R: A Language and Environment for Statistical Computing

Reference Index

The R Development Core Team

Version 2.2.0 (2005-10-06)
## Contents

1 The **base** package .......................................................... 1
   .Device ................................................................. 1
   .Machine .............................................................. 1
   .Platform ............................................................. 3
   .Script ............................................................... 4
   abbreviate ........................................................... 5
   abs ................................................................... 6
   agrep ................................................................. 7
   all ................................................................... 8
   all.equal ............................................................ 9
   all.names .......................................................... 10
   any .................................................................. 11
   aperm .............................................................. 12
   append ............................................................. 13
   apply .............................................................. 13
   args ............................................................... 15
   Arithmetic .......................................................... 16
   array ............................................................... 17
   as.data.frame ........................................................ 18
   as.environment ...................................................... 19
   as.function ........................................................ 20
   as.POSIX* ........................................................... 21
   AsIs ................................................................. 22
   assign .............................................................. 23
   assignOps .......................................................... 24
   attach .............................................................. 26
   attr ................................................................. 27
   attributes ........................................................ 28
   autoload ........................................................... 29
   backsolve .......................................................... 30
   base-deprecated ................................................... 31
   base-package ....................................................... 32
   basename .......................................................... 32
   Bessel ............................................................... 33
   body ................................................................. 35
   bquote .............................................................. 36
   browser ............................................................ 37
   builtins ............................................................ 37
   by ................................................................. 38
   c ................................................................. 39
   call ............................................................... 40
capabilities ........................................... 41
cat ....................................................... 42
cbind .................................................... 43
char.expand ............................................. 45
character ............................................... 46
charmatch ............................................... 47
chartr .................................................... 48
chol ...................................................... 49
chol2inv .................................................. 51
class ..................................................... 52
col ....................................................... 53
colSums ................................................... 54
commandArgs .......................................... 55
comment .................................................. 56
Comparison .............................................. 57
complex ................................................... 58
conditions ............................................... 59
conflicts .................................................. 63
connections .............................................. 63
Constants ............................................... 69
contributors ............................................ 70
Control ................................................... 70
copyright ............................................... 71
count.fields ............................................. 72
crossprod ............................................... 73
cumsum ................................................... 73
cut ....................................................... 74
cut.POSIXt ............................................... 76
data.class ............................................... 77
data.frame ............................................... 78
data.matrix ............................................. 79
date ....................................................... 80
Dates ..................................................... 81
DateTimeClasses ........................................ 81
dcf ........................................................ 84
debug ..................................................... 85
Defunct ................................................... 86
delayedAssign .......................................... 86
deparse ................................................... 87
deparseOpt .............................................. 89
Deprecated ............................................... 90
det ........................................................ 90
detach .................................................... 91
diag ....................................................... 93
diff ........................................................ 94
difftime .................................................... 95
dim ........................................................ 96
dimnames ............................................... 97
do.call .................................................... 98
double .................................................... 99
dput ....................................................... 100
drop ..................................................... 102
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dump</td>
<td>102</td>
</tr>
<tr>
<td>duplicated</td>
<td>104</td>
</tr>
<tr>
<td>dyn.load</td>
<td>105</td>
</tr>
<tr>
<td>eapply</td>
<td>107</td>
</tr>
<tr>
<td>eigen</td>
<td>108</td>
</tr>
<tr>
<td>encodeString</td>
<td>109</td>
</tr>
<tr>
<td>environment</td>
<td>111</td>
</tr>
<tr>
<td>eval</td>
<td>113</td>
</tr>
<tr>
<td>exists</td>
<td>115</td>
</tr>
<tr>
<td>expand.grid</td>
<td>116</td>
</tr>
<tr>
<td>expression</td>
<td>117</td>
</tr>
<tr>
<td>Extract</td>
<td>118</td>
</tr>
<tr>
<td>Extract.data.frame</td>
<td>121</td>
</tr>
<tr>
<td>Extract.factor</td>
<td>123</td>
</tr>
<tr>
<td>Extremes</td>
<td>124</td>
</tr>
<tr>
<td>factor</td>
<td>126</td>
</tr>
<tr>
<td>file.access</td>
<td>128</td>
</tr>
<tr>
<td>file.choose</td>
<td>129</td>
</tr>
<tr>
<td>file.info</td>
<td>130</td>
</tr>
<tr>
<td>file.path</td>
<td>131</td>
</tr>
<tr>
<td>file.show</td>
<td>131</td>
</tr>
<tr>
<td>files</td>
<td>132</td>
</tr>
<tr>
<td>findInterval</td>
<td>134</td>
</tr>
<tr>
<td>force</td>
<td>135</td>
</tr>
<tr>
<td>Foreign</td>
<td>136</td>
</tr>
<tr>
<td>formals</td>
<td>139</td>
</tr>
<tr>
<td>format</td>
<td>140</td>
</tr>
<tr>
<td>format.Date</td>
<td>142</td>
</tr>
<tr>
<td>format.info</td>
<td>144</td>
</tr>
<tr>
<td>format.pval</td>
<td>145</td>
</tr>
<tr>
<td>formatC</td>
<td>146</td>
</tr>
<tr>
<td>formatDL</td>
<td>148</td>
</tr>
<tr>
<td>function</td>
<td>149</td>
</tr>
<tr>
<td>gc</td>
<td>150</td>
</tr>
<tr>
<td>gc.time</td>
<td>151</td>
</tr>
<tr>
<td>gctorture</td>
<td>152</td>
</tr>
<tr>
<td>get</td>
<td>152</td>
</tr>
<tr>
<td>getCallingDLL</td>
<td>154</td>
</tr>
<tr>
<td>getDLLRegisteredRoutines</td>
<td>155</td>
</tr>
<tr>
<td>getLoadedDLLs</td>
<td>156</td>
</tr>
<tr>
<td>getNativeSymbolInfo</td>
<td>157</td>
</tr>
<tr>
<td>getNumCConverters</td>
<td>158</td>
</tr>
<tr>
<td>getpid</td>
<td>160</td>
</tr>
<tr>
<td>gettext</td>
<td>160</td>
</tr>
<tr>
<td>getwd</td>
<td>162</td>
</tr>
<tr>
<td>gl</td>
<td>162</td>
</tr>
<tr>
<td>grep</td>
<td>163</td>
</tr>
<tr>
<td>groupGeneric</td>
<td>163</td>
</tr>
<tr>
<td>gzcon</td>
<td>166</td>
</tr>
<tr>
<td>Hyperbolic</td>
<td>169</td>
</tr>
<tr>
<td>iconv</td>
<td>170</td>
</tr>
<tr>
<td>identical</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>172</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>ifelse</td>
<td>174</td>
</tr>
<tr>
<td>integer</td>
<td>175</td>
</tr>
<tr>
<td>interaction</td>
<td>176</td>
</tr>
<tr>
<td>interactive</td>
<td>177</td>
</tr>
<tr>
<td>Internal</td>
<td>177</td>
</tr>
<tr>
<td>InternalMethods</td>
<td>178</td>
</tr>
<tr>
<td>invisible</td>
<td>178</td>
</tr>
<tr>
<td>is.finite</td>
<td>179</td>
</tr>
<tr>
<td>is.function</td>
<td>180</td>
</tr>
<tr>
<td>is.language</td>
<td>181</td>
</tr>
<tr>
<td>is.object</td>
<td>182</td>
</tr>
<tr>
<td>is.R</td>
<td>182</td>
</tr>
<tr>
<td>is.recursive</td>
<td>183</td>
</tr>
<tr>
<td>is.single</td>
<td>184</td>
</tr>
<tr>
<td>jitter</td>
<td>184</td>
</tr>
<tr>
<td>kappa</td>
<td>186</td>
</tr>
<tr>
<td>kronecker</td>
<td>187</td>
</tr>
<tr>
<td>l10n_info</td>
<td>188</td>
</tr>
<tr>
<td>labels</td>
<td>189</td>
</tr>
<tr>
<td>lapply</td>
<td>189</td>
</tr>
<tr>
<td>Last.value</td>
<td>191</td>
</tr>
<tr>
<td>length</td>
<td>191</td>
</tr>
<tr>
<td>levels</td>
<td>192</td>
</tr>
<tr>
<td>libPaths</td>
<td>194</td>
</tr>
<tr>
<td>library</td>
<td>195</td>
</tr>
<tr>
<td>library.dynam</td>
<td>198</td>
</tr>
<tr>
<td>license</td>
<td>200</td>
</tr>
<tr>
<td>list</td>
<td>200</td>
</tr>
<tr>
<td>list.files</td>
<td>202</td>
</tr>
<tr>
<td>load</td>
<td>203</td>
</tr>
<tr>
<td>localeconv</td>
<td>204</td>
</tr>
<tr>
<td>locales</td>
<td>205</td>
</tr>
<tr>
<td>log</td>
<td>206</td>
</tr>
<tr>
<td>Logic</td>
<td>208</td>
</tr>
<tr>
<td>logical</td>
<td>209</td>
</tr>
<tr>
<td>lower.tri</td>
<td>210</td>
</tr>
<tr>
<td>ls</td>
<td>211</td>
</tr>
<tr>
<td>make.names</td>
<td>212</td>
</tr>
<tr>
<td>make.unique</td>
<td>213</td>
</tr>
<tr>
<td>manglePackageName</td>
<td>214</td>
</tr>
<tr>
<td>mapply</td>
<td>214</td>
</tr>
<tr>
<td>margin.table</td>
<td>215</td>
</tr>
<tr>
<td>mat.or.vec</td>
<td>216</td>
</tr>
<tr>
<td>match</td>
<td>216</td>
</tr>
<tr>
<td>match.arg</td>
<td>218</td>
</tr>
<tr>
<td>match.call</td>
<td>219</td>
</tr>
<tr>
<td>match.fun</td>
<td>220</td>
</tr>
<tr>
<td>matmult</td>
<td>221</td>
</tr>
<tr>
<td>matrix</td>
<td>222</td>
</tr>
<tr>
<td>maxCol</td>
<td>223</td>
</tr>
<tr>
<td>mean</td>
<td>224</td>
</tr>
<tr>
<td>Memory</td>
<td>225</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>v</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---</td>
</tr>
<tr>
<td>Memory-limits</td>
<td>227</td>
</tr>
<tr>
<td>memory.profile</td>
<td>227</td>
</tr>
<tr>
<td>merge</td>
<td>228</td>
</tr>
<tr>
<td>message</td>
<td>230</td>
</tr>
<tr>
<td>missing</td>
<td>230</td>
</tr>
<tr>
<td>mode</td>
<td>231</td>
</tr>
<tr>
<td>NA</td>
<td>233</td>
</tr>
<tr>
<td>name</td>
<td>234</td>
</tr>
<tr>
<td>names</td>
<td>235</td>
</tr>
<tr>
<td>nargs</td>
<td>236</td>
</tr>
<tr>
<td>nchar</td>
<td>237</td>
</tr>
<tr>
<td>nlevels</td>
<td>238</td>
</tr>
<tr>
<td>noquote</td>
<td>239</td>
</tr>
<tr>
<td>NotYet</td>
<td>240</td>
</tr>
<tr>
<td>nrow</td>
<td>240</td>
</tr>
<tr>
<td>ns-dblcolon</td>
<td>241</td>
</tr>
<tr>
<td>ns-hooks</td>
<td>242</td>
</tr>
<tr>
<td>ns-load</td>
<td>243</td>
</tr>
<tr>
<td>ns-topenv</td>
<td>244</td>
</tr>
<tr>
<td>NULL</td>
<td>245</td>
</tr>
<tr>
<td>numeric</td>
<td>245</td>
</tr>
<tr>
<td>NumericConstants</td>
<td>246</td>
</tr>
<tr>
<td>octmode</td>
<td>247</td>
</tr>
<tr>
<td>on.exit</td>
<td>248</td>
</tr>
<tr>
<td>options</td>
<td>249</td>
</tr>
<tr>
<td>order</td>
<td>254</td>
</tr>
<tr>
<td>outer</td>
<td>256</td>
</tr>
<tr>
<td>package-version</td>
<td>277</td>
</tr>
<tr>
<td>Paren</td>
<td>258</td>
</tr>
<tr>
<td>parse</td>
<td>258</td>
</tr>
<tr>
<td>paste</td>
<td>259</td>
</tr>
<tr>
<td>path.expand</td>
<td>260</td>
</tr>
<tr>
<td>pmatch</td>
<td>261</td>
</tr>
<tr>
<td>polyroot</td>
<td>262</td>
</tr>
<tr>
<td>pos.to.env</td>
<td>263</td>
</tr>
<tr>
<td>pretty</td>
<td>263</td>
</tr>
<tr>
<td>Primitive</td>
<td>265</td>
</tr>
<tr>
<td>print</td>
<td>265</td>
</tr>
<tr>
<td>print.data.frame</td>
<td>267</td>
</tr>
<tr>
<td>print.default</td>
<td>267</td>
</tr>
<tr>
<td>prmatrix</td>
<td>269</td>
</tr>
<tr>
<td>proc.time</td>
<td>270</td>
</tr>
<tr>
<td>prod</td>
<td>271</td>
</tr>
<tr>
<td>prop.table</td>
<td>272</td>
</tr>
<tr>
<td>pushBack</td>
<td>272</td>
</tr>
<tr>
<td>qr</td>
<td>273</td>
</tr>
<tr>
<td>QR.Auxiliaries</td>
<td>276</td>
</tr>
<tr>
<td>quit</td>
<td>277</td>
</tr>
<tr>
<td>Quotes</td>
<td>278</td>
</tr>
<tr>
<td>R.home</td>
<td>279</td>
</tr>
<tr>
<td>R.Version</td>
<td>280</td>
</tr>
<tr>
<td>r2dtable</td>
<td>281</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
</tr>
<tr>
<td>Random</td>
<td>282</td>
</tr>
<tr>
<td>Random.user</td>
<td>285</td>
</tr>
<tr>
<td>range</td>
<td>287</td>
</tr>
<tr>
<td>rank</td>
<td>288</td>
</tr>
<tr>
<td>raw</td>
<td>289</td>
</tr>
<tr>
<td>rawConversion</td>
<td>290</td>
</tr>
<tr>
<td>RdUtils</td>
<td>291</td>
</tr>
<tr>
<td>read.table</td>
<td>292</td>
</tr>
<tr>
<td>readBin</td>
<td>295</td>
</tr>
<tr>
<td>readChar</td>
<td>298</td>
</tr>
<tr>
<td>readline</td>
<td>299</td>
</tr>
<tr>
<td>readLines</td>
<td>300</td>
</tr>
<tr>
<td>real</td>
<td>301</td>
</tr>
<tr>
<td>Recall</td>
<td>302</td>
</tr>
<tr>
<td>reg.finalizer</td>
<td>302</td>
</tr>
<tr>
<td>regex</td>
<td>303</td>
</tr>
<tr>
<td>remove</td>
<td>307</td>
</tr>
<tr>
<td>rep</td>
<td>308</td>
</tr>
<tr>
<td>replace</td>
<td>310</td>
</tr>
<tr>
<td>rev</td>
<td>310</td>
</tr>
<tr>
<td>rle</td>
<td>311</td>
</tr>
<tr>
<td>Round</td>
<td>312</td>
</tr>
<tr>
<td>round.POSIXt</td>
<td>313</td>
</tr>
<tr>
<td>row</td>
<td>314</td>
</tr>
<tr>
<td>row.names</td>
<td>315</td>
</tr>
<tr>
<td>row/collnames</td>
<td>316</td>
</tr>
<tr>
<td>rowsum</td>
<td>317</td>
</tr>
<tr>
<td>sample</td>
<td>318</td>
</tr>
<tr>
<td>save</td>
<td>319</td>
</tr>
<tr>
<td>scale</td>
<td>321</td>
</tr>
<tr>
<td>scan</td>
<td>322</td>
</tr>
<tr>
<td>search</td>
<td>325</td>
</tr>
<tr>
<td>seek</td>
<td>326</td>
</tr>
<tr>
<td>seq</td>
<td>327</td>
</tr>
<tr>
<td>seq.Date</td>
<td>329</td>
</tr>
<tr>
<td>seq.POSIXt</td>
<td>330</td>
</tr>
<tr>
<td>sequence</td>
<td>331</td>
</tr>
<tr>
<td>sets</td>
<td>332</td>
</tr>
<tr>
<td>shell</td>
<td>333</td>
</tr>
<tr>
<td>shell.exec</td>
<td>334</td>
</tr>
<tr>
<td>showConnections</td>
<td>335</td>
</tr>
<tr>
<td>shQuote</td>
<td>336</td>
</tr>
<tr>
<td>sign</td>
<td>337</td>
</tr>
<tr>
<td>sink</td>
<td>338</td>
</tr>
<tr>
<td>slice.index</td>
<td>339</td>
</tr>
<tr>
<td>slotOp</td>
<td>340</td>
</tr>
<tr>
<td>socketSelect</td>
<td>341</td>
</tr>
<tr>
<td>solve</td>
<td>341</td>
</tr>
<tr>
<td>sort</td>
<td>343</td>
</tr>
<tr>
<td>source</td>
<td>345</td>
</tr>
<tr>
<td>Special</td>
<td>346</td>
</tr>
<tr>
<td>split</td>
<td>348</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>sprintf</td>
<td>350</td>
</tr>
<tr>
<td>sQuote</td>
<td>353</td>
</tr>
<tr>
<td>stack</td>
<td>354</td>
</tr>
<tr>
<td>Startup</td>
<td>355</td>
</tr>
<tr>
<td>stop</td>
<td>357</td>
</tr>
<tr>
<td>stopifnot</td>
<td>358</td>
</tr>
<tr>
<td>strftime</td>
<td>359</td>
</tr>
<tr>
<td>strsplit</td>
<td>363</td>
</tr>
<tr>
<td>strtrim</td>
<td>364</td>
</tr>
<tr>
<td>structure</td>
<td>365</td>
</tr>
<tr>
<td>strwrap</td>
<td>365</td>
</tr>
<tr>
<td>subset</td>
<td>366</td>
</tr>
<tr>
<td>substitute</td>
<td>368</td>
</tr>
<tr>
<td>substr</td>
<td>369</td>
</tr>
<tr>
<td>sum</td>
<td>371</td>
</tr>
<tr>
<td>summary</td>
<td>371</td>
</tr>
<tr>
<td>svd</td>
<td>373</td>
</tr>
<tr>
<td>sweep</td>
<td>374</td>
</tr>
<tr>
<td>switch</td>
<td>375</td>
</tr>
<tr>
<td>Syntax</td>
<td>376</td>
</tr>
<tr>
<td>Sys.getenv</td>
<td>377</td>
</tr>
<tr>
<td>Sys.info</td>
<td>378</td>
</tr>
<tr>
<td>sys.parent</td>
<td>379</td>
</tr>
<tr>
<td>Sys.putenv</td>
<td>381</td>
</tr>
<tr>
<td>Sys.sleep</td>
<td>382</td>
</tr>
<tr>
<td>sys.source</td>
<td>383</td>
</tr>
<tr>
<td>Sys.time</td>
<td>384</td>
</tr>
<tr>
<td>system</td>
<td>385</td>
</tr>
<tr>
<td>system.file</td>
<td>386</td>
</tr>
<tr>
<td>system.time</td>
<td>387</td>
</tr>
<tr>
<td>t</td>
<td>388</td>
</tr>
<tr>
<td>table</td>
<td>389</td>
</tr>
<tr>
<td>tabulate</td>
<td>391</td>
</tr>
<tr>
<td>tapply</td>
<td>392</td>
</tr>
<tr>
<td>taskCallback</td>
<td>393</td>
</tr>
<tr>
<td>taskCallbackManager</td>
<td>395</td>
</tr>
<tr>
<td>taskCallbackNames</td>
<td>397</td>
</tr>
<tr>
<td>tempfile</td>
<td>398</td>
</tr>
<tr>
<td>textConnection</td>
<td>399</td>
</tr>
<tr>
<td>tilde</td>
<td>400</td>
</tr>
<tr>
<td>toString</td>
<td>401</td>
</tr>
<tr>
<td>trace</td>
<td>402</td>
</tr>
<tr>
<td>traceback</td>
<td>405</td>
</tr>
<tr>
<td>transform</td>
<td>406</td>
</tr>
<tr>
<td>Trig</td>
<td>407</td>
</tr>
<tr>
<td>try</td>
<td>408</td>
</tr>
<tr>
<td>type.convert</td>
<td>409</td>
</tr>
<tr>
<td>typeof</td>
<td>410</td>
</tr>
<tr>
<td>unique</td>
<td>411</td>
</tr>
<tr>
<td>unlink</td>
<td>412</td>
</tr>
<tr>
<td>unlist</td>
<td>413</td>
</tr>
<tr>
<td>unname</td>
<td>414</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>UseMethod</td>
<td>415</td>
</tr>
<tr>
<td>UserHooks</td>
<td>416</td>
</tr>
<tr>
<td>utf8Conversion</td>
<td>418</td>
</tr>
<tr>
<td>vector</td>
<td>418</td>
</tr>
<tr>
<td>warning</td>
<td>420</td>
</tr>
<tr>
<td>warnings</td>
<td>421</td>
</tr>
<tr>
<td>weekdays</td>
<td>422</td>
</tr>
<tr>
<td>which</td>
<td>423</td>
</tr>
<tr>
<td>which.min</td>
<td>424</td>
</tr>
<tr>
<td>with</td>
<td>425</td>
</tr>
<tr>
<td>write</td>
<td>426</td>
</tr>
<tr>
<td>write.table</td>
<td>427</td>
</tr>
<tr>
<td>writeLines</td>
<td>429</td>
</tr>
<tr>
<td>zip.file.extract</td>
<td>430</td>
</tr>
<tr>
<td>zpackages</td>
<td>431</td>
</tr>
<tr>
<td>zutils</td>
<td>432</td>
</tr>
</tbody>
</table>

2 The datasets package

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ability.cov</td>
<td>433</td>
</tr>
<tr>
<td>airmiles</td>
<td>434</td>
</tr>
<tr>
<td>AirPassengers</td>
<td>434</td>
</tr>
<tr>
<td>airquality</td>
<td>435</td>
</tr>
<tr>
<td>anscombe</td>
<td>436</td>
</tr>
<tr>
<td>attenu</td>
<td>437</td>
</tr>
<tr>
<td>attitude</td>
<td>438</td>
</tr>
<tr>
<td>austres</td>
<td>439</td>
</tr>
<tr>
<td>beavers</td>
<td>439</td>
</tr>
<tr>
<td>BJsales</td>
<td>441</td>
</tr>
<tr>
<td>BOD</td>
<td>441</td>
</tr>
<tr>
<td>cars</td>
<td>442</td>
</tr>
<tr>
<td>ChickWeight</td>
<td>443</td>
</tr>
<tr>
<td>chickwts</td>
<td>444</td>
</tr>
<tr>
<td>CO2</td>
<td>445</td>
</tr>
<tr>
<td>co2</td>
<td>446</td>
</tr>
<tr>
<td>datasets-package</td>
<td>446</td>
</tr>
<tr>
<td>discoveries</td>
<td>447</td>
</tr>
<tr>
<td>DNase</td>
<td>447</td>
</tr>
<tr>
<td>esoph</td>
<td>448</td>
</tr>
<tr>
<td>euro</td>
<td>449</td>
</tr>
<tr>
<td>eurodist</td>
<td>450</td>
</tr>
<tr>
<td>faithful</td>
<td>451</td>
</tr>
<tr>
<td>Formaldehyde</td>
<td>452</td>
</tr>
<tr>
<td>freeny</td>
<td>453</td>
</tr>
<tr>
<td>HairEyeColor</td>
<td>454</td>
</tr>
<tr>
<td>Harman23.cor</td>
<td>455</td>
</tr>
<tr>
<td>Harman74.cor</td>
<td>455</td>
</tr>
<tr>
<td>Indometh</td>
<td>456</td>
</tr>
<tr>
<td>infert</td>
<td>457</td>
</tr>
<tr>
<td>InsectSprays</td>
<td>458</td>
</tr>
<tr>
<td>iris</td>
<td>458</td>
</tr>
<tr>
<td>islands</td>
<td>460</td>
</tr>
<tr>
<td>JohnsonJohnson</td>
<td>460</td>
</tr>
<tr>
<td>LakeHuron</td>
<td>461</td>
</tr>
<tr>
<td>Contents</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>3 The <em>grDevices</em> package</td>
<td>499</td>
</tr>
<tr>
<td>boxplot.stats</td>
<td>499</td>
</tr>
<tr>
<td>bringToTop</td>
<td>501</td>
</tr>
<tr>
<td>check.options</td>
<td>501</td>
</tr>
<tr>
<td>chull</td>
<td>502</td>
</tr>
<tr>
<td>cm</td>
<td>503</td>
</tr>
<tr>
<td>col2rgb</td>
<td>504</td>
</tr>
<tr>
<td>colors</td>
<td>504</td>
</tr>
<tr>
<td>contourLines</td>
<td>504</td>
</tr>
<tr>
<td>convertColor</td>
<td>505</td>
</tr>
<tr>
<td>dev.interactive</td>
<td>506</td>
</tr>
<tr>
<td>dev.xxx</td>
<td>506</td>
</tr>
<tr>
<td>dev2</td>
<td>507</td>
</tr>
<tr>
<td>dev2bitmap</td>
<td>507</td>
</tr>
<tr>
<td>Devices</td>
<td>509</td>
</tr>
<tr>
<td>extendrange</td>
<td>509</td>
</tr>
<tr>
<td>getGraphicsEvent</td>
<td>510</td>
</tr>
<tr>
<td>gray</td>
<td>510</td>
</tr>
<tr>
<td>gray.colors</td>
<td>510</td>
</tr>
<tr>
<td>hcl</td>
<td>511</td>
</tr>
<tr>
<td>Hershey</td>
<td>512</td>
</tr>
<tr>
<td>hsv</td>
<td>512</td>
</tr>
<tr>
<td>Japanese</td>
<td>513</td>
</tr>
<tr>
<td>make.rgb</td>
<td>514</td>
</tr>
<tr>
<td>nclass</td>
<td>514</td>
</tr>
<tr>
<td>palette</td>
<td>516</td>
</tr>
<tr>
<td>Palettes</td>
<td>516</td>
</tr>
<tr>
<td>pdf</td>
<td>517</td>
</tr>
<tr>
<td>pictex</td>
<td>517</td>
</tr>
<tr>
<td>plotmath</td>
<td>518</td>
</tr>
<tr>
<td>png</td>
<td>518</td>
</tr>
<tr>
<td>postscript</td>
<td>519</td>
</tr>
<tr>
<td>postscriptFonts</td>
<td>519</td>
</tr>
<tr>
<td>recordGraphics</td>
<td>520</td>
</tr>
<tr>
<td>recordPlot</td>
<td>520</td>
</tr>
<tr>
<td>rgb</td>
<td>521</td>
</tr>
<tr>
<td>rgb2hsv</td>
<td>521</td>
</tr>
<tr>
<td>savePlot</td>
<td>522</td>
</tr>
<tr>
<td>trans3d</td>
<td>522</td>
</tr>
<tr>
<td>windows</td>
<td>523</td>
</tr>
<tr>
<td>windowsFonts</td>
<td>523</td>
</tr>
<tr>
<td>xfig</td>
<td>524</td>
</tr>
<tr>
<td>xy.coords</td>
<td>524</td>
</tr>
<tr>
<td>xyz.coords</td>
<td>524</td>
</tr>
<tr>
<td>4 The <em>graphics</em> package</td>
<td>559</td>
</tr>
<tr>
<td>abline</td>
<td>559</td>
</tr>
<tr>
<td>arrows</td>
<td>560</td>
</tr>
<tr>
<td>assocplot</td>
<td>561</td>
</tr>
<tr>
<td>axis</td>
<td>562</td>
</tr>
<tr>
<td>axis.POSIXct</td>
<td>565</td>
</tr>
<tr>
<td>aXTicks</td>
<td>566</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>barplot</td>
<td>567</td>
</tr>
<tr>
<td>box</td>
<td>570</td>
</tr>
<tr>
<td>boxplot</td>
<td>571</td>
</tr>
<tr>
<td>bxp</td>
<td>574</td>
</tr>
<tr>
<td>cdplot</td>
<td>576</td>
</tr>
<tr>
<td>contour</td>
<td>578</td>
</tr>
<tr>
<td>coplot</td>
<td>581</td>
</tr>
<tr>
<td>curve</td>
<td>583</td>
</tr>
<tr>
<td>dotchart</td>
<td>585</td>
</tr>
<tr>
<td>filled.contour</td>
<td>586</td>
</tr>
<tr>
<td>fourfoldplot</td>
<td>588</td>
</tr>
<tr>
<td>frame</td>
<td>590</td>
</tr>
<tr>
<td>graphics-package</td>
<td>591</td>
</tr>
<tr>
<td>grid</td>
<td>591</td>
</tr>
<tr>
<td>hist</td>
<td>592</td>
</tr>
<tr>
<td>hist.POSIXt</td>
<td>595</td>
</tr>
<tr>
<td>identify</td>
<td>596</td>
</tr>
<tr>
<td>image</td>
<td>597</td>
</tr>
<tr>
<td>layout</td>
<td>599</td>
</tr>
<tr>
<td>legend</td>
<td>601</td>
</tr>
<tr>
<td>lines</td>
<td>605</td>
</tr>
<tr>
<td>locator</td>
<td>606</td>
</tr>
<tr>
<td>matplot</td>
<td>607</td>
</tr>
<tr>
<td>mosaicplot</td>
<td>610</td>
</tr>
<tr>
<td>mtext</td>
<td>612</td>
</tr>
<tr>
<td>pairs</td>
<td>614</td>
</tr>
<tr>
<td>panel.smooth</td>
<td>616</td>
</tr>
<tr>
<td>par</td>
<td>617</td>
</tr>
<tr>
<td>persp</td>
<td>624</td>
</tr>
<tr>
<td>pie</td>
<td>626</td>
</tr>
<tr>
<td>plot</td>
<td>628</td>
</tr>
<tr>
<td>plot.data.frame</td>
<td>629</td>
</tr>
<tr>
<td>plot.default</td>
<td>630</td>
</tr>
<tr>
<td>plot.design</td>
<td>632</td>
</tr>
<tr>
<td>plot.factor</td>
<td>634</td>
</tr>
<tr>
<td>plot.formula</td>
<td>635</td>
</tr>
<tr>
<td>plot.histogram</td>
<td>636</td>
</tr>
<tr>
<td>plot.table</td>
<td>637</td>
</tr>
<tr>
<td>plot.window</td>
<td>638</td>
</tr>
<tr>
<td>plot.xy</td>
<td>639</td>
</tr>
<tr>
<td>points</td>
<td>640</td>
</tr>
<tr>
<td>polygon</td>
<td>642</td>
</tr>
<tr>
<td>rect</td>
<td>644</td>
</tr>
<tr>
<td>rug</td>
<td>645</td>
</tr>
<tr>
<td>screen</td>
<td>646</td>
</tr>
<tr>
<td>segments</td>
<td>648</td>
</tr>
<tr>
<td>spineplot</td>
<td>649</td>
</tr>
<tr>
<td>stars</td>
<td>652</td>
</tr>
<tr>
<td>stem</td>
<td>654</td>
</tr>
<tr>
<td>stripchart</td>
<td>655</td>
</tr>
<tr>
<td>strwidth</td>
<td>656</td>
</tr>
<tr>
<td>sunflowerplot</td>
<td>658</td>
</tr>
<tr>
<td>text</td>
<td>660</td>
</tr>
<tr>
<td>title</td>
<td>661</td>
</tr>
<tr>
<td>units</td>
<td>663</td>
</tr>
</tbody>
</table>

5 The **grid package** 665

- absolute.size ........................................ 665
- convertNative ........................................ 666
- dataViewport ......................................... 667
- drawDetails .......................................... 668
- editDetails .......................................... 669
- gEdit .................................................. 669
- getNames ............................................. 670
- gpar .................................................... 671
- gPath ................................................... 673
- Grid .................................................... 674
- Grid Viewports ........................................ 674
- grid-package .......................................... 678
- grid.add .............................................. 678
- grid.arrows ........................................... 679
- grid.circle ........................................... 682
- grid.collection ....................................... 683
- grid.convert .......................................... 684
- grid.copy ............................................. 686
- grid.display.list ..................................... 686
- grid.draw .............................................. 687
- grid.edit .............................................. 688
- grid.frame ............................................. 689
- grid.get ................................................ 690
- grid.grab .............................................. 691
- grid.grill ............................................. 693
- grid.grob .............................................. 693
- grid.layout ............................................ 695
- grid.lines ............................................. 697
- grid.locator ........................................... 698
- grid.move.to .......................................... 699
- grid.newpage .......................................... 700
- grid.pack ............................................... 701
- grid.place ............................................. 702
- grid.plot.and.legend .................................. 703
- grid.points ........................................... 704
- grid.polygon .......................................... 705
- grid.pretty ............................................ 706
- grid.promp ............................................. 707
- grid.record ............................................ 707
- grid.rect ................................................ 708
- grid.refresh ........................................... 710
- grid.remove ............................................ 710
- grid.segments ......................................... 711
- grid.set ................................................ 712
- grid.show.layout ...................................... 713
- grid.show.viewport ..................................... 714
- grid.text .............................................. 715
- grid.xaxis ............................................. 717
6 The methods package

.grid.yaxis ................................................................. 719
.grid.yaxis ................................................................. 719
grobWidth ................................................................. 720
.plotViewport .............................................................. 720
.pop.viewport .............................................................. 721
.push.viewport ............................................................ 722
Querying the Viewport Tree ............................................... 722
.stringWidth ................................................................. 724
.unit ......................................................................... 724
.unit.c ...................................................................... 726
.unit.length ................................................................. 727
.unit.pmin ................................................................. 727
.unit.rep ................................................................. 728
.validDetails .............................................................. 729
.vpPath ................................................................... 730
.widthDetails ............................................................. 731
Working with Viewports .................................................. 731

.methods package ........................................................ 735
.as ................................................................. 735
.BasicClasses ............................................................ 739
callNextMethod .......................................................... 740
cbind2 ................................................................. 742
.Classes ................................................................. 743
classRepresentation-class ............................................... 744
.Documentation ............................................................ 746
.environment-class ....................................................... 747
.fixPre1.8 ................................................................. 748
genericFunction-class ..................................................... 749
.GenericFunctions .......................................................... 750
getClass ................................................................. 753
getMethod ................................................................. 754
.getPackageName ............................................................ 757
.hasArg ................................................................. 758
.initialize-methods ....................................................... 759
.is ................................................................. 760
.isSealedMethod .......................................................... 763
.language-class ........................................................... 764
.LinearMethodsList-class ................................................. 765
.makeClassRepresentation ............................................... 766
.MethodDefinition-class .................................................. 767
.MethodsWithNext-class .................................................. 772
.new ................................................................. 773
.ObjectsWithPackage-class ............................................... 774
.promptClass ............................................................... 775
.promptMethods ........................................................... 777
.representation ........................................................... 778
.SClassExtension-class .................................................... 779
.setClass ............................................................... 780
.setClassUnion .......................................................... 784
7 The stats package

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>.checkMFClasses</td>
<td>805</td>
</tr>
<tr>
<td>acf</td>
<td>805</td>
</tr>
<tr>
<td>acf2AR</td>
<td>806</td>
</tr>
<tr>
<td>add1</td>
<td>808</td>
</tr>
<tr>
<td>addmargins</td>
<td>808</td>
</tr>
<tr>
<td>aggregate</td>
<td>810</td>
</tr>
<tr>
<td>AIC</td>
<td>812</td>
</tr>
<tr>
<td>alias</td>
<td>814</td>
</tr>
<tr>
<td>anova</td>
<td>815</td>
</tr>
<tr>
<td>anova.glm</td>
<td>816</td>
</tr>
<tr>
<td>anova.lm</td>
<td>817</td>
</tr>
<tr>
<td>anova.mlm</td>
<td>819</td>
</tr>
<tr>
<td>ansari.test</td>
<td>820</td>
</tr>
<tr>
<td>aov</td>
<td>822</td>
</tr>
<tr>
<td>approxfun</td>
<td>824</td>
</tr>
<tr>
<td>ar</td>
<td>826</td>
</tr>
<tr>
<td>ar.ols</td>
<td>828</td>
</tr>
<tr>
<td>arima</td>
<td>831</td>
</tr>
<tr>
<td>arima.sim</td>
<td>833</td>
</tr>
<tr>
<td>arima0</td>
<td>836</td>
</tr>
<tr>
<td>ARMAtocf</td>
<td>837</td>
</tr>
<tr>
<td>ARMAtoMA</td>
<td>840</td>
</tr>
<tr>
<td>as.hclust</td>
<td>841</td>
</tr>
<tr>
<td>asOneSidedFormula</td>
<td>842</td>
</tr>
<tr>
<td>ave</td>
<td>843</td>
</tr>
<tr>
<td>bandwidth</td>
<td>844</td>
</tr>
<tr>
<td>bartlett.test</td>
<td>845</td>
</tr>
<tr>
<td>Beta</td>
<td>846</td>
</tr>
<tr>
<td>binom.test</td>
<td>847</td>
</tr>
<tr>
<td>Binomial</td>
<td>849</td>
</tr>
<tr>
<td>biplot</td>
<td>850</td>
</tr>
<tr>
<td>biplot.princomp</td>
<td>851</td>
</tr>
<tr>
<td>birthday</td>
<td>853</td>
</tr>
<tr>
<td>Box.test</td>
<td>854</td>
</tr>
<tr>
<td>C</td>
<td>855</td>
</tr>
<tr>
<td>cancro</td>
<td>856</td>
</tr>
<tr>
<td>case/variable.names</td>
<td>857</td>
</tr>
<tr>
<td>Cauchy</td>
<td>858</td>
</tr>
<tr>
<td>chisq.test</td>
<td>859</td>
</tr>
<tr>
<td>Chisquare</td>
<td>860</td>
</tr>
<tr>
<td>clearNames</td>
<td>862</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
</tr>
<tr>
<td>cmdscale</td>
<td>865</td>
</tr>
<tr>
<td>coef</td>
<td>867</td>
</tr>
<tr>
<td>complete.cases</td>
<td>868</td>
</tr>
<tr>
<td>confint</td>
<td>868</td>
</tr>
<tr>
<td>constrOptim</td>
<td>869</td>
</tr>
<tr>
<td>contrast</td>
<td>871</td>
</tr>
<tr>
<td>contrasts</td>
<td>873</td>
</tr>
<tr>
<td>convolve</td>
<td>874</td>
</tr>
<tr>
<td>cophenetic</td>
<td>875</td>
</tr>
<tr>
<td>cor</td>
<td>876</td>
</tr>
<tr>
<td>cor.test</td>
<td>879</td>
</tr>
<tr>
<td>cov.wt</td>
<td>881</td>
</tr>
<tr>
<td>cpgram</td>
<td>882</td>
</tr>
<tr>
<td>cutree</td>
<td>883</td>
</tr>
<tr>
<td>decompose</td>
<td>884</td>
</tr>
<tr>
<td>delete.response</td>
<td>885</td>
</tr>
<tr>
<td>dendrapply</td>
<td>886</td>
</tr>
<tr>
<td>dendrogram</td>
<td>887</td>
</tr>
<tr>
<td>density</td>
<td>890</td>
</tr>
<tr>
<td>deriv</td>
<td>893</td>
</tr>
<tr>
<td>deviance</td>
<td>896</td>
</tr>
<tr>
<td>df.residual</td>
<td>896</td>
</tr>
<tr>
<td>diffinv</td>
<td>897</td>
</tr>
<tr>
<td>dist</td>
<td>898</td>
</tr>
<tr>
<td>dummy.coef</td>
<td>901</td>
</tr>
<tr>
<td>ecdf</td>
<td>902</td>
</tr>
<tr>
<td>eff.aovlist</td>
<td>904</td>
</tr>
<tr>
<td>effects</td>
<td>905</td>
</tr>
<tr>
<td>embed</td>
<td>906</td>
</tr>
<tr>
<td>expand.model.frame</td>
<td>907</td>
</tr>
<tr>
<td>Exponential</td>
<td>908</td>
</tr>
<tr>
<td>factanal</td>
<td>909</td>
</tr>
<tr>
<td>factor.scope</td>
<td>912</td>
</tr>
<tr>
<td>family</td>
<td>913</td>
</tr>
<tr>
<td>FDist</td>
<td>915</td>
</tr>
<tr>
<td>fft</td>
<td>916</td>
</tr>
<tr>
<td>filter</td>
<td>917</td>
</tr>
<tr>
<td>fisher.test</td>
<td>919</td>
</tr>
<tr>
<td>fitted</td>
<td>921</td>
</tr>
<tr>
<td>five.num</td>
<td>922</td>
</tr>
<tr>
<td>fligner.test</td>
<td>922</td>
</tr>
<tr>
<td>formula</td>
<td>924</td>
</tr>
<tr>
<td>formula.nls</td>
<td>926</td>
</tr>
<tr>
<td>friedman.test</td>
<td>926</td>
</tr>
<tr>
<td>ftable</td>
<td>928</td>
</tr>
<tr>
<td>ftable.formula</td>
<td>930</td>
</tr>
<tr>
<td>GammaDist</td>
<td>931</td>
</tr>
<tr>
<td>Geometric</td>
<td>933</td>
</tr>
<tr>
<td>getInitial</td>
<td>934</td>
</tr>
<tr>
<td>glm</td>
<td>935</td>
</tr>
<tr>
<td>glm.control</td>
<td>939</td>
</tr>
<tr>
<td>glm.summaries</td>
<td>940</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
</tr>
<tr>
<td>hclust</td>
<td>941</td>
</tr>
<tr>
<td>heatmap</td>
<td>944</td>
</tr>
<tr>
<td>HoltWinters</td>
<td>947</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>949</td>
</tr>
<tr>
<td>identify.hclust</td>
<td>950</td>
</tr>
<tr>
<td>influence.measures</td>
<td>951</td>
</tr>
<tr>
<td>integrate</td>
<td>954</td>
</tr>
<tr>
<td>interaction.plot</td>
<td>956</td>
</tr>
<tr>
<td>IQR</td>
<td>958</td>
</tr>
<tr>
<td>is.empty.model</td>
<td>959</td>
</tr>
<tr>
<td>isoreg</td>
<td>959</td>
</tr>
<tr>
<td>KalmanLike</td>
<td>961</td>
</tr>
<tr>
<td>kernapply</td>
<td>962</td>
</tr>
<tr>
<td>kernel</td>
<td>963</td>
</tr>
<tr>
<td>kmeans</td>
<td>965</td>
</tr>
<tr>
<td>kruskal.test</td>
<td>966</td>
</tr>
<tr>
<td>ks.test</td>
<td>968</td>
</tr>
<tr>
<td>ksmooth</td>
<td>969</td>
</tr>
<tr>
<td>lag</td>
<td>970</td>
</tr>
<tr>
<td>lag.plot</td>
<td>971</td>
</tr>
<tr>
<td>line</td>
<td>973</td>
</tr>
<tr>
<td>lm</td>
<td>973</td>
</tr>
<tr>
<td>lm.fit</td>
<td>976</td>
</tr>
<tr>
<td>lm.influence</td>
<td>978</td>
</tr>
<tr>
<td>lm.summaries</td>
<td>979</td>
</tr>
<tr>
<td>loadings</td>
<td>981</td>
</tr>
<tr>
<td>loess</td>
<td>981</td>
</tr>
<tr>
<td>loess.control</td>
<td>983</td>
</tr>
<tr>
<td>Logistic</td>
<td>984</td>
</tr>
<tr>
<td>logLik</td>
<td>985</td>
</tr>
<tr>
<td>loglin</td>
<td>987</td>
</tr>
<tr>
<td>Lognormal</td>
<td>988</td>
</tr>
<tr>
<td>lowess</td>
<td>990</td>
</tr>
<tr>
<td>ls.diag</td>
<td>991</td>
</tr>
<tr>
<td>ls.print</td>
<td>992</td>
</tr>
<tr>
<td>lsfit</td>
<td>992</td>
</tr>
<tr>
<td>mad</td>
<td>994</td>
</tr>
<tr>
<td>mahalanobis</td>
<td>995</td>
</tr>
<tr>
<td>make.link</td>
<td>996</td>
</tr>
<tr>
<td>makepredictcall</td>
<td>996</td>
</tr>
<tr>
<td>manova</td>
<td>997</td>
</tr>
<tr>
<td>mantelhaen.test</td>
<td>998</td>
</tr>
<tr>
<td>mauchley.test</td>
<td>1000</td>
</tr>
<tr>
<td>mcnemar.test</td>
<td>1002</td>
</tr>
<tr>
<td>median</td>
<td>1003</td>
</tr>
<tr>
<td>medpolish</td>
<td>1004</td>
</tr>
<tr>
<td>model.extract</td>
<td>1005</td>
</tr>
<tr>
<td>model.frame</td>
<td>1006</td>
</tr>
<tr>
<td>model.matrix</td>
<td>1008</td>
</tr>
<tr>
<td>model.tables</td>
<td>1010</td>
</tr>
<tr>
<td>monthplot</td>
<td>1011</td>
</tr>
<tr>
<td>mood.test</td>
<td>1013</td>
</tr>
<tr>
<td>CONTENTS</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>Multinomial</td>
<td>1014</td>
</tr>
<tr>
<td>na.action</td>
<td>1016</td>
</tr>
<tr>
<td>na.contiguous</td>
<td>1016</td>
</tr>
<tr>
<td>na.fail</td>
<td>1017</td>
</tr>
<tr>
<td>naprint</td>
<td>1018</td>
</tr>
<tr>
<td>naresid</td>
<td>1018</td>
</tr>
<tr>
<td>NegBinomial</td>
<td>1019</td>
</tr>
<tr>
<td>nextn</td>
<td>1021</td>
</tr>
<tr>
<td>nlm</td>
<td>1022</td>
</tr>
<tr>
<td>nlminb</td>
<td>1024</td>
</tr>
<tr>
<td>nls</td>
<td>1026</td>
</tr>
<tr>
<td>nls.control</td>
<td>1029</td>
</tr>
<tr>
<td>nlsModel</td>
<td>1030</td>
</tr>
<tr>
<td>NLSstAsymptotic</td>
<td>1031</td>
</tr>
<tr>
<td>NLSstClosestX</td>
<td>1032</td>
</tr>
<tr>
<td>NLSstLfAsymptote</td>
<td>1033</td>
</tr>
<tr>
<td>NLSstRtAsymptote</td>
<td>1033</td>
</tr>
<tr>
<td>Normal</td>
<td>1034</td>
</tr>
<tr>
<td>numericDeriv</td>
<td>1036</td>
</tr>
<tr>
<td>offset</td>
<td>1036</td>
</tr>
<tr>
<td>oneway.test</td>
<td>1037</td>
</tr>
<tr>
<td>optim</td>
<td>1038</td>
</tr>
<tr>
<td>optimize</td>
<td>1043</td>
</tr>
<tr>
<td>order.dendrogram</td>
<td>1045</td>
</tr>
<tr>
<td>p.adjust</td>
<td>1046</td>
</tr>
<tr>
<td>pairwise.prop.test</td>
<td>1048</td>
</tr>
<tr>
<td>pairwise.t.test</td>
<td>1049</td>
</tr>
<tr>
<td>pairwise.table</td>
<td>1049</td>
</tr>
<tr>
<td>pairwise.wilcox.test</td>
<td>1050</td>
</tr>
<tr>
<td>plot.acf</td>
<td>1051</td>
</tr>
<tr>
<td>plot.density</td>
<td>1052</td>
</tr>
<tr>
<td>plot.HoltWinters</td>
<td>1053</td>
</tr>
<tr>
<td>plot.isoreg</td>
<td>1054</td>
</tr>
<tr>
<td>plot.lm</td>
<td>1055</td>
</tr>
<tr>
<td>plot.ppr</td>
<td>1057</td>
</tr>
<tr>
<td>plot.profile.nls</td>
<td>1058</td>
</tr>
<tr>
<td>plot.spec</td>
<td>1059</td>
</tr>
<tr>
<td>plot.stepfun</td>
<td>1060</td>
</tr>
<tr>
<td>plot.ts</td>
<td>1062</td>
</tr>
<tr>
<td>Poisson</td>
<td>1063</td>
</tr>
<tr>
<td>power.t.test</td>
<td>1065</td>
</tr>
<tr>
<td>power</td>
<td>1066</td>
</tr>
<tr>
<td>PP.test</td>
<td>1067</td>
</tr>
<tr>
<td>ppoints</td>
<td>1068</td>
</tr>
<tr>
<td>ppr</td>
<td>1069</td>
</tr>
<tr>
<td>pcomp</td>
<td>1072</td>
</tr>
<tr>
<td>predict</td>
<td>1074</td>
</tr>
<tr>
<td>predict.Arima</td>
<td>1075</td>
</tr>
<tr>
<td>predict.glm</td>
<td>1076</td>
</tr>
<tr>
<td>predict.HoltWinters</td>
<td>1078</td>
</tr>
<tr>
<td>predict.lm</td>
<td>1079</td>
</tr>
<tr>
<td>predict.loess</td>
<td>1080</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
</tr>
<tr>
<td>predict.nls</td>
<td>1082</td>
</tr>
<tr>
<td>predict.smooth.spline</td>
<td>1083</td>
</tr>
<tr>
<td>preplot</td>
<td>1084</td>
</tr>
<tr>
<td>princomp</td>
<td>1085</td>
</tr>
<tr>
<td>print.power.htest</td>
<td>1087</td>
</tr>
<tr>
<td>print.ts</td>
<td>1088</td>
</tr>
<tr>
<td>printCoefmat</td>
<td>1089</td>
</tr>
<tr>
<td>profile</td>
<td>1090</td>
</tr>
<tr>
<td>profile.nls</td>
<td>1091</td>
</tr>
<tr>
<td>profiler</td>
<td>1092</td>
</tr>
<tr>
<td>profiler.nls</td>
<td>1093</td>
</tr>
<tr>
<td>proj</td>
<td>1094</td>
</tr>
<tr>
<td>prop.test</td>
<td>1096</td>
</tr>
<tr>
<td>prop.trend.test</td>
<td>1098</td>
</tr>
<tr>
<td>qqnorm</td>
<td>1099</td>
</tr>
<tr>
<td>quade.test</td>
<td>1100</td>
</tr>
<tr>
<td>quantile</td>
<td>1101</td>
</tr>
<tr>
<td>read.ftable</td>
<td>1103</td>
</tr>
<tr>
<td>rect.hclust</td>
<td>1105</td>
</tr>
<tr>
<td>relevel</td>
<td>1106</td>
</tr>
<tr>
<td>reorder</td>
<td>1107</td>
</tr>
<tr>
<td>reorder.factor</td>
<td>1108</td>
</tr>
<tr>
<td>replications</td>
<td>1109</td>
</tr>
<tr>
<td>reshape</td>
<td>1110</td>
</tr>
<tr>
<td>runmed</td>
<td>1112</td>
</tr>
<tr>
<td>scatter.smooth</td>
<td>1114</td>
</tr>
<tr>
<td>screeplot</td>
<td>1115</td>
</tr>
<tr>
<td>sd</td>
<td>1116</td>
</tr>
<tr>
<td>se.contrast</td>
<td>1117</td>
</tr>
<tr>
<td>selfStart</td>
<td>1119</td>
</tr>
<tr>
<td>setNames</td>
<td>1120</td>
</tr>
<tr>
<td>shapiro.test</td>
<td>1121</td>
</tr>
<tr>
<td>SignRank</td>
<td>1122</td>
</tr>
<tr>
<td>simulate</td>
<td>1123</td>
</tr>
<tr>
<td>smooth</td>
<td>1124</td>
</tr>
<tr>
<td>smooth.spline</td>
<td>1126</td>
</tr>
<tr>
<td>smoothEnds</td>
<td>1129</td>
</tr>
<tr>
<td>sortedXyData</td>
<td>1130</td>
</tr>
<tr>
<td>spec.ar</td>
<td>1131</td>
</tr>
<tr>
<td>spec.pgram</td>
<td>1132</td>
</tr>
<tr>
<td>spec.taper</td>
<td>1134</td>
</tr>
<tr>
<td>spectrum</td>
<td>1135</td>
</tr>
<tr>
<td>splinefun</td>
<td>1137</td>
</tr>
<tr>
<td>SSasymp</td>
<td>1138</td>
</tr>
<tr>
<td>SSasympOff</td>
<td>1139</td>
</tr>
<tr>
<td>SSasympOrig</td>
<td>1140</td>
</tr>
<tr>
<td>SSbiexp</td>
<td>1141</td>
</tr>
<tr>
<td>SSD</td>
<td>1142</td>
</tr>
<tr>
<td>SSfol</td>
<td>1143</td>
</tr>
<tr>
<td>SSfpl</td>
<td>1144</td>
</tr>
<tr>
<td>SSGompertz</td>
<td>1145</td>
</tr>
<tr>
<td>SSlogis</td>
<td>1146</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SSmicmen</td>
<td>1147</td>
</tr>
<tr>
<td>SSweibull</td>
<td>1148</td>
</tr>
<tr>
<td>start</td>
<td>1149</td>
</tr>
<tr>
<td>stat.anova</td>
<td>1150</td>
</tr>
<tr>
<td>stats-package</td>
<td>1151</td>
</tr>
<tr>
<td>step</td>
<td>1151</td>
</tr>
<tr>
<td>stepfun</td>
<td>1153</td>
</tr>
<tr>
<td>stl</td>
<td>1155</td>
</tr>
<tr>
<td>stlmethods</td>
<td>1157</td>
</tr>
<tr>
<td>StructTS</td>
<td>1158</td>
</tr>
<tr>
<td>summary.aov</td>
<td>1160</td>
</tr>
<tr>
<td>summary(glm)</td>
<td>1162</td>
</tr>
<tr>
<td>summary.lm</td>
<td>1164</td>
</tr>
<tr>
<td>summary.manova</td>
<td>1165</td>
</tr>
<tr>
<td>summary.princomp</td>
<td>1167</td>
</tr>
<tr>
<td>supsmu</td>
<td>1168</td>
</tr>
<tr>
<td>symnum</td>
<td>1169</td>
</tr>
<tr>
<td>t.test</td>
<td>1171</td>
</tr>
<tr>
<td>TDist</td>
<td>1173</td>
</tr>
<tr>
<td>termplot</td>
<td>1174</td>
</tr>
<tr>
<td>terms</td>
<td>1176</td>
</tr>
<tr>
<td>terms.formula</td>
<td>1177</td>
</tr>
<tr>
<td>terms.object</td>
<td>1178</td>
</tr>
<tr>
<td>time</td>
<td>1179</td>
</tr>
<tr>
<td>toeplitz</td>
<td>1180</td>
</tr>
<tr>
<td>ts</td>
<td>1180</td>
</tr>
<tr>
<td>ts-methods</td>
<td>1182</td>
</tr>
<tr>
<td>ts.plot</td>
<td>1183</td>
</tr>
<tr>
<td>ts.union</td>
<td>1184</td>
</tr>
<tr>
<td>tsdiag</td>
<td>1185</td>
</tr>
<tr>
<td>tsp</td>
<td>1186</td>
</tr>
<tr>
<td>tsSmooth</td>
<td>1186</td>
</tr>
<tr>
<td>Tukey</td>
<td>1187</td>
</tr>
<tr>
<td>TukeyHSD</td>
<td>1188</td>
</tr>
<tr>
<td>Uniform</td>
<td>1190</td>
</tr>
<tr>
<td>uniroot</td>
<td>1191</td>
</tr>
<tr>
<td>update</td>
<td>1192</td>
</tr>
<tr>
<td>update.formula</td>
<td>1193</td>
</tr>
<tr>
<td>var.test</td>
<td>1194</td>
</tr>
<tr>
<td>varimax</td>
<td>1195</td>
</tr>
<tr>
<td>vcov</td>
<td>1196</td>
</tr>
<tr>
<td>Weibull</td>
<td>1197</td>
</tr>
<tr>
<td>weighted.mean</td>
<td>1198</td>
</tr>
<tr>
<td>weighted.residuals</td>
<td>1199</td>
</tr>
<tr>
<td>wilcox.test</td>
<td>1200</td>
</tr>
<tr>
<td>Wilcoxon</td>
<td>1203</td>
</tr>
<tr>
<td>window</td>
<td>1204</td>
</tr>
<tr>
<td>xtabs</td>
<td>1206</td>
</tr>
</tbody>
</table>
8 The tools package

buildVignettes ................................................. 1209
checkFF ......................................................... 1210
checkMD5sums ................................................. 1211
checkTnF ......................................................... 1211
checkVignettes ................................................. 1212
codoc ............................................................... 1213
delimMatch ....................................................... 1215
encoded_text_to_latex ........................................... 1215
fileutils ......................................................... 1217
getDepList ....................................................... 1218
installFoundDepends ........................................... 1219
makeLazyLoading ............................................... 1220
md5sum ............................................................. 1221
package.dependencies ........................................... 1222
QC ................................................................. 1222
Rdindex ........................................................... 1224
Rdutils ........................................................... 1224
read.00Index ..................................................... 1226
texi2dvi .......................................................... 1226
tools-deprecated ............................................... 1227
tools-package .................................................... 1227
undoc ............................................................... 1228
vignetteDepends ............................................... 1229
write_PACKAGE .................................................. 1230
xgettext .......................................................... 1231

9 The utils package

alarm ............................................................... 1233
apropos ............................................................ 1233
BATCH ............................................................. 1235
browseEnv ......................................................... 1235
browseURL ........................................................ 1237
bug.report ......................................................... 1237
capture.output ................................................... 1240
choose.files ...................................................... 1241
chooseCRANmirror .............................................. 1242
citation ........................................................... 1242
citEntry ........................................................... 1244
clipboard ........................................................ 1246
close.socket ...................................................... 1246
compareVersion .................................................. 1247
data ............................................................... 1247
dataentry ........................................................ 1249
debugger ........................................................... 1251
demo ............................................................... 1253
DLL.version ....................................................... 1254
download.file .................................................... 1255edit ............................................................... 1257
edit.data.frame ................................................. 1258
example ........................................................... 1259
file.edit ........................................................... 1261
fix ................................................................. 1262
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>flush.console</td>
<td>1262</td>
</tr>
<tr>
<td>getAnywhere</td>
<td>1263</td>
</tr>
<tr>
<td>getFromNamespace</td>
<td>1264</td>
</tr>
<tr>
<td>getS3method</td>
<td>1265</td>
</tr>
<tr>
<td>getWindowsHandle</td>
<td>1266</td>
</tr>
<tr>
<td>glob2rx</td>
<td>1267</td>
</tr>
<tr>
<td>head</td>
<td>1268</td>
</tr>
<tr>
<td>help</td>
<td>1269</td>
</tr>
<tr>
<td>help.search</td>
<td>1272</td>
</tr>
<tr>
<td>help.start</td>
<td>1274</td>
</tr>
<tr>
<td>index_search</td>
<td>1275</td>
</tr>
<tr>
<td>INSTALL</td>
<td>1276</td>
</tr>
<tr>
<td>installed.packages</td>
<td>1277</td>
</tr>
<tr>
<td>link.html.help</td>
<td>1278</td>
</tr>
<tr>
<td>localeToCharset</td>
<td>1279</td>
</tr>
<tr>
<td>ls.str</td>
<td>1280</td>
</tr>
<tr>
<td>make.socket</td>
<td>1281</td>
</tr>
<tr>
<td>memory.size</td>
<td>1282</td>
</tr>
<tr>
<td>menu</td>
<td>1283</td>
</tr>
<tr>
<td>methods</td>
<td>1284</td>
</tr>
<tr>
<td>mirrorAdmin</td>
<td>1285</td>
</tr>
<tr>
<td>normalizePath</td>
<td>1286</td>
</tr>
<tr>
<td>object.size</td>
<td>1286</td>
</tr>
<tr>
<td>package.skeleton</td>
<td>1287</td>
</tr>
<tr>
<td>packageDescription</td>
<td>1288</td>
</tr>
<tr>
<td>packageStatus</td>
<td>1289</td>
</tr>
<tr>
<td>page</td>
<td>1291</td>
</tr>
<tr>
<td>person</td>
<td>1291</td>
</tr>
<tr>
<td>PkgUtils</td>
<td>1292</td>
</tr>
<tr>
<td>prompt</td>
<td>1293</td>
</tr>
<tr>
<td>promptData</td>
<td>1295</td>
</tr>
<tr>
<td>promptPackage</td>
<td>1296</td>
</tr>
<tr>
<td>read.fortran</td>
<td>1297</td>
</tr>
<tr>
<td>read.fwf</td>
<td>1298</td>
</tr>
<tr>
<td>read.socket</td>
<td>1300</td>
</tr>
<tr>
<td>recover</td>
<td>1301</td>
</tr>
<tr>
<td>REMOVE</td>
<td>1302</td>
</tr>
<tr>
<td>remove.packages</td>
<td>1303</td>
</tr>
<tr>
<td>Rprof</td>
<td>1304</td>
</tr>
<tr>
<td>RSiteSearch</td>
<td>1305</td>
</tr>
<tr>
<td>Rtangle</td>
<td>1306</td>
</tr>
<tr>
<td>RweaveLatex</td>
<td>1307</td>
</tr>
<tr>
<td>Rwin configuration</td>
<td>1309</td>
</tr>
<tr>
<td>savehistory</td>
<td>1310</td>
</tr>
<tr>
<td>select.list</td>
<td>1311</td>
</tr>
<tr>
<td>sessionInfo</td>
<td>1312</td>
</tr>
<tr>
<td>setRepositories</td>
<td>1312</td>
</tr>
<tr>
<td>setWindowTitle</td>
<td>1313</td>
</tr>
<tr>
<td>SHLIB</td>
<td>1314</td>
</tr>
<tr>
<td>str</td>
<td>1315</td>
</tr>
<tr>
<td>Sweave</td>
<td>1316</td>
</tr>
<tr>
<td>SweaveSyntConv</td>
<td>1318</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
</tr>
<tr>
<td>toLatex</td>
<td>1319</td>
</tr>
<tr>
<td>update.packages</td>
<td>1320</td>
</tr>
<tr>
<td>url.show</td>
<td>1323</td>
</tr>
<tr>
<td>URLencode</td>
<td>1324</td>
</tr>
<tr>
<td>utils-deprecated</td>
<td>1325</td>
</tr>
<tr>
<td>utils-package</td>
<td>1325</td>
</tr>
<tr>
<td>vignette</td>
<td>1326</td>
</tr>
<tr>
<td>winDialog</td>
<td>1327</td>
</tr>
<tr>
<td>winextras</td>
<td>1328</td>
</tr>
<tr>
<td>winMenus</td>
<td>1328</td>
</tr>
</tbody>
</table>

Index 1331
Chapter 1

The base package

.Device

Lists of Open Graphics Devices

Description

A list of the names of the open graphics devices is stored in .Devices. The name of the active device is stored in .Device.

.Machine

Numerical Characteristics of the Machine

Description

.Machine is a variable holding information on the numerical characteristics of the machine R is running on, such as the largest double or integer and the machine’s precision.

Usage

.Machine

Details

The algorithm is based on Cody’s (1988) subroutine MACHAR.

Value

A list with components (for simplicity, the prefix “double” is omitted in the explanations)

double.eps

the smallest positive floating-point number \( x \) such that \( 1 + x \neq 1 \). It equals \( \text{base}^{\text{ulp.digits}} \) if either \( \text{base} \) is 2 or rounding is 0; otherwise, it is \( (\text{base}^{\text{ulp.digits}}) / 2 \).
double.neg.eps

A small positive floating-point number $x$ such that $1 - x \neq 1$. It equals $\text{base}^{\text{neg.ulp.digits}}$ if base is 2 or round is 0; otherwise, it is $(\text{base}^{\text{neg.ulp.digits}}) / 2$. As $\text{neg.ulp.digits}$ is bounded below by $-(\text{digits} + 3)$, $\text{neg.eps}$ may not be the smallest number that can alter 1 by subtraction.

double.xmin

The smallest non-vanishing normalized floating-point power of the radix, i.e., $\text{base}^{\text{min.exp}}$.

double.xmax

The largest finite floating-point number. Typically, it is equal to $(1 - \text{neg.eps}) \times \text{base}^{\text{max.exp}}$, but on some machines it is only the second, or perhaps third, largest number, being too small by 1 or 2 units in the last digit of the significand.

double.base

The radix for the floating-point representation

double.digits

The number of base digits in the floating-point significand

double.rounding

The rounding action.

- 0 if floating-point addition chops;
- 1 if floating-point addition rounds, but not in the IEEE style;
- 2 if floating-point addition rounds in the IEEE style;
- 3 if floating-point addition chops, and there is partial underflow;
- 4 if floating-point addition rounds, but not in the IEEE style, and there is partial underflow;
- 5 if floating-point addition rounds in the IEEE style, and there is partial underflow.

double.guard

The number of guard digits for multiplication with truncating arithmetic. It is 1 if floating-point arithmetic truncates and more than $\text{digits}$ base base digits participate in the post-normalization shift of the floating-point significand in multiplication, and 0 otherwise.

double.ulp.digits

The largest negative integer $i$ such that $1 + \text{base}^i \neq 1$, except that it is bounded below by $-(\text{digits} + 3)$.

double.neg.ulp.digits

The largest negative integer $i$ such that $1 - \text{base}^i \neq 1$, except that it is bounded below by $-(\text{digits} + 3)$.

double.exponent

The number of bits (decimal places if base is 10) reserved for the representation of the exponent (including the bias or sign) of a floating-point number

double.min.exp

The largest in magnitude negative integer $i$ such that $\text{base}^i$ is positive and normalized.

double.max.exp

The smallest positive power of base that overflows.

integer.max

The largest integer which can be represented.

sizeof.long

The number of bytes in a C long type.

sizeof.longlong

The number of bytes in a C long long type. Will be zero if there is no such type.
**Description**

`.Platform` is a list with some details of the platform under which R was built. This provides means to write OS-portable R code.

**Usage**

`.Platform`

**Value**

A list with at least the following components:

- `OS.type` character, giving the Operating System (family) of the computer. One of "unix" or "windows".
- `file.sep` character, giving the file separator used on your platform: "/" on both Unix-alikes and on Windows (but not on the now abandoned port to Classic MacOS).
- `dynlib.ext` character, giving the file name extension of dynamically loadable libraries, e.g., ".dll" on Windows and ".so" or ".sl" on Unix-alikes. (Note for MacOS X users: these are shared objects as loaded by `dyn.load` and not dylibs.)
- `GUI` character, giving the type of GUI in use, or "unknown" if no GUI can be assumed.
- `_endian` character, "big" or "little", giving the endianness of the processor in use. This is relevant when it is necessary to know the order to read/write bytes of e.g. an integer or double from/to a connection: see `readBin`.
- `pkgType` character, the preferred setting for `options("pkgType")`. Values "source", "mac.binary" and "win.binary" are currently in use.

**Examples**

```r
`.Machine
## or for a neat printout
noquote(unlist(format(.Machine)))
```

**References**

path.sep character, giving the path separator, used on your platform, e.g., ":" on Unix-alikes and ";" on Windows. Used to separate paths in variables such as PATH and TEXINPUTS.

See Also

R.version and Sys.info give more details about the OS. In particular, R.version$platform is the canonical name of the platform under which R was compiled.

 Machine for details of the arithmetic used, and system for invoking platform-specific system commands.

Examples

## Note: this can be done in a system-independent way by file.info()$isdir
if(.Platform$OS.type == "unix") {
  system.test <- function(...) { system(paste("test", ...)) == 0 }
  dir.exists <- function(dir) sapply(dir, function(d)system.test("-d", d))
  dir.exists(c(R.home(), "/tmp", "/", "/NO"))# > T T T F
}

Description

Run a script through its interpreter with given arguments.

Usage

.Script(interpreter, script, args, ...)

Arguments

interpreter a character string naming the interpreter for the script.

script a character string with the base file name of the script, which must be located in the ‘interpreter’ subdirectory of ‘R_HOME/share’.

args a character string giving the arguments to pass to the script.

... further arguments to be passed to system when invoking the interpreter on the script.

Note

This function is for R internal use only.

Examples

.Script("perl", "massage-Examples.pl",
paste("tools", system.file("R-ex", package = "tools")))
Abbreviate Strings

Description

Abbreviate strings to at least `minlength` characters, such that they remain unique (if they were).

Usage

```r
abbreviate(names.arg, minlength = 4, use.classes = TRUE, dot = FALSE)
```

Arguments

- `names.arg`: a vector of names to be abbreviated.
- `minlength`: the minimum length of the abbreviations.
- `use.classes`: logical (currently ignored by R).
- `dot`: logical; should a dot (".") be appended?

Details

The algorithm used is similar to that of S. First spaces at the beginning of the word are stripped. Then any other spaces are stripped. Next lower case vowels are removed followed by lower case consonants. Finally if the abbreviation is still longer than `minlength` upper case letters are stripped.

Letters are always stripped from the end of the word first. If an element of `names.arg` contains more than one word (words are separated by space) then at least one letter from each word will be retained. If a single string is passed it is abbreviated in the same manner as a vector of strings.

Missing (NA) values are not abbreviated.

If `use.classes` is FALSE then the only distinction is to be between letters and space. This has NOT been implemented.

Value

A character vector containing abbreviations for the strings in its first argument. Duplicates in the original `names.arg` will be given identical abbreviations. If any non-duplicated elements have the same `minlength` abbreviations then `minlength` is incremented by one and new abbreviations are found for those elements only. This process is repeated until all unique elements of `names.arg` have unique abbreviations.

The character version of `names.arg` is attached to the returned value as a names argument.

Warning

This is really only suitable for English, and does not work correctly with non-ASCII characters in UTF-8 locales. It will warns if used with non-ASCII characters.

See Also

`substr`
Examples

```r
x <- c("abcd", "efgh", "abce")
abbreviate(x, 2)

(st.abb <- abbreviate(state.name, 2))
table(nchar(st.abb)) # out of 50, 3 need 4 letters
```

---

**Miscellaneous Mathematical Functions**

**Description**

These functions compute miscellaneous mathematical functions. The naming follows the standard for computer languages such as C or Fortran.

**Usage**

```r
abs(x)
sqrt(x)
```

**Arguments**

`x`  
a numeric or complex vector or array.

**Details**

These are generic functions: methods can be defined for them individually or via the `Math` group generic. For complex arguments (and the default method), `z`, `abs(z) == Mod(z)` and `sqrt(z) == z^0.5`.

**References**


**See Also**

`Arithmetic` for simple, `log` for logarithmic, `sin` for trigonometric, and `Special` for special mathematical functions.

**Examples**

```r
require(stats) # for spline
xx <- -9:9
plot(xx, sqrt(abs(xx)), col = "red")
lines(spline(xx, sqrt(abs(xx)), n=101), col = "pink")
```
**agrep**

Approximate String Matching (Fuzzy Matching)

**Description**

Searches for approximate matches to `pattern` (the first argument) within the string `x` (the second argument) using the Levenshtein edit distance.

**Usage**

```r
agrep(pattern, x, ignore.case = FALSE, value = FALSE, max.distance = 0.1)
```

**Arguments**

- `pattern`: a non-empty character string to be matched (*not* a regular expression!)
- `x`: character vector where matches are sought.
- `ignore.case`: if `FALSE`, the pattern matching is *case sensitive* and if `TRUE`, case is ignored during matching.
- `value`: if `FALSE`, a vector containing the (integer) indices of the matches determined is returned and if `TRUE`, a vector containing the matching elements themselves is returned.
- `max.distance`: Maximum distance allowed for a match. Expressed either as integer, or as a fraction of the pattern length (will be replaced by the smallest integer not less than the corresponding fraction), or a list with possible components
  - `all`: maximal (overall) distance
  - `insertions`: maximum number/fraction of insertions
  - `deletions`: maximum number/fraction of deletions
  - `substitutions`: maximum number/fraction of substitutions
  
  If `all` is missing, it is set to 10%, the other components default to `all`. The component names can be abbreviated.

**Details**

The Levenshtein edit distance is used as measure of approximateness: it is the total number of insertions, deletions and substitutions required to transform one string into another.

The function is a simple interface to the `aprop` library developed by Jarkko Hietaniemi (also used in the Perl String::Approx module).

**Value**

Either a vector giving the indices of the elements that yielded a match, or, if `value` is `TRUE`, the matched elements.

**Author(s)**

David Meyer (David.Meyer@wu-wien.ac.at) (based on C code by Jarkko Hietaniemi); modifications by Kurt Hornik.
See Also

grep

Examples

agrep("lasy", "1 lazy 2")
agrep("lasy", "1 lazy 2", max = list(sub = 0))
agrep("laysy", c("1 lazy", "1", "1 LAZY"), max = 2)
agrep("laysy", c("1 lazy", "1", "1 LAZY"), max = 2, value = TRUE)
agrep("laysy", c("1 lazy", "1", "1 LAZY"), max = 2, ignore.case = TRUE)

Description

Are All Values True?

Given a set of logical vectors, are all of the values true?

Usage

all(..., na.rm = FALSE)

Arguments

... one or more logical vectors. Other objects are coerced in a similar way as as.logical.default.

na.rm logical. If true NA values are removed before the result is computed.

Details

This is a generic function: methods can be defined for it directly or via the Summary group generic.

Value

Given a sequence of logical arguments, a logical value indicating whether or not all of the elements of x are TRUE.

The value returned is TRUE if all the values in x are TRUE, and FALSE if any the values in x are FALSE.

If na.rm = FALSE and x consists of a mix of TRUE and NA values, the value is NA.

Note

Prior to R 2.1.0, only NULL and logical, integer, numeric and complex vectors were accepted.

References


See Also

any, the “complement” of all, and stopifnot(*) which is an all(*) “insurance”.
all.equal

Examples

```r
range(x <- sort(round(rnorm(10) - 1.2, 1)))
if(all(x < 0)) cat("all x values are negative\n")
```

Description

all.equal(x, y) is a utility to compare R objects x and y testing “near equality”. If they
are different, comparison is still made to some extent, and a report of the differences is returned.
Don’t use all.equal directly in if expressions—either use isTRUE(all.equal(....))
or identical if appropriate.

Usage

```r
all.equal(target, current, ...)
## S3 method for class 'numeric':
all.equal(target, current,
  tolerance = .Machine$double.eps ^ 0.5,
  scale = NULL, ...)
attr.all.equal(target, current, ...)
```

Arguments

target R object.
current other R object, to be compared with target.
... Further arguments for different methods, notably the following two, for numerical comparison:
tolerance numeric ≥ 0. Differences smaller than tolerance are not considered.
scale numeric scalar > 0 (or NULL). See Details.

Details

There are several methods available, most of which are dispatched by the default method, see
methods("all.equal"). all.equal.list and all.equal.language provide comparison of recursive objects.
Numerical comparisons for `scale = NULL` (the default) are done by first computing the mean
absolute difference of the two numerical vectors. If this is smaller than `tolerance` or not finite,
absolute differences are used, otherwise relative differences scaled by the mean absolute difference.
If `scale` is positive, absolute comparisons are after scaling (dividing) by `scale`.
For complex arguments, the modulus `Mod` of the difference is used: all.numeric.numeric is
called so arguments `tolerance` and `scale` are available.
`attr.all.equal` is used for comparing attributes, returning NULL or character.
Value

Either TRUE or a vector of mode "character" describing the differences between target and current. Numerical differences are reported by relative error.

References


See Also

identical, isTRUE, ==, and all for exact equality testing.

Examples

all.equal(pi, 355/113) # not precise enough (default tol) > relative error
d45 <- pi*(1/4 + 1:10)
stopifnot(all.equal(tan(d45), rep(1,10))) # TRUE, but
all((tan(d45) == rep(1,10))) # FALSE, since not exactly
all.equal(tan(d45), rep(1,10), tol=0) # to see difference

all.names

Find All Names in an Expression

Description

Return a character vector containing all the names which occur in an expression or call.

Usage

all.names(expr, functions = TRUE, max.names = 200, unique = FALSE)
all.vars(expr, functions = FALSE, max.names = 200, unique = TRUE)

Arguments

expr

an expression or call from which the names are to be extracted.

functions

a logical value indicating whether function names should be included in the result.

max.names

the maximum number of names to be returned.

unique

a logical value which indicates whether duplicate names should be removed from the value.

Details

These functions differ only in the default values for their arguments.

Value

A character vector with the extracted names.
any

Examples

```r
all.names(expression(sin(x+y)))
all.vars(expression(sin(x+y)))
```

---

any Are Some Values True?

Description

Given a set of logical vectors, are any of the values true?

Usage

```r
any(..., na.rm = FALSE)
```

Arguments

... one or more logical vectors. Other objects are coerced in a similar way as `as.logical.default`.

na.rm logical. If true NA values are removed before the result is computed.

Details

This is a generic function: methods can be defined for it directly or via the `Summary` group generic.

Value

Given a sequence of logical arguments, a logical value indicating whether or not any of the elements of `x` are TRUE.

The value returned is TRUE if any the values in `x` are TRUE, and FALSE if all the values in `x` are FALSE.

If `na.rm = FALSE` and `x` consists of a mix of FALSE and NA values, the value is NA.

Note

Prior to R 2.1.0, only NULL and logical, integer, numeric and complex vectors were accepted.

References


See Also

`all`, the “complement” of any.

Examples

```r
range(x <- sort(round(rnorm(10) - 1.2,1)))
if(any(x < 0)) cat("x contains negative values\n")
```
aperm

Array Transposition

Description

Transpose an array by permuting its dimensions and optionally resizing it.

Usage

aperm(a, perm, resize = TRUE)

Arguments

a          the array to be transposed.
perm       the subscript permutation vector, which must be a permutation of the integers 1:n, where n is the number of dimensions of a. The default is to reverse the order of the dimensions.
resize     a flag indicating whether the vector should be resized as well as having its elements reordered (default TRUE).

Value

A transposed version of array a, with subscripts permuted as indicated by the array perm. If resize is TRUE, the array is reshaped as well as having its elements permuted, the dimnames are also permuted; if FALSE then the returned object has the same dimensions as a, and the dimnames are dropped.

The function t provides a faster and more convenient way of transposing matrices.

Author(s)

Jonathan Rougier, (J.C.Rougier@durham.ac.uk) did the faster C implementation.

References


See Also

t, to transpose matrices.

Examples

# interchange the first two subscripts on a 3-way array x
x <- array(1:24, 2:4)
x <- aperm(x, c(2,1,3))
stopifnot(t(x[,2]) == x[,,2],
         t(x[,3]) == x[,,3],
         t(x[,4]) == x[,,4])
**append**

**Vector Merging**

**Description**

Add elements to a vector.

**Usage**

```r
append(x, values, after = length(x))
```

**Arguments**

- `x` the vector to be modified.
- `values` to be included in the modified vector.
- `after` a subscript, after which the values are to be appended.

**Value**

A vector containing the values in `x` with the elements of `values` appended after the specified element of `x`.

**References**


**Examples**

```r
append(1:5, 0:1, after=3)
```

---

**apply**

**Apply Functions Over Array Margins**

**Description**

Returns a vector or array or list of values obtained by applying a function to margins of an array.

**Usage**

```r
apply(X, MARGIN, FUN, ...)
```

**Arguments**

- `X` the array to be used.
- `MARGIN` a vector giving the subscripts which the function will be applied over. 1 indicates rows, 2 indicates columns, `c(1, 2)` indicates rows and columns.
- `FUN` the function to be applied. In the case of functions like `+`, `%*%`, etc., the function name must be quoted.
- `...` optional arguments to `FUN`. 
Details

If \( X \) is not an array but has a dimension attribute, \texttt{apply} attempts to coerce it to an array via \texttt{as.matrix} if it is two-dimensional (e.g., data frames) or via \texttt{as.array}.

Value

If each call to \( \text{FUN} \) returns a vector of length \( n \), then \texttt{apply} returns an array of dimension \( c(n, \dim(X)[\text{MARGIN}]) \) if \( n > 1 \). If \( n \) equals 1, \texttt{apply} returns a vector if \text{MARGIN} has length 1 and an array of dimension \( \dim(X)[\text{MARGIN}] \) otherwise. If \( n \) is 0, the result has length 0 but not necessarily the “correct” dimension.

If the calls to \( \text{FUN} \) return vectors of different lengths, \texttt{apply} returns a list of length \( \prod(\dim(X)[\text{MARGIN}]) \) with \texttt{dim} set to \text{MARGIN} if this has length greater than one.

References


See Also

\texttt{lapply, tapply}, and convenience functions \texttt{sweep} and \texttt{aggregate}.

Examples

```r
## Compute row and column sums for a matrix:
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
dimnames(x)[[1]] <- letters[1:8]
apply(x, 2, mean, trim = .2)
col.sums <- apply(x, 2, sum)
row.sums <- apply(x, 1, sum)
rbind(cbind(x, Rtot = row.sums), Ctot = c(col.sums, sum(col.sums)))

stopifnot( apply(x, 2, is.vector)) # not ok in R <= 0.63.2

## Sort the columns of a matrix
apply(x, 2, sort)

##- function with extra args:
cave <- function(x, c1,c2) c(mean(x[c1]),mean(x[c2]))
apply(x, 1, cave, c1="x1", c2="x1","x2")

ma <- matrix(c(1:4, 1, 6:8), nr = 2)

apply(ma, 1, table) #---> a list of length 2
apply(ma, 1, quantile)# 5 x n matrix with rownames

stopifnot(dim(ma) == dim(apply(ma, 1:2, sum)))## wasn't ok before R 0.63.1

## Example with different lengths for each call
z <- array(1:24, dim=2:4)
zseq <- apply(z, 1:2, function(x)seq(length=max(x)))
zseq # a 2 x 3 matrix
typeof(zseq) # list
dim(zseq) # 2 3
zseq[1,]
```
apply(z, 3, function(x)seq(length=max(x))) # a list without a dim attribute

args

Argument List of a Function

Description

Displays the argument names and corresponding default values of a function.

Usage

args(name)

Arguments

name

an interpreted function. If name is a character string then the function with that
name is found and used.

Details

This function is mainly used interactively. For programming, use formals instead.

Value

A function with identical formal argument list but an empty body if given an interpreted function;
NULL in case of a variable or primitive (non-interpreted) function.

References

Brooks/Cole.

See Also

formals, help.

Examples

args(c)    # -> NULL (c is a 'primitive' function)
args(graphics::plot.default)
Arithmetic Operators

Description

These binary operators perform arithmetic on numeric or complex vectors (or objects which can be coerced to them).

Usage

\[ \begin{align*}
\times + y \\
\times - y \\
\times \times y \\
\times / y \\
\times ^ y \\
\times \%\% y \\
\times \%/ y
\end{align*} \]

Arguments

\[ \times, y \]
numeric or complex vectors or objects which can be coerced to such, or other objects for which methods have been written.

Details

The binary arithmetic operators are generic functions: methods can be written for them individually or via the \texttt{Ops}) group generic function.

If applied to arrays the result will be an array if this is sensible (for example it will not if the recycling rule has been invoked).

Logical vectors will be coerced to numeric vectors, \texttt{FALSE} having value 0 and \texttt{TRUE} having value one.

\( 1 ^ y \) and \( y ^ 0 \) are 1, always. \( \times ^ y \) should also give the proper “limit” result when either argument is infinite (i.e., \( +\text{-}\text{Inf} \)).

Objects such as arrays or time-series can be operated on this way provided they are conformable.

For real arguments, \% can be subject to catastrophic loss of accuracy if \( x \) is much larger than \( y \), and a warning is given if this is detected.

Value

These operators return vectors containing the result of the element by element operations. The elements of shorter vectors are recycled as necessary (with a warning when they are recycled only fractionally). The operators are + for addition, − for subtraction, * for multiplication, / for division and ^ for exponentiation.

\% indicates \( x \mod y \) and \%/\% indicates integer division. It is guaranteed that \( x == (x \% y) + y \ast (x \%/y) \) (up to rounding error) unless \( y == 0 \) where the result is \texttt{NA} or \texttt{NaN} (depending on the \texttt{typeof} of the arguments).

If either argument is complex the result will be complex, and if one or both arguments are numeric, the result will be numeric. If both arguments are integer, the result of / and \(^\) is numeric and of the other operators integer (with overflow returned as \texttt{NA} with a warning).
The rules for determining the attributes of the result are rather complicated. Most attributes are
taken from the longer argument, the first if they are of the same length. Names will be copied from
the first if it is the same length as the answer, otherwise from the second if that is. For time series,
these operations are allowed only if the series are compatible, when the class and tsp attribute
of whichever is a time series (the same, if both are) are used. For arrays (and an array result) the
dimensions and dimnames are taken from first argument if it is an array, otherwise the second.

References

Brooks/Cole.

See Also

sqrt for miscellaneous and Special for special mathematical functions.
Syntax for operator precedence.
%% for matrix multiplication.

Examples

```r
x <- -1:12
x + 1
2 * x + 3
x %% 2  #-- is periodic
x %/% 5
```

---

**array**  

*Multi-way Arrays*

**Description**

Creates or tests for arrays.

**Usage**

```r
array(data = NA, dim = length(data), dimnames = NULL)
as.array(x)
is.array(x)
```

**Arguments**

- `data` a vector (including a list) giving data to fill the array.
- `dim` the dim attribute for the array to be created, that is a vector of length one or more
giving the maximal indices in each dimension.
- `dimnames` the names for the dimensions. This is a list with one component for each di-
mension, either NULL or a character vector of the length given by `dim` for that
dimension. The list can be names, and the names will be used as names for the
dimensions.
- `x` an R object.
as.data.frame

Value
array returns an array with the extents specified in dim and naming information in dimnames. The values in data are taken to be those in the array with the leftmost subscript moving fastest. If there are too few elements in data to fill the array, then the elements in data are recycled. If data has length zero, NA of an appropriate type is used for atomic vectors (0 for raw vectors) and NULL for lists.

as.array() coerces its argument to be an array by attaching a dim attribute to it. It also attaches dimnames if x has names. The sole purpose of this is to make it possible to access the dim[names] attribute at a later time.
is.array returns TRUE or FALSE depending on whether its argument is an array (i.e., has a dim attribute) or not. It is generic: you can write methods to handle specific classes of objects, see InternalMethods.

References

See Also
aperm, matrix, dim, dimnames.

Examples
dim(as.array(letters))
array(1:3, c(2,4)) # recycle 1:3 *2 2/3 times"
[1,] 1 3 2 1
[2,] 2 1 3 2

as.data.frame Coerce to a Data Frame

Description
Functions to check if an object is a data frame, or coerce it if possible.

Usage
as.data.frame(x, row.names = NULL, optional = FALSE)
is.data.frame(x)

Arguments
x any R object.
row.names NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional logical. If TRUE, setting row names and converting column names (to syntactic names) is optional.
Details

as.data.frame is a generic function with many methods, and users and packages can supply further methods.

If a list is supplied, each element is converted to a column in the data frame. Similarly, each column of a matrix is converted separately. This can be overridden if the object has a class which has a method for as.data.frame: two examples are matrices of class "model.matrix" (which are included as a single column) and list objects of class "POSIXlt" which are coerced to class "POSIXct".

As from R 1.9.0 arrays can be converted. One-dimensional arrays are treated like vectors and two-dimensional arrays like matrices. Arrays with more than two dimensions are converted to matrices by 'flattening' all dimensions after the first and creating suitable column labels.

Character variables are converted to factor columns unless protected by I.

If a data frame is supplied, all classes preceding "data.frame" are stripped, and the row names are changed if that argument is supplied.

If row.names = NULL, row names are constructed from the names or dimnames of x, otherwise are the integer sequence starting at one. Few of the methods check for duplicated row names. Names are removed from vector columns unless I.

Value

as.data.frame returns a data frame, normally with all row names "" if optional = TRUE.

is.data.frame returns TRUE if its argument is a data frame (that is, has "data.frame" amongst its classes) and FALSE otherwise.

Note

In versions of R prior to 1.4.0 logical columns were converted to factors (as in S3 but not S4).

References


See Also

data.frame, as.data.frame.table for the table method (which has additional arguments if called directly).

---

as.environment Coerce to an Environment Object

Description

Converts a number or a character string to the corresponding environment on the search path.

Usage

as.environment(object)
as.function

Arguments

- **object**: the object to convert. If it is already an environment, just return it. If it is a number, return the environment corresponding to that position on the search list. If it is a character string, match the string to the names on the search list.

Value

The corresponding environment object.

Author(s)

John Chambers

See Also

- environment for creation and manipulation, search.

Examples

```r
as.environment(1) ## the global environment
identical(globalenv(), as.environment(1)) ## is TRUE
try(as.environment("package:stats")) ## stats need not be loaded
```

```
## Default S3 method:
as.function(x, envir = parent.frame(), ...)
```

Arguments

- **x**: object to convert, a list for the default method.
- **...**: additional arguments, depending on object
- **envir**: environment in which the function should be defined

Value

The desired function.
as.POSIX*

Note

For ancient historical reasons, `envir = NULL` uses the global environment rather than the base environment. Please use `envir = globalenv()` instead if this is what you want, as the special handling of NULL may change in a future release.

Author(s)

Peter Dalgaard

See Also

`function; alist` which is handy for the construction of argument lists, etc.

Examples

```r
as.function(alist(a=,b=2,a+b))
as.function(alist(a=,b=2,a+b))(3)
```

---

as.POSIX*  

Date-time Conversion Functions

Description

Functions to manipulate objects of classes "POSIXlt" and "POSIXct" representing calendar dates and times (to the nearest second).

Usage

```r
as.POSIXct(x, tz = "")
as.POSIXlt(x, tz = "")
```

Arguments

- `x`: An object to be converted.
- `tz`: A timezone specification to be used for the conversion, if one is required. System-specific, but "" is the current timezone, and "GMT" is UTC (Coordinated Universal Time, in French).

Details

The `as.POSIX*` functions convert an object to one of the two classes used to represent date/times (calendar dates plus time to the nearest second). They can take convert a wide variety of objects, including objects of the other class and of classes "Date", "date" (from package `date` or `survival"), "chron" and "dates" (from package `chron`) to these classes. Dates are treated as being at midnight UTC.

They can also convert character strings of the formats "2001-02-03" and "2001/02/03" optionally followed by white space and a time in the format "14:52" or "14:52:03". (Formats such as "01/02/03" are ambiguous but can be converted via a format specification by `strptime`.)

Logical NA's can be converted to either of the classes, but no other logical vectors can be.
Value

`as.POSIXct` and `as.POSIXlt` return an object of the appropriate class. If `tz` was specified, `as.POSIXlt` will give an appropriate "tzone" attribute.

Note

If you want to extract specific aspects of a time (such as the day of the week) just convert it to class "POSIXlt" and extract the relevant component(s) of the list, or if you want a character representation (such as a named day of the week) use `format.POSIXlt` or `format.POSIXct`. If a timezone is needed and that specified is invalid on your system, what happens is system-specific but it will probably be ignored.

See Also

`DateTimeClasses` for details of the classes; `strptime` for conversion to and from character representations.

Examples

```r
(z <- Sys.time()) # the current date, as class "POSIXct"
unclass(z) # a large integer
floor(unclass(z)/86400) # the number of days since 1970-01-01
(z <- as.POSIXlt(Sys.time())) # the current date, as class "POSIXlt"
unlist(unclass(z)) # a list shown as a named vector

as.POSIXlt(Sys.time(), "GMT") # the current time in GMT
```

---

**AsIs**

Inhibit Interpretation/Conversion of Objects

Description

Change the class of an object to indicate that it should be treated "as is".

Usage

```r
I(x)
```

Arguments

- **x**
  - an object

Details

Function `I` has two main uses.

- In function `data.frame`. Protecting an object by enclosing it in `I()` in a call to `data.frame` inhibits the conversion of character vectors to factors, and the dropping of names. `I` can also be used to protect objects which are to be added to a data frame, or converted to a data frame via `as.data.frame.`

  It achieves this by prepending the class "AsIs" to the object’s classes. Class "AsIs" has a few of its own methods, including for `[, as.data.frame, print` and `format`.
• In function `formula`. There it is used to inhibit the interpretation of operators such as "+", "-", "*" and "^" as formula operators, so they are used as arithmetical operators. This is interpreted as a symbol by `terms.formula`.

**Value**

A copy of the object with class "AsIs" prepended to the class(es).

**References**


**See Also**

`data.frame`, `formula`
assignOps

Value

This function is invoked for its side effect, which is assigning value to the variable x. If no envir is specified, then the assignment takes place in the currently active environment.

If inherits is TRUE, enclosing environments of the supplied environment are searched until the variable x is encountered. The value is then assigned in the environment in which the variable is encountered. If the symbol is not encountered then assignment takes place in the user’s workspace (the global environment).

If inherits is FALSE, assignment takes place in the initial frame of envir.

References


See Also

<-, get, exists, environment.

Examples

```r
for(i in 1:6) { #-- Create objects 'r1', 'r2', ... 'r6' --
  nam <- paste("r",i, sep=".")
  assign(nam, 1:i)
}
ls(pat="^r..$")
##-- Global assignment within a function:
myf <- function(x) {
  innerf <- function(x) assign("Global.res", x^2, env = .GlobalEnv)
  innerf(x+1)
}
myf(3)
Global.res # 16

a <- 1:4
assign("a[1]", 2)
a[1] == 2 #FALSE
get("a[1]") == 2 #TRUE
```

assignOps

Assignment Operators

Description

Assign a value to a name.
**Usage**

\[
x \leftarrow \text{value} \\
x \leftarrow \leftarrow \text{value} \\
\text{value} \rightarrow x \\
\text{value} \rightarrow> x \\
x = \text{value}
\]

**Arguments**

- `x` a variable name (possibly quoted).
- `value` a value to be assigned to `x`.

**Details**

There are three different assignment operators: two of them have leftwards and rightwards forms.

The operators `<-` and `=` assign into the environment in which they are evaluated. The `<-` can be used anywhere, but the `=` is only allowed at the top level (that is, in the complete expression typed by the user) or as one of the subexpressions in a braced list of expressions.

The operators `<<-` and `->>` cause a search to made through the environment for an existing definition of the variable being assigned. If such a variable is found then its value is redefined, otherwise assignment takes place globally. Note that their semantics differ from that in the S language, but are useful in conjunction with the scoping rules of R. See ‘The R Language Definition’ manual for further details and examples.

In all the assignment operator expressions, `x` can be a name or an expression defining a part of an object to be replaced (e.g., `z[[1]]`). The name does not need to be quoted, though it can be.

The leftwards forms of assignment `<-` = `<<-` group right to left, the other from left to right.

**Value**

`value`. Thus one can use `a <- b <- c <- 6`.

**References**


**See Also**

`assign`, `environment`. 
attach

Attach Set of R Objects to Search Path

Description

The database is attached to the R search path. This means that the database is searched by R when evaluating a variable, so objects in the database can be accessed by simply giving their names.

Usage

```
attach(what, pos = 2, name = deparse(substitute(what)), warn.conflicts=TRUE)
```

Arguments

- `what` 
  "database". This may currently be a data.frame or list or a R data file created with `save`. 

- `pos` 
  integer specifying position in `search()` where to attach. 

- `name` 
  alternative way to specify the database to be attached. 

- `warn.conflicts` 
  logical. If TRUE, warnings are printed about `conflicts` from attaching the database, unless that database contains an object `.conflicts.OK`. 

Details

When evaluating a variable or function name R searches for that name in the databases listed by `search`. The first name of the appropriate type is used.

By attaching a data frame to the search path it is possible to refer to the variables in the data frame by their names alone, rather than as components of the data frame (eg in the example below, `height` rather than `women$height`). 

By default the database is attached in position 2 in the search path, immediately after the user’s workspace and before all previously loaded packages and previously attached databases. This can be altered to attach later in the search path with the `pos` option, but you cannot attach at `pos=1`. 

The database is not actually attached. Rather, a new environment is created on the search path and the elements of a list (including columns of a data frame) or objects in a save file are *copied* into the new environment. If you use `<<-` or `assign` to assign to an attached database, you only alter the attached copy, not the original object. (Normal assignment will place a modified version in the user’s workspace: see the examples.) For this reason `attach` can lead to confusion. 

One useful ‘trick’ is to use `what = NULL` (or equivalently a length-zero list) to create a new environment on the search path into which objects can be assigned by `assign` or `sys.source`. 

Value

The `environment` is returned invisibly with a "name" attribute. 

References

attr

See Also

library, detach, search, objects, environment, with.

Examples

summary(women$height)  # refers to variable 'height' in the data frame
attach(women)
summary(height)  # The same variable now available by name
height <- height*2.54  # Don't do this. It creates a new variable
                   # in the user's workspace
find("height")
summary(height)  # The new variable in the workspace
rm(height)
summary(height)  # The original variable.
height <<- height*25.4 # Change the copy in the attached environment
find("height")
summary(height)  # The changed copy
detach("women")
summary(women$height)  # unchanged

## Not run:
## create an environment on the search path and populate it
sys.source("myfuns.R", envir=attach(NULL, name="myfuns"))
## End(Not run)

attr

Object Attributes

Description

Get or set specific attributes of an object.

Usage

attr(x, which)
attr(x, which) <- value

Arguments

x  an object whose attributes are to be accessed.
which  a character string specifying which attribute is to be accessed.
value  an object, the new value of the attribute.

Value

This function provides access to a single object attribute. The simple form above returns the value of the named attribute. The assignment form causes the named attribute to take the value on the right of the assignment symbol.

The first form first looks for an exact match to code amongst the attributed of x, then a partial match. If no exact match is found and more than one partial match is found, the result is NULL.
References


See Also

attributes

Examples

# create a 2 by 5 matrix
x <- 1:10
attr(x,"dim") <- c(2, 5)
Examples

```r
x <- cbind(a=1:3, pi=pi) # simple matrix w/ dimnames
attributes(x)

## strip an object's attributes:
attributes(x) <- NULL
x # now just a vector of length 6

mostattributes(x) <- list(mycomment = "really special", dim = 3:2,
     dimnames = list(LETTERS[1:3], letters[1:5]), names = paste(1:6))
x # dim(), but not (dim)names
```

autoload

---

On-demand Loading of Packages

Description

autoload creates a promise-to-evaluate autoloader and stores it with name name in .AutoloadEnv environment. When R attempts to evaluate name, autoloader is run, the package is loaded and name is re-evaluated in the new package’s environment. The result is that R behaves as if file was loaded but it does not occupy memory.

.Autoloaded contains the “base names” of the packages for which autoloading has been promised.

Usage

```r
autoload(name, package, reset = FALSE, ...)
autoloader(name, package, ...)
```

.AutoloadEnv
.Autoloaded

Arguments

- `name` string giving the name of an object.
- `package` string giving the name of a package containing the object.
- `reset` logical: for internal use by autoloader.
- `...` other arguments to `library`.

Value

This function is invoked for its side-effect. It has no return value as of R 1.7.0.

See Also

`delayedAssign, library`
Examples

```R
require(stats)
autoload("interpSpline", "splines")
search()
ls("Autoloads")
.Autoloaded

x <- sort(rnorm(12))
y <- x^2
is <- interpSpline(x,y)
search() ## now has splines
detach("package:splines")
search()
is2 <- interpSpline(x,y+x)
search() ## and again
detach("package:splines")
```

backsolve

**Solve an Upper or Lower Triangular System**

Description

Solves a system of linear equations where the coefficient matrix is upper or lower triangular.

Usage

```R
backsolve(r, x, k= ncol(r), upper.tri = TRUE, transpose = FALSE)
forwardsolve(l, x, k= ncol(l), upper.tri = FALSE, transpose = FALSE)
```

Arguments

- `r, l` an upper (or lower) triangular matrix giving the coefficients for the system to be solved. Values below (above) the diagonal are ignored.
- `x` a matrix whose columns give "right-hand sides" for the equations.
- `k` The number of columns of `r` and rows of `x` to use.
- `upper.tri` logical; if `TRUE` (default), the upper triangular part of `r` is used. Otherwise, the lower one.
- `transpose` logical; if `TRUE`, solve $r' * y = x$ for $y$, i.e., `t(r) %*% y == x`.

Value

The solution of the triangular system. The result will be a vector if `x` is a vector and a matrix if `x` is a matrix.

References


See Also

chol, qr, solve.

Examples

```r
## upper triangular matrix 'r':
r <- rbind(c(1,2,3),
c(0,1,1),
c(0,0,2))
(y <- backsolve(r, x <- c(8,4,2)) ) # -1 3 1
r %*% y # == x = (8,4,2)
backsolve(r, x, transpose = TRUE) # 8 -12 -5
```

Description

These functions are provided for compatibility with older versions of R only, and may be defunct as soon as the next release.

Usage

```r
format.char(x, width = NULL, flag = "-")
```

Arguments

- `x` a character vector.
- `width, flag` see `formatC`.

Details

`write.table0` is an R-level version of `write.table` provided for temporary back
compatibility.

`format.char` was an auxiliary function for `formatC` whose functionality (and more) is pro-
vided by `format.default`.

The original help page for these functions is often available at `help("oldName-
deprecated")` (note the quotes). Functions in packages other than the base package are listed in `help("pkg-deprecated")`.

See Also

`Deprecated`
The R Base Package

Description
Base R functions

Details
This package contains the basic functions which let R function as a language: arithmetic, input/output, basic programming support, etc. Its contents are available through inheritance from any environment.

For a complete list of functions, use `library(help="base")`.

Manipulate File Paths

Description
basename removes all of the path up to the last path separator (if any).
dirname returns the part of the path up to (but excluding) the last path separator, or "." if there is no path separator.

Usage
basename(path)
dirname(path)

Arguments
path character vector, containing path names.

Details
For dirname tilde expansion is done: see the description of `path.expand`.
Trailing file separators are removed before dissecting the path, and for dirname any trailing file separators are removed from the result.
On Windows this will accept either \ or / as the directory separator, but dirname will return a path using /.

Value
A character vector of the same length as path. A zero-length input will give a zero-length output with no error.

See Also
`file.path`, `path.expand`
Bessel Functions

Examples

```r
basename(file.path("", "p1", "p2", "p3", c("file1", "file2")))
dirname(file.path("", "p1", "p2", "p3", "filename"))
```

Description

Bessel Functions of integer and fractional order, of first and second kind, \( J_\nu \) and \( Y_\nu \), and Modified Bessel functions (of first and third kind), \( I_\nu \) and \( K_\nu \).

gammaCody is the \((\Gamma)\) function as from the Specfun package and originally used in the Bessel code.

Usage

```r
besselI(x, nu, expon.scaled = FALSE)
besselK(x, nu, expon.scaled = FALSE)
besselJ(x, nu)
besselY(x, nu)
gammaCody(x)
```

Arguments

- **x**: numeric, \( \geq 0 \).
- **nu**: numeric; The order (maybe fractional!) of the corresponding Bessel function.
- **expon.scaled**: logical; if TRUE, the results are exponentially scaled in order to avoid overflow (\( I_\nu \)) or underflow (\( K_\nu \)), respectively.

Details


If \( \text{expon.scaled} = \text{TRUE} \), \( e^{-x}I_\nu(x) \), or \( e^{x}K_\nu(x) \) are returned.

gammaCody may be somewhat faster but less precise and/or robust than R’s standard \( \text{gamma} \). It is here for experimental purpose mainly, and may be defunct very soon.

For \( \nu < 0 \), formulae 9.1.2 and 9.6.2 from the reference below are applied (which is probably suboptimal), unless for \( \text{besselK} \) which is symmetric in \( \nu \).

Value

Numeric vector of the same length of \( x \) with the (scaled, if \( \text{expon.scaled}=\text{TRUE} \)) values of the corresponding Bessel function.

Author(s)

Original Fortran code: W. J. Cody, Argonne National Laboratory
Translation to C and adaption to R: Martin Maechler (maechler@stat.math.ethz.ch)
References


See Also

Other special mathematical functions, such as *gamma*, $\Gamma(x)$, and *beta*, $B(x)$.

Examples

\begin{verbatim}
nus <- c(0:5, 10, 20)
x <- seq(0, 4, len = 501)
plot(x, x, ylim = c(0, 6), ylab = "", type = "n",
     main = "Bessel Functions I_nu(x)")
for(nu in nus) lines(x, besselI(x, nu=nu), col = nu+2)
legend(0, 6, legend = paste("nu=", nus), col = nus+2, lwd = 1)

x <- seq(0, 40, len=801); yl <- c(-.8, .8)
plot(x, x, ylim = yl, ylab = "", type = "n",
     main = "Bessel Functions J_nu(x)")
for(nu in nus) lines(x, besselJ(x, nu=nu), col = nu+2)
legend(32,-.18, legend = paste("nu=", nus), col = nus+2, lwd = 1)

## Negative nu's :
xx <- 2:7
nu <- seq(-10, 9, len = 2001)
op <- par(lab = c(16, 5, 7))
matplot(nu, t(outer(xx, nu, besselI)), type = "l", ylim = c(-50, 200),
       main = expression(paste("Bessel I[nu](x), as \( f(nu) \).
                                 xlab = expression(nu))
       abline(v=0, col = "light gray", lty = 3)
legend(5, 200, legend = paste("nu=", paste(nus, nus+.5, sep=",")),
       col = nus + 2, lwd = 1)
par(op)

x0 <- 2^(-20:10)
plot(x0, x0^-8, log="xy", ylab="", type="n",
     main = "Bessel Functions J_nu(x) near 0\n
for(nu in sort(c(nus, nus+.5)))
   lines(x0, besselJ(x0, nu=nu), col = nu+2)
legend(3, 1e50, legend = paste("nu=", paste(nus, nus+.5, sep=",")),
       col = nus + 2, lwd = 1)

plot(x0, x0^-8, log="xy", ylab="", type="n",
     main = "Bessel Functions K_nu(x) near 0\n
for(nu in sort(c(nus, nus+.5)))
   lines(x0, besselK(x0, nu=nu), col = nu+2)
legend(3, 1e50, legend = paste("nu=", paste(nus, nus+.5, sep=",")),
       col = nus + 2, lwd = 1)

x <- x[x > 0]
plot(x, x, ylim=c(1e-18, 1e11), log = "y", ylab = "", type = "n",
     main = "Bessel Functions K_nu(x)")
for(nu in nus) lines(x, besselK(x, nu=nu), col = nu+2)
legend(0, 1e-5, legend=paste("nu=", nus), col = nus+2, lwd = 1)
\end{verbatim}
yl <- c(-1.6, .6)
plot(x, x, ylim = yl, ylab = "", type = "n",
    main = "Bessel Functions \( Y_\nu(x) \)"
for(nu in nus){
    xx <- x[x > .6*nu]
    lines(xx, besselY(xx, nu=nu), col = nu+2)
}
legend(25, -.5, legend = paste("nu=", nus), col = nus+2, lwd = 1)

---

**body**  
*Access to and Manipulation of the Body of a Function*

### Description
Get or set the body of a function.

### Usage

```r
body(fun = sys.function(sys.parent()))
body(fun, envir = parent.frame()) <- value
```

### Arguments

- **fun**: a function object, or see Details.
- **envir**: environment in which the function should be defined.
- **value**: an expression or a list of \( \mathbb{R} \) expressions.

### Details
For the first form, `fun` can be a character string naming the function to be manipulated, which is searched for from the parent environment. If it is not specified, the function calling `body` is used.

### Value
`body` returns the body of the function specified.

The assignment form sets the body of a function to the list on the right hand side.

### Note
For ancient historical reasons, `envir = NULL` uses the global environment rather than the base environment. Please use `envir = globalenv()` instead if this is what you want, as the special handling of `NULL` may change in a future release.

### See Also
`alist`, `args`, `function`. 
Examples

body(body)
f <- function(x) x^5
body(f) <- expression(5^x)
## or equivalently body(f) <- list(quote(5^x))
f(3) # = 125
body(f)

bquote

Partial substitution in expressions

Description

An analogue of the LISP backquote macro. `bquote` quotes its argument except that terms wrapped in .() are evaluated in the specified `where` environment.

Usage

`bquote(expr, where = parent.frame())`

Arguments

- `expr` An expression
- `where` An environment

Value

An expression

See Also

`quote, substitute`

Examples

a <- 2

bquote(a==a)
quote(a==a)

bquote(a==.(a))
substitute(a==A, list(A=a))

plot(1:10, a*(1:10), main = bquote(a==.(a)))
browser

Environment Browser

Description

Interrupt the execution of an expression and allow the inspection of the environment where browser was called from.

Usage

browser()

Details

A call to browser causes a pause in the execution of the current expression and runs a copy of the R interpreter which has access to variables local to the environment where the call took place.

Local variables can be listed with ls, and manipulated with R expressions typed to this sub-interpreter. The sub-interpreter can be exited by typing c. Execution then resumes at the statement following the call to browser.

Typing n causes the step-through-debugger, to start and it is possible to step through the remainder of the function one line at a time. In this mode c will continue to the end of the current context (to the next loop iteration if within a loop).

Typing Q quits the current execution and returns you to the top-level prompt.

References


See Also

d debug, and traceback for the stack on error.

builtins

Returns the names of all built-in objects

Description

Return the names of all the built-in objects. These are fetched directly from the symbol table of the R interpreter.

Usage

builtins(internal = FALSE)

Arguments

internal a logical indicating whether only “internal” functions (which can be called via .Internal) should be returned.
by

Description

Function by is an object-oriented wrapper for tapply applied to data frames.

Usage

by(data, INDICES, FUN, ...)

Arguments

data an R object, normally a data frame, possibly a matrix.
INDICES a factor or a list of factors, each of length nrow(data).
FUN a function to be applied to data frame subsets of data.
... further arguments to FUN.

Details

A data frame is split by row into data frames subsetted by the values of one or more factors, and function FUN is applied to each subset in turn.

Object data will be coerced to a data frame by default.

Value

A list of class "by", giving the results for each subset.

See Also

tapply

Examples

require(stats)
attach(warpbreaks)
by(warpbreaks[, 1:2], tension, summary)
by(warpbreaks[, 1], list(wool=wool, tension=tension), summary)
by(warpbreaks, tension, function(x) lm(breaks ~ wool, data=x))

## now suppose we want to extract the coefficients by group
tmp <- by(warpbreaks, tension, function(x) lm(breaks ~ wool, data=x))
sapply(tmp, coef)

detach("warpbreaks")
Description

This is a generic function which combines its arguments.

The default method combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value.

Usage

c(..., recursive=FALSE)

Arguments

... objects to be concatenated.

recursive logical. If recursive=TRUE, the function recursively descends through lists combining all their elements into a vector.

References


See Also

unlist and as.vector to produce attribute-free vectors.

Examples

c(1,7:9)
c(1:5, 10.5, "next")

## append to a list:
ll <- list(A = 1, c="C")
## do *not* use
c(ll, d = 1:3) # which is == c(ll, as.list(c(d=1:3))
## but rather
c(ll, d = list(1:3))# c() combining two lists

c(list(A=c(B=1)), recursive=TRUE)
c(options(), recursive=TRUE)
c(list(A=c(B=1,C=2), B=c(E=7)), recursive=TRUE)
call

Function Calls

Description

Create or test for objects of mode "call".

Usage

call(name, ...) is.call(x) as.call(x)

Arguments

name a character string naming the function to be called.
... arguments to be part of the call.
x an arbitrary R object.

Details

call returns an unevaluated function call, that is, an unevaluated expression which consists of the named function applied to the given arguments (name must be a quoted string which gives the name of a function to be called).
is.call is used to determine whether x is a call (i.e., of mode "call"). It is generic: you can write methods to handle specific classes of objects, see InternalMethods. Objects of mode "list" can be coerced to mode "call". The first element of the list becomes the function part of the call, so should be a function or the name of one (as a symbol; a quoted string will not do).

References


See Also

do.call for calling a function by name and argument list; Recall for recursive calling of functions; further is.language, expression, function.

Examples

is.call(call) #-> FALSE: Functions are NOT calls

# set up a function call to round with argument 10.5
c1 <- call("round", 10.5)
is.call(c1)# TRUE
c1
# such a call can also be evaluated.
eval(c1)# [1] 10
Report Capabilities of this Build of R

Description

Report on the optional features which have been compiled into this build of R.

Usage

capabilities(what = NULL)

Arguments

what character vector or NULL, specifying required components. NULL implies that all are required.

Value

A named logical vector. Current components are

- `jpeg` Is the `jpeg` function operational?
- `png` Is the `png` function operational?
- `tcltk` Is the `tcltk` package operational?
- `X11` (Unix) Are the X11 graphics device and the X11-based data editor available? As from R 2.1.0 this loads the X11 module if not already loaded, and checks that the default display can be contacted unless a X11 device has already been used.
- `http/ftp` Are `url` and the internal method for `download.file` available?
- `sockets` Are `make.socket` and related functions available?
- `libxml` Is there support for integrating `libxml` with the R event loop?
- `fifo` are FIFO connections supported?
- `cledit` Is command-line editing available in the current R session? This is false in non-interactive sessions.
- `iconv` is internationalization conversion via `iconv` supported?
- `NLS` is there Natural Language Support (for message translations)?

See Also

.Platform

Examples

capabilities()

if(!capabilities("http/ftp"))
    warning("internal download.file() is not available")

## See also the examples for 'connections'.
**cat**

*Concatenate and Print*

**Description**

Prints the arguments, coercing them if necessary to character mode first.

**Usage**

```r
cat(..., file = "", sep = " ", fill = FALSE, labels = NULL, append = FALSE)
```

**Arguments**

- `...`: R objects which are coerced to character strings, concatenated, and printed, with the remaining arguments controlling the output.
- `file`: A connection, or a character string naming the file to print to. If "" (the default), `cat` prints to the standard output connection, the console unless redirected by `sink`.
- `sep`: character string to insert between the objects to print.
- `fill`: a logical or numeric controlling how the output is broken into successive lines. If `FALSE` (default), only newlines created explicitly by `\n` are printed. Otherwise, the output is broken into lines with print width equal to the option `width` if `fill` is `TRUE`, or the value of `fill` if this is numeric.
- `labels`: character vector of labels for the lines printed. Ignored if `fill` is `FALSE`.
- `append`: logical. Only used if the argument `file` is the name of file (and not a connection or "|cmd"). If `TRUE` output will be appended to `file`; otherwise, it will overwrite the contents of `file`.

**Details**

cat converts its arguments to character strings, concatenates them, separating them by the given `sep` string, and then prints them.

No line feeds are printed unless explicitly requested by `\n` or if generated by filling (if argument `fill` is `TRUE` or numeric.)

cat is useful for producing output in user-defined functions.

**Value**

None (invisible `NULL`).

**References**


**See Also**

`print`, `format`, and `paste` which concatenates into a string.
Examples

```r
iter <- rpois(1, lambda=10)
## print an informative message
cat("iteration = ", iter <- iter + 1, "\n")
## 'fill' and label lines:
cat(paste(letters, 100*1:26), fill = TRUE,
labels = paste("","1:10",""",sep=""))
```

**cbind**

*Combine R Objects by Rows or Columns*

**Description**

Take a sequence of vector, matrix or data frames arguments and combine by columns or rows, respectively. These are generic functions with methods for other R classes.

**Usage**

```r
cbind(..., deparse.level = 1)
rbind(..., deparse.level = 1)
```

**Arguments**

- `...` vectors or matrices. These can be given as named arguments.
- `deparse.level` integer controlling the construction of labels in the case of “non-matrix-like” arguments (i.e., for the default method):
  - `deparse.level = 0` constructs no labels; the default, `deparse.level = 1` or 2 constructs labels from the argument names, see the ‘Value’ section below.

**Details**

The functions `cbind` and `rbind` are S3 generic, with methods for data frames. The data frame method will be used if an argument is a data frame and the rest are vectors or matrices. There can be other methods; in particular, there is one for time series objects.

In the matrix case, all the vectors/matrices must be atomic (see `vector`) or lists (e.g., not expressions).

The `rbind` data frame method takes the classes of the columns from the first data frame. Factors have their levels expanded as necessary (in the order of the levels of the levelsets of the factors encountered) and the result is an ordered factor if and only if all the components were ordered factors. (The last point differs from S-PLUS.)

If there are several matrix arguments, they must all have the same number of columns (or rows) and this will be the number of columns (or rows) of the result. If all the arguments are vectors, the number of columns (rows) in the result is equal to the length of the longest vector. Values in shorter arguments are recycled to achieve this length (with a `warning` if they are recycled only fractionally).

When the arguments consist of a mix of matrices and vectors the number of columns (rows) of the result is determined by the number of columns (rows) of the matrix arguments. Any vectors have their values recycled or subsetted to achieve this length.
For `cbind` (rbind), vectors of zero length (including NULL) are ignored unless the result would have zero rows (columns), for S compatibility. (Zero-extent matrices do not occur in S3 and are not ignored in R.)

**Value**

A matrix or data frame combining the ... arguments column-wise or row-wise.

For `cbind` (rbind) the column (row) names are taken from the `colnames` (rownames) of the arguments if these are matrix-like. Otherwise from the names of the arguments or where those are not supplied and `deparse.level > 0`, by deparsing the expressions given, for `deparse.level = 1` only if that gives a sensible name (a ‘symbol’, see `is.symbol`).

The names will depend on whether data frames are included: see the examples.

**Note**

The method dispatching is not done via `UseMethod()`, but by C-internal dispatching. Therefore, there is no need for, e.g., `rbind.default`.

The dispatch algorithm is described in the source file (`./src/main/bind.c`) as

1. For each argument we get the list of possible class memberships from the class attribute.
2. We inspect each class in turn to see if there is an an applicable method.
3. If we find an applicable method we make sure that it is identical to any method determined for prior arguments. If it is identical, we proceed, otherwise we immediately drop through to the default code.

If you want to combine other objects with data frames, it may be necessary to coerce them to data frames first. (Note that this algorithm can result in calling the data frame method if the arguments are all either data frames or vectors, and this will result in the coercion of character vectors to factors.)

**References**


**See Also**

c to combine vectors (and lists) as vectors, `data.frame` to combine vectors and matrices as a data frame.

**Examples**

```r
m <- cbind(1, 1:7) # the '1' (= shorter vector) is recycled
m
m <- cbind(m, 8:14)[, c(1, 3, 2)] # insert a column
m
cbind(1:7, diag(3)) # vector is subset -> warning

cbind(0, rbind(1, 1:3))
```

```r
cbind(I=0, X=rbind(a=1, b=1:3)) # use some names
xx <- data.frame(I=rep(0,2))
```

```r
cbind(xx, X=rbind(a=1, b=1:3)) # named differently
```
char.expand

Expand a String with Respect to a Target Table

Description

Seeks a unique match of its first argument among the elements of its second. If successful, it returns this element; otherwise, it performs an action specified by the third argument.

Usage

char.expand(input, target, nomatch = stop("no match"))

Arguments

input a character string to be expanded.
target a character vector with the values to be matched against.
nomatch an R expression to be evaluated in case expansion was not possible.

Details

This function is particularly useful when abbreviations are allowed in function arguments, and need to be uniquely expanded with respect to a target table of possible values.

See Also

charmatch and pmatch for performing partial string matching.

Examples

locPars <- c("mean", "median", "mode")
char.expand("me", locPars, warning("Could not expand!"))
char.expand("mo", locPars)
**Description**

Create or test for objects of type "character".

**Usage**

\[
\begin{align*}
\text{character}(\text{length} = 0) \\
\text{as.character}(x, \ldots) \\
\text{is.character}(x)
\end{align*}
\]

**Arguments**

- **length**: desired length.
- **x**: object to be coerced or tested.
- **...**: further arguments passed to or from other methods.

**Details**

as.character and is.character are generic: you can write methods to handle specific classes of objects, see InternalMethods.

as.character represents real and complex numbers to 15 decimal places (technically the compiler's setting of the ISO C constant DBL_DIG, which will be 15 on machines supporting IEC60559 arithmetic according to the C99 standard). This ensures that all the digits in the result will be reliable (and not the result of representation error), but does mean that conversion to character and back to numeric may change the number. If you want to convert numbers to character with the maximum possible precision, use format.

**Value**

character creates a character vector of the specified length. The elements of the vector are all equal to "".

as.character attempts to coerce its argument to character type; like as.vector it strips attributes including names.

is.character returns TRUE or FALSE depending on whether its argument is of character type or not.

**Note**

as.character truncates components of language objects to 500 characters (was about 70 before 1.3.1).

**References**

charmatch

See Also

paste, substr and strsplit for character concatenation and splitting, chartr for character translation and casefolding (e.g., upper to lower case) and sub, grep etc for string matching and substitutions. Note that help.search(keyword = "character") gives even more links.
deparse, which is normally preferable to as.character for language objects.

Examples

form <- y ~ a + b + c
as.character(form) ## length 3
deparse(form) ## like the input

a0 <- 11/999 # has a repeating decimal representation
(a1 <- as.character(a0))
format(a0, digits=16) # shows one more digit
a2 <- as.numeric(a1)
a2 - a0 # normally around -1e-17
as.character(a2) # normally different from a1
print(c(a0, a2), digits = 16)

charmatch Partial String Matching

Description

ccharmatch seeks matches for the elements of its first argument among those of its second.

Usage

charmatch(x, table, nomatch = NA)

Arguments

x the values to be matched.
table the values to be matched against.
nomatch the value returned at non-matching positions.

Details

Exact matches are preferred to partial matches (those where the value to be matched has an exact match to the initial part of the target, but the target is longer).

If there is a single exact match or no exact match and a unique partial match then the index of the matching value is returned; if multiple exact or multiple partial matches are found then 0 is returned and if no match is found then NA is returned.

Author(s)

This function is based on a C function written by Terry Therneau.
chartr

Character Translation and Casefolding

Description

Translate characters in character vectors, in particular from upper to lower case or vice versa.

Usage

chartr(old, new, x)
tolower(x)
toupper(x)
casefold(x, upper = FALSE)

Arguments

x a character vector.
old a character string specifying the characters to be translated.
new a character string specifying the translations.
upper logical: translate to upper or lower case?

Details

chartr translates each character in x that is specified in old to the corresponding character specified in new. Ranges are supported in the specifications, but character classes and repeated characters are not. If old contains more characters than new, an error is signaled; if it contains fewer characters, the extra characters at the end of new are ignored.
tolower and toupper convert upper-case characters in a character vector to lower-case, or vice versa. Non-alphabetic characters are left unchanged.
casefold is a wrapper for tolower and toupper provided for compatibility with S-PLUS.

See Also

sub and gsub for other substitutions in strings.

Examples

charmatch("", ") # returns 1
charmatch("m", c("mean", "median", "mode")) # returns 0
charmatch("med", c("mean", "median", "mode")) # returns 2
Examples

```r
x <- "MiXeD cAsE 123"
chartr("iXs", "why", x)
chartr("a-cX", "D-Fw", x)
tolower(x)
toupper(x)

## "MiXed Case" Capitalizing - toupper( every first letter of a word ) :

.simpleCap <- function(x) {
  s <- strsplit(x, " ")[[1]]
  paste(toupper(substring(s, 1,1)), substring(s, 2), sep="", collapse=" ")
}
.simpleCap("the quick red fox jumps over the lazy brown dog")
## -> [1] "The Quick Red Fox Jumps Over The Lazy Brown Dog"

## and the better, more sophisticated version:
capwords <- function(s, strict = FALSE) {
  cap <- function(s) paste(toupper(substring(s,1,1)),
                           (s <- substring(s,2); if(strict) tolower(s) else s),
                           sep = "", collapse = " ")
  sapply(strsplit(s, split = " "), cap, USE.NAMES = !is.null(names(s)))
}
capwords(c("using AIC for model selection"))
## -> [1] "Using AIC For Model Selection"
capwords(c("using AIC", "for MODEL selection"), strict=TRUE)
## -> [1] "Using Aic" "For Model Selection"

## ^^^ ^^^^^
## 'bad' 'good'
```

---

`chol`  
*The Choleski Decomposition*

**Description**

Compute the Choleski factorization of a real symmetric positive-definite square matrix.

**Usage**

```r
chol(x, pivot = FALSE, LINPACK = pivot)
La.chol(x)
```

**Arguments**

- `x`  
a real symmetric, positive-definite matrix
- `pivot`  
Should pivoting be used?
- `LINPACK`  
logical. Should LINPACK be used (for compatibility with R < 1.7.0)?
Details

chol(pivot = TRUE) provides an interface to the LINPACK routine DCHDC. La.chol provides an interface to the LAPACK routine DPOTRF.

Note that only the upper triangular part of x is used, so that \( R^T R = x \) when \( x \) is symmetric.

If pivot = FALSE and \( x \) is not non-negative definite an error occurs. If \( x \) is positive semi-definite (i.e., some zero eigenvalues) an error will also occur, as a numerical tolerance is used.

If pivot = TRUE, then the Choleski decomposition of a positive semi-definite \( x \) can be computed. The rank of \( x \) is returned as attr(Q, "rank"), subject to numerical errors. The pivot is returned as attr(Q, "pivot"). It is no longer the case that \( t(Q) \%*% Q \) equals \( x \). However, setting pivot <- attr(Q, "pivot") and oo <- order(pivot), it is true that \( t(Q[,, oo]) \%*% Q[,, oo] \) equals \( x \), or, alternatively, \( t(Q) \%*% Q \) equals \( x[pivot, pivot] \).

See the examples.

Value

The upper triangular factor of the Choleski decomposition, i.e., the matrix \( R \) such that \( R^T R = x \) (see example).

If pivoting is used, then two additional attributes "pivot" and "rank" are also returned.

Warning

The code does not check for symmetry.

If pivot = TRUE and \( x \) is not non-negative definite then there will be no error message but a meaningless result will occur. So only use pivot = TRUE when \( x \) is non-negative definite by construction.

References


See Also

chol2inv for its inverse (without pivoting), backsolve for solving linear systems with upper triangular left sides.

qr, svd for related matrix factorizations.

Examples

```r
( m <- matrix(c(5,1,1,3),2,2) )
( cm <- chol(m) )
t(cm) \%*% cm #-- = 'm'
crossprod(cm) #-- = 'm'

# now for something positive semi-definite
x <- matrix(c(1:5, (1:5)^2), 5, 2)
x <- cbind(x, x[, 1] + 3*x[, 2])
```
m <- crossprod(x)
qr(m)$rank # is 2, as it should be

# chol() may fail, depending on numerical rounding:
# chol() unlike qr() does not use a tolerance.
try(chol(m))

(Q <- chol(m, pivot = TRUE)) # NB wrong rank here ... see Warning section.
## we can use this by
pivot <- attr(Q, "pivot")
oo <- order(pivot)
t(Q[, oo]) %*% Q[, oo] # recover m

---

chol2inv  

**Inverse from Choleski Decomposition**

### Description

Invert a symmetric, positive definite square matrix from its Choleski decomposition.

### Usage

```r
chol2inv(x, size = NCOL(x), LINPACK = FALSE)
La.chol2inv(x, size = ncol(x))
```

### Arguments

- `x`: a matrix. The first `nc` columns of the upper triangle contain the Choleski decomposition of the matrix to be inverted.
- `size`: the number of columns of `x` containing the Choleski decomposition.
- `LINPACK`: logical. Should LINPACK be used (for compatibility with R < 1.7.0)?

### Details

`chol2inv(LINPACK=TRUE)` provides an interface to the LINPACK routine DPODI.
`La.chol2inv` provides an interface to the LAPACK routine DPOTRI.

### Value

The inverse of the decomposed matrix.

### References


### See Also

`chol, solve`
Examples

cma <- chol(ma <- cbind(1, 1:3, c(1,3,7)))
ma %*% chol2inv(cma)

Object Classes

description

R possesses a simple generic function mechanism which can be used for an object-oriented style of
programming. Method dispatch takes place based on the class of the first argument to the generic
function.

Usage

class(x)
class(x) <- value
unclass(x)
inherits(x, what, which = FALSE)

oldClass(x)
oldClass(x) <- value

Arguments

x a R object
what, value a character vector naming classes.
which logical affecting return value: see Details.

details

Many R objects have a class attribute, a character vector giving the names of the classes which
the object “inherits” from. If the object does not have a class attribute, it has an implicit class,
"matrix", "array" or the result of mode(x). (Functions oldClass and oldClass<- get
and set the attribute, which can also be done directly.)

When a generic function fun is applied to an object with class attribute c("first",
"second"), the system searches for a function called fun.first and, if it finds it, applies
it to the object. If no such function is found, a function called fun.second is tried. If no class
name produces a suitable function, the function fun.default is used (if it exists). If there is no
class attribute, the implicit class is tried, then the default method.

The function class prints the vector of names of classes an object inherits from. Correspondingly,
class<- sets the classes an object inherits from.

unclass returns (a copy of) its argument with its class attribute removed. (It is not allowed for
objects which cannot be copied, namely environments and external pointers.)

inherits indicates whether its first argument inherits from any of the classes specified in the
what argument. If which is TRUE then an integer vector of the same length as what is returned.
Each element indicates the position in the class(x) matched by the element of what; zero
indicates no match. If which is FALSE then TRUE is returned by inherits if any of the names
in what match with any class.
Formal classes

An additional mechanism of formal classes is available in packages methods which is attached by default. For objects which have a formal class, its name is returned by class as a character vector of length one.

The replacement version of the function sets the class to the value provided. For classes that have a formal definition, directly replacing the class this way is strongly deprecated. The expression as(object, value) is the way to coerce an object to a particular class.

Note

Functions oldClass and oldClass<- behave in the same way as functions of those names in S-PLUS 5/6, but in R UseMethod dispatches on the class as returned by class (with some interpolated classes: see the link) rather than oldClass. However, group generics dispatch on the oldClass for efficiency.

See Also

UseMethod, NextMethod, group generic.

Examples

x <- 10
inherits(x, "a") # FALSE
class(x) <- c("a", "b")
inherits(x,"a") # TRUE
inherits(x, "a", TRUE) # 1
inherits(x, c("a", "b", "c"), TRUE) # 1 2 0

\begin{tabular}{ll}
\textbf{col} & \textbf{Column Indexes} \\
\hline
\end{tabular}

Description

Returns a matrix of integers indicating their column number in the matrix.

Usage

col(x, as.factor = FALSE)

Arguments

- \texttt{x} a matrix.
- \texttt{as.factor} a logical value indicating whether the value should be returned as a factor rather than as numeric.

Value

An integer matrix with the same dimensions as \texttt{x} and whose \texttt{i} \texttt{j}-th element is equal to \texttt{j}.

References

See Also

row to get rows.

Examples

# extract an off-diagonal of a matrix
ma <- matrix(1:12, 3, 4)
ma[row(ma) == col(ma) + 1]

# create an identity 5-by-5 matrix
x <- matrix(0, nr = 5, nc = 5)
x[row(x) == col(x)] <- 1

colSums

Form Row and Column Sums and Means

Description

Form row and column sums and means for numeric arrays.

Usage

colSums (x, na.rm = FALSE, dims = 1)
rowSums (x, na.rm = FALSE, dims = 1)
colMeans(x, na.rm = FALSE, dims = 1)
rowMeans(x, na.rm = FALSE, dims = 1)

Arguments

x       an array of two or more dimensions, containing numeric, complex, integer or logical values, or a numeric data frame.
na.rm   logical. Should missing values (including NaN) be omitted from the calculations?
dims    Which dimensions are regarded as “rows” or “columns” to sum over. For row*, the sum or mean is over dimensions dims+1, ...; for col* it is over dimensions 1:dims.

Details

These functions are equivalent to use of apply with FUN = mean or FUN = sum with appropriate margins, but are a lot faster. As they are written for speed, they blur over some of the subtleties of NaN and NA. If na.rm = FALSE and either NaN or NA appears in a sum, the result will be one of NaN or NA, but which might be platform-dependent.

Value

A numeric or complex array of suitable size, or a vector if the result is one-dimensional. The dimnames (or names for a vector result) are taken from the original array.

If there are no values in a range to be summed over (after removing missing values with na.rm = TRUE), that component of the output is set to 0 (*Sums) or NA (*Means), consistent with sum and mean.
commandArgs

See Also

apply, rowsum

Examples

## Compute row and column sums for a matrix:
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
rowSums(x); colSums(x)
dimnames(x)[[1]] <- letters[1:8]
rowSums(x); colSums(x); rowMeans(x); colMeans(x)
x[] <- as.integer(x)
rowSums(x); colSums(x)
x[] <- x < 3
rowSums(x); colSums(x)
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
x[3, ] <- NA; x[4, 2] <- NA
rowSums(x); colSums(x); rowMeans(x); colMeans(x);
rowSums(x, na.rm = TRUE); colSums(x, na.rm = TRUE);
rowMeans(x, na.rm = TRUE); colMeans(x, na.rm = TRUE)

## an array
dim(UCBAdmissions)
rowSums(UCBAdmissions); rowSums(UCBAdmissions, dims = 2)
colSums(UCBAdmissions); colSums(UCBAdmissions, dims = 2)

## complex case
x <- cbind(x1 = 3 + 2i, x2 = c(4:1, 2:5) - 5i)
x[3, ] <- NA; x[4, 2] <- NA
rowSums(x); colSums(x); rowMeans(x); colMeans(x);
rowSums(x, na.rm = TRUE); colSums(x, na.rm = TRUE);
rowMeans(x, na.rm = TRUE); colMeans(x, na.rm = TRUE)

commandArgs

Extract Command Line Arguments

Description

Provides access to a copy of the command line arguments supplied when this R session was invoked.

Usage

commandArgs()

Details

These arguments are captured before the standard R command line processing takes place. This means that they are the unmodified values. If it were useful, we could provide support an argument which indicated whether we want the unprocessed or processed values.

This is especially useful with the --args command-line flag to R, as all of the command line after than flag is skipped.
Value

A character vector containing the name of the executable and the user-supplied command line arguments. The first element is the name of the executable by which R was invoked. As far as I am aware, the exact form of this element is platform dependent. It may be the fully qualified name, or simply the last component (or basename) of the application.

See Also

Startup

Examples

commandArgs()

## Spawn a copy of this application as it was invoked.
## system(paste(commandArgs(), collapse=" "))

comment

Query or Set a ‘Comment’ Attribute

Description

These functions set and query a comment attribute for any R objects. This is typically useful for data.frames or model fits.

Contrary to other attributes, the comment is not printed (by print or print.default).

Usage

comment(x)

comment(x) <- value

Arguments

x

any R object

value

a character vector

See Also

attributes and attr for “normal” attributes.

Examples

x <- matrix(1:12, 3,4)

comment(x) <- c("This is my very important data from experiment #0234",
"Jun 5, 1998")

x

comment(x)
Comparison

**Relational Operators**

**Description**

Binary operators which allow the comparison of values in atomic vectors.

**Usage**

- `x < y`
- `x > y`
- `x <= y`
- `x >= y`
- `x == y`
- `x != y`

**Arguments**

`x, y`  atomic vectors, or other objects for which methods have been written.

**Details**

The binary comparison operators are generic functions: methods can be written for them individually or via the `Ops` group generic function.

Comparison of strings in character vectors is lexicographic within the strings using the collating sequence of the locale in use: see `locales`. The collating sequence of locales such as ‘en_US’ is normally different from ‘C’ (which should use ASCII) and can be surprising.

At least one of `x` and `y` must be an atomic vector, but if the other is a list R attempts to coerce it to the type of the atomic vector: this will succeed if the list is made up of elements of length one that can be coerced to the correct type.

If the two arguments are atomic vectors of different types, they are both coerced to the first of character, complex, numeric, integer and logical.

**Value**

A vector of logicals indicating the result of the element by element comparison. The elements of shorter vectors are recycled as necessary.

Objects such as arrays or time-series can be compared this way provided they are conformable.

**Note**

Don’t use `==` and `!=` for tests, such as in `if` expressions, where you must get a single `TRUE` or `FALSE`. Unless you are absolutely sure that nothing unusual can happen, you should use the `identical` function instead.

For numerical values, remember `==` and `!=` do not allow for the finite representation of fractions, nor for rounding error. Using `all.equal` with `identical` is almost always preferable. See the examples.
References


See Also

*Syntax* for operator precedence.

Examples

```r
x <- rnorm(20)
x < 1
x[x > 0]
x1 <- 0.5 - 0.3
x2 <- 0.3 - 0.1
x1 == x2 # FALSE on most machines
identical(all.equal(x1, x2), TRUE) # TRUE everywhere
```

---

**complex**

**Complex Vectors**

Description

Basic functions which support complex arithmetic in R.

Usage

```r
complex(length.out = 0, real = numeric(), imaginary = numeric(),
    modulus = 1, argument = 0)
as.complex(x, ...)
is.complex(x)

Re(x)
Im(x)
Mod(x)
Arg(x)
Conj(x)
```

Arguments

- `length.out`: numeric. Desired length of the output vector, inputs being recycled as needed.
- `real`: numeric vector.
- `imaginary`: numeric vector.
- `modulus`: numeric vector.
- `argument`: numeric vector.
- `x`: an object, probably of mode `complex`.
- `...`: further arguments passed to or from other methods.
Details

Complex vectors can be created with complex. The vector can be specified either by giving its length, its real and imaginary parts, or modulus and argument. (Giving just the length generates a vector of complex zeroes.)

as.complex attempts to coerce its argument to be of complex type: like as.vector it strips attributes including names.

Note that is.complex and is.numeric are never both TRUE.

The functions Re, Im, Mod, Arg and Conj have their usual interpretation as returning the real part, imaginary part, modulus, argument and complex conjugate for complex values. Modulus and argument are also called the polar coordinates. If \( z = x + iy \) with real \( x \) and \( y \), Mod\((z) = \sqrt{x^2 + y^2}\), and for \( \phi = \text{Arg}(z) \), \( x = \cos(\phi) \) and \( y = \sin(\phi) \). They are all generic functions: methods can be defined for them individually or via the Complex group generic.

In addition, the elementary trigonometric, logarithmic and exponential functions are available for complex values.

is.complex is generic: you can write methods to handle specific classes of objects, see Internal-Methods.

References


Examples

0i ^ (-3:3)

matrix(1i^ (-6:5), nr=4)#- all columns are the same
0 ^ 1i # a complex NaN

## or also (less efficiently):
z2 <- 1:2 + 1i*(8:9)

zz.shift <- complex(modulus = Mod(zz), argument= Arg(zz) + pi)
plot(zz, xlim=c(-1,1), ylim=c(-1,1), col="red", asp = 1,
    main = expression(paste("Rotation by ", \pi == 180^o)))
abline(h=0,v=0, col="blue", lty=3)
points(zz.shift, col="orange")
## conditions

### Usage

- `tryCatch(expr, ..., finally)`
- `withCallingHandlers(expr, ...)`
- `signalCondition(cond)`
- `simpleCondition(message, call = NULL)`
- `simpleError(message, call = NULL)`
- `simpleWarning(message, call = NULL)`
- `simpleMessage(message, call = NULL)`
- `## S3 method for class 'condition': as.character(x, ...)`
- `## S3 method for class 'error': as.character(x, ...)`
- `## S3 method for class 'condition': print(x, ...)`
- `## S3 method for class 'restart': print(x, ...)`
- `conditionCall(c)`
- `## S3 method for class 'condition': conditionCall(c)`
- `conditionMessage(c)`
- `## S3 method for class 'condition': conditionMessage(c)`
- `withRestarts(expr, ...)`
- `computeRestarts(cond = NULL)`
- `findRestart(name, cond = NULL)`
- `invokeRestart(r, ...)`
- `invokeRestartInteractively(r)`
- `isRestart(x)`
- `restartDescription(r)`
- `restartFormals(r)`
- `.signalSimpleWarning(msg, call)`
- `.handleSimpleError(h, msg, call)`

### Arguments

- `c` a condition object.
- `call` call expression.
- `cond` a condition object.
- `expr` expression to be evaluated.
- `finally` expression to be evaluated before returning or exiting.
- `h` function.
- `message` character string.
**Details**

The condition system provides a mechanism for signaling and handling unusual conditions, including errors and warnings. Conditions are represented as objects that contain information about the condition that occurred, such as a message and the call in which the condition occurred. Currently conditions are S3-style objects, though this may eventually change.

Conditions are objects inheriting from the abstract class `condition`. Errors and warnings are objects inheriting from the abstract subclasses `error` and `warning`. The class `simpleError` is the class used by `stop` and all internal error signals. Similarly, `simpleWarning` is used by `warning`, and `simpleMessage` is used by `message`. The constructors by the same names take a string describing the condition as argument and an optional call. The functions `conditionMessage` and `conditionCall` are generic functions that return the message and call of a condition.

Conditions are signaled by `signalCondition`. In addition, the `stop` and `warning` functions have been modified to also accept condition arguments.

The function `tryCatch` evaluates its expression argument in a context where the handlers provided in the `. . .` argument are available. The `finally` expression is then evaluated in the context in which `tryCatch` was called; that is, the handlers supplied to the current `tryCatch` call are not active when the `finally` expression is evaluated.

Handlers provided in the `. . .` argument to `tryCatch` are established for the duration of the evaluation of `expr`. If no condition is signaled when evaluating `expr` then `tryCatch` returns the value of the expression.

If a condition is signaled while evaluating `expr` then established handlers are checked, starting with the most recently established ones, for one matching the class of the condition. When several handlers are supplied in a single `tryCatch` then the first one is considered more recent than the second. If a handler is found then control is transferred to the `tryCatch` call that established the handler, the handler found and all more recent handlers are disestablished, the handler is called with the condition as its argument, and the result returned by the handler is returned as the value of the `tryCatch` call.

Calling handlers are established by `withCallingHandlers`. If a condition is signaled and the applicable handler is a calling handler, then the handler is called by `signalCondition` in the context where the condition was signaled but with the available handlers restricted to those below the handler called in the handler stack. If the handler returns, then the next handler is tried; once the last handler has been tried, `signalCondition` returns `NULL`.

User interrupts signal a condition of class `interrupt` that inherits directly from class `condition` before executing the default interrupt action.

Restarts are used for establishing recovery protocols. They can be established using `withRestarts`. One pre-established restart is an `abort` restart that represents a jump to top level.

`findRestart` and `computeRestarts` find the available restarts. `findRestart` returns the most recently established restart of the specified name. `computeRestarts` returns a list of all restarts. Both can be given a condition argument and will then ignore restarts that do not apply to the condition.
invokeRestart transfers control to the point where the specified restart was established and calls the restart's handler with the arguments, if any, given as additional arguments to invokeRestart. The restart argument to invokeRestart can be a character string, in which case findRestart is used to find the restart.

New restarts for withRestarts can be specified in several ways. The simplest is in name=function form where the function is the handler to call when the restart is invoked. Another simple variant is as name=string where the string is stored in the description field of the restart object returned by findRestart; in this case the handler ignores its arguments and returns NULL. The most flexible form of a restart specification is as a list that can include several fields, including handler, description, and test. The test field should contain a function of one argument, a condition, that returns TRUE if the restart applies to the condition and FALSE if it does not; the default function returns TRUE for all conditions.

One additional field that can be specified for a restart is interactive. This should be a function of no arguments that returns a list of arguments to pass to the restart handler. The list could be obtained by interacting with the user if necessary. The function invokeRestartInteractively calls this function to obtain the arguments to use when invoking the restart. The default interactive method queries the user for values for the formal arguments of the handler function.

.signalSimpleWarning and .handleSimpleError are used internally and should not be called directly.

References

The tryCatch mechanism is similar to Java error handling. Calling handlers are based on Common Lisp and Dylan. Restarts are based on the Common Lisp restart mechanism.

See Also

stop and warning signal conditions, and try is essentially a simplified version of tryCatch.

Examples

tryCatch(1, finally=print("Hello"))
e <- simpleError("test error")
## Not run:
  stop(e)
  tryCatch(stop(e), finally=print("Hello"))
  tryCatch(stop("fred"), finally=print("Hello"))
## End(Not run)
tryCatch(stop(e), error = function(e) e, finally=print("Hello"))
tryCatch(stop("fred"), error = function(e) e, finally=print("Hello"))
withCallingHandlers({ warning("A"); 1+2 }, warning = function(w) {})
  { try(invokeRestart("tryRestart")); 1}
## Not run:
  withRestarts(stop("A"), abort = function() {}); 1
## End(Not run)
withRestarts(invokeRestart("foo", 1, 2), foo = function(x, y) (x + y))
conflicts  

**Search for Masked Objects on the Search Path**

**Description**

`conflicts` reports on objects that exist with the same name in two or more places on the search path, usually because an object in the user’s workspace or a package is masking a system object of the same name. This helps discover unintentional masking.

**Usage**

```r
conflicts(where = search(), detail = FALSE)
```

**Arguments**

- `where` A subset of the search path, by default the whole search path.
- `detail` If `TRUE`, give the masked or masking functions for all members of the search path.

**Value**

If `detail=FALSE`, a character vector of masked objects. If `detail=TRUE`, a list of character vectors giving the masked or masking objects in that member of the search path. Empty vectors are omitted.

**Examples**

```r
lm <- 1:3
conflicts(where = search(), detail = TRUE)
## gives something like
# $.GlobalEnv
# [1] "lm"
#
# $package:base
# [1] "lm"

## Remove things from your "workspace" that mask others:
remove(list = conflicts(detail=TRUE)$.GlobalEnv)
```

connections  

**Functions to Manipulate Connections**

**Description**

Functions to create, open and close connections.
connections

Usage

file(description = "", open = "", blocking = TRUE,
encoding = getOption("encoding"))
pipe(description, open = "", encoding = getOption("encoding"))
fifo(description = "", open = "", blocking = FALSE,
encoding = getOption("encoding"))
gzfile(description, open = "", encoding = getOption("encoding"),
compression = 6)
unz(description, filename, open = "", encoding = getOption("encoding"))
bzfile(description, open = "", encoding = getOption("encoding"))
url(description, open = "", blocking = TRUE,
encoding = getOption("encoding"))
socketConnection(host = "localhost", port, server = FALSE,
blocking = FALSE, open = "a+",
encoding = getOption("encoding"))

open(con, ...)
## S3 method for class 'connection':
open(con, open = "r", blocking = TRUE, ...)

close(con, ...)
## S3 method for class 'connection':
close(con, type = "rw", ...)

flush(con)

isOpen(con, rw = "")
isIncomplete(con)

Arguments

description character. A description of the connection. For file and pipe this is a path
to the file to be opened. For url it is a complete URL, including schemes
(http://, ftp:// or file:// – see Details). file also accepts complete
URLs.
filename a filename within a zip file.
con a connection.
host character. Host name for port.
port integer. The TCP port number.
server logical. Should the socket be a client or a server?
open character. A description of how to open the connection (if at all). See Details
for possible values.
blocking logical. See the ‘Blocking’ section below.
encoding The name of the encoding to be used. See the ‘Encoding’ section below.
compression integer in 0–9. The amount of compression to be applied when writing, from
none to maximal. The default is a good space/time compromise.
type character. Currently ignored.
rw character. Empty or "read" or "write", partial matches allowed.
... arguments passed to or from other methods.
Details

The first eight functions create connections. By default the connection is not opened (except for module `socketConnection`), but may be opened by setting a non-empty value of argument `open`.

Module `gzfile` applies to files compressed by `gzip`, and `bzfile` to those compressed by `bzip2`: such connections can only be binary.

Module `unz` reads (only) single files within zip files, in binary mode. The description is the full path, with `.zip` extension if required.

All platforms support module `file`, `gzfile`, `bzfile`, `unz` and `url("file://")` connections. The other types may be partially implemented or not implemented at all. (They do work on most Unix platforms, and all but `fifo` on Windows.)

Proxies can be specified for `url` connections: see module `download.file`.

Open, close and seek are generic functions: the following applies to the methods relevant to connections.

Open opens a connection. In general functions using connections will open them if they are not open, but then close them again, so to leave a connection open call `open` explicitly.

Possible values for the mode `open` to open a connection are

"r" or "rt" Open for reading in text mode.
"w" or "wt" Open for writing in text mode.
"a" or "at" Open for appending in text mode.
"rb" Open for reading in binary mode.
"wb" Open for writing in binary mode.
"ab" Open for appending in binary mode.
"r+", "r+b" Open for reading and writing.
"w+", "w+b" Open for reading and writing, truncating file initially.
"a+", "a+b" Open for reading and appending.

Not all modes are applicable to all connections: for example URLs can only be opened for reading. Only file and socket connections can be opened for reading and writing/append. For many connections there is little or no difference between text and binary modes, but there is for file-like connections on Windows, and module `pushBack` is text-oriented and is only allowed on connections open for reading in text mode.

Close closes and destroys a connection.

Flush flushes the output stream of a connection open for write/append (where implemented).

If for a `file` connection the description is ",", the file is immediately opened (in "w+" mode unless `open="w+b` is specified) and unlinked from the file system. This provides a temporary file to write to and then read from.

A note on `file://` URLs. The most general form (from RFC1738) is `file://host/path/to/file`, but R only accepts the form with an empty `host` field referring to the local machine. In this form the path is relative to the root of the filesystem, not a Windows concept. The standard form is `file://d:/R/repos`: for compatibility with earlier versions of R and Unix versions, any other form is parsed as R as `file://` plus `/path/to/file`. Also, backslashes are accepted within the path even though RFC1738 does not allow them. Also, no attempt is made to decode an encoded URL: call `URLdecode` if necessary.
Value

file, pipe, fifo, url, gzfile, bzfile, unz and socketConnection return a connection object which inherits from class "connection" and has a first more specific class.

isOpen returns a logical value, whether the connection is currently open.

isIncomplete returns a logical value, whether last read attempt was blocked, or for an output text connection whether there is unflushed output.

Encoding

Note: prior to R 2.1.0 there was a byte-by-byte encoding option applied to input only. This has been replaced by a more comprehensive scheme.

The encoding of the input/output stream of a connection in text mode can be specified by name, in the same way as it would be given to iconv: see that help page for how to find out what names are recognized on your platform. Additionally, "" and "native.enc" both mean the ‘native’ encoding, that is the internal encoding of the current locale and hence no translation is done.

Re-encoding only works for connections in text mode.

The encoding "UCS-2LE" is treated specially, as it is the appropriate value for Windows ‘Unicode’ text files. If the first two bytes are the Byte Order Mark 0xFFFE then these are removed as most implementations of iconv do not accept BOMs. Note that some implementations (including that used on Windows) will handle BOMs using encoding "UCS2" but many (including that in glibc) will not.

Exactly what happens when the requested translation cannot be done is in general undocumented. Requesting a conversion that is not supported is an error, reported when the connection is opened.

On output the result is likely to be that up to the error, with a warning. On input, it will most likely be all or some of the input up to the error.

The encoding for stdin when redirected from a file can be set by the command-line flag --encoding.

Blocking

The default condition for all but fifo and socket connections is to be in blocking mode. In that mode, functions do not return to the R evaluator until they are complete. In non-blocking mode, operations return as soon as possible, so on input they will return with whatever input is available (possibly none) and for output they will return whether or not the write succeeded.

The function readLines behaves differently in respect of incomplete last lines in the two modes: see its help page.

Even when a connection is in blocking mode, attempts are made to ensure that it does not block the event loop and hence the operation of GUI parts of R. These do not always succeed, and the whole process will be blocked during a DNS lookup on Unix, for example.

Most blocking operations on URLs and sockets are subject to the timeout set by options("timeout"). Note that this is a timeout for no response at all, not for the whole operation.

Fifos

Fifos default to non-blocking. That follows Svr4 and it probably most natural, but it does have some implications. In particular, opening a non-blockingfifo connection for writing (only) will fail unless some other process is reading on the fifo.

Opening a fifo for both reading and writing (in any mode: one can only append to fifos) connects both sides of the fifo to the R process, and provides an similar facility to file().
Clipboard

file can also be used with description = "clipboard" in modes "r" and "w" only.

When the clipboard is opened for reading, the contents are immediately copied to internal storage in the connection.

When writing to the clipboard, the output is copied to the clipboard only when the connection is closed or flushed. There is a 32Kb limit on the text to be written to the clipboard. This can be raised by using e.g. file("clipboard-128") on NT-based versions of Windows, to give 128Kb.

At least on NT-based versions of Windows the clipboard works in Unicode wide characters, so encodings are likely not to work as one might expect.

Note

R’s connections are modelled on those in S version 4 (see Chambers, 1998). However R goes well beyond the Svr4 model, for example in output text connections and URL, gzfile, bzfile and socket connections.

The default mode in R is "r" except for socket connections. This differs from Svr4, where it is the equivalent of "r+", known as "*".

On platforms where vsnprintf does not return the needed length of output (e.g., Windows) there is a 100,000 character output limit on the length of line for fifo, gzfile and bzfile connections: longer lines will be truncated with a warning.

References


See Also

textConnection, seek, readLines, readBin, writeLines, writeBin, showConnections, pushBack.
capabilities to see if url, fifo and socketConnection are supported by this build of R.

To flush output to the console, see flush.console.

Examples

```
zz <- file("ex.data", "w")  # open an output file connection
cat("TITLE extra line", "2 3 5 7", "", "11 13 17", file = zz, sep = "\n")
cat("One more line\n", file = zz)
close(zz)
readLines("ex.data")
unlink("ex.data")

zz <- gzfile("ex.gz", "w")  # compressed file
cat("TITLE extra line", "2 3 5 7", "", "11 13 17", file = zz, sep = "\n")
close(zz)
readLines(gzfile("ex.gz"))
unlink("ex.gz")

zz <- bzfile("ex.bz2", "w")  # bzip2-ed file
cat("TITLE extra line", "2 3 5 7", "", "11 13 17", file = zz, sep = "\n")
close(zz)
print(readLines(bzfile("ex.bz2")))
```
connections

unlink("ex.bz2")

## An example of a file open for reading and writing
Tfile <- file("test1", "w+")
c(isOpen(Tfile, "r"), isOpen(Tfile, "w")) # both TRUE
cat("abc\ndef\n", file=Tfile)
readLines(Tfile)
seek(Tfile, 0, rw="r") # reset to beginning
readLines(Tfile)
cat("ghi\n", file=Tfile)
readLines(Tfile)
close(Tfile)
unlink("test1")

## We can do the same thing with an anonymous file.
Tfile <- file()
cat("abc\ndef\n", file=Tfile)
readLines(Tfile)
close(Tfile)

if(capabilities("fifo")) {
z <- fifo("foo", "w+")
writeLines("abc", z)
print(readLines(z))
close(z)
unlink("foo")
}

## Not run: ## Unix examples of use of pipes
# read listing of current directory
readLines(pipe("ls -l"))

# remove trailing commas. Suppose
% cat data2
450, 390, 467, 654, 30, 542, 334, 432, 421,
357, 497, 493, 550, 549, 467, 575, 578, 342,
446, 547, 534, 495, 979, 479
# Then read this by
scan(pipe("sed -e s/,$// data2"), sep="",")

# convert decimal point to comma in output
# both R strings and (probably) the shell need \ doubled
z <- pipe(paste("sed s/\ ./,/>", "outfile"), "w")
cat(format(round(rnorm(100), 4)), sep = "\n", file = z)
close(z)
file.show("outfile", delete.file=TRUE)## End(Not run)

## Not run: ## example for Unix machine running a finger daemon
con <- socketConnection(port = 79, blocking = TRUE)
writeLines(paste(system("whoami", intern=TRUE), "\r", sep=""), con)
gsub(" *$", "", readLines(con))
close(con)## End(Not run)

## Not run: ## two R processes communicating via non-blocking sockets
# R process 1
```r
con1 <- socketConnection(port = 6011, server=TRUE)
writeLines(LETTERS, con1)
close(con1)

# R process 2
con2 <- socketConnection(Sys.info()["nodename"], port = 6011)
# as non-blocking, may need to loop for input
readLines(con2)
while(isIncomplete(con2)) {Sys.sleep(1); readLines(con2)}
close(con2)
## End(Not run)
## Not run:
## examples of use of encodings
cat(x, file = file("foo", "w", encoding="UTF-8"))
# read a 'Windows Unicode' file including names
A <- read.table(file("students", encoding="UCS-2LE"))
## End(Not run)
```

---

### Constants

<table>
<thead>
<tr>
<th>Description</th>
<th>Built-in Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constants built into R.</td>
<td></td>
</tr>
</tbody>
</table>

### Usage

- **LETTERS**
- **letters**
- **month.abb**
- **month.name**
- **pi**

### Details

R has a limited number of built-in constants (there is also a rather larger library of data sets which can be loaded with the function `data`).

The following constants are available:

- **LETTERS**: the 26 upper-case letters of the Roman alphabet;
- **letters**: the 26 lower-case letters of the Roman alphabet;
- **month.abb**: the three-letter abbreviations for the English month names;
- **month.name**: the English names for the months of the year;
- **pi**: the ratio of the circumference of a circle to its diameter.

### References

Control

See Also
data, DateTimeClasses. Quotes for the parsing of character constants, NumericConstants for numeric constants.

Examples

# John Machin (1705) computed 100 decimals of pi :
pi - 4* (4*atan(1/5) - atan(1/239))

## months in English
month.name
## months in your current locale
format(ISOdate(2000, 1:12, 1), "%B")
format(ISOdate(2000, 1:12, 1), "%b")

contributors          R Project Contributors

Description

The R Who-is-who, describing who made significant contributions to the development of R.

Usage

contributors()

Control Flow

Description

These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like language.

Usage

if(cond) expr
if(cond) cons.expr else alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next
Arguments

cond
A length-one logical vector that is not NA. Conditions of length greater than one are accepted with a warning, but only the first element is used.

var
A syntactical name for a variable.

seq
An expression evaluating to a vector (including a list).

expr, cons.expr, alt.expr
An expression in a formal sense. This is either a simple expression or a so called compound expression, usually of the form \{ expr1 ; expr2 \}.

Details

break breaks out of a for, while or repeat loop; control is transferred to the first statement outside the inner-most loop. next halts the processing of the current iteration and advances the looping index. Both break and next apply only to the innermost of nested loops.

Note that it is a common mistake to forget to put braces (\{ \ldots \}) around your statements, e.g., after if(\ldots) or for(\ldots). In particular, you should not have a newline between } and else to avoid a syntax error in entering a if \ldots else construct at the keyboard or via source. For that reason, one (somewhat extreme) attitude of defensive programming is to always use braces, e.g., for if clauses.

The index seq in a for loop is evaluated at the start of the loop; changing it subsequently does not affect the loop. The variable var has the same type as seq. If seq is a factor (which is not strictly allowed) then its internal codes are used: the effect is that of \texttt{as.integer} not \texttt{as.vector}.

References


See Also

Syntax for the basic \texttt{R} syntax and operators, Paren for parentheses and braces; further, ifelse, switch.

Examples

```r
for(i in 1:5) print(1:i)
for(n in c(2,5,10,20,50)) {
  x <- rnorm(n)
  cat(n,":", sum(x^2),"\n")
}
```

Description

\texttt{R} is released under the ‘GNU Public License’: see license for details. The license describes your right to use \texttt{R}. Copyright is concerned with ownership of intellectual rights, and some of the software used has conditions that the copyright must be explicitly stated: see the Details section.

We are grateful to these people and other contributors (see contributors) for the ability to use their work.
count.fields

Count the Number of Fields per Line

Description

count.fields counts the number of fields, as separated by sep, in each of the lines of file read.

Usage

count.fields(file, sep = "", quote = "\"", skip = 0,
    blank.lines.skip = TRUE, comment.char = "#")

Arguments

file a character string naming an ASCII data file, or a connection, which will be opened if necessary, and if so closed at the end of the function call.
sep the field separator character. Values on each line of the file are separated by this character. By default, arbitrary amounts of whitespace can separate fields.
quote the set of quoting characters
skip the number of lines of the data file to skip before beginning to read data.
blank.lines.skip logical: if TRUE blank lines in the input are ignored.
comment.char character: a character vector of length one containing a single character or an empty string.

Details

This used to be used by read.table and can still be useful in discovering problems in reading a file by that function.

For the handling of comments, see scan.

Value

A vector with the numbers of fields found.

See Also

read.table

Examples

cat("NAME", "1:John", "2:Paul", file = "foo", sep = "\n")
count.fields("foo", sep = ":")
unlink("foo")
crossprod  

Matrix Crossproduct

Description
Given matrices x and y as arguments, crossprod returns their matrix cross-product. This is formally equivalent to, but faster than, the call t(x) %*% y.

Usage
crossprod(x, y = NULL)

Arguments
x, y matrices: y = NULL is taken to be the same matrix as x.

References

See Also
%*% and outer product %o%.

Examples
(z <- crossprod(1:4))  # = sum(1 + 2^2 + 3^2 + 4^2)
drop(z)                # scalar

---

cumsum  

Cumulative Sums, Products, and Extremes

Description
Returns a vector whose elements are the cumulative sums, products, minima or maxima of the elements of the argument.

Usage
cumsum(x)
cumprod(x)
cummax(x)
cummin(x)

Arguments
x a numeric object.
Details
An NA value in x causes the corresponding and following elements of the return value to be NA.
These are generic functions: methods can be defined for them individually or via the Math group generic.

References

Examples
```
cumsum(1:10)
cumprod(1:10)
cummin(c(3:1, 2:0, 4:2))
cummax(c(3:1, 2:0, 4:2))
```

---

**cut**

Convert Numeric to Factor

Description
cut divides the range of x into intervals and codes the values in x according to which interval they fall. The leftmost interval corresponds to level one, the next leftmost to level two and so on.

Usage
cut(x, ...)

## Default S3 method:
cut(x, breaks, labels = NULL,
    include.lowest = FALSE, right = TRUE, dig.lab = 3, ...)

Arguments
x
a numeric vector which is to be converted to a factor by cutting.

breaks
either a vector of cut points or number giving the number of intervals which x is to be cut into.

labels
labels for the levels of the resulting category. By default, labels are constructed using ",(a,b)" interval notation. If labels = FALSE, simple integer codes are returned instead of a factor.

include.lowest
logical, indicating if an 'x[i]' equal to the lowest (or highest, for right = FALSE) 'breaks' value should be included.

right
logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa.

dig.lab
integer which is used when labels are not given. It determines the number of digits used in formatting the break numbers.

... further arguments passed to or from other methods.
**Details**

If a `labels` parameter is specified, its values are used to name the factor levels. If none is specified, the factor level labels are constructed as "(b1, b2)". "(b2, b3)" etc. for `right = TRUE` and as "[b1, b2)"... if `right = FALSE`. In this case, `dig.lab` indicates the minimum number of digits should be used in formatting the numbers b1, b2, .... A larger value (up to 12) will be used if needed to distinguish between any pair of endpoints: if this fails labels such as "Range3" will be used.

**Value**

A factor is returned, unless `labels = FALSE` which results in the mere integer level codes.

**Note**

Instead of `table(cut(x, br))`, `hist(x, br, plot = FALSE)` is more efficient and less memory hungry. Instead of `cut(*, labels = FALSE)`, `findInterval()` is more efficient.

**References**


**See Also**

`split` for splitting a variable according to a group factor; `factor`, `tabulate`, `table`, `findInterval()`.

**Examples**

```r
Z <- rnorm(10000)
table(cut(Z, br = -6:6))
sum(table(cut(Z, br = -6:6, labels=FALSE)))
sum( hist (Z, br = -6:6, plot=FALSE)$counts)

xa <- cut(rep(1,5),4)#-- dummy
txo <- c(9, 4, 6, 5, 3, 10, 5, 3, 5)
x <- rep(0:8, tx0)
stopifnot(table(x) == tx0)
table(cut(x, b = 8))
table( cut(x, br = 3*(-2:5)))
table( cut(x, br = 3*(-2:5), right = FALSE))

##--- some values OUTSIDE the breaks :
table(cx <- cut(x, br = 2*(0:4)))
table(cxl <- cut(x, br = 2*(0:4), right = FALSE))
which(is.na(cx)); x[is.na(cx)]  ## the first 9 values 0
which(is.na(cxl)); x[is.na(cxl)]  ## the last 5 values 8

## Label construction:
y <- rnorm(100)
table(cut(y, breaks = pi/3*(-3:3)))
table(cut(y, breaks = pi/3*(-3:3), dig.lab=4))
```
table(cut(y, breaks = 1*(-3:3), dig.lab=4))
# extra digits don't "harm" here
# the same, since no exact INT!
table(cut(y, breaks = 1*(-3:3), right = FALSE))

## sometimes the default dig.lab is not enough to be avoid confusion:
## sometimes the default dig.lab is not enough to be avoid confusion:

### S3 method for class 'POSIXt':

```r
cut(x, breaks, labels = NULL, start.on.monday = TRUE,
    right = FALSE, ...)
```

### S3 method for class 'Date':

```r
cut(x, breaks, labels = NULL, start.on.monday = TRUE,
    right = FALSE, ...)
```

### Arguments

- **x**: an object inheriting from class "POSIXt" or "Date".
- **breaks**: a vector of cut points or number giving the number of intervals which x is to be cut into or an interval specification, one of "sec", "min", "hour", "day", "DSTday", "week", "month" or "year", optionally preceded by an integer and a space, or followed by "s". For "Date" objects only "day", "week", "month" and "year" are allowed.
- **labels**: labels for the levels of the resulting category. By default, labels are constructed from the left-hand end of the intervals (which are include for the default value of right). If labels = FALSE, simple integer codes are returned instead of a factor.
- **start.on.monday**: logical. If breaks = "weeks", should the week start on Mondays or Sundays?
- **right**, **...**: arguments to be passed to or from other methods.

### Details

Using both right = TRUE and include.lowest = TRUE will include both ends of the range of dates.

### Value

A factor is returned, unless labels = FALSE which returns the integer level codes.
**data.class**

See Also

seq.POSIXt, seq.Date, cut

Examples

```r
## random dates in a 10-week period
cut(ISOdate(2001, 1, 1) + 70*86400*runif(100), "weeks")
cut(as.Date("2001/1/1") + 70*runif(100), "weeks")
```

Description

Determine the class of an arbitrary R object.

Usage

data.class(x)

Arguments

x

an R object.

Value

character string giving the “class” of x.

The “class” is the (first element) of the class attribute if this is non-NULL, or inferred from the object’s dim attribute if this is non-NULL, or mode(x).

Simply speaking, data.class(x) returns what is typically useful for method dispatching. (Or, what the basic creator functions already and maybe eventually all will attach as a class attribute.)

Note

For compatibility reasons, there is one exception to the rule above: When x is integer, the result of data.class(x) is "numeric" even when x is classed.

See Also

class

Examples

```r
x <- LETTERS
data.class(factor(x))  # has a class attribute
data.class(matrix(x, nc = 13))  # has a dim attribute
data.class(list(x))  # the same as mode(x)
data.class(x)  # the same as mode(x)

stopifnot(data.class(1:2) == "numeric")  # compatibility "rule"
```
data.frame

Data Frames

Description
This function creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of R's modeling software.

Usage
```
data.frame(..., row.names = NULL, check.rows = FALSE,
           check.names = TRUE)
```

Arguments
- `...` these arguments are of either the form `value` or `tag=value`. Component names are created based on the tag (if present) or the deparsed argument itself.
- `row.names` NULL or an integer or character string specifying a column to be used as row names, or a character vector giving the row names for the data frame.
- `check.rows` if `TRUE` then the rows are checked for consistency of length and names.
- `check.names` logical. If `TRUE` then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names and are not duplicated. If necessary they are adjusted (by `make.names`) so that they are.

Details
A data frame is a list of variables of the same length with unique row names, given class "data.frame". data.frame converts each of its arguments to a data frame by calling `as.data.frame(optional=TRUE)`. As that is a generic function, methods can be written to change the behaviour of arguments according to their classes: R comes with many such methods. Character variables passed to `data.frame` are converted to factor columns unless protected by `I`. If a list or data frame or matrix is passed to `data.frame` it is as if each component or column had been passed as a separate argument.

Objects passed to `data.frame` should have the same number of rows, but atomic vectors, factors and character vectors protected by `I` will be recycled a whole number of times if necessary.

If row names are not supplied in the call to `data.frame`, the row names are taken from the first component that has suitable names, for example a named vector or a matrix with rownames or a data frame. (If that component is subsequently recycled, the names are discarded with a warning.) If `row.names` was supplied as `NULL` or no suitable component was found the row names are the integer sequence starting at one.

If row names are supplied of length one and the data frame has a single row, the `row.names` is taken to specify the row names and not a column (by name or number).

Names are removed from vector inputs not protected by `I`.

Value
A data frame, a matrix-like structure whose columns may be of differing types (numeric, logical, factor and character and so on).
Note

In versions of R prior to 1.4.0 logical columns were converted to factors (as in S3 but not S4).

References


See Also

I.plot.data.frame, print.data.frame, row.names, [.data.frame for subsetting methods, Math.data.frame etc, about Group methods for data.frames; read.table, make.names.

Examples

L3 <- LETTERS[1:3]
(d <- data.frame(cbind(x=1, y=1:10), fac=sample(L3, 10, repl=TRUE)))

## The same with automatic column names:
data.frame(cbind( 1, 1:10), sample(L3, 10, repl=TRUE))

is.data.frame(d)

## do not convert to factor, using I() :
(dd <- cbind(d, char = I(letters[1:10])))
rbind(class=sapply(dd, class), mode=sapply(dd, mode))

stopifnot(1:10 == row.names(d))# {coercion}

(d0  <- d[, FALSE]) # NULL data frame with 10 rows
(d.0 <- d[FALSE, ]) # <0 rows> data frame (3 cols)
(d00 <- d0[FALSE,]) # NULL data frame with 0 rows

---

**data.matrix**

Data Frame to Numeric Matrix

Description

Return the matrix obtained by converting all the variables in a data frame to numeric mode and then binding them together as the columns of a matrix. Factors and ordered factors are replaced by their internal codes.

Usage

data.matrix(frame)

Arguments

frame a data frame whose components are logical vectors, factors or numeric vectors.
Suppling a data frame with columns which are not numerical or logical is an error. A warning is given if any non-factor column as a class, as then information can be lost.

References


See Also

`as.matrix`, `data.frame`, `matrix`.

Examples

```r
DF <- data.frame(a=1:3, b=letters[10:12],
                 c=seq(as.Date("2004-01-01"), by = "week", len = 3))
data.matrix(DF[1:2])
data.matrix(DF) # gives a warning and quotes dates as #days since 1970.
```

---

## date

### System Date and Time

**Description**

Returns a character string of the current system date and time.

**Usage**

```r
date()
```

**Value**

The string has the form "Fri Aug 20 11:11:00 1999", i.e., length 24, since it relies on POSIX's `ctime` ensuring the above fixed format. Timezone and Daylight Saving Time are taken account of, but not indicated in the result.

The day and month abbreviations are always in English, irrespective of locale.

**References**


**See Also**

`Sys.time`

**Examples**

```r
(d <- date())
nchar(d) == 24

## something similar in the current locale
format(Sys.time(), "%a %b %d %H:%M:%S %Y")
```
### Dates

#### Date Class

**Description**

Description of the class "Date" representing calendar dates.

**Details**

Dates are represented as the number of days since 1970-01-01, with negative values for earlier dates. They are always printed following the rules of the current Gregorian calendar, even though that calendar was not in use long ago (it was adopted in 1752 in Great Britain and its colonies).

It is intended that the date should be an integer, but this is not enforced in the internal representation. Fractional days will be ignored when printing. It is possible to produce fractional days via the `mean` method or by adding or subtracting an object of class "difftime".

**See Also**

- `Sys.Date` for the current date.
- `format.Date` for conversion to and from character strings.
- `plot.Date` and `hist.Date` for plotting.
- `weekdays` for convenience extraction functions.
- `seq.Date`, `cut.Date`, `round.Date` for utility operations.
- `DateTimeClasses` for date-time classes.

**Examples**

```r
(today <- Sys.Date())
format(today, "%d %b %Y")  # with month as a word
(tenweeks <- seq(today, len=10, by="1 week"))  # next ten weeks
weekdays(today)
months(tenweeks)
as.Date(.leap.seconds)
```

### DateTimeClasses

#### Date-Time Classes

**Description**

Description of the classes "POSIXlt" and "POSIXct" representing calendar dates and times (to the nearest second).
Usage

## S3 method for class 'POSIXct':
print(x, ...)

## S3 method for class 'POSIXct':
summary(object, digits = 15, ...)

time + number
time - number
time1 lop time2

Arguments

x, object An object to be printed or summarized from one of the date-time classes.
digits Number of significant digits for the computations: should be high enough to
represent the least important time unit exactly.
...
Further arguments to be passed from or to other methods.
time, time1, time2 date-time objects.
number a numeric object.
lop One of ==, !=, <, <=, > or >=.

Details

There are two basic classes of date/times. Class "POSIXct" represents the (signed) number of
seconds since the beginning of 1970 as a numeric vector. Class "POSIXlt" is a named list of
vectors representing

- **sec** 0–61: seconds
- **min** 0–59: minutes
- **hour** 0–23: hours
- **mday** 1–31: day of the month
- **mon** 0–11: months after the first of the year.
- **year** Years since 1900.
- **wday** 0–6 day of the week, starting on Sunday.
- **yday** 0–365: day of the year.
- **isdst** Daylight savings time flag. Positive if in force, zero if not, negative if unknown.

The classes correspond to the ANSI C constructs of “calendar time” (the time_t data type) and
“local time” (or broken-down time, the struct tm data type), from which they also inherit their
names.

"POSIXct" is more convenient for including in data frames, and "POSIXlt" is closer to human-
readable forms. A virtual class "POSIXt" inherits from both of the classes: it is used to allow
operations such as subtraction to mix the two classes.

Logical comparisons and limited arithmetic are available for both classes. One can add or subtract
a number of seconds or a difftime object from a date-time object, but not add two date-time
objects. Subtraction of two date-time objects is equivalent to using difftime. Be aware that
"POSIXlt" objects will be interpreted as being in the current timezone for these operations, unless
a timezone has been specified.
"POSIXlt" objects will often have an attribute "tzone", a character vector of length 3 giving the timezone name from the TZ environment variable and the names of the base timezone and the alternate (daylight-saving) timezone. Sometimes this may just be of length one, giving the timezone name.

"POSIXct" objects may also have an attribute "tzone", a character vector of length one. If set, it will determine how the object is converted to class "POSIXlt" and in particular how it is printed. This is usually desirable, but if you want to specify an object in a particular timezone but to be printed in the current timezone you may want to remove the "tzone" attribute (e.g. by \texttt{c(x)}).

Unfortunately, the conversion is complicated by the operation of time zones and leap seconds (22 days have been 86401 seconds long so far: the times of the extra seconds are in the object \texttt{.leap.seconds}). The details of this are entrusted to the OS services where possible. This will usually cover the period 1970–2037, and on Unix machines back to 1902 (when time zones were in their infancy). Outside those ranges we use our own C code. This uses the offset from GMT in use in the timezone in 2000, and uses the alternate (daylight-saving) timezone only if isdst is positive.

It seems that some systems use leap seconds but most do not. This is detected and corrected for at build time, so all "POSIXct" times used by \texttt{R} do not include leap seconds. (Conceivably this could be wrong if the system has changed since build time, just possibly by changing locales.)

Using \texttt{c} on "POSIXlt" objects converts them to the current time zone.

**Warning**

Some Unix-like systems (especially Linux ones) do not have "TZ" set, yet have internal code that expects it (as does POSIX). We have tried to work around this, but if you get unexpected results try setting "TZ".

**See Also**

\textit{Dates} for dates without times.

\texttt{as.POSIXct} and \texttt{as.POSIXlt} for conversion between the classes.

\texttt{strptime} for conversion to and from character representations.

\texttt{Sys.time} for clock time as a "POSIXct" object.

\texttt{difftime} for time intervals.

\texttt{cut.POSIXt}, \texttt{seq.POSIXt}, \texttt{round.POSIXt} and \texttt{trunc.POSIXt} for methods for these classes.

\texttt{weekdays.POSIXt} for convenience extraction functions.

**Examples**

\begin{verbatim}
(z <- Sys.time()) # the current date, as class "POSIXct"
Sys.time() - 3600 # an hour ago
as.POSIXlt(Sys.time(), "GMT") # the current time in GMT
format(.leap.seconds) # all 22 leapseconds in your timezone
print(.leap.seconds, tz="PST8PDT") # and in Seattle's
\end{verbatim}
Read and Write Data in DCF Format

Description
Reads or writes an R object from/to a file in Debian Control File format.

Usage

```r
read.dcf(file, fields=NULL)
write.dcf(x, file = "", append = FALSE,
           indent = 0.1 * getOption("width"),
           width = 0.9 * getOption("width"))
```

Arguments

- `file`: either a character string naming a file or a connection. "" indicates output to the console. For `read.dcf` this can name a gzip-compressed file.
- `fields`: Fields to read from the DCF file. Default is to read all fields.
- `x`: the object to be written, typically a data frame. If not, it is attempted to coerce `x` to a data frame.
- `append`: logical. If TRUE, the output is appended to the file. If FALSE, any existing file of the name is destroyed.
- `indent`: a positive integer specifying the indentation for continuation lines in output entries.
- `width`: a positive integer giving the target column for wrapping lines in the output.

Details
DCF is a simple format for storing databases in plain text files that can easily be directly read and written by humans. DCF is used in various places to store R system information, like descriptions and contents of packages.

The DCF rules as implemented in R are:

1. A database consists of one or more records, each with one or more named fields. Not every record must contain each field, a field may appear only once in a record.
2. Regular lines start with a non-whitespace character.
3. Regular lines are of form `tag:value`, i.e., have a name tag and a value for the field, separated by : (only the first : counts). The value can be empty (=whitespace only).
4. Lines starting with whitespace are continuation lines (to the preceding field) if at least one character in the line is non-whitespace.
5. Records are separated by one or more empty (=whitespace only) lines.

`read.dcf` returns a character matrix with one line per record and one column per field. Leading and trailing whitespace of field values is ignored. If a tag name is specified, but the corresponding value is empty, then an empty string of length 0 is returned. If the tag name of a fields is never used in a record, then `NA` is returned. If there are multiple records with the same tag name, the last one encountered is returned.
debug

See Also

write.table.

Examples

```r
## Create a reduced version of the 'CONTENTS' file in package 'splines'
x <- read.dcf(file = system.file("CONTENTS", package = "splines"),
             fields = c("Entry", "Description"))
write.dcf(x)
```

dialog

Description

Set or unset the debugging flag on a function.

Usage

```r
depug(fun)
undebug(fun)
```

Arguments

fun any interpreted R function.

Details

When a function flagged for debugging is entered, normal execution is suspended and the body of function is executed one statement at a time. A new browser context is initiated for each step (and the previous one destroyed). You take the next step by typing carriage return, n or next. You can see the values of variables by typing their names. Typing c or cont causes the debugger to continue to the end of the function (or loop if within a loop). You can debug new functions before you step in to them from inside the debugger. Typing Q quits the current execution and returns you to the top–level prompt. Typing where causes the debugger to print out the current stack trace (all functions that are active). If you have variables with names that are identical to the controls (eg. c or n) then you need to use print(c) and print(n) to evaluate them.

In order to debug S4 methods (see Methods), you need to use trace, typically calling browser, e.g., as

```r
trace("plot", browser, exit=browser, signature = c("track", "missing"))
```

See Also

browser, trace; traceback to see the stack after an Error: ... message; recover for another debugging approach.
Marking Objects as Defunct

Description
When an object is removed from R it should be replaced by a call to .Defunct.

Usage
```r
.Defunct(new, package = NULL)
```

Arguments
- `new` character string: A suggestion for a replacement function.
- `package` character string: The package to be used when suggesting where the defunct function might be listed.

Details
`.Defunct` is called from defunct functions. Functions should be listed in help("pkg-defunct") for an appropriate pkg, including base.

See Also
- Deprecated.
- base-defunct and so on which list the defunct functions in the packages.

Delay Evaluation

delayedAssign

Description
delayedAssign creates a promise to evaluate the given expression if its value is requested. This provides direct access to the lazy evaluation mechanism used by R for the evaluation of (interpreted) functions.

Usage
delayedAssign(x, value, eval.env = parent.frame(1),
              assign.env = parent.frame(1))

Arguments
- `x` a variable name (given as a quoted string in the function call)
- `value` an expression to be assigned to x
- `eval.env` an environment in which to evaluate value
- `assign.env` an environment in which to assign x
Details

Both `eval.env` and `assign.env` default to the currently active environment.
The expression assigned to a promise by `delayedAssign` will not be evaluated until it is eventually “forced”. This happens when the variable is first accessed.
When the promise is eventually forced, it is evaluated within the environment specified by `eval.env` (whose contents may have changed in the meantime). After that, the value is fixed and the expression will not be evaluated again.
This function is meant to replace the `delay()` function, to make it more difficult for R code to see “naked” promises.

Value

This function is invoked for its side effect, which is assigning a promise to evaluate `value` to the variable `x`.

See Also

`substitute`, to see the expression associated with a promise.

Examples

```r
msg <- "old"
delayedAssign("x", msg)
msg <- "new!"
x # new!
substitute(x) #- msg

delayedAssign("x", {
  for(i in 1:3)
    cat("yippee!\n")
  10
})
x^2 # yippee
x^2 # simple number

e <- (function(x, y = 1, z) environment())(1+2, "y", (cat(" HO! "); pi+2))
(le <- as.list(e)) # evaluates the promises
```

---

**Expression Deparsing**

**Description**

Turn unevaluated expressions into character strings.

**Usage**

```r
deparse(expr, width.cutoff = 60,
        backtick = mode(expr) %in% c("call", "expression", "("),
        control = "showAttributes")
```
Arguments

expr any \texttt{R} expression.
width.cutoff integer in \([20, 500]\) determining the cutoff at which line-breaking is tried.
backtick logical indicating whether symbolic names should be enclosed in backticks if they do not follow the standard syntax.
control character vector of deparsing options. See \texttt{deparseOpt}.

Details

This function turns unevaluated expressions (where “expression” is taken in a wider sense than the strict concept of a vector of mode "expression" used in \texttt{expression}) into character strings (a kind of inverse \texttt{parse}).

A typical use of this is to create informative labels for data sets and plots. The example shows a simple use of this facility. It uses the functions \texttt{deparse} and \texttt{substitute} to create labels for a plot which are character string versions of the actual arguments to the function \texttt{myplot}.

The default for the \texttt{backtick} option is not to quote single symbols but only composite expressions. This is a compromise to avoid breaking existing code.

Using \texttt{control = "all"} comes closest to making \texttt{deparse()} an inverse of \texttt{parse()}. However, not all objects are deparseable even with this option and a warning will be issued if the function recognizes that it is being asked to do the impossible.

Note

To avoid the risk of a source attribute out of sync with the actual function definition, the source attribute of a function will never be deparsed as an attribute.

References


See Also

\texttt{substitute, parse, expression}.

Quotes for quoting conventions, including backticks.

Examples

\begin{verbatim}
require(stats)
deparse(args(lm))
deparse(args(lm), width = 500)
myplot <- function(x, y) {
  plot(x, y, xlab=deparse(substitute(x)),
       ylab=deparse(substitute(y)))
}
e <- quote(`foo bar')
deparse(e)
deparse(e, backtick=TRUE)
e <- quote(`foo bar'+1)
deparse(e)
deparse(e, control = "all")
\end{verbatim}
Options for Expression Deparsing

**Description**

Process the deparsing options for `deparse`, `dput` and `dump`.

**Usage**

`deparseOpts(control)`

**Arguments**

- `control` character vector of deparsing options.

**Details**

This is called by `deparse`, `dput` and `dump` to process their `control` argument. The `control` argument is a vector containing zero or more of the following strings. Partial string matching is used.

- `keepInteger` Surround integer vectors by `as.integer()`, so they are not converted to floating point when re-parsed.
- `quoteExpressions` Surround expressions with `quote()`, so they are not evaluated when re-parsed.
- `showAttributes` If the object has attributes (other than a `source` attribute), use `structure()` to display them as well as the object value. This is the default for `deparse` and `dput`.
- `useSource` If the object has a `source` attribute, display that instead of deparsing the object. Currently only applies to function definitions.
- `warnIncomplete` Some exotic objects such as `environment`s, external pointers, etc. can not be deparsed properly. This option causes a warning to be issued if any of those may give problems.
- `all` An abbreviated way to specify all of the options listed above. May not be used with other options. This is the default for `dump`.
- `delayPromises` Deparse promises in the form `<promise: expression>` rather than evaluating them. The value and the environment of the promise will not be shown and the deparsed code cannot be sourced.

For the most readable (but perhaps incomplete) display, use `control = NULL`. This displays the object’s value, but not its attributes. The default is to display the attributes as well, but not to use any of the other options to make the result parseable.

Using `control = "all"` comes closest to making `deparse()` an inverse of `parse()`. However, not all objects are deparseable even with this option. A warning will be issued if the function recognizes that it is being asked to do the impossible.

**Value**

A numerical value corresponding to the options selected.
Deprecation

Marking Objects as Deprecated

Description

When an object is about removed from R it is first deprecated and should include a call to .Deprecated.

Usage

 Dereprecated(new, package=NULL)

Arguments

new character string: A suggestion for a replacement function.
package character string: The package to be used when suggesting where the deprecated function might be listed.

Details

.Degradated("<new name>") is called from deprecated functions. The original help page for these functions is often available at help("oldName-deprecated") (note the quotes). Functions should be listed in help("pkg-deprecated") for an appropriate pkg, including base.

See Also

Defunct
base-deprecated and so on which list the deprecated functions in the packages.

det

Calculate the Determinant of a Matrix

Description

det calculates the determinant of a matrix. determinant is a generic function that returns separately the modulus of the determinant, optionally on the logarithm scale, and the sign of the determinant.

Usage

det(x, ...)
determinant(x, logarithm = TRUE, ...)
Arguments

x numeric matrix.

logarithm logical; if TRUE (default) return the logarithm of the modulus of the determinant.

... Optional arguments. At present none are used. Previous versions of det allowed an optional method argument. This argument will be ignored but will not produce an error.

Value

For det, the determinant of x. For determinant, a list with components

modulus a numeric value. The modulus (absolute value) of the determinant if logarithm is FALSE; otherwise the logarithm of the modulus.

sign integer; either +1 or −1 according to whether the determinant is positive or negative.

Note

Often, computing the determinant is not what you should be doing to solve a given problem.

Prior to version 1.8.0 the det function had a method argument to allow use of either a QR decomposition or an eigenvalue-eigenvector decomposition. The determinant function now uses an LU decomposition and the det function is simply a wrapper around a call to determinant.

Examples

(x <- matrix(1:4, ncol=2))
unlist(determinant(x))
det(x)

det(print(cbind(1,1:3,c(2,0,1))))

...........................

detach Detach Objects from the Search Path

Description

Detach a database, i.e., remove it from the search path of available R objects. Usually, this is either a data.frame which has been attached or a package which was required previously.

Usage

detach(name, pos = 2, version)
Arguments

name  The object to detach. Defaults to search() [pos]. This can be an unquoted name or a character string but not a character vector. If a number is supplied this is taken as pos.

pos  Index position in search() of database to detach. When name is a number, pos = name is used.

version  A character string denoting a version number of the package to be removed. This should be used only with a versioned installation of the package: see library.

Details

This most commonly used with a single number argument referring to a position on the search list, and can also be used with a unquoted or quoted name of an item on the search list such as package:tools.

When a package have been loaded with an explicit version number it can be detached using the name shown by search or by supplying name and version: see the examples.

Value

The attached database is returned invisibly, either as data.frame or as list.

Note

You cannot detach either the workspace (position 1) or the base package (the last item in the search list).

References


See Also

attach, library, search, objects.

Examples

require(splines)#package
detach(package:splines)
  ## could equally well use detach("package:splines")
  ## but NOT pkg <- "package:splines"; detach(pkg)
  ## Instead, use
  library(splines)
  pkg <- "package:splines"
  detach(pos = match(pkg, search()))

  ## careful: do not do this unless 'splines' is not already loaded.
  library(splines)
detach(2)# 'pos' used for 'name'

  ## an example of the name argument to attach
  ## and of detaching a database named by a character vector
  attach_and_detach <- function(db, pos=2)
  {
name <- deparse(substitute(db))
attach(db, pos=pos, name=name)
print(search()[pos])
eval(substitute(detach(n), list(n=name)))
}
attach_and_detach(women, pos=3)

## Not run:
## Using a versioned install
library(ash, version="1.0-9")  # or perhaps just library(ash)
# then one of
detach("package:ash", version="1.0-9")
# or
detach("package:ash_1.0-9")
## End(Not run)

---

**diag**

**Matrix Diagonals**

**Description**

Extract or replace the diagonal of a matrix, or construct a diagonal matrix.

**Usage**

```r
diag(x = 1, nrow, ncol = )
diag(x) <- value
```

**Arguments**

- **x**
  - a matrix, vector or 1D array.
- **nrow, ncol**
  - Optional dimensions for the result.
- **value**
  - either a single value or a vector of length equal to that of the current diagonal. Should be of a mode which can be coerced to that of `x`.

**Value**

If `x` is a matrix then `diag(x)` returns the diagonal of `x`. The resulting vector will have `names` if the matrix `x` has matching column and row names.

If `x` is a vector (or 1D array) of length two or more, then `diag(x)` returns a diagonal matrix whose diagonal is `x`.

If `x` is a vector of length one then `diag(x)` returns an identity matrix of order the nearest integer to `x`. The dimension of the returned matrix can be specified by `nrow` and `ncol` (the default is square).

The assignment form sets the diagonal of the matrix `x` to the given `value(s)`.

**Note**

Using `diag(x)` can have unexpected effects if `x` is a vector that could be of length one. Use `diag(x, nrow = length(x))` for consistent behaviour.
References


See Also

`upper.tri`, `lower.tri`, `matrix`.

Examples

```r
require(stats)

# Example

dim(diag(3))
diag(10, 3, 4) # guess what?
all(diag(1:3) == (m <- matrix(0, 3, 3); diag(m) <- 1:3; m))

diag(var(M <- cbind(X=1:5, Y=rnorm(5)))) # vector with names "X" and "y"
rownames(M) <- c(colnames(M), rep("", 3));
M; diag(M) # named as well
```

---

**diff**

**Lagged Differences**

Description

Returns suitably lagged and iterated differences.

Usage

```r
diff(x, ...)
```

## Default S3 method:

diff(x, lag = 1, differences = 1, ...)

## S3 method for class 'POSIXt':

diff(x, lag = 1, differences = 1, ...)

## S3 method for class 'Date':

diff(x, lag = 1, differences = 1, ...)

Arguments

- `x`: a numeric vector or matrix containing the values to be differenced.
- `lag`: an integer indicating which lag to use.
- `differences`: an integer indicating the order of the difference.
- `...`: further arguments to be passed to or from methods.

Details

`diff` is a generic function with a default method and ones for classes "ts", "POSIXt" and "Date". NA's propagate.
Value

If \( x \) is a vector of length \( n \) and \( \text{differences}=1 \), then the computed result is equal to the successive differences \( x[1+(1+\text{lag}):n] - x[1:(n-\text{lag})] \).

If \( \text{difference} \) is larger than one this algorithm is applied recursively to \( x \). Note that the returned value is a vector which is shorter than \( x \).

If \( x \) is a matrix then the difference operations are carried out on each column separately.

References


See Also

diff.ts, diffinv.

Examples

diff(1:10, 2)
diff(1:10, 2, 2)
\( x \leftarrow \text{cumsum}(\text{cumsum}(1:10)) \)
diff(\( x \), lag = 2)
diff(\( x \), differences = 2)
diff(.leap.seconds)

difftime

Time Intervals

Description

Create, print and round time intervals.

Usage

time1 - time2
difftime(time1, time2, tz = "", units = c("auto", "secs", "mins", "hours", "days", "weeks"))
as.difftime(tim, format = "%X")

## S3 method for class 'difftime':
round(x, digits = 0)

Arguments

time1, time2  date-time objects.
tz  a timezone specification to be used for the conversion. System-specific, but "" is the current time zone, and "GMT" is UTC.
units  character. Units in which the results are desired. Can be abbreviated.
Function `difftime` takes a difference of two date/time objects (of either class) and returns an object of class "difftime" with an attribute indicating the units. There is a `round` method for objects of this class, as well as methods for the group-generic (see `Ops`) logical and arithmetic operations.

If `units = "auto"`, a suitable set of units is chosen, the largest possible (excluding "weeks") in which all the absolute differences are greater than one.

Subtraction of two date-time objects gives an object of this class, by calling `difftime` with `units="auto"`. Alternatively, `as.difftime()` works on character-coded time intervals.

Limited arithmetic is available on "difftime" objects: they can be added or subtracted, and multiplied or divided by a numeric vector. In addition, adding or subtracting a numeric vector implicitly converts the numeric vector to a "difftime" object with the same units as the "difftime" object.

See Also

`DateTimeClasses`.

Examples

```
(z <- Sys.time() - 3600)
Sys.time() - z # just over 3600 seconds.

## time interval between releases of 1.2.2 and 1.2.3.

as.difftime(c("0:3:20", "11:23:15"))

as.difftime(c("3:20", "23:15", "2:"), format= "%H:%M") # 3rd gives NA
```

---

**dim**

Dimensions of an Object

Description

Retrieve or set the dimension of an object.

Usage

```r
dim(x)
dim(x) <- value
```

Arguments

- `x` an R object, for example a matrix, array or data frame.
- `value` For the default method, either `NULL` or a numeric vector which coerced to integer (by truncation).
dimnames

Details

The functions dim and dim<- are generic.

dim has a method for data.frames, which returns the length of the row.names attribute of x and the length of x (the numbers of “rows” and “columns”).

Value

For an array (and hence in particular, for a matrix) dim retrieves the dim attribute of the object. It is NULL or a vector of mode integer.

References


See Also

ncol, nrow and dimnames.

Examples

x <- 1:12 ; dim(x) <- c(3,4)
x

# simple versions of nrow and ncol could be defined as follows
nrow0 <- function(x) dim(x)[1]
col0 <- function(x) dim(x)[2]

---

dimnames

Dimnames of an Object

Description

Retrieve or set the dimnames of an object.

Usage

dimnames(x)
dimnames(x) <- value

Arguments

x an R object, for example a matrix, array or data frame.
value a possible value for dimnames(x): see “Value”.

Details

The functions `dimnames` and `dimnames<-` are generic.

For an array (and hence in particular, for a matrix), they retrieve or set the `dimnames` attribute (see `attributes`) of the object. The list value can have names, and these will be used to label the dimensions of the array where appropriate.

Both have methods for data frames. The dimnames of a data frame are its `row.names` attribute and its `names`.

As from R 1.8.0 factor components of `value` will be coerced to character.

Value

The `dimnames` of a matrix or array can be NULL or a list of the same length as `dim(x)`. If a list, its components are either NULL or a character vector the length of the appropriate dimension of `x`.

References


See Also

`rownames`, `colnames`; `array`, `matrix`, `data.frame`.

Examples

```r
## simple versions of rownames and colnames
## could be defined as follows
rownames0 <- function(x) dimnames(x)[[1]]
colnames0 <- function(x) dimnames(x)[[2]]
```

---

### do.call

**Execute a Function Call**

**Description**

`do.call` constructs and executes a function call from the name of the function and a list of arguments to be passed to it.

If `quote` is `FALSE`, the default, then the arguments are evaluated. If `quote` is `TRUE` then each argument is quoted (see `quote`) so that the effect of argument evaluation is to remove the quote - leaving the original argument unevaluated.

The behavior of some functions, such as `substitute`, will not be the same for functions evaluated using `do.call` as if they were evaluated from the interpreter. The precise semantics are currently undefined and subject to change.

**Usage**

```r
do.call(what, args, quote=FALSE)
```
Arguments

what
either a function or a character string naming the function to be called.

args
a list of arguments to the function call. The names attribute of args gives the argument names.

quote
a logical value indicating whether to quote the arguments.

Value

The result of the (evaluated) function call.

References


See Also

call which creates an unevaluated call.

Examples

do.call("complex", list(imag = 1:3))

## if we already have a list (e.g. a data frame)
## we need c() to add further arguments
tmp <- expand.grid(letters[1:2], 1:3, c("+", "-"))
do.call("paste", c(tmp, sep=""))

do.call(paste, list(as.name("A"), as.name("B")), quote=TRUE)

---

double

Double Precision Vectors

Description

Create, coerce to or test for a double-precision vector.

Usage

double(length = 0)
as.double(x, ...)
is.double(x)

single(length = 0)
as.single(x, ...)

Arguments

length
desired length.
x
object to be coerced or tested.
...
further arguments passed to or from other methods.
Value

double creates a double precision vector of the specified length. The elements of the vector are all equal to 0.
as.double attempts to coerce its argument to be of double type: like as.vector it strips attributes including names. Character strings containing either a decimal representation or a hexadecimal representation (starting with 0x or 0X) can be converted.
is.double returns TRUE or FALSE depending on whether its argument is of double type or not. It is generic: you can write methods to handle specific classes of objects, see InternalMethods.

Note

R has no single precision data type. All real numbers are stored in double precision format. The functions as.single and single are identical to as.double and double except they set the attribute Csingle that is used in the .C and .Fortran interface, and they are intended only to be used in that context.

References


See Also

integer, numeric.

Examples

is.double(1)
all(double(3) == 0)

dput

Write an Internal Object to a File

Description

Writes an ASCII text representation of an R object to a file or connection, or uses one to recreate the object.

Usage

dput(x, file = "", control = "showAttributes")
dget(file)

Arguments

x an object.
file either a character string naming a file or a connection. "" indicates output to the console.
control character vector indicating deparsing options. See .deparseOpts for their description.
Details

dput opens file and deparses the object x into that file. The object name is not written (contrary to dump). If x is a function the associated environment is stripped. Hence scoping information can be lost.

Deparsing an object is difficult, and not always possible. With the default control = c("showAttributes"), dput() attempts to deparse in a way that is readable, but for more complex or unusual objects, not likely to be parsed as identical to the original. Use control = "all" for the most complete deparsing; use control = NULL for the simplest deparsing, not even including attributes.

dput will warn if fewer characters were written to a file than expected, which may indicate a full or corrupt file system.

To display saved source rather than deparsing the internal representation include "useSource" in control. R currently saves source only for function definitions.

Note

To avoid the risk of a source attribute out of sync with the actual function definition, the source attribute of a function will never be written as an attribute.

References


See Also

deparse, dump, write.

Examples

```r
## Write an ASCII version of mean to the file "foo"
dput(mean, "foo")
## And read it back into 'bar'
bar <- dget("foo")
unlink("foo")
## Create a function with comments
baz <- function(x) {
  # Subtract from one
  1-x
}
## and display it
dput(baz)
## and now display the saved source
dput(baz, control = "useSource")
```
Drop Redundant Extent Information

Description
Delete the dimensions of an array which have only one level.

Usage
\texttt{drop(x)}

Arguments
\texttt{x} \\ an array (including a matrix).

Value
If \texttt{x} is an object with a \texttt{dim} attribute (e.g., a matrix or \texttt{array}), then \texttt{drop} returns an object like \texttt{x}, but with any extents of length one removed. Any accompanying \texttt{dimnames} attribute is adjusted and returned with \texttt{x}.

Array subsetting (\texttt{[}) performs this reduction unless used with \texttt{drop = FALSE}, but sometimes it is useful to invoke \texttt{drop} directly.

See Also
\texttt{drop1} which is used for dropping terms in models.

Examples
\begin{verbatim}
dim(drop(array(1:12, dim=c(1,3,1,1,2,1,2)))) # = 3 2 2
drop(1:3 %*% 2:4) # scalar product
\end{verbatim}

Text Representations of R Objects

Description
This function takes a vector of names of R objects and produces text representations of the objects on a file or connection. A dump file can usually be \texttt{sourced} into another R (or S) session.

Usage
\begin{verbatim}
dump(list, file = "dumpdata.R", append = FALSE,
     control = "all", envir = parent.frame(), evaluate = TRUE)
\end{verbatim}
dump

Arguments

list character. The names of one or more R objects to be dumped.
file either a character string naming a file or a connection. "" indicates output to the console.
append if TRUE, output will be appended to file; otherwise, it will overwrite the contents of file.
control character vector indicating deparsing options. See .deparseOpts for their description.
envir the environment to search for objects.
evaluate logical. Should promises be evaluated?

Details

If some of the objects named do not exist (in scope), they are omitted, with a warning. If file is a file and no objects exist then no file is created.

At present sourcing may not produce an identical copy of dumped objects. A warning is issued if it is likely that problems will arise, for example when dumping exotic objects such as environments and external pointers.
dump will also warn if fewer characters were written to a file than expected, which may indicate a full or corrupt file system.
A dump file can be sourced into another R (or perhaps S) session, but the function save is designed to be used for transporting R data, and will work with R objects that dump does not handle.

To produce a more readable representation of an object, use control = NULL. This will skip attributes, and will make other simplifications that make source less likely to produce an identical copy. See deparse for details.

To deparse the internal function representation rather than displaying the saved source, use control = c("keepInteger", "quoteExpressions", "showAttributes", "warnIncomplete"). This will lose all formatting and comments, but may be useful in those cases where the saved source is no longer correct.

Promises will normally only be encountered by users as a result of lazy-loading (when the default evaluate = TRUE is essential) and after the use of delayedAssign, when evaluate = FALSE might be intended.

Value

An invisible character vector containing the names of the objects which were dumped.

Note

The envir argument was added at version 1.7.0, and changed the default search path for named objects to include the environment from which dump was called.

As dump is defined in the base namespace, the base package will be searched before the global environment unless dump is called from the top level or the envir argument is given explicitly.

To avoid the risk of a source attribute out of sync with the actual function definition, the source attribute of a function will never be dumped as an attribute.
References


See Also
dput, dget, write.
save for a more reliable way to save R objects.

Examples

```r
x <- 1; y <- 1:10
dump(ls(patt='^[xyz]\'), "xyz.Rdump")
print(.Last.value)
unlink("xyz.Rdump")
```

---

**duplicated**

*Determine Duplicate Elements*

**Description**

Determines which elements of a vector of data frame are duplicates of elements with smaller subscripts, and returns a logical vector indicating which elements (rows) are duplicates.

**Usage**

```r
duplicated(x, incomparables = FALSE, ...)
```

```r
## S3 method for class 'array':
duplicated(x, incomparables = FALSE, MARGIN = 1, ...)
```

**Arguments**

- `x` a vector or a data frame or an array or NULL.
- `incomparables` a vector of values that cannot be compared. Currently, FALSE is the only possible value, meaning that all values can be compared.
- `...` arguments for particular methods.
- `MARGIN` the array margin to be held fixed: see apply.

**Details**

This is a generic function with methods for vectors (including lists), data frames and arrays (including matrices).

The data frame method works by pasting together a character representation of the rows separated by \r, so may be imperfect if the data frame has characters with embedded carriage returns or columns which do not reliably map to characters.

The array method calculates for each element of the sub-array specified by MARGIN if the remaining dimensions are identical to those for an earlier element (in row-major order). This would most commonly be used to find duplicated rows (the default) or columns (with MARGIN = 2).
Warning

Using this for lists is potentially slow, especially if the elements are not atomic vectors (see \texttt{vector}) or differ only in their attributes. In the worst case it is $O(n^2)$.

References


See Also

\texttt{unique}.

Examples

\begin{verbatim}
x <- c(9:20, 1:5, 3:7, 0:8)
## extract unique elements
(xu <- x[!duplicated(x)])
## xu == unique(x) but unique(x) is more efficient
duplicated(iris)[140:143]
duplicated(iris3, MARGIN = c(1, 3))
\end{verbatim}

---

dyn.load \hspace{1cm} Foreign Function Interface

Description

Load or unload shared libraries, and test whether a C function or Fortran subroutine is available.

Usage

\begin{verbatim}
dyn.load(x, local = TRUE, now = TRUE)
dyn.unload(x)

is.loaded(symbol, PACKAGE = "")
symbol.C(name)
symbol.For(name)
\end{verbatim}

Arguments

\begin{verbatim}
x \hspace{1cm} a character string giving the pathname to a shared library or DLL.
local \hspace{1cm} a logical value controlling whether the symbols in the shared library are stored in their own local table and not shared across shared libraries, or added to the global symbol table. Whether this has any effect is system-dependent. It is ignored on Windows.
now \hspace{1cm} a logical controlling whether all symbols are resolved (and relocated) immediately the library is loaded or deferred until they are used. This control is useful for developers testing whether a library is complete and has all the necessary symbols, and for users to ignore missing symbols. Whether this has any effect is system-dependent. It is ignored on Windows.
\end{verbatim}
symbol  a character string giving a symbol name.

PACKAGE if supplied, confine the search for the name to the DLL given by this argument (plus the conventional extension, .so, .sl, .dll,...). This is intended to add safety for packages, which can ensure by using this argument that no other package can override their external symbols. Use PACKAGE="base" for symbols linked in to R. This is used in the same way as in .C, .Call, .Fortran and .External functions.

name a character string giving either the name of a C function or Fortran subroutine. Fortran names probably need to be given entirely in lower case (but this may be system-dependent).

Details

See ‘See Also’ and the Writing R Extensions and R Installation and Administration manuals for how to create and install a suitable shared library. Note that unlike some versions of S-PLUS, dyn.load does not load an object (.o) file but a shared library or DLL.

Unfortunately a very few platforms (Compaq Tru64) do not handle the PACKAGE argument correctly, and may incorrectly find symbols linked into R.

The additional arguments to dyn.load mirror the different aspects of the mode argument to the dlopen() routine on UNIX systems. They are available so that users can exercise greater control over the loading process for an individual library. In general, the defaults values are appropriate and you should override them only if there is good reason and you understand the implications.

External code must not change the floating point control word, but many DLLs do so. Common changes are to set it to use 53 bit precision instead of R’s default 64 bit precision, or to unmask some exceptions. dyn.load detects such changes, and restores R’s control word to its default value of hex 8001F. This may cause the DLL to malfunction; if so, it should be rewritten to save and restore the control word itself. If warn.FPU is set to TRUE using the options function, a warning will be printed. (If the warning says that the control word was changed from some other value than 8001F, please report the circumstances to the Windows maintainers: that probably indicates an internal bug.)

Value

The function dyn.load is used for its side effect which links the specified shared library to the executing R image. Calls to .C, .Call, .Fortran and .External can then be used to execute compiled C functions or Fortran subroutines contained in the library. The return value of dyn.load is an object of class DLLInfo. See getLoadedDLLs for information about this class.

The function dyn.unload unlinks the shared library.

Functions symbol.C and symbol.For map function or subroutine names to the symbol name in the compiled code. These are no longer of much use in R.

is.loaded checks if the symbol name is loaded and hence available for use in .C or .Fortran: nowadays it needs the name you would give to .C or .Fortran and not that remapped by symbol.C and symbol.For.

Note

The creation of shared libraries and the runtime linking of them into executing programs is very platform dependent. In recent years there has been some simplification in the process because the C subroutine call dlopen has become the standard for doing this under UNIX. Under UNIX
dyn.load uses the dlopen mechanism and should work on all platforms which support it. On Windows it uses the standard mechanisms for loading 32-bit DLLs.

The original code for loading DLLs in UNIX was provided by Heiner Schwarte. The compatibility code for HP-UX was provided by Luke Tierney.

References


See Also

`library.dynam` to be used inside a package’s `.First.lib` initialization.

`SHLIB` for how to create suitable DLLs.


Examples

```r
is.loaded("hcass2") #-> probably TRUE, as stats is loaded
```

---

**eapply**

*Apply a Function over values in an environment*

**Description**

`eapply` applies `FUN` to the named values from an environment and returns the results as a list. The user can request that all named objects are used (normally names that begin with a dot are not). The output is not sorted and no parent environments are searched.

**Usage**

```r
eapply(env, FUN, ..., all.names = FALSE)
```

**Arguments**

- `env` environment to be used.
- `FUN` the function to be applied. In the case of functions like `+`, `%*%`, etc., the function name must be quoted.
- `...` optional arguments to `FUN`.
- `all.names` a logical indicating whether to apply the function to all values

**See Also**

`lapply`. 
Examples

```r
env <- new.env()
env$a <- 1:10
env$beta <- exp(-3:3)
env$logic <- c(TRUE, FALSE, FALSE, TRUE)
# compute the list mean for each list element
eapply(env, mean)
# median and quartiles for each list element
eapply(env, quantile, probs = 1:3/4)
eapply(env, quantile)
```

---

**eigen**

*Spectral Decomposition of a Matrix*

Description

Computes eigenvalues and eigenvectors.

Usage

```r
eigen(x, symmetric, only.values = FALSE, EISPACK = FALSE)
```

Arguments

- `x`  
  A matrix whose spectral decomposition is to be computed.
- `symmetric`  
  If `TRUE`, the matrix is assumed to be symmetric (or Hermitian if complex) and only its lower triangle is used. If `symmetric` is not specified, the matrix is inspected for symmetry.
- `only.values`  
  If `TRUE`, only the eigenvalues are computed and returned, otherwise both eigenvalues and eigenvectors are returned.
- `EISPACK`  
  Logical. Should EISPACK be used (for compatibility with R < 1.7.0)?

Details

By default `eigen` uses the LAPACK routines DSYEVR, DGEEV, ZHEEV and ZGEEV whereas `eigen(EISPACK=TRUE)` provides an interface to the EISPACK routines RS, RG, CH and CG.

If `symmetric` is unspecified, the code attempts to determine if the matrix is symmetric up to plausible numerical inaccuracies. It is faster and surer to set the value yourself.

`eigen` is preferred to `eigen(EISPACK = TRUE)` for new projects, but its eigenvectors may differ in sign and (in the asymmetric case) in normalization. (They may also differ between methods and between platforms.)

Computing the eigenvectors is the slow part for large matrices.

Value

The spectral decomposition of `x` is returned as components of a list with components

- `values`  
  A vector containing the `p` eigenvalues of `x`, sorted in decreasing order, according to `Mod(values)` in the asymmetric case when they might be complex (even for real matrices). For real asymmetric matrices the vector will be complex only if complex conjugate pairs of eigenvalues are detected.
either a \( p \times p \) matrix whose columns contain the eigenvectors of \( x \), or NULL if only.values is TRUE.

For `eigen()`, symmetric = FALSE, EISPACK = TRUE) the choice of length of the eigenvectors is not defined by EISPACK. In all other cases the vectors are normalized to unit length.

Recall that the eigenvectors are only defined up to a constant: even when the length is specified they are still only defined up to a scalar of modulus one (the sign for real matrices).

References


See Also

`svd`, a generalization of `eigen`; `qr`, and `chol` for related decompositions.

To compute the determinant of a matrix, the `qr` decomposition is much more efficient: `det`. `capabilities` to test for IEEE 754 arithmetic.

Examples

```r
eigen(cbind(c(1,-1),c(-1,1)))
eigen(cbind(c(1,-1),c(-1,1)), symmetric = FALSE)# same (different algorithm).

eigen(cbind(1,c(1,-1)), only.values = TRUE)
eigen(cbind(-1,2:1)) # complex values
eigen(print(cbind(c(0,1i), c(-1i,0))))# Hermite ==> real Eigen values
## 3 x 3:
eigen(cbind( 1,3:1,1:3))
eigen(cbind(-1,c(1:2,0),0:2)) # complex values
```

---

**encodeString**  
**Encode Character Vector as for Printing**

**Description**

`encodeString` escapes the strings in a character vector in the same way `print.default` does, and optionally fits the encoded strings within a field width.

**Usage**

```r
encodeString(x, width = 0, quote = "", na.encode = TRUE, justify = c("left", "right", "centre", "none"))
```
encodeString

Arguments

- `x`: A character vector, or an object that can be coerced to one by `as.character`.
- `width`: integer: the minimum field width. If NULL or NA, this is taken to be the largest field width needed for any element of `x`.
- `quote`: character: quoting character, if any.
- `na.encode`: logical: should NA strings be encoded?
- `justify`: character: partial matches are allowed. If padding to the minimum field width is needed, how should spaces be inserted? `justify == "none"` is equivalent to `width = 0`, for consistency with `format.default`.

Details

This escapes backslash and the control characters
- `a` (bell),
- `b` (backspace),
- `f` (formfeed),
- `n` (line feed),
- `r` (carriage return),
- `t` (tab),
- `v` (vertical tab) and
- `0` (nul) as well as any non-printable characters in a single-byte locale, which are printed in octal notation (`xyz` with leading zeroes). (Windows' reporting of printable characters is unreliable, so all other control characters are regarded as non-printable, and all characters with codes 32–255 as printable.)

See `print.default` for how non-printable characters are handled in multi-byte locales.

If `quote` is a single or double quote any embedded quote of the same type is escaped. Note that justification is of the quoted string, hence spaces are added outside the quotes.

Value

A character vector of the same length as `x`, with the same attributes (including names and dimensions) but with no class set.

Note

The default for `width` is different from `format.default`, which as of R 2.2.0 does similar things for character vectors (but without encoding using escapes).

See Also

`print.default`

Examples

```r
x <- "ab\bc\ndef"
print(x)
cat(x) # interprets escapes
cat(encodeString(x), "\n", sep="") # similar to print()

factor(x) # makes use of this to print the levels
x <- c("a", "ab", "abcde")
```
```r
encodeString(x, w = NA) # left justification
encodeString(x, w = NA, justify = "c")
encodeString(x, w = NA, justify = "r")
encodeString(x, w = NA, quote = "", justify = "r")
```

### environment

**Description**

Get, set, test for and create environments.

**Usage**

```r
evironment(fun = NULL)
evironment(fun) <- value

is.environment(obj)

.env <- .GlobalEnv
globalenv()
.env <- .BaseNamespaceEnv
baseenv()

new.env(hash = FALSE, parent = parent.frame())

parent.env(env)
parent.env(env) <- value
```

**Arguments**

- `fun` a function, a formula, or NULL, which is the default.
- `value` an environment to associate with the function
- `obj` an arbitrary R object.
- `hash` a logical, if `TRUE` the environment will be hashed
- `parent` an environment to be used as the enclosure of the environment created.
- `env` an environment

**Details**

Environments consist of a frame, or collection of named objects, and a pointer to an enclosing environment. The most common example is the frame of variables local to a function call; its enclosure is the environment where the function was defined. The enclosing environment is distinguished from the parent frame: the latter (returned by `parent.frame()`) refers to the environment of the caller of a function.

When get or exists search an environment with the default `inherits = TRUE`, they look for the variable in the frame, then in the enclosing frame, and so on.

The global environment `.GlobalEnv`, more often known as the user’s workspace, is the first item on the search path. It can also be accessed by `globalenv()`. On the search path, each item’s enclosure is the next item.
The object .BaseNamespaceEnv is the namespace environment for the base package. The environment of the base package itself is available as baseenv(), currently NULL, the ultimate enclosure of any environment: If one follows the parent.env() chain of enclosures back far enough from any environment, eventually one reaches baseenv(). This means that arithmetic operators and the base package functions will be always be found by eval() or get(..., inherits = TRUE).

The replacement function parent.env<- is extremely dangerous as it can be used to destructively change environments in ways that violate assumptions made by the internal C code. It may be removed in the near future.

is.environment is generic: you can write methods to handle specific classes of objects, see InternalMethods.

Value

If fun is a function or a formula then environment(fun) returns the environment associated with that function or formula. If fun is NULL then the current evaluation environment is returned.

The assignment form sets the environment of the function or formula fun to the value given.

is.environment(obj) returns TRUE iff obj is an environment.

new.env returns a new (empty) environment enclosed in the parent’s environment, by default.

parent.env returns the parent environment of its argument.

parent.env<- sets the enclosing environment of its first argument.

See Also

The envir argument of eval, get, and exists.

ls may be used to view the objects in an environment.

Examples

```r
## all three give the same:
environment()
environment(environment)
.GlobalEnv

ls(envir=environment(approxfun(1:2,1:2, method="const")))

is.environment(.GlobalEnv) # TRUE

el <- new.env(parent = baseenv()) # this one has enclosure package:base.
e2 <- new.env(parent = el)
assign("a", 3, env=el)
ls(el)
ls(e2)
e1 <- new.env(parent = baseenv()) # this succeeds by inheritance
exists("a", env=e1)
e2 exists("a", env=e2, inherits = FALSE)
e2 exists("+", env=e2) # this succeeds by inheritance
```
Description

Evaluate an R expression in a specified environment.

Usage

```r
eval(expr, envir = parent.frame(),
     enclos = if(is.list(envir) || is.pairlist(envir))
                parent.frame() else baseenv())
evalq(expr, envir, enclos)
eval.parent(expr, n = 1)
local(expr, envir = new.env())
```

Arguments

- `expr`: object of mode `expression` or call or an “unevaluated expression”.
- `envir`: the `environment` in which `expr` is to be evaluated. May also be, NULL, a list, a data frame, or an integer as in `sys.call`.
- `enclos`: Relevant when `envir` is a list or a data frame. Specifies the enclosure, i.e., where R looks for objects not found in `envir`.
- `n`: parent generations to go back

Details

eval evaluates the expression `expr` argument in the environment specified by `envir` and returns the computed value. If `envir` is not specified, then `sys.frame(sys.frame())`, the environment where the call to eval was made is used.

The `evalq` form is equivalent to `eval(quote(expr), ...)`.

As eval evaluates its first argument before passing it to the evaluator, it allows you to assign complicated expressions to symbols and then evaluate them. evalq avoids this.

eval.parent(expr, n) is a shorthand for `eval(expr, parent.frame(n))`.

If `envir` is a data frame or list, it is copied into a temporary environment, and the copy is used for evaluation. So if `expr` changes any of the components named in the data frame/list, the changes are lost.

If `envir` is NULL it is treated as an empty list or data frame: no values will be found in `envir`, so look-up goes directly to `enclos`.

A value of NULL for `enclos` is interpreted as the environment of the base package.

local evaluates an expression in a local environment. It is equivalent to `evalq` except that its default argument creates a new, empty environment. This is useful to create anonymous recursive functions and as a kind of limited namespace feature since variables defined in the environment are not visible from the outside.
Note

Due to the difference in scoping rules, there are some differences between R and S in this area. In particular, the default enclosure in S is the global environment.

When evaluating expressions in data frames that has been passed as argument to a function, the relevant enclosure is often the caller’s environment, i.e., one needs `eval(x, data, parent.frame())`.

References


See Also

`expression`, `quote`, `sys.frame`, `parent.frame`, `environment`.

Further, `force` to force evaluation, typically of function arguments.

Examples

eval(2 ^ 2 ^ 3)
mEx <- expression(2^2^3); mEx; 1 + eval(mEx)
eval(xx <- pi; xx^2))
a <- 3 ; aa <- 4 ; evalq(evalq(a+b+aa, list(a=1)), list(b=5)) # == 10
a <- 3 ; aa <- 4 ; evalq(evalq(a+b+aa, -1), list(b=5)) # == 12
ev <- function() {
e1 <- parent.frame()
## Evaluate a in e1
aa <- eval(expression(a),e1)
## evaluate the expression bound to a in e1
a <- expression(x+y)
list(aa = aa, eval = eval(a, e1))
}
tst.ev <- function(a = 7) { x <- pi; y <- 1; ev() }
tst.ev()#-> aa : 7, eval : 4.14

##
## Uses of local()
##
##
# Mutual recursives.
# gg gets value of last assignment, an anonymous version of f.
gg <- local({
  k <- function(y)f(y)
  f <- function(x) if(x) x*k(x-1) else 1
})
gg(10)
sapply(1:5, gg)

# Nesting locals. a is private storage accessible to k
gg <- local({
k <- local({
a <- 1
})
})
exists

function(y) {print(a <<- a+1);f(y)}
})
)
f <- function(x) if (x) x * k(x-1) else 1
})
sapply(1:5, gg)
ls(envir = environment(gg))
ls(envir = environment(get("k", envir = environment(gg)))))

exists Is an Object Defined?

Description

Look for an R object of the given name.

Usage

exists(x, where = -1, envir = , frame, mode = "any", inherits = TRUE)

Arguments

x a variable name (given as a character string).
where where to look for the object (see the details section); if omitted, the function will search as if the name of the object appeared unquoted in an expression.
envir an alternative way to specify an environment to look in, but it’s usually simpler to just use the where argument.
frame a frame in the calling list. Equivalent to giving where as sys.frame(frame).
mode the mode of object sought. For the meaning, see the Details section of the help for exists.
inherits should the enclosing frames of the environment be searched?

Details

The where argument can specify the environment in which to look for the object in any of several ways: as an integer (the position in the search list); as the character string name of an element in the search list; or as an environment (including using sys.frame to access the currently active function calls). The envir argument is an alternative way to specify an environment, but is primarily there for back compatibility.

This function looks to see if the name x has a value bound to it in the specified environment. If inherits is TRUE and a value is not found for x in the specified environment, the enclosures of the environment are searched until the name x is encountered. See environment and the ‘R Language Definition’ manual for details about the structure of environments and their enclosures.

Warning: inherits=TRUE is the default behaviour for R but not for S.

If mode is specified then only objects of that mode are sought. The mode may specify collections such as "numeric" and "function": any member of the collection will suffice.
Value

Logical, true if and only if an object of the correct name and mode is found.

References


See Also

*get*.

Examples

```r
## Define a substitute function if necessary:
if(!exists("some.fun", mode="function"))
  some.fun <- function(x) { cat("some.fun(x)\n"); x }
search()
exists("ls", 2) # true even though ls is in pos=3
exists("ls", 2, inherits = FALSE) # false
```

---

**expand.grid**

Create a Data Frame from All Combinations of Factors

Description

Create a data frame from all combinations of the supplied vectors or factors. See the description of the return value for precise details of the way this is done.

Usage

```r
expand.grid(...) 
```

Arguments

*...* Vectors, factors or a list containing these.

Value

A data frame containing one row for each combination of the supplied factors. The first factors vary fastest. The columns are labelled by the factors if these are supplied as named arguments or named components of a list.

Attribute "out.attrs" is a list which gives the dimension and dimnames for use by *predict* methods.

References


Examples

```r
expand.grid(height = seq(60, 80, 5), weight = seq(100, 300, 50),
            sex = c("Male","Female"))
```
**expression**  

*Unevaluated Expressions*

**Description**

Creates or tests for objects of mode "expression".

**Usage**

expression(...)

is.expression(x)

as.expression(x, ...)

**Arguments**

... valid R expressions.

x an arbitrary R object.

**Value**

expression returns a vector of mode "expression" containing its arguments as unevaluated "calls".

is.expression returns TRUE if expr is an expression object and FALSE otherwise.

as.expression attempts to coerce its argument into an expression object.

**References**


**See Also**

call, eval, function. Further, text and legend for plotting math expressions.

**Examples**

```r
length(ex1 <- expression(1+ 0:9)) # 1
ex1
eval(ex1) # 1:10

length(ex3 <- expression(u,v, 1+ 0:9)) # 3
mode(ex3 [3]) # expression
mode(ex3[[3]]) # call
rm(ex3)
```
Extract

Extract or Replace Parts of an Object

Description

Operators acting on vectors, matrices, arrays and lists to extract or replace subsets.

Usage

\[
x[i]
\]
\[
x[i, \ j, \ ...\ ,\ \text{drop} = \TRUE]
\]
\[
x[[i]]
\]
\[
x[[i, \ j, \ ...]]
\]
\[
x$name$
\]

Arguments

\textbf{x} \quad \text{object from which to extract elements or in which to replace elements.}

\textbf{i, j, \ ..., name} \quad \text{indices specifying elements to extract or replace. i, j are numeric or character or empty whereas name must be character or an (unquoted) name. Numeric values are coerced to integer as by \texttt{as.integer}. For \texttt{[[} and \$ character strings are normally partially matched to the names of the object if exact matching does not succeed. For [-indexing only: i, j, \ ... can be logical vectors, indicating elements/slices to select. Such vectors are recycled if necessary to match the corresponding extent. When indexing arrays, i can be a (single) matrix with as many columns as there are dimensions of \texttt{x}; the result is then a vector with elements corresponding to the sets of indices in each row of i. i, j, \ ... can also be negative integers, indicating elements/slices to leave out of the selection.}

\textbf{drop} \quad \text{For matrices and arrays. If TRUE the result is coerced to the lowest possible dimension (see examples below). This only works for extracting elements, not for the replacement forms.}

Details

These operators are generic. You can write methods to handle subsetting of specific classes of objects, see \texttt{InternalMethods} as well as \texttt{[.data.frame} and \texttt{[.factor}. The descriptions here apply only to the default methods.

The most important distinction between [, [[ and \$ is that the [ can select more than one element whereas the other two select a single element.

The default methods work somewhat differently for atomic vectors, matrices/arrays and for recursive (list-like, see \texttt{is.recursive}) objects. \$ returns \texttt{NULL} except for recursive objects, and is only discussed there.

Indexing can occur on the right-hand-side of an expression for extraction, or on the left-hand-side for replacement. When an index expression appears on the left side of an assignment then that part of \texttt{x} is set to the value of the right hand side of the assignment.
Atomic vectors

The usual form of indexing is "[ ". 

"[[ can be used to select a single element, but "[ 

" can also do so (but will not partially match a character index).

The index object i can be numeric, logical, character or empty. Indexing by factors is allowed and is equivalent to indexing by the numeric codes (see factor) and not by the character values which are printed (for which use as.character(i)).

An empty index selects all values: this is most often used to replace all the entries but keep the attributes.

Matrices and arrays

Matrices and arrays are vectors with a dimension attribute and so all the vector forms of indexing can be used with a single index. The result will be an unnamed vector unless x is one-dimensional when it will be a one-dimensional array.

The most common form of indexing a k-dimensional array is to specify k indices to [ . As for vector indexing, the indices can be numeric, logical, character, empty or even factor. An empty index (a comma separated blank) indicates that all entries in that dimension are selected. The argument drop applies to this form of indexing.

A third form of indexing is via a numeric matrix with the one column for each dimension: each row of the index matrix then selects a single element of the array, and the result is a vector. Negative indices are not allowed in the index matrix. NA and zero values are allowed: rows of an index matrix containing a zero are ignored, whereas rows containing an NA produce an NA in the result.

A vector obtained by matrix indexing will be unnamed unless x is one-dimensional when the row names (if any) will be indexed to provide names for the result.

List-like objects

Indexing by [ is similar to atomic vectors and selects a list of the specified element(s).

Both [ [ and $ select a single element of the list. The main difference is that $ does not allow computed indices, whereas [ [ does. x$name is equivalent to x[["name"]].

[ and [ are sometimes applied to other recursive objects such as calls and expressions. Pairlists are coerced to lists for extraction by [ , but all three operators can be used for replacement.

As from R 1.7.0 [ [ can be applied recursively to lists, so that if the single index i is a vector of length p, alist[[i]] is equivalent to alist[[i1]]...[[ip]] providing all but the final indexing results in a list.

When $<- is applied to a NULL x, it first coerces x to list(). This is what also happens with [<- if the replacement value value is of length greater than one: if value has length 1 or 0, x is first coerced to a zero-length vector of the type of value.

Environments

As from R 1.9.0 both $ and [ [ can be applied to environments. Only character arguments are allowed and no partial matching is done (this is in contrast to the behavior for lists). The semantics of these operations is basically that of get(i, env=x, inherits=FALSE). If no match is found then NULL is returned. The assignment versions, $<- and [<- can also be used. Again, only character arguments are allowed and no partial matching is done. The semantics in this case are those of assign(i, value, env=x, inherits=FALSE). Such an assignment will either create a new binding or change the existing binding in x.
NAs in indexing

When extracting, a numerical, logical or character NA picks an unknown element and so returns NA in the corresponding element of a logical, integer, numeric, complex or character result, and NULL for a list.

When replacing (that is using subscripting on the lhs of an assignment) NA does not select any element to be replaced. As there is ambiguity as to whether an element of the rhs should be used or not (and R handled this inconsistently prior to R 2.0.0), this is only allowed if the rhs value is of length one (so the two interpretations would have the same outcome).

Argument matching

Note that these operations do not match their index arguments in the standard way: argument names are ignored and positional matching only is used. So m[j=2, i=1] is equivalent to m[2, 1] and not to m[1, 2].

This may not be true for methods defined for them; for example it is not true for the data.frame methods described in [.data.frame.

To avoid confusion, do not name index arguments (but drop must be named).

References


See Also

list, array, matrix.

[.data.frame and [.factor for the behaviour when applied to data.frame and factors.


Examples

```r
x <- 1:12; m <- matrix(1:6, nr=2); li <- list(pi=pi, e = exp(1))
x[10] # the tenth element of x
x <- x[-1] # delete the 1st element of x
m[1,] # the first row of matrix m
m[1, , drop = FALSE] # is a 1-row matrix
m[,c(TRUE,FALSE,TRUE)] # logical indexing
m[cbind(c(1,2,1),3:1)] # matrix index
m <- m[-1] # delete the first column of m
li[[1]] # the first element of list li
y <- list(1,2,a=4,5)
y[c(3,4)] # a list containing elements 3 and 4 of y
y$a # the element of y named a

## non-integer indices are truncated:
i <- 3.999999999 # "4" is printed
(1:5)[i] # 3

## recursive indexing into lists
z <- list( a=list( b=9, c='hello' ), d=1:5)
unlist(z)
z[[c(1, 2)]]
z[[c(1, 2, 1)]] # both "hello"
```
z[[c("a", "b")]] <- "new"

unlist(z)

## check $ and [[] for environments
e1 <- new.env()
e1$a <- 10
e1[["a"]]
e1[["b"]]] <- 20

e1$b

ls(e1)

---

**Extract.data.frame**  *Extract or Replace Parts of a Data Frame*

**Description**

Extract or replace subsets of data frames.

**Usage**

```r
x[i]
x[i] <- value
x[i, j, drop = TRUE]
x[i, j] <- value

x[[i]]
x[[i]] <- value
x[[i, j]]
x[[i, j]] <- value

x$name
x$name <- value
```

**Arguments**

- `x` data frame.
- `i, j` elements to extract or replace. `i, j` are numeric or character or, for `[` only, empty. Numeric values are coerced to integer as if by `as.integer`. For replacement by `[]`, a logical matrix is allowed.
- `drop` logical. If `TRUE` the result is coerced to the lowest possible dimension: however, see the Warning below.
- `value` A suitable replacement value: it will be repeated a whole number of times if necessary and it may be coerced: see the Coercion section. If `NULL`, deletes the column if a single column is selected.
- `name` name or literal character string.
Data frames can be indexed in several modes. When \([\) and \([\) are used with a single index, they index the data frame as if it were a list. In this usage a drop argument is ignored, with a warning. Using $ is equivalent to using \([\) with a single index.

When \([\) and \([\) are used with two indices they act like indexing a matrix: \([\) can only be used to select one element.

If \([\) returns a data frame it will have unique (and non-missing) row names, if necessary transforming the row names using make.unique. Similarly, column names will be transformed (if columns are selected more than once).

When drop =TRUE, this is applied to the subsetting of any matrices contained in the data frame as well as to the data frame itself.

The replacement methods can be used to add whole column(s) by specifying non-existent column(s), in which case the column(s) are added at the right-hand edge of the data frame and numerical indices must be contiguous to existing indices. On the other hand, rows can be added at any row after the current last row, and the columns will be in-filled with missing values. Missing values in the indices are not allowed for replacement.

For \([\) the replacement value can be a list: each element of the list is used to replace (part of) one column, recycling the list as necessary. If columns specified by number are created, the names (if any) of the corresponding list elements are used to name the columns. If the replacement is not selecting rows, list values can contain NULL elements which will cause the corresponding columns to be deleted. (See the Examples.)

Matrixing indexing using \([\) is not recommended, and barely supported. For extraction, \(x\) is first coerced to a matrix. For replacement a logical matrix (only) can be used to select the elements to be replaced in the same way as for a matrix.

For \([\) a data frame, list or a single column (the latter two only when dimensions have been dropped). If matrix indexing is used for extraction a matrix results.

For \([\) a column of the data frame (extraction with one index) or a length-one vector (extraction with two indices).

For \([<-,\ [<-,\ $<-,\ [<-,\ [[<-,\ \$<-,\ a\ data\ frame.

The story over when replacement values are coerced is a complicated one, and one that has changed during R’s development. This section is a guide only.

When \([\) and \([\) are used to add or replace a whole column, no coercion takes place but value will be replicated (by calling the generic function rep) to the right length if an exact number of repeats can be used.

When \([\) is used with a logical matrix, each value is coerced to the type of the column in which it is to be placed.

When \([\) and \([\) are used with two indices, the column will be coerced as necessary to accommodate the value.

Note that when the replacement value is an array (including a matrix) it is not treated as a series of columns (as data.frame and as.data.frame do) but inserted as a single column.
Warning

Although the default for drop is TRUE, the default behaviour when only one row is left is equivalent to specifying drop = FALSE. To drop from a data frame to a list, drop = TRUE has to be specified explicitly.

See Also

subset which is often easier for extraction, data.frame, Extract.

Examples

```r
sw <- swiss[1:5, 1:4] # select a manageable subset

sw[1:3]       # select columns
sw[1, 1:3]    # same
sw[4:5, 1:3]  # select rows and columns
sw[1]         # a one-column data frame
sw[, 1, drop = FALSE] # the same
sw[, 1]       # a (unnamed) vector
sw[[1]]       # the same

sw[1,]        # a one-row data frame
sw[1,, drop=TRUE] # a list

swiss[ c(1, 1:2), ] # duplicate row, unique row names are created

sw[sw <= 6] <- 6 # logical matrix indexing

sw

## adding a column
sw["new1"] <- LETTERS[1:5] # adds a character column
sw[['new2']] <- letters[1:5] # ditto
sw[, "new3"] <- LETTERS[1:5] # ditto
sw$new4 <- 1:5
sapply(sw, class)
sw$new4 <- NULL # delete the column

sw

sw[6:8] <- list(letters[10:14], NULL, aa=1:5) # delete col7, update 6, append

sw

## matrices in a data frame
A <- data.frame(x=1:3, y=I(matrix(4:6)), z=I(matrix(letters[1:9],3,3)))
A[1:3, "y"] # a matrix, was a vector prior to 1.8.0
A[1:3, "z"] # a matrix
A[, "y"]    # a matrix
```

Description

Extract or replace subsets of factors.
Usage

\[ x[i, \text{drop} = \text{FALSE}] \]
\[ x[i] \leftarrow \text{value} \]

Arguments

- \( x \): a factor
- \( i \): a specification of indices – see Extract.
- \( \text{drop} \): logical. If true, unused levels are dropped.
- \( \text{value} \): character: a set of levels. Factor values are coerced to character.

Details

When unused levels are dropped the ordering of the remaining levels is preserved.

If \( \text{value} \) is not in \( \text{levels}(x) \), a missing value is assigned with a warning.

Any contrasts assigned to the factor are preserved unless \( \text{drop}=\text{TRUE} \).

Value

A factor with the same set of levels as \( x \) unless \( \text{drop}=\text{TRUE} \).

See Also

factor, Extract.

Examples

```r
## following example(factor)
(ff <- factor(substring("statistics", 1:10, 1:10), levels=letters))
ff[, drop=TRUE]
factor(letters[7:10])[2:3, drop = TRUE]
```

Description

Returns the (parallel) maxima and minima of the input values.

Usage

- \( \text{max}(...) \), \( \text{na.rm=FALSE} \)
- \( \text{min}(...) \), \( \text{na.rm=FALSE} \)
- \( \text{pmax}(...) \), \( \text{na.rm=FALSE} \)
- \( \text{pmin}(...) \), \( \text{na.rm=FALSE} \)
Arguments

... numeric arguments.

na.rm a logical indicating whether missing values should be removed.

Value

max and min return the maximum or minimum of all the values present in their arguments, as integer if all are integer, or as double otherwise.

The minimum and maximum of an empty set are +Inf and -Inf (in this order!) which ensures transitivity, e.g., \( \min(x_1, \min(x_2)) = \min(x_1, x_2) \). In R versions before 1.5, \( \min(\text{integer}(0)) = \text{.	ext{Machine}\$\text{integer}\text{.max}} \), and analogously for max, preserving argument type, whereas from R version 1.5.0, \( \max(x) = -\text{Inf} \) and \( \min(x) = +\text{Inf} \) whenever \( \text{length}(x) = 0 \) (after removing missing values if requested).

If na.rm is FALSE an NA value in any of the arguments will cause a value of NA to be returned, otherwise NA values are ignored.

pmax and pmin take several vectors (or matrices) as arguments and return a single vector giving the “parallel” maxima (or minima) of the vectors. The first element of the result is the maximum (minimum) of the first elements of all the arguments, the second element of the result is the maximum (minimum) of the second elements of all the arguments and so on. Shorter vectors are recycled if necessary. If na.rm is FALSE, NA values in the input vectors will produce NA values in the output. If na.rm is TRUE, NA values are ignored. attributes (such as names or dim) are transferred from the first argument (if applicable).

max and min are generic functions: methods can be defined for them individually or via the Summary group generic.

References


See Also

range (both min and max) and which.min (which.max) for the arg min, i.e., the location where an extreme value occurs.

Examples

```r
require(stats)
min(5:1, pi) #-> one number
pmin(5:1, pi) #-> 5 numbers

x <- sort(rnorm(100)); cH <- 1.35
pmin(cH, quantile(x)) # no names
pmin(quantile(x), cH) # has names
plot(x, pmin(cH, pmax(-cH, x), type='b', main= "Huber's function")
```
The function `factor` is used to encode a vector as a factor (the terms ‘category’ and ‘enumerated type’ are also used for factors). If `ordered` is `TRUE`, the factor levels are assumed to be ordered. For compatibility with S there is also a function `ordered`. `is.factor`, `is.ordered`, `as.factor` and `as.ordered` are the membership and coercion functions for these classes.

**Usage**

```r
factor(x, levels = sort(unique.default(x), na.last = TRUE),
       labels = levels, exclude = NA, ordered = is.ordered(x))

ordered(x, ...)

is.factor(x)

is.ordered(x)

as.factor(x)

as.ordered(x)
```

**Arguments**

- `x` a vector of data, usually taking a small number of distinct values
- `levels` an optional vector of the values that `x` might have taken. The default is the set of values taken by `x`, sorted into increasing order.
- `labels` *either* an optional vector of labels for the levels (in the same order as `levels` after removing those in `exclude`), *or* a character string of length 1.
- `exclude` a vector of values to be excluded when forming the set of levels. This should be of the same type as `x`, and will be coerced if necessary.
- `ordered` logical flag to determine if the levels should be regarded as ordered (in the order given).
- `...` (in `ordered(.)`): any of the above, apart from `ordered` itself.

**Details**

The type of the vector `x` is not restricted.

Ordered factors differ from factors only in their class, but methods and the model-fitting functions treat the two classes quite differently.

The encoding of the vector happens as follows. First all the values in `exclude` are removed from `levels`. If `x[i]` equals `levels[j]`, then the `i`-th element of the result is `j`. If no match is found for `x[i]` in `levels`, then the `i`-th element of the result is set to `NA`.

Normally the ‘levels’ used as an attribute of the result are the reduced set of levels after removing those in `exclude`, but this can be altered by supplying `labels`. This should either be a set of new labels for the levels, or a character string, in which case the levels are that character string with a sequence number appended.
factor(x, exclude=NULL) applied to a factor is a no-operation unless there are unused levels: in that case, a factor with the reduced level set is returned. If exclude is used it should also be a factor with the same level set as x or a set of codes for the levels to be excluded.

The codes of a factor may contain NA. For a numeric x, set exclude=NULL to make NA an extra level ("NA"), by default the last level.

If "NA" is a level, the way to set a code to be missing is to use is.na on the left-hand-side of an assignment. Under those circumstances missing values are printed as <NA>.

is.factor is generic: you can write methods to handle specific classes of objects, see Internal-Methods.

Value

factor returns an object of class "factor" which has a set of integer codes the length of x with a "levels" attribute of mode character. If ordered is true (or ordered is used) the result has class c("ordered", "factor").

Applying factor to an ordered or unordered factor returns a factor (of the same type) with just the levels which occur: see also .factor for a more transparent way to achieve this.

is.factor returns TRUE or FALSE depending on whether its argument is of type factor or not. Correspondingly, is.ordered returns TRUE when its argument is ordered and FALSE otherwise.

as.factor coerces its argument to a factor. It is an abbreviated form of factor.

as.ordered(x) returns x if this is ordered, and ordered(x) otherwise.

Warning

The interpretation of a factor depends on both the codes and the "levels" attribute. Be careful only to compare factors with the same set of levels (in the same order). In particular, as.numeric applied to a factor is meaningless, and may happen by implicit coercion. To “revert” a factor f to its original numeric values, as.numeric(levels(f))[f] is recommended and slightly more efficient than as.numeric(as.character(f)).

The levels of a factor are by default sorted, but the sort order may well depend on the locale at the time of creation, and should not be assumed to be ASCII.

Note

Storing character data as a factor is more efficient storage if there is even a small proportion of repeats. On a 32-bit machine storing a string of n bytes takes $28 + 8\lceil(n + 1)/8\rceil$ bytes whereas storing a factor code takes 4 bytes. (On a 64-bit machine 28 is replaced by 56 or more.) Only if they were computed from the same values (rather than, say, read from a file) will identical strings share storage.

References


See Also

.[.factor for subsetting of factors.
gl for construction of “balanced” factors and C for factors with specified contrasts. levels and nlevels for accessing the levels, and unclass to get integer codes.
Examples

```r
(ff <- factor(substring("statistics", 1:10, 1:10), levels=letters))
as.integer(ff) # the internal codes
factor(ff) # drops the levels that do not occur
ff[, drop=TRUE] # the same, more transparently

factor(letters[1:20], label="letter")
class(ordered(4:1)) # "ordered", inheriting from "factor"

## suppose you want "NA" as a level, and to allowing missing values.
(x <- factor(c(1, 2, "NA"), exclude = ""))
is.na(x)[2] <- TRUE
x # [1] 1  <NA>  NA, <NA> used because NA is a level.
is.na(x)
# [1] FALSE TRUE FALSE
```

file.access

Ascertain File Accessibility

Description

Utility function to access information about files on the user’s file systems.

Usage

```r
file.access(names, mode = 0)
```

Arguments

- `names`: character vector containing file names.
- `mode`: integer specifying access mode required.

Details

The mode value can be the exclusive or of the following values

- `0` test for existence.
- `1` test for execute permission.
- `2` test for write permission.
- `4` test for read permission.

Execute permission is ignored (always granted) under Windows, and all directories have both read and write permissions.

Please note that it is not good to use this function to test before trying to open a file. On a multi-tasking system, it is possible that the accessibility of a file will change between the time you call `file.access()` and the time you try to open the file, and in recent Windows versions the underlying function in `msvcrtd.dll` sometimes returns inaccurate values. It is better to wrap file open attempts in `try` instead.
**file.choose**

**Value**

An integer vector with values 0 for success and -1 for failure.

**Note**

This is intended as a replacement for the S-PLUS function `access`, a wrapper for the C function of the same name, which explains the return value encoding. Note that the return value is false for success.

**See Also**

`file.info`, `try`

**Examples**

```r
fa <- file.access(dir(".*"))
table(fa) # count successes & failures
```

---

**file.choose** *Choose a File Interactively*

**Description**

Choose a file interactively.

**Usage**

`file.choose(new = FALSE)`

**Arguments**

- `new` Logical: choose the style of dialog box presented to the user: at present only `new = FALSE` is used.

**Value**

A character vector of length one giving the file path.

**See Also**

`list.files` for non-interactive selection. `choose.files` for selecting multiple files interactively.
Description
Utility function to extract information about files on the user's file systems.

Usage
file.info(...)

Arguments
... character vectors containing file names.

Details
What is meant by "file access" and hence the last access time is system-dependent.
File modes are probably only useful on Windows NT/2000/XP machines.

Value
A data frame with row names the file names and columns
size double: File size in bytes.
isdir logical: Is the file a directory?
mode integer of class "octmode": The file permissions, printed in octal, for example 644.
mtime, ctime, atime integer of class "POSIXct": file modification, creation and last access times.

Entries for non-existent or non-readable files will be NA.

Note
This function will only be operational on systems with the stat system call, but that seems very widely available.
Some (broken) systems allow files of more than 2Gb to be created but not accessed by the stat system call. Such files will show up as non-readable (and very likely not be readable by any of R's input functions).

See Also
files, file.access, list.files, and DateTimeClasses for the date formats.

Examples
ncol(finf <- file.info(dir())) # at least six
## Not run: finf # the whole list
## Those that are more than 100 days old:
finf[diff(time(Sys.time()), finf[, "mtime"], units="days") > 100 , 1:4]

file.info("no-such-file-exists")
Construct Path to File

Description
Construct the path to a file from components in a platform-independent way.

Usage
```r
call.file(path(..., fsep = .Platform$file.sep))
```

Arguments
- `...`: character vectors.
- `fsep`: the path separator to use.

Value
A character vector of the arguments concatenated term-by-term and separated by `fsep` if all arguments have positive length; otherwise, an empty character vector.

Note
The components are separated by `/` (not `\`) on Windows.

Display One or More Files

Description
Display one or more files.

Usage
```r
call.file.show(..., header = rep('',nfiles), title = "R Information", delete.file=FALSE, pager=getOption("pager"))
```

Arguments
- `...`: one or more character vectors containing the names of the files to be displayed.
- `header`: character vector (of the same length as the number of files specified in `...`) giving a header for each file being displayed. Defaults to empty strings.
- `title`: an overall title for the display. If a single separate window is used for the display, `title` will be used as the window title. If multiple windows are used, their titles should combine the title and the file-specific header.
- `delete.file`: should the files be deleted after display? Used for temporary files.
- `pager`: the pager to be used.
Details

This function provides the core of the R help system, but it can be used for other purposes as well.

Note

How the pager is implemented is highly system dependent.

The basic Unix version concatenates the files (using the headers) to a temporary file, and displays it in the pager selected by the pager argument, which is a character vector specifying a system command to run on the set of files.

Most GUI systems will use a separate pager window for each file, and let the user leave it up while R continues running. The selection of such pagers could either be done using “magic” pager names being intercepted by lower-level code (such as "internal" and "console" on Windows), or by letting pager be an R function which will be called with the same arguments as file.show and take care of interfacing to the GUI.

Not all implementations will honour delete.file. In particular, using an external pager on Windows does not, as there is no way to know when the external application has finished with the file.

Author(s)

Ross Ihaka, Brian Ripley.

See Also

files, list.files, help.

Examples

file.show(file.path(R.home(), "COPYRIGHTS"))

files

File and Directory Manipulation

Description

These functions provide a low-level interface to the computer’s file system.

Usage

file.create(...)  
file.exists(...)  
file.remove(...)  
file.rename(from, to)  
file.append(file1, file2)  
file.copy(from, to, overwrite = FALSE)  
file.symlink(from, to)  
dir.create(path, showWarnings = TRUE, recursive = FALSE)
Arguments

..., file1, file2, from, to
class vectors, containing file names.

path:
a character vector containing a single path name.

overwrite:
logical; should the destination files be overwritten?

showWarnings:
logical; should the warnings on failure be shown?

recursive:
logical: should elements of the path other than the last be created? If true, like
Unix’s mkdir -p.

Details

The ... arguments are concatenated to form one character string: you can specify the files separately or as one vector. All of these functions expand path names: see path.expand.

defile.create creates files with the given names if they do not already exist and truncates them if they do.

defile.exists returns a logical vector indicating whether the files named by its argument exist.

defile.remove attempts to remove the files named in its argument.

defile.rename attempts to rename a single file. On Windows 9x/ME rename is not atomic, so it is possible that to will be deleted but from will not be renamed.

defile.append attempts to append the files named by its second argument to those named by its first. The R subscript recycling rule is used to align names given in vectors of different lengths.

defile.copy works in a similar way to file.append but with the arguments in the natural order for copying. Copying to existing destination files is skipped unless overwrite = TRUE. The to argument can specify a single existing directory.

defile.symlink makes symbolic links on those Unix-like platforms which support them. The to argument can specify a single existing directory.

dir.create creates the last element of the path, unless recursive = TRUE.

Value

dir.create and file.rename return a logical, true for success.

The remaining functions return a logical vector indicating which operation succeeded for each of the files attempted.

dir.create will return failure if the directory already exists.

Author(s)

Ross Ihaka, Brian Ripley

See Also

file.info, file.access, file.path, file.show, list.files, unlink, basename, path.expand.
Examples

```r
cat("file A\n", file="A")
cat("file B\n", file="B")
file.append("A", "B")
file.create("A")
file.append("A", rep("B", 10))
if(interactive()) file.show("A")
dir.create("tmp")
file.copy("A", "C")
dir.create("tmp")
list.files("tmp")
unlink("tmp", recursive=TRUE)
file.remove("A", "B", "C")
```

---

**findInterval**

*Find Interval Numbers or Indices*

Description

Find the indices of \(x\) in \(vec\), where \(vec\) must be sorted (non-decreasingly); i.e., if \(i < -\)**

\[
\text{findInterval}(x, vec) \text{ where } v_i \leq x_i < v_{i+1} \text{ where } v_0 := -\infty, v_{N+1} := +\infty, \text{ and } N < -\text{ length}(vec).
\]

At the two boundaries, the returned index may differ by 1, depending on the optional arguments `rightmost.closed` and `all.inside`.

Usage

```r
findInterval(x, vec, rightmost.closed = FALSE, all.inside = FALSE)
```

Arguments

- **x** numeric.
- **vec** numeric, sorted (weakly) increasingly, of length \(N\), say.
- **rightmost.closed** logical; if true, the rightmost interval, \(vec[N-1] \ldots vec[N]\) is treated as closed, see below.
- **all.inside** logical; if true, the returned indices are coerced into \(\{1, \ldots, N - 1\}\), i.e., 0 is mapped to 1 and \(N\) to \(N - 1\).

Details

The function `findInterval` finds the index of one vector \(x\) in another, \(vec\), where the latter must be non-decreasing. Where this is trivial, equivalent to \(\text{apply}(\text{outer}(x, vec, ">="), 1, \text{sum})\). as a matter of fact, the internal algorithm uses interval search ensuring \(O(n \log N)\) complexity where \(n < -\text{length}(x)\) (and \(N < -\text{length}(vec)\)). For (almost) sorted \(x\), it will be even faster, basically \(O(n)\).

This is the same computation as for the empirical distribution function, and indeed, `findInterval(t, sort(X))` is identical to \(nF_n(t; X_1, \ldots, X_n)\) where \(F_n\) is the empirical distribution function of \(X_1, \ldots, X_n\).

When `rightmost.closed = TRUE`, the result for \(x[j] = vec[N] (= \max(vec))\), is \(N - 1\) as for all other values in the last interval.
Value

vector of length length(x) with values in 0:N (and NA) where N <- length(vec), or values coerced to 1:(N-1) if all.inside = TRUE (equivalently coercing all x values inside the intervals). Note that NAs are propagated from x, and Inf values are allowed in both x and vec.

Author(s)

Martin Maechler

See Also

approx(*, method = "constant") which is a generalization of findInterval(), ecdf for computing the empirical distribution function which is (up to a factor of n) also basically the same as findInterval().

Examples

N <- 100
X <- sort(round(rt(N, df=2), 2))
tt <- c(-100, seq(-2,2, len=201), +100)
it <- findInterval(tt, X)
tt[it < 1 | it >= N] # only first and last are outside range(X)

Description

Forces the evaluation of a function argument.

Usage

force(x)

Arguments

x  
a formal argument.

Details

force forces the evaluation of a formal argument. This can be useful if the argument will be captured in a closure by the lexical scoping rules and will later be altered by an explicit assignment or an implicit assignment in a loop or an apply function.

Note

force does not force the evaluation of promises.
Examples

```r
f <- function(y) function() y
def <- vector("list", 5)
for (i in seq(along = def)) def[[i]] <- f(i)
def[[1]]() # returns 5

g <- function(y) { force(y); function() y }
def <- vector("list", 5)
for (i in seq(along = def)) def[[i]] <- g(i)
def[[1]]() # returns 1
```

Description

Functions to make calls to compiled code that has been loaded into R.

Usage

```r
.C(name, ..., NAOK = FALSE, DUP = TRUE, PACKAGE)
.Fortran(name, ..., NAOK = FALSE, DUP = TRUE, PACKAGE)
.External(name, ..., PACKAGE)
.Call(name, ..., PACKAGE)
.External.graphics(name, ..., PACKAGE)
.Call.graphics(name, ..., PACKAGE)
```

Arguments

- `name`: a character string giving the name of a C function or Fortran subroutine.
- `...`: arguments to be passed to the foreign function.
- `NAOK`: if `TRUE` then any `NA` or `NaN` or `Inf` values in the arguments are passed on to the foreign function. If `FALSE`, the presence of `NA` or `NaN` or `Inf` values is regarded as an error.
- `DUP`: if `TRUE` then arguments are “duplicated” before their address is passed to C or Fortran.
- `PACKAGE`: if supplied, confine the search for the `name` to the DLL given by this argument (plus the conventional extension, `.so`, `.sl`, `.dll`, ...). This is intended to add safety for packages, which can ensure by using this argument that no other package can override their external symbols. Use `PACKAGE="base"` for symbols linked in to R.

Details

The functions `.C` and `.Fortran` can be used to make calls to C and Fortran code.

`.External` and `.External.graphics` can be used to call compiled code that uses R objects in the same way as internal R functions.

`.Call` and `.Call.graphics` can be used call compiled code which makes use of internal R objects. The arguments are passed to the C code as a sequence of R objects. It is included to provide compatibility with S version 4.
For details about how to write code to use with .Call and .External, see the chapter on “System and foreign language interfaces” in “Writing R Extensions” in the ‘doc/manual’ subdirectory of the R source tree.

For Windows-specific details on producing the external code, see the “R Installation and Administration” manual.

**Value**

The functions .C and .Fortran return a list similar to the ... list of arguments passed in, but reflecting any changes made by the C or Fortran code.


These calls are typically made in conjunction with `dyn.load` which links DLLs to R.

The .graphics versions of .Call and .External are used when calling code which makes low-level graphics calls. They take additional steps to ensure that the device driver display lists are updated correctly.

**Argument types**

The mapping of the types of R arguments to C or Fortran arguments in .C or .Fortran is

<table>
<thead>
<tr>
<th>R Type</th>
<th>C Type</th>
<th>Fortran Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>int *</td>
<td>integer</td>
</tr>
<tr>
<td>numeric</td>
<td>double *</td>
<td>double precision</td>
</tr>
<tr>
<td>logical</td>
<td>int *</td>
<td>integer</td>
</tr>
<tr>
<td>character</td>
<td>char **</td>
<td>[see below]</td>
</tr>
<tr>
<td>raw</td>
<td>unsigned char *</td>
<td>not allowed</td>
</tr>
<tr>
<td>list</td>
<td>SEXP *</td>
<td>not allowed</td>
</tr>
<tr>
<td>other</td>
<td>SEXP</td>
<td>not allowed</td>
</tr>
</tbody>
</table>

Numeric vectors in R will be passed as type double * to C (and as double precision to Fortran) unless (i) .C or .Fortran is used, (ii) DUP is false and (iii) the argument has attribute Csingle set to TRUE (use `as.single` or `single`). This mechanism is only intended to be used to facilitate the interfacing of existing C and Fortran code.

The C type Rcomplex is defined in ‘Complex.h’ as a typedef struct {double r; double i;}. Fortran type double complex is an extension to the Fortran standard, and the availability of a mapping of complex to Fortran may be compiler dependent.

Note: The C types corresponding to integer and logical are int, not long as in S.

The first character string of a character vector is passed as a C character array to Fortran: that string may be usable as character *255 if its true length is passed separately. Only up to 255 characters of the string are passed back. (How well this works, or even if it works at all, depends on the C and Fortran compilers and the platform.)

Missing (NA) string values are passed to .C as the string "NA". As the C char type can represent all possible bit patterns there appears to be no way to distinguish missing strings from the string "NA". If this distinction is important use .Call.

Functions, expressions, environments and other language elements are passed as the internal R pointer type SEXP. This type is defined in ‘Rinternals.h’ or the arguments can be declared as generic pointers, void *. Lists are passed as C arrays of SEXP and can be declared as void * or SEXP *. Note that you cannot assign values to the elements of the list within the C routine.
Assigning values to elements of the array corresponding to the list bypasses R’s memory management/garbage collection and will cause problems. Essentially, the array corresponding to the list is read-only. If you need to return S objects created within the C routine, use the .Call interface.

R functions can be invoked using call_S or call_R and can be passed lists or the simple types as arguments.

Header files for external code

Writing code for use with .External and .Call will use internal R structures. If possible use just those defined in ‘Rinternals.h’ and/or the macros in ‘Rdefines.h’, as other header files are not installed and are even more likely to be changed.

Note

DUP=FALSE is dangerous.

There are two dangers with using DUP=FALSE.

The first is that if you pass a local variable to .C/.Fortran with DUP=FALSE, your compiled code can alter the local variable and not just the copy in the return list. Worse, if you pass a local variable that is a formal parameter of the calling function, you may be able to change not only the local variable but the variable one level up. This will be very hard to trace.

The second is that lists are passed as a single R SEXP with DUP=FALSE, not as an array of SEXP. This means the accessor macros in ‘Rinternals.h’ are needed to get at the list elements and the lists cannot be passed to call_S/call_R. New code using R objects should be written using .Call or .External, so this is now only a minor issue.

(Prior to R version 1.2.0 there has a third danger, that objects could be moved in memory by the garbage collector. The current garbage collector never moves objects.)

It is safe and useful to set DUP=FALSE if you do not change any of the variables that might be affected, e.g.,

.C("Cfunction", input=x, output=numeric(10)).

In this case the output variable did not exist before the call so it cannot cause trouble. If the input variable is not changed in the C code of Cfunction you are safe.

Neither .Call nor .External copy their arguments. You should treat arguments you receive through these interfaces as read-only.

References


See Also

dyn.load.
formals

Access to and Manipulation of the Formal Arguments

Description

Get or set the formal arguments of a function.

Usage

formals(fun = sys.function(sys.parent()))
formals(fun, envir = parent.frame()) <- value

Arguments

fun a function object, or see Details.
envir environment in which the function should be defined.
value a list of R expressions.

Details

For the first form, fun can be a character string naming the function to be manipulated, which is
searched for from the parent environment. If it is not specified, the function calling formals is
used.

Value

formals returns the formal argument list of the function specified.
The assignment form sets the formals of a function to the list on the right hand side.

See Also

args for a “human-readable” version, alist.body.function.

Examples

length(formals(lm)) # the number of formal arguments
names(formals(boxplot)) # formal arguments names

f <- function(x)a+b
formals(f) <- alist(a=,b=3) # function(a,b=3)a+b
f(2) # result = 5
Description

Format an R object for pretty printing.

Usage

format(x, ...)

## Default S3 method:
format(x, trim = FALSE, digits = NULL,
    nsmall = 0, justify = c("left", "right", "centre", "none"),
    width = NULL, na.encode = TRUE, scientific = NA,
    big.mark = ",", big.interval = 3,
    small.mark = ",", small.interval = 5,
    decimal.mark = ".", ...)  

## S3 method for class 'data.frame':
format(x, ..., justify = "none")  

## S3 method for class 'factor':
format(x, ...)  

## S3 method for class 'AsIs':
format(x, width = 12, ...)  

Arguments

x any R object (conceptually); typically numeric.
trim logical; if FALSE, logical, numeric and complex values are right-justified to
a common width: if TRUE the leading blanks for justification are suppressed.
digits how many significant digits are to be used for numeric and complex x. The
default, NULL, uses getOption(digits). This is a suggestion: enough
decimal places will be used so that the smallest (in magnitude) number has this
many significant digits, and also to satisfy nsmall. (For the interpretation
for complex numbers see signif.)
nsmall number of digits which will always appear to the right of the decimal point
in formatting real/complex numbers in non-scientific formats. Allowed values are
0 <= nsmall <= 20.
justify should a character vector be left-justified (the default), right-justified, centred
or left alone.
width default method: the minimum field width or NULL or 0 for no restriction.
AsIs method: the maximum field width for non-character objects. NULL cor-
responds to the default 12.
na.encode logical: should NA strings be encoded?
Either a logical specifying whether elements of a real or complex vector should be encoded in scientific format, or an integer penalty (see options("scipen")). Missing values correspond to the current default penalty.

... further arguments passed to or from other methods.

big.mark, big.interval, small.mark, small.interval, decimal.mark used for prettying longer decimal sequences, passed to prettyNum: that help page explains the details.

Details

format is a generic function. Apart from the methods described here there are methods for dates (see format.Date), date-times (see format.POSIXct) and for other classes such as format.octmode and format.dist.

format.data.frame formats the data frame column by column, applying the appropriate method of format for each column. Methods for columns are often similar to as.character but offer more control. Matrix and data-frame columns will be converted to separate columns in the result, and character columns (normally all) will be given class "AsIs".

format.factor converts the factor to a character vector and then calls the default method (and so justify applies).

format.AsIs deals with columns of complicated objects that have been extracted from a data frame. Character objects are passed to the default method (and so width does not apply). Otherwise it calls toString to convert the object to character (if a vector or list, element by element) and then right-justifies the result.

Justification for character vectors (and objects converted to character vectors by their methods) is done on display width (see nchar), taking double-width characters and the rendering of special characters (as escape sequences, including escaping backslash: see print.default) into account. Character strings are padded with blanks to the display width of the widest. (If na.encode = FALSE missing character strings are not included in the width computations and are not encoded.)

Numeric vectors are encoded with the minimum number of decimal places needed to display all the elements to at least the digit significant digits. However, if all the elements then have trailing zeroes, the number of decimal places is reduced until at least one element has a non-zero final digit.

Raw vectors are converted to their 2-digit hexadecimal representation by as.character.

Value

An object of similar structure to x containing character representations of the elements of the first argument x in a common format.

For numeric or complex x, dims and dimnames are preserved on matrices/arrays and names on vectors: as from R 2.2.0 no other attributes are copied.

If x is a list, the result is a character vector obtained by applying format.default(x, ...) to each element of the list (after unlist ing elements which are themselves lists), and then collapsing the result for each element with paste(collapse = ",", ")). The defaults in this case are trim = TRUE, justify = "none" since one does not usually want alignment in the collapsed strings.

References

See Also

`format.info` indicates how an atomic vector would be formatted.

`formatC`, `paste`, `as.character`, `sprintf`, `print`, `toString`, `encodeString`.

Examples

```r
format(1:10)
format(1:10L, trim = TRUE)

zz <- data.frame("(row names)" = c("aaaaa", "b"), check.names=FALSE)
format(zz)
format(zz, justify = "left")

## use of nsmall
format(13.7)
format(13.7L, nsmall = 3)
format(c(6.0, 13.1), digits = 2)
format(c(6.0, 13.1L), digits = 2, nsmall = 1)

## use of scientific
format(2^31L-1)
format(2^31L-1, sci = TRUE)

## a list
z <- list(a=letters[1:3], b=(-pi+0i)^((-2:2)/2),
          c=c(1,10,100,1000),
          d=c("a", "longer", "character", "string"))
format(z)
format(z, digits = 2)
format(z, digits = 2, justify = "left", trim = FALSE)
```

---

**format.Date**

Date Conversion Functions to and from Character

Description

Functions to convert between character representations and objects of class "Date" representing calendar dates.

Usage

```
as.Date(x, ...)
```

### S3 method for class 'character':

```r
as.Date(x, format = "", ...)
```

### S3 method for class 'Date':

```r
format(x, ...)
```

```
## S3 method for class 'Date':
as.character(x, ...)
```
Arguments

- **x**: An object to be converted.
- **format**: A character string. The default is "%Y-%m-%d". For details see **strftime**.
- **...**: Further arguments to be passed from or to other methods, including `format` for `as.character` and `as.Date` methods.

Details

The usual vector re-cycling rules are applied to `x` and `format` so the answer will be of length that of the longer of the vectors.

Locale-specific conversions to and from character strings are used where appropriate and available. This affects the names of the days and months.

The `as.Date` methods accept character strings, factors, logical NA and objects of classes "POSIXlt" and "POSIXct". (The last are converted to days by ignoring the time after midnight in the representation of the time in UTC.) Also objects of class "date" (from package `date` or `survival`) and "dates" (from package `chron`).

The `format` and `as.character` methods ignore any fractional part of the date.

Value

The `format` and `as.character` methods return a character vector representing the date.

The `as.Date` methods return an object of class "Date".

Note

The default formats follow the rules of the ISO 8601 international standard which expresses a day as "2001-02-03".

If the date string does not specify the date completely, the returned answer may be system-specific. The most common behaviour is to assume that a missing year, month or day is the current one. If it specifies a date incorrectly, reliable implementations will give an error and the date is reported as NA. Unfortunately some common implementations (such as 'glibc') are unreliable and guess at the intended meaning.

Years before 1CE (aka 1AD) will probably not be handled correctly.

References


See Also

- `Date` for details of the date class; `locales` to query or set a locale.

Your system’s help pages on `strftime` and `strptime` to see how to specify their formats. Windows users will find no help page for `strptime`: code based on ‘glibc’ is used (with corrections), so all the format specifiers described here are supported, but with no alternative number representation nor era available in any locale.
Examples

```r
## locale-specific version of the date
format(Sys.Date(), "%a %b %d")

## read in date info in format 'ddmmmyyyy'
## This will give NA(s) in some locales; setting the C locale
## as in the commented lines will overcome this on most systems.
## lct <- Sys.getlocale("LC_TIME"); Sys.setlocale("LC_TIME", "C")
x <- c("1jan1960", "2jan1960", "31mar1960", "30jul1960")
z <- as.Date(x, "%d%b%Y")
## Sys.setlocale("LC_TIME", lct)
##
z

## read in date/time info in format 'm/d/y'
dates <- c("02/27/92", "02/27/92", "01/14/92", "02/28/92", "02/01/92")
as.Date(dates, "%m/%d/%y")
```

---

**Description**

Information is returned on how `format(x, digits, nsmall)` would be formatted.

**Usage**

```r
format.info(x, digits = NULL, nsmall = 0)
```

**Arguments**

- `x`: an atomic vector; a potential argument of `format(x, ...)`.
- `digits`: how many significant digits are to be used for numeric and complex `x`. The default, NULL, uses `getOption(digits)`.
- `nsmall`: (see `format(..., nsmall)`).

**Value**

An integer vector of length 1, 3 or 6, say `r`.

For logical, integer and character vectors a single element, the width which would be used by `format(width = NULL).

For numeric vectors:

- `r[1]`: width (in characters) used by `format(x)`
- `r[3]`:
  - in `0:2:if ≥1, exponential representation would be used, with exponent length of `r[3]+1`.

For a complex vector the first three elements refer to the real parts, and there are three further elements corresponding to the imaginary parts.
format.pval

Description

format.pval is intended for formatting p-values.

Usage

format.pval(pv, digits = max(1, getOption("digits") - 2),
eps = .Machine$double.eps, na.form = "NA")

Arguments

pv
a numeric vector.
digits
how many significant digits are to be used.
eps
a numerical tolerance: see Details.
na.form
character representation of NAs.

Details

format.pval is mainly an auxiliary function for print.summary.lm etc., and does separate
formatting for fixed, floating point and very small values; those less than eps are formatted as "<
[eps]" (where "[eps]" stands for format(eps, digits)).

Value

A character vector.
146

formatC

Examples
format.pval(c(runif(5), pi^-100, NA))
format.pval(c(0.1, 0.0001, 1e-27))

formatC

Formatting Using C-style Formats

Description
Formatting numbers individually and flexibly, using C style format specifications.
Usage
formatC(x, digits = NULL, width = NULL,
format = NULL, flag = "", mode = NULL,
big.mark = "", big.interval = 3,
small.mark = "", small.interval = 5,
decimal.mark = ".")
prettyNum(x, big.mark = "",
big.interval = 3,
small.mark = "", small.interval = 5,
decimal.mark = ".", ...)
Arguments
x

an atomic numerical or character object, typically a vector of real numbers.

digits

the desired number of digits after the decimal point (format = "f") or significant digits (format = "g", = "e" or = "fg").
Default: 2 for integer, 4 for real numbers. If less than 0, the C default of 6 digits
is used.

width

the total field width; if both digits and width are unspecified, width
defaults to 1, otherwise to digits + 1. width = 0 will use width =
digits, width < 0 means left justify the number in this field (equivalent to
flag ="-"). If necessary, the result will have more characters than width.

format

equal to "d" (for integers), "f", "e", "E", "g", "G", "fg" (for reals), or
"s" (for strings). Default is "d" for integers, "g" for reals.
"f" gives numbers in the usual xxx.xxx format; "e" and "E" give
n.ddde+nn or n.dddE+nn (scientific format); "g" and "G" put x[i] into
scientific format only if it saves space to do so.
"fg" uses fixed format as "f", but digits as the minimum number of significant digits. That this can lead to quite long result strings, see examples below.
Note that unlike signif this prints large numbers with more significant digits
than digits.

flag

For formatC, a character string giving a format modifier as in Kernighan and
Ritchie (1988, page 243). "0" pads leading zeros; "-" does left adjustment,
others are "+", " ", and "#". There can be more than one of these, in any
order.

mode

"double" (or "real"), "integer" or "character". Default: Determined from the storage mode of x.


Character arguments given to formatC are handled in the same way as
the corresponding arguments of format, but with some minor dialectic dif-
fierences (see format for details). The arguments are:

- `big.mark`: character; if not empty used as mark between every `big.interval` decimals before (hence `big`) the decimal point.
- `big.interval`: see `big.mark` above; defaults to 3.
- `small.mark`: character; if not empty used as mark between every `small.interval` decimals after (hence `small`) the decimal point.
- `small.interval`: see `small.mark` above; defaults to 5.
- `decimal.mark`: the character to be used to indicate the numeric decimal point.
- `...`: arguments passed to `format`.

**Details**

If you set `format` it overrides the setting of `mode`, so `formatC(123.45, mode="double", format="d")` gives 123.

The rendering of scientific format is platform-dependent: some systems use `n.ddde+nnn` or `n.dddenn` rather than `n.ddde+nn`.

`formatC` does not necessarily align the numbers on the decimal point, so `formatC(c(6.11, 13.1), digits=2, format="fg")` gives `c("6.1", "13")`. If you want common formatting for several numbers, use `format`.

`prettyNum` is the utility function for prettifying `x`. If `x` is not a character, `format(x[i], ...)` is applied to each element, and then it is left unchanged if all the other arguments are at their defaults. Note that `prettyNum(x)` may behave unexpectedly if `x` is a character vector not resulting from something like `format(<number>):` in particular it assumes that a period is a decimal mark.

**Value**

A character object of same size and attributes as `x`. Unlike `format`, each number is formatted individually. Looping over each element of `x`, the C function `sprintf(...)` is called (inside the C function `str_signif`).

`formatC`: for character `x`, do simple (left or right) padding with white space.

**Author(s)**

`formatC` was originally written by Bill Dunlap, later much improved by Martin Maechler. It was first adapted for R by Friedrich Leisch.

**References**


**See Also**

`format`, `sprintf` for more general C like formatting.
Examples

```r
xx <- pi * 10^(-5:4)
cbind(format(xx, digits=4), formatC(xx))
cbind(formatC(xx, wid = 9, flag = "-"))
cbind(formatC(xx, dig = 5, wid = 8, format = "f", flag = "0"))
cbind(formatC(xx, dig = 4, format = "fg"))

formatC( c("a", "Abc", "no way"), wid = -7) # <= flag = "-"
formatC(c((-1:1)/0,c(1,100)*pi), wid=8, dig=1)

xx <- c(1e-12,-3.98765e-10,1.45645e-69,1e-70,pi*1e37,3.44e4)
## 1 2 3 4 5 6
formatC(xx)
formatC(xx, format="fg") # special "fixed" format.
formatC(xx, format="f", dig=80) #>> also long strings

r <- c("76491283764.97430", "29.12345678901", "+7.1234", "+100.1","1123")
## American:
prettyNum(r, big.mark = ",")
## Some Europeans:
prettyNum(r, big.mark = ",", decimal.mark = ",")

(dd <- sapply(1:10, function(i)paste((9:0)[1:i],collapse="")))
prettyNum(dd, big.mark="'")

## examples of 'small.mark'
pN <- stats::pnorm(1:7, lower=FALSE)
cbind(format (pN, small.mark = " ", digits = 15))
cbind(formatC(pN, small.mark = " ", digits = 17, format = "f"))
```

---

**formatDL**

Format Description Lists

Description

Format vectors of items and their descriptions as 2-column tables or LaTeX-style description lists.

Usage

```r
formatDL(x, y, style = c("table", "list"),
width = 0.9 * getOption("width"), indent = NULL)
```

Arguments

- **x**
  - a vector giving the items to be described, or a list of length 2 or a matrix with 2 columns giving both items and descriptions.
- **y**
  - a vector of the same length as x with the corresponding descriptions. Only used if x does not already give the descriptions.
- **style**
  - a character string specifying the rendering style of the description information. If "table", a two-column table with items and descriptions as columns is produced (similar to Texinfo’s @table environment. If "list", a LaTeX-style tagged description list is obtained.
width

indent

Details

After extracting the vectors of items and corresponding descriptions from the arguments, both are coerced to character vectors.

In table style, items with more than \( \text{indent} - 3 \) characters are displayed on a line of their own.

Value

a character vector with the formatted entries.

Examples

```r
## Use R to create the 'INDEX' for package 'splines' from its 'CONTENTS'
x <- read.dcf(file = system.file("CONTENTS", package = "splines"),
          fields = c("Entry", "Description"))
x <- as.data.frame(x)
writeLines(formatDL(x$Entry, x$Description))
## or equivalently: writeLines(formatDL(x))
## Same information in tagged description list style:
writeLines(formatDL(x$Entry, x$Description, style = "list"))
## or equivalently: writeLines(formatDL(x, style = "list"))
```

Description

These functions provide the base mechanisms for defining new functions in the R language.

Usage

```r
function( arglist ) expr
return(value)
```

Arguments

```r
arglist
    Empty or one or more name or name=expression terms.
value
    An expression.
```

Details

In R (unlike S) the names in an argument list cannot be quoted non-standard names.

If `value` is missing, `NULL` is returned. If it is a single expression, the value of the evaluated expression is returned.

If the end of a function is reached without calling `return`, the value of the last evaluated expression is returned.
Warning

Prior to R 1.8.0, value could be a series of non-empty expressions separated by commas. In that case the value returned is a list of the evaluated expressions, with names set to the expressions where these are the names of R objects. That is, a=foo() names the list component a and gives it value the result of evaluating foo().

This has been deprecated (and a warning is given), as it was never documented in S, and whether or not the list is named differs by S versions.

References


See Also

args and body for accessing the arguments and body of a function.
debug for debugging; invisible for return(.)ing invisibly.

Examples

```r
norm <- function(x) sqrt(x%^*%x)
norm(1:4)

## An anonymous function:
(function(x,y){ z <- x^2 + y^2; x+y+z })(0:7, 1)
```

Description

A call of gc causes a garbage collection to take place. gcinfo sets a flag so that automatic collection is either silent (verbose=FALSE) or prints memory usage statistics (verbose=TRUE).

Usage

```r
gc(verbose = getOption("verbose"), reset=FALSE)
gcinfo(verbose)
```

Arguments

- **verbose**: logical; if TRUE, the garbage collection prints statistics about cons cells and the vector heap.
- **reset**: logical; if TRUE the values for maximum space used are reset to the current values

Details

A call of gc causes a garbage collection to take place. This takes place automatically without user intervention, and the primary purpose of calling gc is for the report on memory usage.

However, it can be useful to call gc after a large object has been removed, as this may prompt R to return memory to the operating system.
Value

\texttt{gc} returns a matrix with rows "Ncells" (\textit{cons cells}), usually 28 bytes each on 32-bit systems and 56 bytes on 64-bit systems, and "Vcells" (\textit{vector cells}, 8 bytes each), and columns "used" and "gc trigger", each also interpreted in megabytes (rounded up to the next 0.1Mb).

If maxima have been set for either "Ncells" or "Vcells", a fifth column is printed giving the current limits in Mb (with \texttt{NA} denoting no limit).

The final two columns show the maximum space used since the last call to \texttt{gc(reset=TRUE)} (or since \texttt{R} started).

\texttt{gcinfo} returns the previous value of the flag.

See Also

\texttt{Memory} on \texttt{R}'s memory management, and \texttt{gctorture} if you are an \texttt{R} hacker. \texttt{reg.finalizer} for actions to happen upon garbage collection.

Examples

\begin{verbatim}
gc()  #-- do it now
gcinfo(TRUE)  #-- in the future, show when \texttt{R} does it
x <- integer(100000); for(i in 1:18) x <- c(x,i)
gcinfo(verbose = FALSE)#-- don't show it anymore
gc(TRUE)
gc(reset=TRUE)
\end{verbatim}

---

\texttt{gc.time}

\textbf{Report Time Spent in Garbage Collection}

Description

This function reports the time spent in garbage collection so far in the \texttt{R} session while GC timing was enabled.

Usage

\texttt{gc.time(on = TRUE)}

Arguments

on  \hspace{1cm} \texttt{logical}; if \texttt{TRUE}, GC timing is enabled.

Value

A numerical vector of length 5 giving the user CPU time, the system CPU time, the elapsed time and children’s user and system CPU times (normally both zero).

Warnings

This is experimental functionality, likely to be removed as soon as the next release.

The timings are rounded up by the sampling interval for timing processes, and so are likely to be over-estimates.
Note

CPU times will be returned as NA on Windows 9x/ME systems, but are genuine times on NT4/2000/XP systems. Times of child processes are not available and will always be given as NA.

See Also

gc, proc.time for the timings for the session.

Examples

gc.time()

---

gctorture  Torture Garbage Collector

Description

Provokes garbage collection on (nearly) every memory allocation. Intended to ferret out memory protection bugs. Also makes \texttt{R} run very slowly, unfortunately.

Usage

gctorture(on = TRUE)

Arguments

on  logical; turning it on/off.

Value

Previous value.

Author(s)

Peter Dalgaard

---

get  Return the Value of a Named Object

Description

Search for an \texttt{R} object with a given name and return it.

Usage

get(x, pos=-1, envir=as.environment(pos), mode="any", inherits=TRUE)
mget(x, envir, mode = "any",
    ifnotfound = list(function(x) stop(paste("value for "
        x, "," not found", sep = ""), call. = FALSE)), inherits = FALSE)
get

Arguments

- x: a variable name (given as a character string).
- pos: where to look for the object (see the details section); if omitted, the function will search as if the name of the object appeared unquoted in an expression.
- envir: an alternative way to specify an environment to look in; see the details section.
- mode: the mode of object sought.
- inherits: should the enclosing frames of the environment be searched?
- ifnotfound: A list of values to be used if the item is not found.

Details

The pos argument can specify the environment in which to look for the object in any of several ways: as an integer (the position in the search list); as the character string name of an element in the search list; or as an environment (including using sys.frame to access the currently active function calls). The envir argument is an alternative way to specify an environment, but is primarily there for back compatibility.

This function looks to see if the name x has a value bound to it in the specified environment. If inherits is TRUE and a value is not found for x in the specified environment, the enclosing frames of the environment are searched until the name x is encountered. See environment and the ‘R Language Definition’ manual for details about the structure of environments and their enclosures.

Warning: inherits=TRUE is the default behaviour for R but not for S.

The mode may specify collections such as "numeric" and "function": any member of the collection will suffice.

Using a NULL environment is equivalent to using the current environment.

For mget multiple values are returned in a named list. This is true even if only one value is requested. The value in mode and ifnotfound can be either the same length as the number of requested items or of length 1. The argument ifnotfound must be a list containing either the value to use if the requested item is not found or a function of one argument which will be called if the item is not found. The argument is the name of the item being requested. The default value for inherits is FALSE, in contrast to the default behavior for get.

mode here is a mixture of the meanings of typeof and mode: "function" covers primitive functions and operators, "numeric", "integer", "real" and "double" all refer to any numeric type, "symbol" and "name" are equivalent but "language" must be used.

Value

The object found. (If no object is found an error results.)

Note

The reverse of a <- get(nam) is assign(nam, a).

References

getCallingDLL

Compute DLL for native interface call

Description

This is an internal function that is called from R’s C code to determine the enclosing namespace of a .C/.Call/.Fortran/.External call which has no PACKAGE argument. If the call has been made from a function within a namespace, then we can find the DLL associated with that namespace. The purpose of this is to avoid having to use the PACKAGE argument in these native calls and so better support versions of packages.

This is an internal function that may be migrated to internal C code in the future and so should not be used by R programmers.

Usage

getCallingDLL(f = sys.function(-1), doStop = FALSE)
getCallingDLL(e)

Arguments

f

the function whose namespace and DLL are to be found. By default, this is the current function being called which is the one in which the native routine is being invoked.

doStop

a logical value indicating whether failure to find a namespace and/or DLL is an error (TRUE) or not (FALSE). The default is FALSE so that when this is called because there is no PACKAGE argument in a .C,.Call,.Fortran,.External call, no error occurs and the regular lookup is performed by searching all DLLs in order.

e

an environment.

See Also

.C,.Call,.Fortran,.External

Examples

if(exists("ansari.test"))
    getCallingDLL(ansari.test)
getDLLRegisteredRoutines

Reflectance Information for C/Fortran routines in a DLL

Description

This function allows us to query the set of routines in a DLL that are registered with R to enhance dynamic lookup, error handling when calling native routines, and potentially security in the future. This function provides a description of each of the registered routines in the DLL for the different interfaces, i.e. `.C`, `.Call`, `.Fortran` and `.External`.

Usage

getDLLRegisteredRoutines(dll)

Arguments

dll a character string or DLLInfo object (as returned by getLoadedDLLs).

Details

This takes the registration information after it has been registered and processed by the R internals. In other words, it uses the extended information.

Value

A list with four elements corresponding to the routines registered for the `.C`, `.Call`, `.Fortran` and `.External` interfaces. Each element is a list with as many elements as there were routines registered for that interface. Each element identifies a routine and is an object of class NativeSymbolInfo. An object of this class has the following fields:

- **name**: the registered name of the routine (not necessarily the name in the C code).
- **address**: the memory address of the routine as resolved in the loaded DLL. This may be NULL if the symbol has not yet been resolved.
- **dll**: an object of class DLLInfo describing the DLL. This is same for all elements returned.
- **numParameters**: the number of arguments the native routine is to be called with. In the future, we will provide information about the types of the parameters also.

Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

References


See Also

ggetLoadedDLLs
getLoadedDLLs

Examples

dlls <- getLoadedDLLs()
getDLLRegisteredRoutines(dlls[["base"]])

getDLLRegisteredRoutines("stats")

Description

This function provides a way to get a list of all the Dynamically Loadable Libraries (DLLs) that are currently loaded in the current R session.

Usage

generateDLLs()

Details

This queries the internal table that manages the DLLs.

Value

An object of class "DLLInfoList" which is a list with an element corresponding to each DLL that is currently loaded in the session. Each element is an object of class "DLLInfo" which has the following entries.

- `name`: the abbreviated name.
- `path`: the fully qualified name of the file which was dynamically loaded.
- `dynamicLookup`: a logical value indicating whether R uses only the registration information to resolve symbols or whether it searches the entire symbol table of the DLL.
- `handle`: a reference to the C-level data structure that provides access to the contents of the DLL. This is an object of class "DLLHandle".

Note

We are starting to use the `handle` elements in the DLL object to resolve symbols more directly in R.

Author(s)

Duncan Temple Lang ⟨duncan@wald.ucdavis.edu⟩.

See Also

generateDLLRegisteredRoutines, getNativeSymbolInfo

Examples

generateDLLs()
getDescription

Obtain a description of a native (C/Fortran) symbol

Description

This finds and returns as comprehensive a description of a dynamically loaded or “exported” built-in native symbol. It returns information about the name of the symbol, the library in which it is located and, if available, the number of arguments it expects and by which interface it should be called (i.e., `.Call`, `.C`, `.Fortran`, or `.External`). Additionally, it returns the address of the symbol and this can be passed to other C routines which can invoke. Specifically, this provides a way to explicitly share symbols between different dynamically loaded package libraries. Also, it provides a way to query where symbols were resolved, and aids diagnosing strange behavior associated with dynamic resolution.

Usage

getNativeSymbolInfo(name, PACKAGE)

Arguments

name the name of the native symbol as used in a call to `is.loaded`, etc. Note that Fortran symbols should be supplied as-is, not wrapped in `symbol.For`.

PACKAGE an optional argument that specifies to which dynamically loaded library we restrict the search for this symbol. If this is "base", we search in the R executable itself.

Details

This uses the same mechanism for resolving symbols as is used in all the native interfaces (`.Call`, etc.). If the symbol has been explicitly registered by the shared library in which it is contained, information about the number of arguments and the interface by which it should be called will be returned. Otherwise, a generic native symbol object is returned.

Value

If the symbol is not found, an error is raised. Otherwise, the value is a list containing the following elements:

name the name of the symbol, as given by the `name` argument.

address the native memory address of the symbol which can be used to invoke the routine, and also compare with other symbol address. This is an external pointer object and of class `NativeSymbol`.

package a list containing 3 elements:

- name the short form of the library name which can be used as the value of the `PACKAGE` argument in the different native interface functions.
- path the fully qualified name of the shared library file.
- `dynamicLookup` a logical value indicating whether dynamic resolution is used when looking for symbols in this library, or only registered routines can be located.
getNumCConverters

numParameters

the number of arguments that should be passed in a call to this routine.

Additionally, the list will have an additional class, being CRoutine, CallRoutine, FortranRoutine or ExternalRoutine corresponding to the R interface by which it should be invoked.

Note

One motivation for accessing this reflectance information is to be able to pass native routines to C routines as “function pointers” in C. This allows us to treat native routines and R functions in a similar manner, such as when passing an R function to C code that makes callbacks to that function at different points in its computation (e.g., \texttt{nls}). Additionally, we can resolve the symbol just once and avoid resolving it repeatedly or using the internal cache. In the future, one may be able to treat NativeSymbol objects directly as callback objects.

Author(s)

Duncan Temple Lang

References

For information about registering native routines, see “In Search of C/C++ & FORTRAN Routines”, R News, volume 1, number 3, 2001, p20–23 (\url{http://CRAN.R-project.org/doc/Rnews/}).

See Also

\texttt{getDLLRegisteredRoutines}, \texttt{is.loaded}, \texttt{.C}, \texttt{.Fortran}, \texttt{.External}, \texttt{.Call}, \texttt{dyn.load}.

Examples

```r
library(stats) # normally loaded
getNativeSymbolInfo("dansari")

generateSymbolInfo("hcass2") # a Fortran symbol
```

getNumCConverters  \textit{Management of .C argument conversion list}

Description

These functions provide facilities to manage the extensible list of converters used to translate R objects to C pointers for use in \texttt{.C} calls. The number and a description of each element in the list can be retrieved. One can also query and set the activity status of individual elements, temporarily ignoring them. And one can remove individual elements.

Usage

```r
getNumCConverters()
getCConverterDescriptions()
getCConverterStatus()
setCConverterStatus(id, status)
removeCConverter(id)
```
Arguments

id       either a number or a string identifying the element of interest in the converter list. A string is matched against the description strings for each element to identify the element. Integers are specified starting at 1 (rather than 0).
status    a logical value specifying whether the element is to be considered active (TRUE) or not (FALSE).

Details

The internal list of converters is potentially used when converting individual arguments in a .C call. If an argument has a non-trivial class attribute, we iterate over the list of converters looking for the first that “matches”. If we find a matching converter, we have it create the C-level pointer corresponding to the R object. When the call to the C routine is complete, we use the same converter for that argument to reverse the conversion and create an R object from the current value in the C pointer. This is done separately for all the arguments.

The functions documented here provide R user-level capabilities for investigating and managing the list of converters. There is currently no mechanism for adding an element to the converter list within the R language. This must be done in C code using the routine R_addToCConverter().

Value

getNumCConverters returns an integer giving the number of elements in the list, both active and inactive.
getCConverterDescriptions returns a character vector containing the description string of each element of the converter list.
getCConverterStatus returns a logical vector with a value for each element in the converter list. Each value indicates whether that converter is active (TRUE) or inactive (FALSE). The names of the elements are the description strings returned by getCConverterDescriptions.
setCConverterStatus returns the logical value indicating the activity status of the specified element before the call to change it took effect. This is TRUE for active and FALSE for inactive.
removeCConverter returns TRUE if an element in the converter list was identified and removed. In the case that no such element was found, an error occurs.

Author(s)

Duncan Temple Lang

References


See Also

.C

Examples

getNumCConverters()
getCConverterDescriptions()
getCConverterStatus()
## Not run:
old <- setCConverterStatus(1, FALSE)
getCConverterStatus(1, old)
## End(Not run)
## Not run:
removeCConverter(1)
removeCConverter(getCConverterDescriptions()[1])
## End(Not run)

getpid

Get the Process ID of the R Session

Description

Get the process ID of the R Session. It is guaranteed by the operating system that two R sessions running simultaneously will have different IDs, but it is possible that R sessions running at different times will have the same ID.

Usage

Sys.getpid()

Value

An integer, usually a small integer between 0 and 32767 under Unix-alikes and a much small integer under Windows.

Examples

Sys.getpid()

gettext

Translate Text Messages

Description

If Native Language Support was enabled in this build of R, attempt to translate character vectors or set where the translations are to be found.

Usage

ggettext(..., domain = NULL)
ngettext(n, msg1, msg2, domain = NULL)
bindtextdomain(domain, dirname = NULL)
gettext

Arguments

... One or more character vectors.
domain The 'domain' for the translation.
n a non-negative integer.
msg1 the message to be used in English for n = 1.
msg2 the message to be used in English for n = 0, 2, 3, ....
dirname The directory in which to find translated message catalogs for the domain.

Details

If domain is NULL or "", a domain is searched for based on the namespace which contains the
function calling gettext or ngettext. If a suitable domain can be found, each character string
is offered for translation, and replaced by its translation into the current language if one is found.
Conventionally the domain for R warning/error messages in package pkg is "R-pkg", and that for
C-level messages is "pkg".
For gettext, leading and trailing whitespace is ignored when looking for the translation.
gettext is used where the message needs to vary by a single integer. Translating such messages
is subject to very specific rules for different languages: see the GNU Gettext Manual. The string
will often contain a single instance of %d to be used in sprintf. If English is used, msg1 is
returned if n == 1 and msg2 in all other cases.

Value

For gettext, a character vector, one element per string in .... If translation is not enabled or no
domain is found or no translation is found in that domain, the original strings are returned.
For ngettext, a character string.
For bindtextdomain, a character string giving the current base directory, or NULL if setting it
failed.

See Also

stop and warning make use of gettext to translate messages.
xgettext for extracting translatable strings from R source files.

Examples

bindtextdomain("R") # non-null iff NLS is enabled
for(n in 0:3)
  print(sprintf(ngettext(n, "%d variable has missing values",
                  "%d variables have missing values"),
              n))

## Not run:
## for translation, those strings should appear in R-pkg.pot as
msgid "%d variable has missing values"
msgid_plural "%d variables have missing values"
msgstr[0]"
msgstr[1]"
## End(Not run)
getwd

Get or Set Working Directory

Description

getwd returns an absolute filename representing the current working directory of the R process; setwd(dir) is used to set the working directory to dir.

Usage

getwd()
setwd(dir)

Arguments

dir A character string.

Value

getwd returns a character vector, or NULL if the working directory is not available on that platform.
setwd returns NULL invisibly. It will give an error if it does not succeed.

Note

These functions are not implemented on all platforms.

See Also

list.files for the contents of a directory.

Examples

(WD <- getwd())
if (!is.null(WD)) setwd(WD)
**gl**

Generate Factor Levels

**Description**

Generate factors by specifying the pattern of their levels.

**Usage**

\[ \text{gl}(n, k, \text{length} = n \times k, \text{labels} = 1:n, \text{ordered} = \text{FALSE}) \]

**Arguments**

- **n**: an integer giving the number of levels.
- **k**: an integer giving the number of replications.
- **length**: an integer giving the length of the result.
- **labels**: an optional vector of labels for the resulting factor levels.
- **ordered**: a logical indicating whether the result should be ordered or not.

**Value**

The result has levels from 1 to \( n \) with each value replicated in groups of length \( k \) out to a total length of \( \text{length} \).

\text{gl} is modelled on the \text{GLIM} function of the same name.

**See Also**

The underlying \text{factor}().

**Examples**

```r
## First control, then treatment:
gl(2, 8, label = c("Control", "Treat"))
## 20 alternating 1s and 2s
gl(2, 1, 20)
## alternating pairs of 1s and 2s
gl(2, 2, 20)
```

---

**grep**

Pattern Matching and Replacement

**Description**

grep searches for matches to \text{pattern} (its first argument) within the character vector \( x \) (second argument). \text{regexpr} and \text{gregexpr} do too, but return more detail in a different format.

\text{sub} and \text{gsub} perform replacement of matches determined by regular expression matching.
Usage

grep(pattern, x, ignore.case = FALSE, extended = TRUE, perl = FALSE, value = FALSE, fixed = FALSE, useBytes = FALSE)

sub(pattern, replacement, x, ignore.case = FALSE, extended = TRUE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

gsub(pattern, replacement, x, ignore.case = FALSE, extended = TRUE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

regexpr(pattern, text, extended = TRUE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

gregexpr(pattern, text, extended = TRUE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

Arguments

pattern character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Coerced to character if possible.

x, text a character vector where matches are sought. Coerced to character if possible.

ignore.case if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

extended if TRUE, extended regular expression matching is used, and if FALSE basic regular expressions are used.

perl logical. Should perl-compatible regexps be used? Has priority over extended.

value if FALSE, a vector containing the (integer) indices of the matches determined by grep is returned, and if TRUE, a vector containing the matching elements themselves is returned.

fixed logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments.

useBytes logical. If TRUE the matching is done byte-by-byte rather than character-by-character. See Details.

replacement a replacement for matched pattern in sub and gsub. Coerced to character if possible. This can include backreferences "\1" to "\9" to parenthesized subexpressions of pattern. For perl = TRUE only, it can also contain "\U" or "\L" to convert the rest of the replacement to upper or lower case.

Details

Arguments which should be character strings or character vectors are coerced to character if possible.

The two *sub* functions differ only in that sub replaces only the first occurrence of a pattern whereas gsub replaces all occurrences.

For regexpr it is an error for pattern to be NA, otherwise NA is permitted and matches only itself.
The regular expressions used are those specified by POSIX 1003.2, either extended or basic, depending on the value of the extended argument, unless perl = TRUE when they are those of PCRE, ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/. (The exact set of patterns supported may depend on the version of PCRE installed on the system in use.)

useBytes is only used if fixed = TRUE or perl = TRUE. For grep its main effect is to avoid errors/warnings about invalid inputs, but for regexp it changes the interpretation of the output.

Value

For grep a vector giving either the indices of the elements of x that yielded a match or, if value is TRUE, the matched elements.

For sub and gsub a character vector of the same length as the original.

For regexp an integer vector of the same length as text giving the starting position of the first match, or -1 if there is none, with attribute "match.length" giving the length of the matched text (or -1 for no match). In a multi-byte locale these quantities are in characters rather than bytes unless useBytes = TRUE is used with fixed = TRUE or perl = TRUE.

For gregexpr a list of the same length as text each element of which is an integer vector as in regexp, except that the starting positions of every match are given.

If in a multi-byte locale the pattern or replacement is not a valid sequence of bytes, an error is thrown. An invalid string in x or text is a non-match with a warning for grep or regexp, but an error for sub or gsub.

Warning

The standard regular-expression code has been reported to be very slow when applied to extremely long character strings (tens of thousands of characters or more): the code used when perl = TRUE seems much faster and more reliable for such usages.

The standard version of gsub does not substitute correctly repeated word-boundaries (e.g. pattern = '\b\b'). Use perl = TRUE for such matches.

The perl = TRUE option is only implemented for single-byte and UTF-8 encodings, and will warn if used in a non-UTF-8 multi-byte locale (unless useBytes = FALSE).

References


See Also

regular expression (aka regexp) for the details of the pattern specification.
agrep for approximate matching.
tolower, toupper and chartr for character translations. charmatch, pmatch, match. apropos uses regexps and has nice examples.

Examples

grep("^[a-z]", letters)

txt <- c("arm","foot","lefoo", "bafoobar")
if(any(i <- grep("foo",txt))
Group Generic Functions

Description

Group generic functions can be defined with either S3 and S4 methods (with different groups). Methods are defined for the group of functions as a whole.

A method defined for an individual member of the group takes precedence over a method defined for the group as a whole.

When package methods is attached there are objects visible with the names of the group generics: these functions should never be called directly (a suitable error message will result if they are).
Usage

## S3 methods have prototypes:
Math(x, ...)
Ops(e1, e2)
Summary(x, ...)
Complex(z)

## S4 methods have prototypes:
Arith(e1, e2)
Compare(e1, e2)
Ops(e1, e2)
Math(x)
Math2(x, digits)
Summary(x, ..., na.rm = FALSE)
Complex(z)

Arguments

x, z, e1, e2 objects.
digits number of digits to be used in round or signif.
... further arguments passed to or from methods.
na.rm logical: should missing values be removed?

S3 Group Dispatching

There are four groups for which S3 methods can be written, namely the "Math", "Ops", "Summary" and "Complex" groups. These are not R objects, but methods can be supplied for them and base R contains \texttt{factor}, \texttt{data.frame} and \texttt{difftime} methods for the first three groups. (There are also a \texttt{ordered} method for Ops, \texttt{POSIXt} methods for Math and Ops, as well as a \texttt{ts} method for Ops in package \texttt{stats}.)

1. Group "Math":
   - abs, sign, sqrt,
     floor, ceiling, trunc,
     round, signif
   - exp, log,
     cos, sin, tan,
     acos, asin, atan
     cosh, sinh, tanh,
     acosh, asinh, atanh
   - lgamma, gamma, gammaCody,
     digamma, trigamma
   - cumsum, cumprod, cummax, cummin

2. Group "Ops":
   - 
     +, -, *, /, ^, \%, \%, \%
   - 
     , , ,
   - 
     =, !=, <, <=, >, >=

3. Group "Summary":
   - all, any
* sum, prod
* min, max
* range

4. Group Complex:
   * Arg, Conj, Im, Mod, Re

Note that a method will be used for either one of these groups or one of its members only if it corresponds to a "class" attribute, as the internal code dispatches on oldClass and not on class. This is for efficiency: having to dispatch on, say, Ops.integer would be too slow.

The number of arguments supplied for "Math" group generic methods is not checked prior to dispatch. (Most have default methods expecting one argument, but three expect two.)

S4 Group Dispatching

When package methods is attached (which it is by default), formal (S4) methods can be defined for groups.

The functions belonging to the various groups are as follows:

**Arith** "+", "-", "\*", "/", "^", "%%", "%/%", "/\"n"

**Compare** "==", ">", "<", "!=", "<="

**Ops** "Arith", "Compare"


**Math2** "round", "signif"

**Summary** "max", "min", "range", "prod", "sum", "any", "all"

**Complex** "Arg", "Conj", "Im", "Mod", "Re"

Functions with the group names exist in the methods package but should not be called directly.

All the functions in these groups (other than the group generics themselves) are basic functions in R. They are not by default S4 generic functions, and many of them are defined as primitives, meaning that they do not have formal arguments. However, you can still define formal methods for them. The effect of doing so is to create an S4 generic function with the appropriate arguments, in the environment where the method definition is to be stored. It all works more or less as you might expect, admittedly via a bit of trickery in the background.

Note: currently those members which are not primitive functions must have been converted to S4 generic functions (preferably before setting an S4 group generic method) as it only sets methods for known S4 generics. This can be done by a call to setGeneric, for example setGeneric("round", group="Math2").

References

Appendix A, Classes and Methods of


See Also

methods for methods of non-Internal generic functions.
Examples

```r
methods("Math")
methods("Ops")
methods("Summary")

d.fr <- data.frame(x=1:9, y=rnorm(9))
data.class(1 + d.fr) == "data.frame" # add to d.f. ...

if(.isMethodsDispatchOn()) {  # package "methods" is attached or loaded
  setClass("testComplex", representation(zz = "complex"))
  ## method for whole group "Complex"
  setMethod("Complex", "testComplex",
            function(z) c("groupMethod", callGeneric(z@zz)))
  ## exception for Arg() :
  setMethod("Arg", "testComplex",
            function(z) c("ArgMethod", Arg(z@zz)))

  z1 <- 1+2i
  z2 <- new("testComplex", zz = z1)
  stopifnot(identical(Mod(z2), c("groupMethod", Mod(z1))))
  stopifnot(identical(Arg(z2), c("ArgMethod", Arg(z1))))
}
```

 gzcon

*(De)compress I/O Through Connections*

Description

gzcon provides a modified connection that wraps an existing connection, and decompresses reads or compresses writes through that connection. Standard gzip headers are assumed.

Usage

```r
gzcon(con, level = 6, allowNonCompressed = TRUE)
```

Arguments

- **con** a connection.
- **level** integer between 0 and 9, the compression level when writing.
- **allowNonCompressed** logical. When reading, should non-compressed files (lacking the gzip magic header) be allowed?

Details

If `con` is open then the modified connection is opened. Closing the wrapper connection will also close the underlying connection.

Reading from a connection which does not supply a gzip magic header is equivalent to reading from the original connection if `allowNonCompressed` is true, otherwise an error.

The original connection is unusable: any object pointing to it will now refer to the modified connection.
Hyperbolic

Value

An object inheriting from class "connection". This is the same connection number as supplied, but with a modified internal structure.

See Also

gzfile

Examples

```r
## Not run:
## This example may not still be available
## print the value to see what objects were created.
con <- url("http://hesweb1.med.virginia.edu/biostat/s/data/sav/kprats.sav")
print(load(con))
## End(Not run)

## gzfile and gzcon can inter-work.
## Of course here one would used gzfile, but file() can be replaced by
## any other connection generator.
zz <- gzfile("ex.gz", "w")
cat("TITLE extra line", "2 3 5 7", "", "11 13 17", file = zz, sep = "\n")
close(zz)
readLines(zz <- gzcon(file("ex.gz")))
close(zz)
unlink("ex.gz")

zz <- gzcon(file("ex.gz", "wb"))
cat("TITLE extra line", "2 3 5 7", "", "11 13 17", file = zz, sep = "\n")
close(zz)
readLines(zz <- gzfile("ex.gz"))
close(zz)
unlink("ex.gz")
```

Description

These functions give the obvious hyperbolic functions. They respectively compute the hyperbolic cosine, sine, tangent, and their inverses, arc-cosine, arc-sine, arc-tangent (or "area cosine", etc).

Usage

```r
cosh(x)
sinh(x)
tanh(x)
acosh(x)
asinh(x)
atanh(x)
```

Arguments

- `x`: a numeric or complex vector
Details

These are generic functions: methods can be defined for them individually or via the Math group generic.

Branch cuts are consistent with the inverse trigonometric functions asin() et seq, and agree with those defined in Abramowitz and Stegun, figure 4.7, page 86.

References


See Also

The trigonometric functions, cos, sin, tan, and their inverses acos, asin, atan.
The logistic distribution function plogis is a shifted version of tanh() for numeric x.

iconv

Convert Character Vector between Encodings

Description

This uses system facilities to convert a character vector between encodings: the ‘i’ stands for ‘internationalization’.

Usage

iconv(x, from, to, sub=NA)

iconvlist()

Arguments

x A character vector.
from A character string describing the current encoding.
to A character string describing the target encoding.
sub character string. If not NA it is used to replace any non-convertible bytes in the input. (This would normally be a single character, but can be more. If "byte", the indication is "<xx>" with the hex code of the byte.

Details

The names of encodings and which ones are available (and indeed, if any are) is platform-dependent. On systems that support R’s iconv you can use "" for the encoding of the current locale, as well as "latin1" and "UTF-8".

iconvlist provides an alphabetical list of the supported encodings.

Elements of x which cannot be converted (perhaps because they are invalid or because they cannot be represented in the target encoding) will be returned as NA unless sub is specified.

Some versions of iconv will allow transliteration by appending //TRANSLIT to the to encoding: see the examples.
identical

Test Objects for Exact Equality

Description

The safe and reliable way to test two objects for being exactly equal. It returns TRUE in this case, FALSE in every other case.

Usage

identical(x, y)

Arguments

x, y any R objects.
Details

A call to identical is the way to test exact equality in if and while statements, as well as in logical expressions that use && or ||. In all these applications you need to be assured of getting a single logical value.

Users often use the comparison operators, such as == or !=, in these situations. It looks natural, but it is not what these operators are designed to do in R. They return an object like the arguments. If you expected x and y to be of length 1, but it happened that one of them wasn’t, you will not get a single FALSE. Similarly, if one of the arguments is NA, the result is also NA. In either case, the expression if(x == y).... won’t work as expected.

The function all.equal is also sometimes used to test equality this way, but was intended for something different: It allows for “reasonable” differences in numeric results. The computations in identical are also reliable and usually fast. There should never be an error. The only known way to kill identical is by having an invalid pointer at the C level, generating a memory fault. It will usually find inequality quickly. Checking equality for two large, complicated objects can take longer if the objects are identical or nearly so, but represent completely independent copies. For most applications, however, the computational cost should be negligible.

As from R 1.6.0, identical sees NaN as different from as.double(NA), but all NaNs are equal (and all NA of the same type are equal).

Value

A single logical value, TRUE or FALSE, never NA and never anything other than a single value.

Author(s)

John Chambers

References


See Also

all.equal for descriptions of how two objects differ; Comparison for operators that generate elementwise comparisons. isTRUE is a simple wrapper based on identical.

Examples

identical(1, NULL) ## FALSE -- don't try this with ==
identical(1, 1.) ## TRUE in R (both are stored as doubles)
identical(1, as.integer(1)) ## FALSE, stored as different types

x <- 1.0; y <- 0.99999999999
## how to test for object equality allowing for numeric fuzz :
(E <- all.equal(x,y))
isTRUE(E) # which is simply defined to just use identical(TRUE, E)
## If all.equal thinks the objects are different, it returns a
## character string, and the above expression evaluates to FALSE

# even for unusual R objects :
identical(.GlobalEnv, environment())
Conditional Element Selection

Description

`ifelse` returns a value with the same shape as `test` which is filled with elements selected from either `yes` or `no` depending on whether the element of `test` is TRUE or FALSE.

Usage

`ifelse(test, yes, no)`

Arguments

- `test`: an object which can be coerced to logical mode.
- `yes`: return values for true elements of `test`.
- `no`: return values for false elements of `test`.

Details

If `yes` or `no` are too short, their elements are recycled. `yes` will be evaluated if and only if any element of `test` is true, and analogously for `no`.

Missing values in `test` giving missing values in the result.

Value

A vector of the same length and attributes (including class) as `test` and data values from the values of `yes` or `no`. The mode of the answer will be coerced from logical to accommodate first any values taken from `yes` and then any values taken from `no`.

Warning

The mode of the result may depend on the value of `test`, and the class attribute of the result is taken from `test` and may be inappropriate for the values selected from `yes` and `no`.

Sometimes it is better to use a construction such as `(tmp <- yes; tmp[!test] <- no[!test]; tmp)`, possibly extended to handle missing values in `test`.

References


See Also

`if`.
Examples

```r
x <- c(6:-4)
sqrt(x)#- gives warning
sqrt(ifelse(x >= 0, x, NA))# no warning

## Note: the following also gives the warning !
ifelse(x >= 0, sqrt(x), NA)

## example of different return modes:
yes <- 1:3
no <- pi^(0:3)
typeof(ifelse(NA, yes, no)) # logical
typeof(ifelse(TRUE, yes, no)) # integer
typeof(ifelse(FALSE, yes, no))# double
```

---

**integer**

**Integer Vectors**

Description

Creates or tests for objects of type "integer".

Usage

```r
integer(length = 0)
as.integer(x, ...)
is.integer(x)
```

Arguments

- **length**: desired length.
- **x**: object to be coerced or tested.
- **...**: further arguments passed to or from other methods.

Details

Integer vectors exist so that data can be passed to C or Fortran code which expects them, and so that small integer data can be represented exactly and compactly.

Note that on almost all implementations of R the range of representable integers is restricted to about ±2 × 10^9: `doubles` can hold much larger integers exactly.

Value

`integer` creates a integer vector of the specified length. Each element of the vector is equal to 0. `as.integer` attempts to coerce its argument to be of integer type. The answer will be `NA` unless the coercion succeeds. Real values larger in modulus than the largest integer are coerced to `NA` (unlike S which gives the most extreme integer of the same sign). Non-integral numeric values are truncated towards zero (i.e., `as.integer(x)` equals `trunc(x)` there), and imaginary parts of complex numbers are discarded (with a warning). Character strings containing either a decimal representation or a hexadecimal representation (starting with 0x or 0X) can be converted, as well
as any allowed by the platform for real numbers. Like `as.vector` it strips attributes including names.

`is.integer` returns TRUE or FALSE depending on whether its argument is of integer type or not. `is.integer` is generic: you can write methods to handle specific classes of objects, see `InternalMethods`. There is a method for factors which returns FALSE. (Prior to R 2.0.0, there was no such method and for most (but not all) factors `is.integer` returned TRUE.)

**References**


**See Also**

`round` (and `ceiling` and `floor` on that help page) to convert to integral values.

**Examples**

```r
## as.integer() truncates:
x <- pi * c(-1:1,10)
as.integer(x)
```

---

**interaction**

**Compute Factor Interactions**

**Description**

`interaction` computes a factor which represents the interaction of the given factors. The result of `interaction` is always unordered.

**Usage**

`interaction(..., drop = FALSE, sep = ".")`

**Arguments**

- `...` the factors for which interaction is to be computed, or a single list giving those factors.
- `drop` if `drop` is TRUE, empty factor levels are dropped from the result. The default is to retain all factor levels.
- `sep` string to construct the new level labels by joining the constituent ones.

**Value**

A factor which represents the interaction of the given factors. The levels are labelled as the levels of the individual factors joined by `sep`, i.e. . by default.

**References**

interactive

See Also

factor; : where f:g is the same as interaction(f,g, sep=":") when f and g are factors.

Examples

a <- gl(2, 4, 8)
b <- gl(2, 2, 8, label = c("ctrl", "treat"))
s <- gl(2, 1, 8, label = c("M", "F"))
interaction(a, b)
interaction(a, b, s, sep = ":")

interactive

Is R Running Interactively?

Description

Return TRUE when R is being used interactively and FALSE otherwise.

Usage

interactive()

See Also

source, .First

Examples

.First <- function() if(interactive()) x11()

Internal

Call an Internal Function

Description

.Internal performs a call to an internal code which is built in to the R interpreter. Only true R wizards should even consider using this function.

Usage

.Internal(call)

Arguments

call a call expression

See Also

Internal Methods  Internal Generic Functions

Description
Many R-internal functions are generic and allow methods to be written for.

Details
The following built-in functions are generic as well, i.e., you can write methods for them:
\[
[. \, [[. \ , [. \ , [<-. \ , [<=-. \ , $<-. \ ,
length, length<-,
dimnames<-, dimnames, dim<-, dim
c, unlist,
as.character, as.vector, is.array, is.atomic, is.call, is.character,
is.complex, is.double, is.environment, is.function, is.integer,
is.language, is.logical, is.list, is.matrix, is.na, is.nan is.null,
is.numeric, is.object, is.pairlist, is.recursive, is.single, is.symbol.
\]

See Also
methods for the methods of non-Internal generic functions.

invisible
Change the Print Mode to Invisible

Description
Return a (temporarily) invisible copy of an object.

Usage
invisible(x)

Arguments
x    an arbitrary R object.

Details
This function can be useful when it is desired to have functions return values which can be assigned, but which do not print when they are not assigned.

References
is.finite

See Also

return, function.

Examples

# These functions both return their argument
f1 <- function(x) x
f2 <- function(x) invisible(x)
f1(1)# prints
f2(1)# does not

is.finite          Finite, Infinite and NaN Numbers

Description

is.finite and is.infinite return a vector of the same length as x, indicating which elements are finite (not infinite and not missing).
Inf and -Inf are positive and negative “infinity” whereas NaN means “Not a Number”. (These apply to numeric values and real and imaginary parts of complex values but not to values of integer vectors.)

Usage

is.finite(x)
is.infinite(x)
Inf
NaN
is.nan(x)

Arguments

x           (numerical) object to be tested.

Details

is.finite returns a vector of the same length as x the jth element of which is TRUE if x[j] is finite (i.e., it is not one of the values NA, NaN, Inf or -Inf). All elements of character and generic (list) vectors are false, so is.finite is only useful for logical, integer, numeric and complex vectors. Complex numbers are finite if both the real and imaginary parts are.

is.infinite returns a vector of the same length as x the jth element of which is TRUE if x[j] is infinite (i.e., equal to one of Inf or -Inf). This will be false unless x is numeric or complex. Complex numbers are infinite if either the real and imaginary part is.

is.nan tests if a numeric value is NaN. Do not test equality to NaN, or even use identical, since systems typically have many different NaN values. In most ports of R one of these is used for the numeric missing value NA. It is generic: you can write methods to handle specific classes of objects, see InternalMethods.
Note

In R, basically all mathematical functions (including basic Arithmetic), are supposed to work properly with +/- Inf and NaN as input or output.

The basic rule should be that calls and relations with Infs really are statements with a proper mathematical limit.

References

The IEC 60559 standard, also know as the ANSI/IEEE 754 Floating-Point Standard.


ACM Computing Surveys, 23(1).


http://grouper.ieee.org/groups/754/ for accessible information.

The C99 function isfinite is used for is.finite if available.

See Also

NA, ‘Not Available’ which is not a number as well, however usually used for missing values and applies to many modes, not just numeric.

Examples

pi / 0 ## = Inf a non-zero number divided by zero creates infinity
0 / 0 ## = NaN

1/0 + 1/0# Inf
1/0 - 1/0# NaN

stopifnot(1/0 == Inf,1/Inf == 0)
sin(Inf)
cos(Inf)
tan(Inf)

is.function  Is an Object of Type (Primitive) Function?

Description

Checks whether its argument is a (primitive) function.

Usage

is.function(x)
is.primitive(x)

Arguments

x  an R object.
is.language

Details

is.function is generic: you can write methods to handle specific classes of objects, see InternalMethods.

is.primitive(x) tests if x is a primitive function (either a "builtin" or "special" as from typeof)?

Value

TRUE if x is a (primitive) function, and FALSE otherwise.

Examples

is.function(1) # FALSE
is.function(is.primitive) # TRUE: it is a function, but ..
is.primitive(is.primitive) # FALSE: it's not a primitive one, whereas
is.primitive(is.function) # TRUE: that one *is*

is.language

Is an Object a Language Object?

Description

is.language returns TRUE if x is either a variable name, a call, or an expression.

Usage

is.language(x)

Arguments

x  object to be tested.

Details

is.language is generic: you can write methods to handle specific classes of objects, see InternalMethods.

References


Examples

ll <- list(a = expression(x^2 - 2*x + 1), b = as.name("Jim"),
c = as.expression(exp(1)), d = call("sin", pi))
sapply(ll, typeof)
sapply(ll, mode)
stopifnot(sapply(ll, is.language))
is.object Is an Object “internally classed”?

Description
A function rather for internal use. It returns TRUE if the object x has the R internal OBJECT attribute set, and FALSE otherwise.

Usage
is.object(x)

Arguments
x object to be tested.

Details
is.object is generic: you can write methods to handle specific classes of objects, see Internal-Methods.

See Also
class, and methods.

Examples
is.object(1) # FALSE
is.object(as.factor(1:3)) # TRUE

is.R Are we using R, rather than S?

Description
Test if running under R.

Usage
is.R()

Details
The function has been written such as to correctly run in all versions of R, S and S-PLUS. In order for code to be runnable in both R and S dialects, either your the code must define is.R or use it as

```r
if (exists("is.R") && is.function(is.R) && is.R()) {
  ## R-specific code
} else {
  ## S-version of code
}
```
is.recursive

Description

is.atomic returns TRUE if x is an atomic vector (or NULL) and FALSE otherwise.

is.recursive returns TRUE if x has a recursive (list-like) structure and FALSE otherwise.

Usage

is.atomic(x)

is.recursive(x)

Arguments

x object to be tested.

Details

These are generic: you can write methods to handle specific classes of objects, see InternalMethods.
The description here applies only to the default method.

is.atomic is true for the atomic vector types ("logical", "integer", "numeric", "complex", "character" and "raw") and NULL.

Most types of language objects are regarded as recursive: those which are not are the atomic vector types, NULL and symbols (as given by as.name).

References


See Also

is.list, is.language, etc, and the demo("is.things").
Examples

```r
is.a.r <- function(x) c(is.atomic(x), is.recursive(x))

is.a.r(c(a=1,b=3)) # TRUE FALSE
is.a.r(list()) # FALSE TRUE ??
is.a.r(list(2)) # FALSE TRUE
is.a.r(lm) # FALSE TRUE
is.a.r(y ~ x) # FALSE TRUE
is.a.r(expression(x+1)) # FALSE TRUE (not in 0.62.3!)
```

---

is.single

Is an Object of Single Precision Type?

Description

is.single reports an error. There are no single precision values in R.

Usage

```r
is.single(x)
```

Arguments

- `x` object to be tested.

Details

is.single is generic: you can write methods to handle specific classes of objects, see Internal-Methods.

References


---

jitter

Add ‘Jitter’ (Noise) to Numbers

Description

Add a small amount of noise to a numeric vector.

Usage

```r
jitter(x, factor=1, amount = NULL)
```
Arguments

- **x**: numeric to which `jitter` should be added.
- **factor**: numeric
- **amount**: numeric; if positive, used as `amount` (see below), otherwise, if = 0 the default is `factor * z/50`.

Default (NULL): `factor * d/5` where `d` is about the smallest difference between `x` values.

Details

The result, say `r`, is `r <- x + runif(n, -a, a)` where `n <- length(x)` and `a` is the `amount` argument (if specified).

Let `z <- max(x) - min(x)` (assuming the usual case). The amount `a` to be added is either provided as positive argument `amount` or otherwise computed from `z`, as follows:

- If `amount == 0`, we set `a <- factor * z/50` (same as S).
- If `amount` is NULL (default), we set `a <- factor * d/5` where `d` is the smallest difference between adjacent unique (apart from fuzz) `x` values.

Value

`jitter(x, ...)` returns a numeric of the same length as `x`, but with an amount of noise added in order to break ties.

Author(s)

Werner Stahel and Martin Maechler, ETH Zurich

References


See Also

`rug` which you may want to combine with `jitter`.

Examples

```r
round(jitter(c(rep(1,3), rep(1.2, 4), rep(3,3))), 3)
## These two 'fail' with S-plus 3.x:
jitter(rep(0, 7));
jitter(rep(10000, 5))
```
**kappa**  

*Estimate the Condition Number*

**Description**

An estimate of the condition number of a matrix or of the $R$ matrix of a $QR$ decomposition, perhaps of a linear fit. The condition number is defined as the ratio of the largest to the smallest *non-zero* singular value of the matrix.

**Usage**

```r
kappa(z, ...)  
## S3 method for class 'lm':
kappa(z, ...)  
## Default S3 method:
kappa(z, exact = FALSE, ...)  
## S3 method for class 'qr':
kappa(z, ...)  

kappa.tri(z, exact = FALSE, ...)
```

**Arguments**

- `z`  
  A matrix or a the result of `qr` or a fit from a class inheriting from "lm".
- `exact`  
  logical. Should the result be exact?
- `...`  
  further arguments passed to or from other methods.

**Details**

If `exact = FALSE` (the default) the condition number is estimated by a cheap approximation. Following S, this uses the LINPACK routine `dtrco.f`. However, in R (or S) the exact calculation is also likely to be quick enough.

`kappa.tri` is an internal function called by `kappa.qr`.

**Value**

The condition number, `kappa`, or an approximation if `exact = FALSE`.

**Author(s)**

The design was inspired by (but differs considerably from) the S function of the same name described in Chambers (1992).

**References**


**See Also**

- `svd` for the singular value decomposition and `qr` for the $QR$ one.
kronecker

Examples

kappa(x1 <- cbind(1,1:10)) # 15.71
kappa(x1, exact = TRUE) # 13.68
kappa(x2 <- cbind(x1,2:11)) # high! [x2 is singular!]

hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
sv9 <- svd(h9 <- hilbert(9))$d
kappa(h9)# pretty high!
kappa(h9, exact = TRUE) == max(sv9) / min(sv9)
kappa(h9, exact = TRUE) / kappa(h9) # .677 (i.e., rel.error = 32%)

kronecker

Kronecker products on arrays

Description

Computes the generalised kronecker product of two arrays, X and Y. kronecker(X, Y) returns an array A with dimensions dim(X) * dim(Y).

Usage

kronecker(X, Y, FUN = "*", make.dimnames = FALSE, ...)

X %x% Y

Arguments

X    A vector or array.
Y    A vector or array.
FUN   a function; it may be a quoted string.
make.dimnames
      Provide dimnames that are the product of the dimnames of X and Y.
...  optional arguments to be passed to FUN.

Details

If X and Y do not have the same number of dimensions, the smaller array is padded with dimensions of size one. The returned array comprises submatrices constructed by taking X one term at a time and expanding that term as FUN(x, Y, ...).

%x% is an alias for kronecker (where FUN is hardwired to "*").

Author(s)

Jonathan Rougier, (J.C.Rougier@durham.ac.uk)

References


See Also

outer, on which kronecker is built and %*% for usual matrix multiplication.
Examples

```r
# simple scalar multiplication
(M <- matrix(1:6, ncol=2))
kronecker(4, M)
# Block diagonal matrix:
kronecker(diag(1, 3), M)

# ask for dimnames

fred <- matrix(1:12, 3, 4, dimnames=list(LETTERS[1:3], LETTERS[4:7]))
bill <- c("happy" = 100, "sad" = 1000)
kronecker(fred, bill, make.dimnames = TRUE)

bill <- outer(bill, c("cat"=3, "dog"=4))
kronecker(fred, bill, make.dimnames = TRUE)
```

---

### l10n_info

#### Localization Information

**Description**

Report on localization information.

**Usage**

```r
l10n_info()
```

**Value**

A list with two logical components:

- **MBCS**: If a multi-byte character set in use?
- **UTF-8**: Is this a UTF-8 locale?

**See Also**

- `Sys.getlocale`, `localeconv`

**Examples**

```r
l10n_info()
```
labels  

Find Labels from Object

Description

Find a suitable set of labels from an object for use in printing or plotting, for example. A generic function.

Usage

labels(object, ...)

Arguments

object  
Any R object: the function is generic.

...  
further arguments passed to or from other methods.

Value

A character vector or list of such vectors. For a vector the results is the names or `seq(along=x)` and for a data frame or array it is the dimnames (with NULL expanded to `seq(len=d[i])`).

References


lapply  

Apply a Function over a List or Vector

Description

lapply returns a list of the same length as X. Each element of which is the result of applying FUN to the corresponding element of X.

sapply is a “user-friendly” version of lapply by default returning a vector or matrix if appropriate.

replicate is a wrapper for the common use of sapply for repeated evaluation of an expression (which will usually involve random number generation).

Usage

lapply(X, FUN, ...)  
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)

replicate(n, expr, simplify = TRUE)
Arguments

X  list or (atomic) vector to be used.
FUN  the function to be applied to each element of X. In the case of functions like +, %*%, etc., the function name must be quoted.
...  optional arguments to FUN.
simplify  logical; should the result be simplified to a vector or matrix if possible?
USE.NAMES  logical; if TRUE and if X is character, use X as names for the result unless it had names already.
n  number of replications.
expr  expression to evaluate repeatedly.

Details

Function FUN must be able to accept as input any of the elements of X. If the latter is an atomic vector, FUN will always be passed a length-one vector.

Simplification in sapply is only attempted if X has length greater than zero and if the return values from all elements of X are all of the same (positive) length. If the common length is one the result is a vector, and if greater than one is a matrix with a column corresponding to each element of X.

The mode of the simplified answer is chosen to accommodate the modes of all the values returned by the calls to FUN: see unlist.

if X has length 0, the return value of sapply is always a 0-length list.

Note

sapply(*, simplify = FALSE, USE.NAMES = FALSE) is equivalent to lapply(*).

References


See Also

apply, tapply.

Examples

x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
# compute the list mean for each list element
lapply(x,mean)
# median and quartiles for each list element
lapply(x, quantile, probs = 1:3/4)
sapply(x, quantile)
i39 <- sapply(3:9, seq) # list of vectors
sapply(i39, fivenum)

hist(replicate(100, mean(rexp(10))))
**Last.value**    
*Value of Last Evaluated Expression*

**Description**

The value of the internal evaluation of a top-level R expression is always assigned to `.Last.value` (in package:base) before further processing (e.g., printing).

**Usage**

`.Last.value`

**Details**

The value of a top-level assignment *is* put in `.Last.value`, unlike S.

Do not assign to `.Last.value` in the workspace, because this will always mask the object of the same name in package:base.

**See Also**

eval

**Examples**

```r
## These will not work correctly from example(),
## but they will in make check or if pasted in,
## as example() does not run them at the top level
gamma(1:15)            # think of some intensive calculation...
fac14 <- .Last.value   # keep them

library("splines")    # returns invisibly
.Last.value            # shows what library(.) above returned
```

---

**length**    
*Length of an Object*

**Description**

Get or set the length of vectors (including lists) and factors, and of any other R object for which a method has been defined.

**Usage**

```r
length(x)
length(x) <- value
```

**Arguments**

- `x`  
  - an R object. For replacement, a vector or factor.

- `value`  
  - an integer.
Details

Both functions are generic: you can write methods to handle specific classes of objects, see InternalMethods. length<- has a "factor" method.

The replacement form can be used to reset the length of a vector. If a vector is shortened, extra values are discarded and when a vector is lengthened, it is padded out to its new length with NAs (null for raw vectors).

Value

The default method currently returns an integer of length 1. Since this may change in the future and may differ for other methods, programmers should not rely on it.

For vectors (including lists) and factors the length is the number of elements. For an environment it is the number of objects in the environment, and NULL has length 0. For expressions and pairlists (including language objects and dotlists) it is the length of the pairlist chain. All other objects (including functions) have length one: note that for functions this differs from S.

References


See Also

nchar for counting the number of characters in character vectors.

Examples

length(diag(4)) # = 16 (4 x 4)
length(options()) # 12 or more
length(y ~ x1 + x2 + x3) # 3
length(expression(x, (y <- x^2; y+2), x^y)) # 3

## from example(warpbreaks)
fm1 <- lm(breaks ~ wool * tension, data = warpbreaks)
length(fm1$call) # 3, lm() and two arguments.
length(formula(fm1)) # 3, ~ lhs rhs

levels

Levels Attributes

Description

levels provides access to the levels attribute of a variable. The first form returns the value of the levels of its argument and the second sets the attribute.

The assignment form ("levels<-") of levels is a generic function and new methods can be written for it. The most important method is that for factors:

Usage

levels(x)
levels(x) <- value
levels

Arguments

x
an object, for example a factor.

value
A valid value for levels(x). For the default method, NULL or a character vector. For the factor method, a vector of character strings with length at least the number of levels of x, or a named list specifying how to rename the levels.

Details

For the factor assignment method, a NA in value causes that level to be removed from the levels and the elements formerly with that level to be replaced by NA.

References


See Also

nlevels.

Examples

### assign individual levels
x <- gl(2, 4, 8)
levels(x)[1] <- "low"
levels(x)[2] <- "high"
x

### or as a group
y <- gl(2, 4, 8)
levels(y) <- c("low", "high")
y

### combine some levels
z <- gl(3, 2, 12)
levels(z) <- c("A", "B", "A")
z

### same, using a named list
z <- gl(3, 2, 12)
levels(z) <- list(A=c(1,3), B=2)
z

### we can add levels this way:
f <- factor(c("a","b"))
levels(f) <- c("c", "a", "b")
f

f <- factor(c("a","b"))
levels(f) <- list(C="C", A="a", B="b")
f
libPaths

Search Paths for Packages

Description
.libPaths gets/sets the library trees within which packages are looked for.

Usage
.libPaths(new)

Arguments
new a character vector with the locations of R library trees.

Details
.Library is a character string giving the location of the default library, the ‘library’ subdirectory of R_HOME.
.libPaths is used for getting or setting the library trees that R knows about (and hence uses when looking for packages). If called with argument new, the library search path is set to the existing files in unique(c(new, .Library)) and this is returned. If given no argument, a character vector with the currently known library trees is returned.

The library search path is initialized at startup from the environment variable R_LIBS (which should be a semicolon-separated list of directories at which R library trees are rooted) by calling .libPaths with the directories specified in R_LIBS.

Value
A character vector of file paths.

References

See Also
library

Examples
.libPaths() # all library trees R knows about
library and require load add-on packages.

.First.lib is called when a package is loaded; .Last.lib is called when a package is detached.

Usage

library(package, help, pos = 2, lib.loc = NULL,
          character.only = FALSE, logical.return = FALSE,
          warn.conflicts = TRUE,
          keep.source = getOption("keep.source.pkgs"),
          verbose = getOption("verbose"),
          version)

require(package, quietly = FALSE, warn.conflicts = TRUE,
         keep.source = getOption("keep.source.pkgs"),
         character.only = FALSE, version, save = TRUE)

.First.lib(libname, pkgname)
.Last.lib(libpath)

Arguments

package, help
  the name of a package, given as a name or literal character string, or a character string, depending on whether character.only is FALSE (default) or TRUE.

pos
  the position on the search list at which to attach the loaded package. Note that .First.lib may attach other packages, and pos is computed after .First.lib has been run. Can also be the name of a position on the current search list as given by search() .

lib.loc
  a character vector describing the location of R library trees to search through, or NULL. The default value of NULL corresponds to all libraries currently known. Non-existent library trees are silently ignored.

character.only
  a logical indicating whether package or help can be assumed to be character strings.

version
  A character string denoting a version number of the package to be loaded, for use with versioned installs: see the section later in this document.

logical.return
  logical. If it is TRUE, FALSE or TRUE is returned to indicate success.

warn.conflicts
  logical. If TRUE, warnings are printed about conflicts from attaching the new package, unless that package contains an object .conflicts.OK.
keep.source logical. If TRUE, functions "keep their source" including comments, see argument keep.source to options. This applies only to the named package, and not to any packages or namespaces which might be loaded to satisfy dependencies or imports. This argument does not apply to packages using lazy-loading or saved images. Whether they have kept source is determined when they are installed (and is most likely false).

verbose a logical. If TRUE, additional diagnostics are printed.

quietly a logical. If TRUE, no message confirming package loading is printed.

save logical or environment. If TRUE, a call to require from the source for a package will save the name of the required package in the variable ".required", allowing function detach to warn if a required package is detached. See section 'Packages that require other packages' below.

libname a character string giving the library directory where the package was found.

pkgname a character string giving the name of the package, including the version number if the package was installed with --with-package-versions.

libpath a character string giving the complete path to the package.

Details

library(package) and require(package) both load the package with name package. require is designed for use inside other functions; it returns FALSE and gives a warning (rather than an error as library() does) if the package does not exist. Both functions check and update the list of currently loaded packages and do not reload code that is already loaded.

If library is called with no package or help argument, it lists all available packages in the libraries specified by lib.loc, and returns the corresponding information in an object of class "libraryIQR". The structure of this class may change in future versions. In earlier versions of R, only the names of all available packages were returned; use .packages(all = TRUE) for obtaining these. Note that installed.packages() returns even more information.

library(help = somename) computes basic information about the package somename, and returns this in an object of class "packageInfo". The structure of this class may change in future versions. When used with the default value (NULL) for lib.loc, the loaded packages are searched before the libraries.

.First.lib is called when a package without a namespace is loaded by library. (For packages with namespaces see .onLoad.) It is called with two arguments, the name of the library directory where the package was found (i.e., the corresponding element of lib.loc), and the name of the package (in that order, and with the package name including the version for a versioned install, e.g. tree_1.0-16). It is a good place to put calls to library.dynam which are needed when loading a package into this function (don’t call library.dynam directly, as this will not work if the package is not installed in a "standard" location). .First.lib is invoked after the search path interrogated by search() has been updated, so as.environment(match("package:name", search())) will return the environment in which the package is stored. If calling .First.lib gives an error the loading of the package is abandoned, and the package will be unavailable. Similarly, if the option " .First.lib" has a list element with the package’s name, this element is called in the same manner as .First.lib when the package is loaded. This mechanism allows the user to set package “load hooks” in addition to startup code as provided by the package maintainers, but setHook is preferred.

.Last.lib is called when a package is detached. Beware that it might be called if .First.lib has failed, so it should be written defensively. (It is called within try, so errors will not stop the package being detached.)
Value

library returns the list of loaded (or available) packages (or TRUE if logical.return is TRUE). require returns a logical indicating whether the required package is available.

Packages that require other packages

The source code for a package that requires one or more other packages should have a call to require, preferably near the beginning of the source, and of course before any code that uses functions, classes or methods from the other package. The default for argument save will save the names of all required packages in the environment of the new package. The saved package names are used by detach when a package is detached to warn if other packages still require the detached package. Also, if a package is installed with saved image (see INSTALL), the saved package names are used to require these packages when the new package is attached.

Formal methods

library takes some further actions when package methods is attached (as it is by default). Packages may define formal generic functions as well as re-defining functions in other packages (notably base) to be generic, and this information is cached whenever such a package is loaded after methods and re-defined functions are excluded from the list of conflicts. The check requires looking for a pattern of objects; the pattern search may be avoided by defining an object .noGenerics (with any value) in the package. Naturally, if the package does have any such methods, this will prevent them from being used.

Versioned installs

Packages can be installed with version information by R CMD INSTALL --with-package-versions or install.packages(installWithVers = TRUE). This allows more than one version of a package to be installed in a library directory, using package directory names like foo_1.5-1. When such packages are loaded, it is this versioned name that search() returns. Some utility functions require the versioned name and some the unversioned name (here foo).

If version is not specified, library looks first for a packages of that name, and then for versioned installs of the package, selecting the one with the latest version number. If version is specified, a versioned install with an exactly matching version is looked for.

If version is not specified, require will accept any version that is already loaded, whereas library will look for an unversioned install even if a versioned install is already loaded.

Loading more than one version of a package into an R session is not currently supported. Support for versioned installs is patchy.

Note

library and require can only load an installed package, and this is detected by having a `DESCRIPTION` file containing a Built: field.

Under Unix-alikes, the code checks that the package was installed under a similar operating system as given by R.version$platform (the canonical name of the platform under which R was compiled), provided it contains compiled code. Packages which do not contain compiled code can be shared between Unix-alikes, but not to other OSes because of potential problems with line endings and OS-specific help files.

As of version 2.0.0, the package name given to library and require must match the name given in the package’s DESCRIPTION file exactly, even on case-insensitive file systems such as MS Windows.
library.dynam

Loading Shared Libraries

Description

Load the specified file of compiled code if it has not been loaded already, or unloads it.

Usage

library.dynam(chname, package = NULL, lib.loc = NULL, verbose =getOption("verbose"), file.ext = .Platform$dynlib.ext, ...) library.dynam.unload(chname, libpath,
library.dynam

```r
verbose =getOption("verbose"),
file.ext = .Platform$dynlib.ext)
.dynLibs(new)
```

Arguments

- `chname` a character string naming a shared library to load.
- `package` a character vector with the names of packages to search through, or NULL. By default, all packages in the search path are used.
- `lib.loc` a character vector describing the location of R library trees to search through, or NULL. The default value of NULL corresponds to all libraries currently known.
- `libpath` the path to the loaded package whose shared library is to be unloaded.
- `verbose` a logical value indicating whether an announcement is printed on the console before loading the shared library. The default value is taken from the verbose entry in the system options.
- `file.ext` the extension to append to the file name to specify the library to be loaded. This defaults to the appropriate value for the operating system.
- `...` additional arguments needed by some libraries that are passed to the call to `dyn.load` to control how the library is loaded.
- `new` a list of DLLInfo objects corresponding to the shared libraries loaded by packages.

Details

`library.dynam` is designed to be used inside a package rather than at the command line, and should really only be used inside `.First.lib` or `.onLoad`. The system-specific extension for shared libraries (`.dll` on Windows) should not be added. Note that to allow for versioned installs, the first argument should not be set to the `pkgname` argument of `.First.lib` or `.onLoad`. `library.dynam.unload` is designed for use in `.Last.lib` or `.onUnload`. 

`.dynLibs` is used for getting or setting the shared libraries which were loaded by packages (using `library.dynam`). Versions of R prior to 2.1.0 simply recorded the (names of) packages which had loaded shared libraries. Versions of R prior to 1.6.0 used an internal global variable `.Dyn.libs` for storing this information: this variable is now defunct.

Value

If `chname` is not specified, `library.dynam` returns an object of class "DLLInfoList" corresponding to the shared libraries loaded by packages.

If `chname` is specified, an object of class "DLLInfo" that identifies the DLL and can be used in future calls is returned invisibly. For packages that have namespaces, a list of these objects is stored in the namespace’s environment for use at run-time.

`library.dynam.unload` invisibly returns an object of class "DLLInfo" identifying the DLL successfully unloaded.

References

See Also

`getLoadedDLLs` for information on DLLInfo and DLLInfoList objects.

`.First.lib`, `library`, `dyn.load`, `.packages`, `.libPaths` for how to create suitable DLLs.

Examples

```r
## Which DLLs were "dynamically loaded" by packages?
library.dynam()
```

---

### license

**The R License Terms**

The license terms under which R is distributed.

**Usage**

```r
license()
licence()
```

**Details**

R is distributed under the terms of the GNU GENERAL PUBLIC LICENSE Version 2, June 1991. A copy of this license is in `$R_HOME/COPYING`.

A small number of files (the API header files and import library) are distributed under the LESSER GNU GENERAL PUBLIC LICENSE version 2.1. A copy of this license is in `$R_HOME/COPYING.LIB`.

---

### list

**Lists – Generic and Dotted Pairs**

Functions to construct, coerce and check for all kinds of R lists.

**Usage**

```r
list(...)
pairlist(...)

as.list(x, ...)
as.pairlist(x)
as.list.environment(x, all.names=FALSE, ...)

is.list(x)
is.pairlist(x)

alist(...)
```
Arguments

... objects.

x object to be coerced or tested.

all.names a logical indicating whether to copy all values in as.list.environment/

Details

Most lists in R internally are Generic Vectors, whereas traditional dotted pair lists (as in LISP) are still available.

The arguments to list or pairlist are of the form value or tag=value. The functions return a list composed of its arguments with each value either tagged or untagged, depending on how the argument was specified.

alist is like list, except in the handling of tagged arguments with no value. These are handled as if they described function arguments with no default (cf. formals), whereas list simply ignores them.

as.list attempts to coerce its argument to list type. For functions, this returns the concatenation of the list of formals arguments and the function body. For expressions, the list of constituent calls is returned.

is.list returns TRUE iff its argument is a list or a pairlist of length > 0, whereas is.pairlist only returns TRUE in the latter case.

is.list and is.pairlist are generic: you can write methods to handle specific classes of objects, see InternalMethods.

as.list.environment copies the named values from an environment to a list. The user can request that all named objects are copied (normally names that begin with a dot are not). The output is not sorted and no parent environments are searched.

An empty pairlist, pairlist() is the same as NULL. This is different from list().

References


See Also

vector("list", length) for creation of a list with empty components; c, for concatenation; formals.

Examples

# create a plotting structure
pts <- list(x=cars[,1], y=cars[,2])
plot(pts)

## "pre-allocate" an empty list of length 5
vector("list", 5)

# Argument lists
f <- function()x
# Note the specification of a "..." argument:
formals(f) <- al <- alist(x=, y=2, ...)
### list.files

#### List the Files in a Directory/Folder

**Description**

This function produces a list containing the names of files in the named directory. `dir` is an alias.

**Usage**

```r
list.files(path = ".", pattern = NULL, all.files = FALSE, full.names = FALSE, recursive = FALSE)
dir(path = ".", pattern = NULL, all.files = FALSE, full.names = FALSE, recursive = FALSE)
```

**Arguments**

- `path` a character vector of full path names; the default corresponds to the working directory `getwd()`.
- `pattern` an optional regular expression. Only file names which match the regular expression will be returned.
- `all.files` a logical value. If `FALSE`, only the names of visible files are returned. If `TRUE`, all file names will be returned.
- `full.names` a logical value. If `TRUE`, the directory path is prepended to the file names. If `FALSE`, only the file names are returned.
- `recursive` logical. Should the listing recurse into directories?

**Value**

A character vector containing the names of the files in the specified directories, or "" if there were no files. If a path does not exist or is not a directory or is unreadable it is skipped, with a warning.

The files are sorted in alphabetical order, on the full path if `full.names = TRUE`.

**Note**

File naming conventions are very platform dependent.

If `recursive = TRUE` is not supported on all platforms, and may be ignored, with a warning.

**Author(s)**

Ross Ihaka, Brian Ripley
See Also

`file.info`, `file.access` and `files` for many more file handling functions and `file.choose` and `choose.files` for interactive selection.

Examples

```r
list.files(R.home())
## Only files starting with a-l or r (*including* uppercase):
  dir("../..", pattern = "^[a-lr]", full.names=TRUE)
```

Description

Reload datasets written with the function `save`.

Usage

```r
load(file, envir = parent.frame())
```

Arguments

- `file`: a connection or a character string giving the name of the file to load.
- `envir`: the environment where the data should be loaded.

Details

`load` can load R objects saved in the current or any earlier format. It can read a compressed file (see `save`) directly from a file or from a suitable connection (including a call to `url`). Only R objects saved in the current format (used since R 1.4.0) can be read from a connection. If no input is available on a connection a warning will be given, but any input not in the current format will result in an error.

Value

A character vector of the names of objects created, invisibly.

Warning

Saved R objects are binary files, even those saved with `ascii = TRUE`, so ensure that they are transferred without conversion of end of line markers. `load` tries to detect this case and give an informative error message.

See Also

`save`, `download.file`.
Examples

```r
## save all data
save(list = ls(all=TRUE), file = "all.Rdata")

## restore the saved values to the current environment
load("all.Rdata")

## restore the saved values to the user's workspace
load("all.Rdata", .GlobalEnv)

## Not run:
## print the value to see what objects were created.
print(load(url("http://some.where.net/R/data/kprats.rda")))
## End(Not run)
```

localecon

Find Details of the Numerical and Monetary Representations in the Current Locale

Description

Get details of the numerical and monetary representations in the current locale.

Usage

`Sys.localeconv()`

Details

These settings are usually controlled by the environment variables `LC_NUMERIC` and `LC_MONETARY` and if not set the values of `LC_ALL` or `LANG`.

Normally R is run without looking at the value of `LC_NUMERIC`, so the decimal point remains `. `. So the first three of these values will not be useful unless you have set `LC_NUMERIC` in the current R session.

Value

A character vector with 18 named components. See your ISO C documentation for details of the meaning.

It is possible to compile R without support for locales, in which case the value will be `NULL`.

See Also

`Sys.setlocale` for ways to set locales.
Examples

Sys.localeconv()
## The results in the C locale are
## decimal_point thousands_sep grouping int_curr_symbol
## "." "" "" ""
## currency_symbol mon_decimal_point mon_thousands_sep mon_grouping
## "" "" "" ""
## positive_sign negative_sign int_frac_digits frac_digits
## "" "" 127 127
## p_cs_precedes p_sep_by_space n_cs_precedes n_sep_by_space
## 127 127 127 127
## p_sign_posn n_sign_posn
## 127 127
## Now try your default locale (which might be "C").
## Not run:
old <- Sys.getlocale()
Sys.setlocale(locale = "")
Sys.localeconv()
Sys.setlocale(locale = old)
## End(Not run)

## Not run: read.table("foo", dec=Sys.localeconv()[["decimal_point"])

locales

Query or Set Aspects of the Locale

Description

Get details of or set aspects of the locale for the R process.

Usage

Sys.getlocale(category = "LC_ALL")
Sys.setlocale(category = "LC_ALL", locale = "")

Arguments

category character string. Must be one of "LC_ALL", "LC_COLLATE", "LC_CTYPE", "LC_MONETARY", "LC_NUMERIC" or "LC_TIME".
locale character string. A valid locale name on the system in use. Normally "" (the default) will pick up the default locale for the system.

Details

The locale describes aspects of the internationalization of a program. Initially most aspects of the locale of R are set to "C" (which is the default for the C language and reflects North-American usage). R sets "LC_CTYPE" and "LC_COLLATE", which allow the use of a different character set and alphabetic comparisons in that character set (including the use of sort), "LC_MONETARY" (for use by Sys.localeconv) and "LC_TIME" may affect the behaviour of as.POSIXlt and strptime and functions which use them (but not date).
\texttt{R} can be built with no support for locales, but it is normally available on Unix and is available on Windows.

Some systems will have other locale categories, but the six described here are those specified by POSIX.

Note that setting "LC\_ALL" as from \texttt{R} 2.1.0 sets only "LC\_COLLATE", "LC\_CTYPE", "LC\_MONETARY" and "LC\_TIME".

**Value**

A character string of length one describing the locale in use (after setting for \texttt{Sys.setlocale}), or an empty character string if the locale is invalid (with a warning) or NULL if locale information is unavailable.

For \texttt{category = "LC\_ALL"} the details of the string are system-specific: it might be a single locale or a set of locales separated by "/" (Solaris) or ";" (Windows, Linux). For portability, it is best to query categories individually. It is guaranteed that the result of \texttt{foo <- Sys.getlocale("LC\_ALL", locale = foo)} can be used in \texttt{Sys.setlocale("LC\_ALL", locale = foo)} on the same machine.

**Warning**

Setting "LC\_NUMERIC" may cause \texttt{R} to function anomalously, so gives a warning. (The known problems are with input conversion in locales with , as the decimal point.) Setting it temporarily to produce graphical or text output may work well enough.

**See Also**

\texttt{strptime} for uses of \texttt{category = "LC\_TIME"}. \texttt{Sys.localeconv} for details of numerical and monetary representations.

**Examples**

\begin{verbatim}
Sys.getlocale()
Sys.getlocale("LC\_TIME")
## Not run:
Sys.setlocale("LC\_TIME", "de")  # Solaris: details are OS-dependent
Sys.setlocale("LC\_TIME", "German")  # Windows
## End(Not run)
Sys.setlocale("LC\_COLLATE", "C")  # turn off locale-specific sorting
\end{verbatim}

**log**

*Logarithms and Exponentials*

**Description**

\texttt{log} computes natural logarithms, \texttt{log10} computes common (i.e., base 10) logarithms, and \texttt{log2} computes binary (i.e., base 2) logarithms. The general form \texttt{logb(x, base)} computes logarithms with base \texttt{base} (\texttt{log10} and \texttt{log2} are only special cases).

\texttt{log1p(x)} computes \(\log(1 + x)\) accurately also for \(|x| \ll 1\) (and less accurately when \(x \approx -1\)).

\texttt{exp} computes the exponential function.

\texttt{expm1(x)} computes \(\exp(x) - 1\) accurately also for \(|x| \ll 1\).
Usage

log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)
exp(x)
expm1(x)
log1p(x)

Arguments

x      a numeric or complex vector.
base   positive number. The base with respect to which logarithms are computed. De-
defaults to \(e^{\exp(1)}\).

Details

\text{exp} \text{ and } \text{log} \text{ are generic functions: methods can be defined for them individually or via the Math

\text{ group generic.}}

Value

A vector of the same length as \(x\) containing the transformed values. \text{log(0)} \text{ gives } -\text{Inf} \text{ (when available).}

Note

\text{log} \text{ and } \text{logb} \text{ are the same thing in R, but } \text{logb} \text{ is preferred if base is specified, for S-PLUS
compatibility.}

References

Brooks/Cole. (for \text{log},

\text{log10} \text{ and } \text{exp}.)


See Also

\text{Trig, sqrt, Arithmetic}.

Examples

log(exp(3))
log10(1e7)# = 7

x <- 10^(-(1+2*1:9)
cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
### Logical Operators

**Description**

These operators act on logical vectors.

**Usage**

- `! x`
- `x & y`
- `x && y`
- `x | y`
- `x || y`
- `xor(x, y)`
- `isTRUE(x)`

**Arguments**

- `x, y` logical vectors, or objects which can be coerced to such or for which methods have been written.

**Details**

- `!` indicates logical negation (NOT).
- `&` and `&&` indicate logical AND and `|` and `||` indicate logical OR. The shorter form performs elementwise comparisons in much the same way as arithmetic operators. The longer form evaluates left to right examining only the first element of each vector. Evaluation proceeds only until the result is determined. The longer form is appropriate for programming control-flow and typically preferred in `if` clauses.
- `xor` indicates elementwise exclusive OR.
- `isTRUE(x)` is an abbreviation of `identical(TRUE, x)`.

Numeric and complex vectors will be coerced to logical values, with zero being false and all non-zero values being true. Raw vectors are handled without any coercion for `!`, `&` and `|`, with these operators being applied bitwise (so `!` is the 1-complement).

The operators `!`, `&` and `|` are generic functions: methods can be written for them individually or via the `Ops` group generic function.

`NA` is a valid logical object. Where a component of `x` or `y` is `NA`, the result will be `NA` if the outcome is ambiguous. In other words `NA & TRUE` evaluates to `NA`, but `NA & FALSE` evaluates to `FALSE`. See the examples below.

**Value**

For `!`, a logical or raw vector of the same length as `x`.

For `|`, `&` and `xor` a logical or raw vector. The elements of shorter vectors are recycled as necessary (with a `warning` when they are recycled only fractionally). The rules for determining the attributes of the result are rather complicated. Most attributes are taken from the longer argument, the first if they are of the same length. Names will be copied from the first if it is the same length as the answer, otherwise from the second if that is. For time series, these operations are allowed only if
logical

the series are compatible, when the class and \texttt{tsp} attribute of whichever is a time series (the same, if both are) are used. For arrays (and an array result) the dimensions and dimnames are taken from first argument if it is an array, otherwise the second.

For \texttt{||}, \texttt{&&} and \texttt{isTRUE}, a length-one logical vector.

References


See Also

\texttt{TRUE} or \texttt{logical}.

\texttt{any} and \texttt{all} for OR and AND on many scalar arguments.

\texttt{Syntax} for operator precedence.

Examples

\begin{verbatim}
y <- 1 + (x <- rpois(50, lambda=1.5) / 4 - 1)
x[(x > 0) & (x < 1)]  # all x values between 0 and 1
if (any(x == 0) || any(y == 0)) "zero encountered"

## construct truth tables :
x <- c(NA, FALSE, TRUE)
names(x) <- as.character(x)
outer(x, x, "&")## AND table
outer(x, x, "|")## OR table
\end{verbatim}

---

logical \hspace{1cm} Logical Vectors

Description

Create or test for objects of type "logical", and the basic logical “constants”.

Usage

\begin{verbatim}
TRUE
FALSE
T; F

logical(length = 0)
as.logical(x, ...)
is.logical(x)
\end{verbatim}

Arguments

\begin{verbatim}
length desired length.
x object to be coerced or tested.
... further arguments passed to or from other methods.
\end{verbatim}
Details

TRUE and FALSE are part of the R language, where T and F are global variables set to these. All four are logical(1) vectors.

is.logical is generic: you can write methods to handle specific classes of objects, see Internal-Methods.

Value

logical creates a logical vector of the specified length. Each element of the vector is equal to FALSE.

as.logical attempts to coerce its argument to be of logical type. For factors, this uses the levels (labels). Like as.vector it strips attributes including names.

is.logical returns TRUE or FALSE depending on whether its argument is of logical type or not.

References


lower.tri

Lower and Upper Triangular Part of a Matrix

Description

Returns a matrix of logicals the same size of a given matrix with entries TRUE in the lower or upper triangle.

Usage

lower.tri(x, diag = FALSE)
upper.tri(x, diag = FALSE)

Arguments

x a matrix.

diag logical. Should the diagonal be included?

See Also

diag.matrix.

Examples

(m2 <- matrix(1:20, 4, 5))
lower.tri(m2)
m2[lower.tri(m2)] <- NA
m2
**ls**  
*List Objects*

**Description**

`ls` and `objects` return a vector of character strings giving the names of the objects in the specified environment. When invoked with no argument at the top level prompt, `ls` shows what data sets and functions a user has defined. When invoked with no argument inside a function, `ls` returns the names of the functions local variables. This is useful in conjunction with `browser`.

**Usage**

```r
ls(name, pos = -1, envir = as.environment(pos),
   all.names = FALSE, pattern)
objects(name, pos= -1, envir = as.environment(pos),
   all.names = FALSE, pattern)
```

**Arguments**

- `name`  
  which environment to use in listing the available objects. Defaults to the current environment. Although called `name` for back compatibility, in fact this argument can specify the environment in any form; see the details section.

- `pos`  
  An alternative argument to `name` for specifying the environment as a position in the search list. Mostly there for back compatibility.

- `envir`  
  an alternative argument to `name` for specifying the environment evaluation environment. Mostly there for back compatibility.

- `all.names`  
  a logical value. If `TRUE`, all object names are returned. If `FALSE`, names which begin with a `.` are omitted.

- `pattern`  
  an optional regular expression. Only names matching `pattern` are returned.

**Details**

The `name` argument can specify the environment from which object names are taken in one of several forms: as an integer (the position in the search list); as the character string name of an element in the search list; or as an explicit environment (including using `sys.frame` to access the currently active function calls). By default, the environment of the call to `ls` or `objects` is used. The `pos` and `envir` arguments are an alternative way to specify an environment, but are primarily there for back compatibility.

**References**


**See Also**

`ls.str` for a long listing based on `str`, `apropos` (or `find`) for finding objects in the whole search path; `grep` for more details on “regular expressions”; `class`, `methods`, etc., for object-oriented programming.
Examples

```
.Ob <- 1
ls(pat="O")
ls(pat="O", all = TRUE)  # also shows ".[foo]"

# shows an empty list because inside myfunc no variables are defined
myfunc <- function() {ls()}
myfunc()

# define a local variable inside myfunc
myfunc <- function() {y <- 1; ls()}
myfunc()  # shows "y"
```

---

### make.names

**Make Syntactically Valid Names**

**Description**

Make syntactically valid names out of character vectors.

**Usage**

```r
make.names(names, unique = FALSE, allow_ = TRUE)
```

**Arguments**

- `names` character vector to be coerced to syntactically valid names. This is coerced to character if necessary.
- `unique` logical; if `TRUE`, the resulting elements are unique. This may be desired for, e.g., column names.
- `allow_` logical. For compatibility with R prior to 1.9.0.

**Details**

A syntactically valid name consists of letters, numbers and the dot or underline characters and starts with a letter or the dot not followed by a number. Names such as ".2way" are not valid, and neither are the reserved words.

The character "X" is prepended if necessary. All invalid characters are translated to ".". A missing value is translated to "NA". Names which match R keywords have a dot appended to them. Duplicated values are altered by `make.unique`.

**Value**

A character vector of same length as `names` with each changed to a syntactically valid name.

**Note**

Prior to R version 1.9.0, underscores were not valid in variable names, and code that relies on them being converted to dots will no longer work. Use `allow_ = FALSE` for back-compatibility. `allow_ = FALSE` is also useful when creating names for export to applications which do not allow underline in names (for example, S-PLUS and some DBMSs).
Make Character Strings Unique

Description

Makes the elements of a character vector unique by appending sequence numbers to duplicates.

Usage

make.unique(names, sep = ".")

Arguments

names

a character vector

sep

a character string used to separate a duplicate name from its sequence number.

Details

The algorithm used by make.unique has the property that make.unique(c(A, B)) == make.unique(c(make.unique(A), B)).

In other words, you can append one string at a time to a vector, making it unique each time, and get the same result as applying make.unique to all of the strings at once.

If character vector A is already unique, then make.unique(c(A, B)) preserves A.

Value

A character vector of same length as names with duplicates changed.

Author(s)

Thomas P Minka

See Also

make.names
Examples

```r
make.unique(c("a", "a", "a"))
make.unique(c(make.unique(c("a", "a")), "a"))

make.unique(c("a", "a", "a.2", "a"))
make.unique(c(make.unique(c("a", "a")), "a.2", "a"))

rbind(data.frame(x=1), data.frame(x=2), data.frame(x=3))
rbind(rbind(data.frame(x=1), data.frame(x=2)), data.frame(x=3))
```

---

### manglePackageName  
**Mangle the Package Name**

**Description**

This function takes the package name and the package version number and pastes them together with a separating underscore.

**Usage**

```r
manglePackageName(pkgName, pkgVersion)
```

**Arguments**

- `pkgName`: The package name, as a character string.
- `pkgVersion`: The package version, as a character string.

**Value**

A character string with the two inputs pasted together.

**Examples**

```r
manglePackageName("foo", "1.2.3")
```

---

### mapply  
**Apply a function to multiple list or vector arguments**

**Description**

A multivariate version of `sapply`. `mapply` applies `FUN` to the first elements of each argument, the second elements, the third elements, and so on. Arguments are recycled if necessary.

**Usage**

```r
mapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```
**Arguments**

- **FUN**  
  Function to apply

- **...**  
  Arguments to vectorise over (list or vector)

- **MoreArgs**  
  A list of other arguments to `FUN`

- **SIMPLIFY**  
  Attempt to reduce the result to a vector or matrix?

- **USE.NAMES**  
  If the first ...argument is character and the result doesn’t already have names, use it as the names

**Value**

A list, vector, or matrix.

**See Also**

`sapply`, `outer`

**Examples**

```r
mapply(rep, 1:4, 4:1)
mapply(rep, times=1:4, x=4:1)
mapply(rep, times=1:4, MoreArgs=list(x=42))
```

**Description**

For a contingency table in array form, compute the sum of table entries for a given index.

**Usage**

```r
margin.table(x, margin=NULL)
```

**Arguments**

- **x**  
  an array

- **margin**  
  index number (1 for rows, etc.)

**Details**

This is really just `apply(x, margin, sum)` packaged up for newbies, except that if `margin` has length zero you get `sum(x)`.

**Value**

The relevant marginal table. The class of `x` is copied to the output table, except in the summation case.
Author(s)
Peter Dalgaard

Examples
```
m <- matrix(1:4, 2)
margin.table(m, 1)
margin.table(m, 2)
```

---

**mat.or.vec**  
Create a Matrix or a Vector

**Description**

`mat.or.vec` creates an `nr` by `nc` zero matrix if `nc` is greater than 1, and a zero vector of length `nr` if `nc` equals 1.

**Usage**

```
mat.or.vec(nr, nc)
```

**Arguments**

- `nr`, `nc` numbers of rows and columns.

**Examples**

```
mat.or.vec(3, 1)
mat.or.vec(3, 2)
```

---

**match**  
Value Matching

**Description**

`match` returns a vector of the positions of (first) matches of its first argument in its second.  
`%in%` is a more intuitive interface as a binary operator, which returns a logical vector indicating if there is a match or not for its left operand.

**Usage**

```
match(x, table, nomatch = NA, incomparables = FALSE)

x %in% table
```
match

Arguments

- **x**: vector: the values to be matched.
- **table**: vector: the values to be matched against.
- **nomatch**: the value to be returned in the case when no match is found. Note that it is coerced to integer.
- **incomparables**: a vector of values that cannot be matched. Any value in x matching a value in this vector is assigned the nomatch value. Currently, FALSE is the only possible value, meaning that all values can be matched.

Details

- `%in%` is currently defined as
  
  ```r
  "%in%" <- function(x, table) match(x, table, nomatch = 0) > 0
  ```

  Factors are converted to character vectors, and then x and table are coerced to a common type (the later of the two types in R’s ordering, logical < integer < numeric < complex < character < list) before matching.

  Matching for lists is potentially very slow and best avoided except in simple cases.

Value

In both cases, a vector of the same length as x.

- **match**: An integer vector giving the position in table of the first match if there is a match, otherwise nomatch.
  
  If x[i] is found to equal table[j] then the value returned in the i-th position of the return value is j for the smallest possible j. If no match is found, the value is nomatch.

- **%in%**: A logical vector, indicating if a match was located for each element of x.

References


See Also

- `pmatch` and `charmatch` for (partial) string matching, `match.arg`, etc for function argument matching.
- `is.element` for an S-compatible equivalent of `%in%`.

Examples

```r
## The intersection of two sets :
intersect <- function(x, y) y[match(x, y, nomatch = 0)]
intersect(1:10, 7:20)

1:10 %in% c(1,3,5,9)
sstr <- c("c","ab","B","bba","c","@","bla","a","Ba","%")
sstr[sstr %in% c(letters, LETTERS)]

"%w/o%" <- function(x,y) x[!x %in% y] #-- x without y
(1:10) %w/o% c(3,7,12)
```
match.arg  Argument Verification Using Partial Matching

Description

match.arg matches arg against a table of candidate values as specified by choices.

Usage

match.arg(arg, choices, several.ok = FALSE)

Arguments

arg  a character string
choices  a character vector of candidate values
several.ok  logical specifying if arg should be allowed to have more than one element.

Details

In the one-argument form match.arg(arg), the choices are obtained from a default setting for the formal argument arg of the function from which match.arg was called.

Matching is done using pmatch, so arg may be abbreviated.

Value

The unabbreviated version of the unique partial match if there is one; otherwise, an error is signalled if several.ok is false, as per default. When several.ok is true and there is more than one match, all unabbreviated versions of matches are returned.

See Also

pmatch, match.fun, match.call.

Examples

require(stats)
## Extends the example for 'switch'
center <- function(x, type = c("mean", "median", "trimmed")) {
  type <- match.arg(type)
  switch(type,
    mean = mean(x),
    median = median(x),
    trimmed = mean(x, trim = .1))
}
x <- rcauchy(10)
center(x, "t")        # Works
center(x, "med")      # Works
try(center(x, "m"))   # Error

## Allowing more than one match:
match.arg(c("gauss", "rect", "ep"),
c("gaussian", "epanechnikov", "rectangular", "triangular"),
several.ok = TRUE)
**match.call**

**Argument Matching**

**Description**

`match.call` returns a call in which all of the specified arguments are specified by their full names.

**Usage**

```r
match.call(definition = NULL, call = sys.call(sys.parent()), expand.dots = TRUE)
```

**Arguments**

- `definition`: a function, by default the function from which `match.call` is called.
- `call`: an unevaluated call to the function specified by `definition`, as generated by `call`.
- `expand.dots`: logical. Should arguments matching `...` in the call be included or left as a `...` argument?

**Details**

`match.call` is most commonly used in two circumstances:

- To record the call for later re-use: for example most model-fitting functions record the call as element `call` of the list they return. Here the default `expand.dots = TRUE` is appropriate.
- To pass most of the call to another function, often `model.frame`. Here the common idiom is that `expand.dots = FALSE` is used, and the `...` element of the matched call is removed. An alternative is to explicitly select the arguments to be passed on, as is done in `lm`.

**Value**

An object of class `call`.

**References**


**See Also**

`call`, `pmatch`, `match.arg`, `match.fun`.

**Examples**

```r
match.call(get, call("get", "abc", i = FALSE, p = 3))
## -> get(x = "abc", pos = 3, inherits = FALSE)
fun <- function(x, lower = 0, upper = 1) {
  structure((x - lower) / (upper - lower), CALL = match.call())
}
fun(4 * atan(1), u = pi)
```
Function Verification for “Function Variables”

Description

When called inside functions that take a function as argument, extract the desired function object while avoiding undesired matching to objects of other types.

Usage

match.fun(FUN, descend = TRUE)

Arguments

FUN item to match as function.

descend logical; control whether to search past non-function objects.

Details

match.fun is not intended to be used at the top level since it will perform matching in the parent of the caller.

If FUN is a function, it is returned. If it is a symbol or a character vector of length one, it will be looked up using get in the environment of the parent of the caller. If it is of any other mode, it is attempted first to get the argument to the caller as a symbol (using substitute twice), and if that fails, an error is declared.

If descend = TRUE, match.fun will look past non-function objects with the given name; otherwise if FUN points to a non-function object then an error is generated.

This is now used in base functions such as apply, lapply, outer, and sweep.

Value

A function matching FUN or an error is generated.

Bugs

The descend argument is a bit of misnomer and probably not actually needed by anything. It may go away in the future.

It is impossible to fully foolproof this. If one attaches a list or data frame containing a character object with the same name of a system function, it will be used.

Author(s)

Peter Dalgaard and Robert Gentleman, based on an earlier version by Jonathan Rougier.

See Also

match.arg, get
Examples

# Same as get("*"):
match.fun("*")
# Overwrite outer with a vector
outer <- 1:5
## Not run:
match.fun(outer, descend = FALSE) #-> Error: not a function
## End(Not run)
match.fun(outer) # finds it anyway
is.function(match.fun("outer")) # as well

Description

Multiplies two matrices, if they are conformable. If one argument is a vector, it will be coerced to either a row or column matrix to make the two arguments conformable. If both are vectors it will return the inner product.

Usage

a %*% b

Arguments

a, b numeric or complex matrices or vectors.

Value

The matrix product. Use drop to get rid of dimensions which have only one level.

References


See Also

matrix, Arithmetic, diag.

Examples

x <- 1:4
(z <- x %*% x) # scalar ("inner") product (1 x 1 matrix)
drop(z) # as scalar

y <- diag(x)
z <- matrix(1:12, ncol = 3, nrow = 4)
y %*% z
y %*% x
x %*% z
Description

matrix creates a matrix from the given set of values.
as.matrix attempts to turn its argument into a matrix.
is.matrix tests if its argument is a (strict) matrix. It is generic: you can write methods to handle specific classes of objects, see InternalMethods.

Usage

matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
as.matrix(x)
is.matrix(x)

Arguments

data an optional data vector.
nrow the desired number of rows
ncol the desired number of columns
byrow logical. If FALSE (the default) the matrix is filled by columns, otherwise the matrix is filled by rows.
dimnames A dimnames attribute for the matrix: a list of length 2 giving the row and column names respectively.
x an R object.

Details

If either of nrow or ncol is not given, an attempt is made to infer it from the length of data and the other parameter.
If there are too few elements in data to fill the array, then the elements in data are recycled. If data has length zero, NA of an appropriate type is used for atomic vectors (0 for raw vectors) and NULL for lists.
is.matrix returns TRUE if x is a matrix (i.e., it is not a data.frame and has a dim attribute of length 2) and FALSE otherwise.
as.matrix is a generic function. The method for data frames will convert any non-numeric/complex column into a character vector using format and so return a character matrix, except that all-logical data frames will be coerced to a logical matrix.

References


See Also

data.matrix, which attempts to convert to a numeric matrix.
maxCol

Examples

```r
is.matrix(as.matrix(1:10))
!is.matrix(warpbreaks)# data.frame, NOT matrix!
warpbreaks[1:10,]
as.matrix(warpbreaks[1:10,]) #using as.matrix.data.frame(.) method
```

# Example of setting row and column names
```r
mdat <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol=3, byrow=TRUE, 
              dimnames = list(c("row1", "row2"), c("C.1", "C.2", "C.3")))
mdat
```

maxCol

Find Maximum Position in Matrix

Description

Find the maximum position for each row of a matrix, breaking ties at random.

Usage

```r
max.col(m, ties.method=c("random", "first", "last"))
```

Arguments

- `m` numerical matrix
- `ties.method` a character string specifying how “ties” are handled, "random" by default; can be abbreviated; see Details.

Details

When `ties.method = "random"`, as per default, ties are broken at random. In this case, the determination of “tie” assumes that the entries are probabilities: there is a relative tolerance of $10^{-5}$, relative to the largest entry in the row.

If `ties.method = "first"`, `max.col` returns the column number of the first of several maxima in every row, the same as `unname(unname(m, 1, unname))`. Correspondingly, `ties.method = "last"` returns the last of possibly several indices.

Value

index of a maximal value for each row, an integer vector of length `nrow(m)`.

References


See Also

`which.max` for vectors.
Examples

```r
table(mc <- max.col(swiss))  # mostly "1" and "5", 5 x "2" and once "4"
swiss[unique(print(mr <- max.col(t(swiss)))) , ]  # 3 33 45 45 33 6

set.seed(1)  # reproducible example:
(mm <- rbind(x = round(2*runif(12)),
             y = round(5*runif(12)),
             z = round(8*runif(12))))

## Not run:
x 1 1 1 2 0 2 2 1 1 0 0 0
y 3 2 4 2 4 5 2 4 5 1 3 1
z 2 3 0 3 7 3 4 5 4 1 7 5
## End(Not run)

# column indices of all row maxima:
str(lapply(1:3, function(i) which(mm[,i] == max(mm[,i])))
max.col(mm) ; max.col(mm)  # "random"
max.col(mm, "first")  # -> 4 6 5
max.col(mm, "last")  # -> 7 9 11
```

---

### mean

**Arithmetic Mean**

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```r
mean(x, ...)
```

Arguments

- `x`: An R object. Currently there are methods for numeric data frames, numeric vectors and dates. A complex vector is allowed for `trim = 0`, only.
- `trim`: the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed.
- `na.rm`: a logical value indicating whether NA values should be stripped before the computation proceeds.
- `...`: further arguments passed to or from other methods.

Value

For a data frame, a named vector with the appropriate method being applied column by column. If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed. If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.
Memory

References


See Also

weighted.mean, mean.POSIXct

Examples

```r
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
mean(USArrests, trim = 0.2)
```

---

**Memory**

**Memory Available for Data Storage**

### Description

Use command line options to control the memory available for R.

### Usage

```r
Rgui --min-vsize=vl --max-vsize=vu --min-nsize=nl --max-nsize=nu --max-ppsize=N
Rterm --min-vsize=vl --max-vsize=vu --min-nsize=nl --max-nsize=nu --max-ppsize=N
mem.limits(nsize = NA, vsize = NA)
```

### Arguments

- `vl, vu, vsize`
  - Heap memory in bytes.
- `nl, nu, nsize`
  - Number of cons cells.
- `N`
  - Number of nested PROTECT calls.

### Details

R has a variable-sized workspace (from version 1.2.0). There is now much less need to set memory options than previously, and most users will never need to set these. They are provided both as a way to control the overall memory usage (which can also be done using the option '--max-mem-size' on Windows), and since setting larger values of the minima will make R slightly more efficient on large tasks.

(On Windows the '--max-mem-size' option sets the maximum memory allocation: it has a minimum allowed value of 16M. This is intended to catch attempts to allocate excessive amounts of memory which may cause other processes to run out of resources. The default is the smaller of the amount of physical RAM in the machine and 1024Mb. See also `memory.limit`.)
To understand the options, one needs to know that R maintains separate areas for fixed and variable sized objects. The first of these is allocated as an array of "cons cells" (Lisp programmers will know what they are, others may think of them as the building blocks of the language itself, parse trees, etc.), and the second are thrown on a "heap" of "Vcells" of 8 bytes each. Effectively, the inputs vl and vu are rounded up to the next multiple of 8.

Each cons cell occupies 28 bytes on a 32-bit machine, (usually) 56 bytes on a 64-bit machine.

The '--nsize' options can be used to specify the number of cons cells and the '--vsize' options specify the size of the vector heap in bytes. Both options must be integers or integers followed by G, M, K, or k meaning Giga \(2^{30} = 1073741824\), Mega \(2^{20} = 1048576\), (computer) Kilo \(2^{10} = 1024\), or regular kilo (1000).

The '--min-' options set the minimal sizes for the number of cons cells and for the vector heap. These values are also the initial values, but thereafter R will grow or shrink the areas depending on usage, but never exceeding the limits set by the '--max-' options nor decreasing below the initial values.

The default values are currently minima of 350k cons cells, 6Mb of vector heap and no maxima (other than machine resources). The maxima can be changed during an R session by calling mem.limits. (If this is called with the default values, it reports the current settings.)

You can find out the current memory consumption (the heap and cons cells used as numbers and megabytes) by typing gc() at the R prompt. Note that following gcinfo(TRUE), automatic garbage collection always prints memory use statistics. Maxima will never be reduced below the current values for triggering garbage collection, and attempts to do so will be silently ignored.

The option '--ppsize' controls the maximum size of the pointer protection stack. This defaults to 50000, but can be increased to allow deep recursion or large and complicated calculations to be done. Note that parts of the garbage collection process goes through the full reserved pointer protection stack and hence becomes slower when the size is increased. Currently the maximum value accepted is 500000.

Value

mem.limits() returns an integer vector giving the current settings of the maxima, possibly NA.

See Also

An Introduction to R for more command-line options

Memory-limits for the design limitations.

gc for information on the garbage collector and total memory usage, object.size(a) for the (approximate) size of R object a. memory.profile for profiling the usage of cons cells.

memory.size to monitor total memory usage, memory.limit for the current limit.

Examples

# Start R with 10MB of heap memory and 500k cons cells, limit to # 100Mb and 1M cells
## Not run:
## Unix
R --min-vsize=10M --max-vsize=100M --min-nsize=500k --max-nsize=1M
## End(Not run)
Description

R holds objects it is using in memory. This help file documents the current design limitations on large objects: these differ between 32-bit and 64-bit builds of R.

Details

R holds all objects in memory, and there are limits based on the amount of memory that can be used by all objects:

- There may be limits on the size of the heap and the number of cons cells allowed – see Memory – but these are usually not imposed.
- There is a limit on the address space of a single process such as the R executable. This is 2Gb under 32-bit Windows unless both the OS and the R executable have been changed to allow more (up to 3Gb?). See the ‘rw-FAQ’ for how to do so.
- Under Windows, R does impose limits on the total memory allocation available to a single run, as the OS provides no way to do so. See memory.size and memory.limit.

Error messages beginning cannot allocate vector of size indicate a failure to obtain memory, either because the size exceeded the address-space limit for a process or, more likely, because the system was unable to provide the memory.

There are also limits on individual objects. On all versions of R, the maximum length (number of elements) of a vector is $2^{31} - 1 \approx 2 \times 10^9$, as lengths are stored as signed integers. In addition, the storage space cannot exceed the address limit, and if you try to exceed that limit, the error message begins cannot allocate vector of length. The number of characters in a character string is in theory only limited by the address space.

See Also

object.size(a) for the (approximate) size of R object a.

memory.profile  Profile the Usage of Cons Cells

Description

Lists the usage of the cons cells by SEXPREC type.

Usage

memory.profile()

Details

The current types and their uses are listed in the include file ‘Rinternals.h’.
merge

Value

A vector of counts, named by the types. See `typeof` for an explanation of types.

See Also

`gc` for the overall usage of cons cells.

Examples

memory.profile()

merge

Merge Two Data Frames

Description

Merge two data frames by common columns or row names, or do other versions of database “join” operations.

Usage

merge(x, y, ...)

## Default S3 method:
merge(x, y, ...)

## S3 method for class 'data.frame':
merge(x, y, by = intersect(names(x), names(y)),
      by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all,
      sort = TRUE, suffixes = c(".x",".y"), ...)

Arguments

x, y  
data frames, or objects to be coerced to one
by, by.x, by.y  
specifications of the common columns. See Details.
all  
logical; all=L is shorthand for all.x=L and all.y=L.
all.x  
logical; if TRUE, then extra rows will be added to the output, one for each row in x that has no matching row in y. These rows will have NAs in those columns that are usually filled with values from y. The default is FALSE, so that only rows with data from both x and y are included in the output.
all.y  
logical; analogous to all.x above.
sort  
logical. Should the results be sorted on the by columns?
suffixes  
character(2) specifying the suffixes to be used for making non-by names() unique.
...  
arguments to be passed to or from methods.
By default the data frames are merged on the columns with names they both have, but separate specifications of the columns can be given by `by.x` and `by.y`. Columns can be specified by name, number or by a logical vector: the name "row.names" or the number 0 specifies the row names. The rows in the two data frames that match on the specified columns are extracted, and joined together. If there is more than one match, all possible matches contribute one row each. If the `by.*` vectors are of length 0, the result, `r`, is the "Cartesian product" of `x` and `y`, i.e., `dim(r) = c(nrow(x) * nrow(y), ncol(x) + ncol(y))`. If `all.x` is true, all the non matching cases of `x` are appended to the result as well, with NA filled in the corresponding columns of `y`; analogously for `all.y`. If the remaining columns in the data frames have any common names, these have suffixes (".x" and ".y" by default) appended to make the names of the result unique.

A data frame. The rows are by default lexicographically sorted on the common columns, but are otherwise in the order in which they occurred in `y`. The columns are the common columns followed by the remaining columns in `x` and then those in `y`. If the matching involved row names, an extra column `Row.names` is added at the left, and in all cases the result has no special row names.

See Also

`data.frame`, `by`, `cbind`

Examples

```r
authors <- data.frame(
  surname = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil"),
  nationality = c("US", "Australia", "US", "UK", "Australia"),
  deceased = c("yes", rep("no", 4)))
books <- data.frame(
  name = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil", "R Core"),
  other.author = c(NA, "Ripley", NA, NA, NA, NA, "Venables & Smith"))

(m1 <- merge(authors, books, by.x = "surname", by.y = "name"))
(m2 <- merge(books, authors, by.x = "name", by.y = "surname"))
stopifnot(as.character(m1[,1]) == as.character(m2[,1]),
  all.equal(m1[, -1], m2[, -1][ names(m1)[-1] ],
  dim(merge(m1, m2, by = integer(0))) == c(36, 10))

## "R core" is missing from authors and appears only here :
merge(authors, books, by.x = "surname", by.y = "name", all = TRUE)
```
**Message**

Generate a diagnostic message from its arguments.

**Usage**

```r
message(..., domain = NULL)
suppressMessages(expr)
```

**Arguments**

- `...`: character vectors (which are pasted together with no separator), a condition object, or `NULL`.
- `domain`: see `gettext`. If `NA`, messages will not be translated.
- `expr`: expression to evaluate.

**Details**

`message` is used for generating “simple” diagnostic messages which are neither warnings nor errors, but nevertheless represented as conditions.

While the message is being processed, a `muffleMessage` restart is available.

`suppressMessages` evaluates its expression in a context that ignores all “simple” diagnostic messages.

**See Also**

`warning` and `stop` for generating warnings and errors; `conditions` for condition handling and recovery.

`gettext` for the mechanisms for the automated translation of text.

**Examples**

```r
message("ABC", "DEF")
suppressMessages(message("ABC"))
```

---

**Missing**

Does a Formal Argument have a Value?

**Description**

`missing` can be used to test whether a value was specified as an argument to a function.

**Usage**

```r
missing(x)
```
Arguments

x  a formal argument.

Details

`missing(x)` is only reliable if `x` has not been altered since entering the function: in particular it will always be false after `x <- match.arg(x)`.

The example shows how a plotting function can be written to work with either a pair of vectors giving `x` and `y` coordinates of points to be plotted or a single vector giving `y` values to be plotted against their indexes.

Currently `missing` can only be used in the immediate body of the function that defines the argument, not in the body of a nested function or a local call. This may change in the future.

References


See Also

`substitute` for argument expression; `NA` for “missing values” in data.

Examples

```r
myplot <- function(x,y) {
  if(missing(y)) {
    y <- x
    x <- 1:length(y)
  }
  plot(x,y)
}
```

---

**mode**

The (Storage) Mode of an Object

Description

Get or set the type or storage mode of an object.

Usage

```r
mode(x)
mode(x) <- value
storage.mode(x)
storage.mode(x) <- value
```

Arguments

- `x`  any R object.
- `value`  a character string giving the desired (storage) mode of the object.
Both `mode` and `storage.mode` return a character string giving the (storage) mode of the object — often the same — both relying on the output of `typeof(x)`, see the example below.

The two assignment versions are currently identical. Both `mode(x) <- "newmode"` and `storage.mode(x) <- "newmode"` change the mode or storage mode of object `x` to `newmode`. This is only supported if there is an appropriate `as.newmode` function, for example "logical", "integer", "double", "complex", "raw", "character", "list", "expression", "name", "symbol" and "function".

As storage mode "single" is only a pseudo-mode in R, it will not be reported by `mode` or `storage.mode`: use `attr(object, "Csingle")` to examine this. However, the assignment versions can be used to set the mode to "single", which sets the real mode to "double" and the "Csingle" attribute to `TRUE`. Setting any other mode will remove this attribute.

Note (in the examples below) that some calls have mode "(" which is S compatible.

### Mode names

Modes have the same set of names as types (see `typeof`) except that

- types "integer" and "double" are returned as "numeric".
- types "special" and "builtin" are returned as "function".
- type "symbol" is called mode "name".
- type "language" is returned as "(" or "call".

### References


### See Also

`typeof` for the R-internal “mode”, `attributes`.

### Examples

```r
sapply(options(), mode)

cex3 <- c("NULL","1","1:1","1i","list(1)","data.frame(x=1)", "pairlist(pi)",
"c","lm", "formals(lm)[[1]]", "formals(lm)[[2]]", 
"y-x","expression(l1)[[1]]", "(y-x)[[1]]", ",expression(x <- pi)[[1]][[1]][[1]]")
lex3 <- sapply(cex3, function(x) eval(parse(text=x)))
mex3 <- t(sapply(lex3, function(x) c(typeof(x), storage.mode(x), mode(x))))
dimnames(mex3) <- list(cex3, c("typeof(.)","storage.mode(.)","mode(.)"))
mex3

## This also makes a local copy of 'pi':
storage.mode(pi) <- "complex"
storage.mode(pi)
rm(pi)
```
Description

NA is a logical constant of length 1 which contains a missing value indicator. NA can be freely coerced to any other vector type except raw.

The generic function is.na indicates which elements are missing.

The generic function is.na<- sets elements to NA.

Usage

is.na(x)

## S3 method for class 'data.frame':
is.na(x)

is.na(x) <- value

Arguments

x an R object to be tested.
value a suitable index vector for use with x.

Details

The NA of character type is distinct from the string "NA". Programmers who need to specify an explicit string NA should use as.character(NA) rather than "NA", or set elements to NA using is.na<-.

is.na(x) works elementwise when x is a list. It is generic: you can write methods to handle specific classes of objects, see InternalMethods.

Function is.na<- may provide a safer way to set missingness. It behaves differently for factors, for example.

Value

The default method for is.na returns a logical vector of the same "form" as its argument x, containing TRUE for those elements marked NA or NaN (!) and FALSE otherwise. dim, dimnames and names attributes are preserved.

The method is.na.data.frame returns a logical matrix with the same dimensions as the data frame, and with dimnames taken from the row and column names of the data frame.

References


See Also

NaN, is.nan, etc., and the utility function complete.cases.
na.action, na.omit, na.fail on how methods can be tuned to deal with missing values.

Examples

```r
is.na(c(1, NA)) #> FALSE TRUE
is.na(paste(c(1, NA))) #> FALSE FALSE
```

Description

as.symbol coerces its argument to be a symbol, or equivalently, a name. The argument must be of mode "character". as.name is an alias for as.symbol.

is.symbol (and is.name equivalently) returns TRUE or FALSE depending on whether its argument is a symbol (i.e., name) or not.

Usage

```r
as.symbol(x)
is.symbol(y)
as.name(x)
is.name(y)
```

Arguments

```r
x, y          objects to be coerced or tested.
```

Details

is.symbol is generic: you can write methods to handle specific classes of objects, see Internal-Methods.

Note

The term “symbol” is from the LISP background of R, whereas “name” has been the standard S term for this.

References


See Also
call, is.language. For the internal object mode, typeof.
### Examples

```r
an <- as.name("arrg")
is.name(an) # TRUE
mode(an) # name
typeof(an) # symbol
```

---

### The Names Attribute of an Object

#### Description

Functions to get or set the names of an object.

#### Usage

```r
names(x)
names(x) <- value
```

#### Arguments

- `x`: an R object.
- `value`: a character vector of up to the same length as `x`, or `NULL`.

#### Details

`names` is a generic accessor function, and `names<-` is a generic replacement function. The default methods get and set the "names" attribute of a vector or list.

If `value` is shorter than `x`, it is extended by character NAs to the length of `x`.

It is possible to update just part of the names attribute via the general rules: see the examples. This works because the expression there is evaluated as `z <- "names<-"(z, "[<-"(names(z), 3, "c2")").`

#### Value

For `names`, `NULL` or a character vector of the same length as `x`.

For `names<-`, the updated object. (Note that the value of `names(x) <- value` is that of the assignment, `value`, not the return value from the left-hand side.)

#### References

nargs

The Number of Arguments to a Function

Examples

# print the names attribute of the islands data set
names(islands)

# remove the names attribute
names(islands) <- NULL
islands
rm(islands) # remove the copy made

z <- list(a=1, b="c", c=1:3)
names(z)
# change just the name of the third element.
names(z)[3] <- "c2"
z

z <- 1:3
names(z)
## assign just one name
names(z)[2] <- "b"
z

nargs

Description

When used inside a function body, nargs returns the number of arguments supplied to that function, including positional arguments left blank.

Usage

nargs()

References


See Also

args, formals and sys.call.

Examples

tst <- function(a, b = 3, ...) {nargs()}
tst() # 0
tst(clicketyclack) # 1 (even non-existing)
tst(c1, a2, rr3) # 3

foo <- function(x, y, z, w) {
  cat("call was", deparse(match.call()), "\n")
nargs()
}
foo()   # 0
foo(,,3) # 3
foo(z=3) # 1, even though this is the same call

nargs()# not really meaningful

---

**nchar**

*Count the Number of Characters (Bytes)*

**Description**

`nchar` takes a character vector as an argument and returns a vector whose elements contain the sizes of the corresponding elements of `x`.

**Usage**

```r
nchar(x, type = c("bytes", "chars", "width"))
```

**Arguments**

- **x**
  - character vector, or a vector to be coerced to a character vector.
- **type**
  - character string: partial matching is allowed. See Details.

**Details**

The ‘size’ of a character string can be measured in one of three ways:

- **bytes** The number of bytes needed to store the string (plus in C a final terminator which is not counted).
- **chars** The number of human-readable characters.
- **width** The number of columns `cat` will use to print the string in a monospaced font. The same as `chars` if this cannot be calculated (which is currently common).

These will often be the same, and always will be in single-byte locales. There will be differences between the first two with multibyte character sequences, e.g. in UTF-8 locales. If the byte stream contains embedded `null` bytes, `type = "bytes"` looks at all the bytes whereas the other two types look only at the string as printed by `cat`, up to the first `null` byte.

The internal equivalent of the default method of `as.character` is performed on `x`. If you want to operate on non-vector objects passing them through `deparse` first will be required.

**Value**

An integer vector giving the size of each string, currently always 2 for missing values (for `NA`).

Not all platforms will return a non-missing value for `type="width"`.

If the string is invalid in a multi-byte character set such as UTF-8, the number of characters and the width will be `NA`. Otherwise the number of characters will be non-negative, so `!is.na(nchar(x, "chars"))` is a test of validity.
Note

This does not by default give the number of characters that will be used to print() the string, although it was documented to do so up to R 2.0.1. Use encodeString to find the characters used to print the string.

As from R 2.1.0 embedded null bytes are included in the byte count (but not the final null): previously the count stopped immediately before the first null.

References


See Also

strwidth giving width of strings for plotting: paste, substr, strsplit

Examples

x <- c("asfef","qwerty","yuiop","b","stuff.blah.yech")
nchar(x)
# 5 6 6 1 15

nchar(deparse(mean))
# 18 17

nlevels

The Number of Levels of a Factor

Description

Return the number of levels which its argument has.

Usage

nlevels(x)

Arguments

x an object, usually a factor.

Details

If the argument is not a factor, NA is returned.

The actual factor levels (if they exist) can be obtained with the levels function.

Examples

nlevels(gl(3,7)) # = 3
Description

Print character strings without quotes.

Usage

noquote(obj)

## S3 method for class 'noquote':
print(x, ...)

## S3 method for class 'noquote':
c(..., recursive = FALSE)

Arguments

obj 
any R object, typically a vector of character strings.
x 
an object of class "noquote".
... 
further options passed to next methods, such as print.
recursive 
for compatibility with the generic c function.

Details

noquote returns its argument as an object of class "noquote". There is a method for c() and subscript method ("[.noquote") which ensures that the class is not lost by subsetting. The print method (print.noquote) prints character strings without quotes ("...").

These functions exist both as utilities and as an example of using (S3) class and object orientation.

Author(s)

Martin Maechler (maechler@stat.math.ethz.ch)

See Also

methods, class, print.

Examples

letters
nql <- noquote(letters)
nql
nql[1:4] <- "oh"
nql[1:12]

cmp.logical <- function(log.v)
{
    ## Purpose: compact printing of logicals
    log.v <- as.logical(log.v)
Description

In order to pinpoint missing functionality, the R core team uses these functions for missing R functions and not yet used arguments of existing R functions (which are typically there for compatibility purposes).
You are very welcome to contribute your code ...

Usage

.NotYetImplemented()
.NotYetUsed(arg, error = TRUE)

Arguments

arg an argument of a function that is not yet used.
error a logical. If TRUE, an error is signalled; if FALSE; only a warning is given.

See Also

the contrary, Deprecated and Defunct for outdated code.

Examples

require(graphics)
require(stats)
plot.mlm # to see how the "NotYetImplemented"
# reference is made automagically
try(plot.mlm())
barplot(1:5, inside = TRUE) # 'inside' is not yet used

nrow

The Number of Rows/Columns of an Array

Description

nrow and ncol return the number of rows or columns present in x. NCOL and NROW do the same treating a vector as 1-column matrix.

Usage

nrow(x)
ncol(x)
NCOL(x)
NROW(x)
**Arguments**

- `x`: a vector, array or data frame

**Value**

an **integer** of length 1 or **NULL**.

**References**


**See Also**

`dim` which returns *all* dimensions; `array, matrix`.

**Examples**

```r
ma <- matrix(1:12, 3, 4)
nrow(ma) # 3
ncol(ma) # 4

ncol(array(1:24, dim = 2:4)) # 3, the second dimension
NCOL(1:12) # 1
NROW(1:12) # 12
```

---

**ns-dblcolon**

**Double Colon and Triple Colon Operators**

**Description**

Accessing exported and internal variables in a name space.

**Usage**

```r
pkg::name
pkg:::name
```

**Arguments**

- `pkg`: package name symbol or literal character string.
- `name`: variable name symbol or literal character string.

**Details**

The expression `pkg:::name` returns the value of the exported variable `name` in package `pkg` if the package has a name space. The expression `pkg::name` returns the value of the internal variable `name` in package `pkg` if the package has a name space. The package will be loaded if it was not loaded already before the call. Assignment into name spaces is not supported.

**See Also**

`get` to access an object masked by another of the same name.
Examples

base::log
base::"+

Description

Packages with name spaces can supply functions to be called when loaded, attached or unloaded.

Usage

.onLoad(libname, pkgname)
.onAttach(libname, pkgname)
.onUnload(libpath)

Arguments

libname a character string giving the library directory where the package defining the namespace was found.
 pkgname a character string giving the name of the package, including the version number if the package was installed with --with-package-versions.
 libpath a character string giving the complete path to the package.

Details

These functions apply only to packages with name spaces.

After loading, `loadNamespace` looks for a hook function named `.onLoad` and runs it before sealing the namespace and processing exports.

If a name space is unloaded (via `unloadNamespace`), a hook function `.onUnload` is run before final unloading.

Note that the code in `.onLoad` and `.onUnload` is run without the package being on the search path, but (unless circumvented) lexical scope will make objects in the namespace and its imports visible. (Do not use the double colon operator in `.onLoad` as exports have not been processed at the point it is run.)

When the package is attached (via `library`), the hook function `.onAttach` is looked for and if found is called after the exported functions are attached and before the package environment is sealed. This is less likely to be useful than `.onLoad`, which should be seen as the analogue of `.First.lib` (which is only used for packages without a name space).

`.onLoad`, `.onUnload` and `.onAttach` are looked for as internal variables in the name space and should not be exported.

If a function `.Last.lib` is visible in the package, it will be called when the package is detached: this does need to be exported.

Anything needed for the functioning of the name space should be handled at load/unload times by the `.onLoad` and `.onUnload` hooks. For example, shared libraries can be loaded (unless done by a `useDynLib` directive in the `NAMESPACE` file) and initialized in `.onLoad` and unloaded in
.onUnload. Use .onAttach only for actions that are needed only when the package becomes visible to the user, for example a start-up message.

If a package was installed with --with-package-versions, the pkgname supplied will be something like tree_1.0-16.

See Also

setHook shows how users can set hooks on the same events.

ns-load Loading and Unloading Name Spaces

Description

Functions to load and unload namespaces.

Usage

attachNamespace(ns, pos = 2, dataPath = NULL)
loadNamespace(package, lib.loc = NULL,
    keep.source = getOption("keep.source.pkgs"),
    partial = FALSE, declarativeOnly = FALSE)
loadedNamespaces()
unloadNamespace(ns)

Arguments

ns    string or namespace object.
pos    integer specifying position to attach.
dataPath    path to directory containing a database of datasets to be lazy-loaded into the attached environment.
package    string naming the package/name space to load.
lib.loc    character vector specifying library search path.
keep.source    logical specifying whether to retain source. This applies only to the specified name space, and not to other name spaces which might be loaded to satisfy imports.
For more details see this argument to library.
partial    logical; if true, stop just after loading code.
declarativeOnly    logical; disables .Import, etc, if true.

Details

The functions loadNamespace and attachNamespace are usually called implicitly when library is used to load a name space and any imports needed. However it may be useful to call these functions directly at times.

loadNamespace loads the specified name space and registers it in an internal data base. A request to load a name space that is already loaded has no effect. The arguments have the same meaning as the corresponding arguments to library. After loading, loadNamespace looks
for a hook function named `.onLoad` as an internal variable in the name space (it should not be exported). This function is called with the same arguments as `.First.lib`. Partial loading is used to support installation with the `--save` and `--lazy` options.

`loadNamespace` does not attach the name space it loads to the search path. `attachNamespace` can be used to attach a frame containing the exported values of a name space to the search path. The hook function `.onAttach` is run after the name space exports are attached.

`loadedNamespaces` returns a character vector of the names of the loaded name spaces.

`unloadNamespace` can be used to force a name space to be unloaded. An error is signaled if the name space is imported by other loaded name spaces. If defined, a hook function `code(.onUnload)` is run before removing the name space from the internal registry. `unloadNamespace` will first `detach` a package of the same name if one is on the path, thereby running a `.Last.lib` function in the package if one is exported.

Author(s)

Luke Tierney

---

### ns-topenv

**Top Level Environment**

#### Description

Finding the top level environment.

#### Usage

```r
topenv(envir = parent.frame(),
      matchThisEnv = getOption("topLevelEnvironment"))
```

#### Arguments

- `envir` environment.
- `matchThisEnv` return this environment, if it matches before any other criterion is satisfied. The default, the option “topLevelEnvironment”, is set by `sys.source`, which treats a specific environment as the top level environment. Supplying the argument as NULL means it will never match.

#### Details

`topenv` returns the first top level environment found when searching `envir` and its parent environments. An environment is considered top level if it is the internal environment of a name space, a package environment in the search path, or `.GlobalEnv`.

#### Examples

```r
topenv(.GlobalEnv)
topenv(new.env())
```
**NULL**  
*The Null Object*

**Description**

NULL represents the null object in R. NULL is used mainly to represent the lists with zero length, and is often returned by expressions and functions whose value is undefined.

as.null ignores its argument and returns the value NULL.

is.null returns TRUE if its argument is NULL and FALSE otherwise.

**Usage**

```r
NULL
as.null(x, ...)
is.null(x)
```

**Arguments**

- `x` an object to be tested or coerced.
- `...` ignored.

**Details**

is.null is generic: you can write methods to handle specific classes of objects, see InternalMethods.

**References**


**Examples**

```r
is.null(list()) # FALSE (on purpose!)
is.null(integer(0))# F
is.null(logical(0))# F
as.null(list(a=1,b='c'))
```

---

**numeric**  
*Numeric Vectors*

**Description**

Creates or tests for objects of type "numeric".

**Usage**

```r
numeric(length = 0)
as.numeric(x, ...)
is.numeric(x)
```
Arguments

- **length**: desired length.
- **x**: object to be coerced or tested.
- **...**: further arguments passed to or from other methods.

Details

- `as.numeric` is a generic function, but methods must be written for `as.double`, which it calls.
- `is.numeric` is generic: you can write methods to handle specific classes of objects, see Internal-Methods.
  
Note that factors are false for `is.numeric` since there is a "factor" method.

Value

- `numeric` creates a real vector of the specified length. The elements of the vector are all equal to 0.
- `as.numeric` attempts to coerce its argument to a numeric type (either integer or real).
- `as.numeric` for factors yields the codes underlying the factor levels, not the numeric representation of the labels.
- `is.numeric` returns TRUE if its argument is of mode "numeric" (type "double" or type "integer") and not a factor, and FALSE otherwise.

Note

- R has no single precision data type. All real numbers are stored in double precision format.

References


Examples

```r
as.numeric(c("-0.1","2.7","B")) # (-0.1, 2.7, NA) + warning
as.numeric(factor(5:10))
```

Description

How R parses numeric constants.
octmode

Details

R parses numeric constants in its input in a very similar way to C floating-point constants. Inf and NaN are numeric constants. All other numeric constants start with a digit or period. Hexadecimal constants start with 0x or 0X followed by a non-empty sequence from 0-9 a-f A-F which is interpreted as a hexadecimal number. Decimal constants consists of a nonempty sequence of digits possibly containing a period (the decimal point), optionally followed by a decimal exponent. A decimal exponent consists of an E or e followed by an optional plus or minus sign followed by a non-empty sequence of digits, and indicates multiplication by a power of ten. A numeric constant immediately followed by i is regarded as an imaginary complex number. Only the ASCII digits 0–9 are recognized as digits, even in languages which have other representations of digits. The ‘decimal separator’ is always a period and never a comma. Note that a leading plus or minus is not part of numeric constant but a unary operator applied to the constant.

See Also

Syntax, Quotes for the parsing of character constants,

octmode

Display Numbers in Octal

Description

Convert or print integers in octal format, with as many digits as are needed to display the largest, using leading zeroes as necessary.

Usage

## S3 method for class 'octmode':
as.character(x, ...)

## S3 method for class 'octmode':
format(x, ...)

## S3 method for class 'octmode':
print(x, ...)

Arguments

x An object inheriting from class "octmode".

... further arguments passed to or from other methods.

Details

Class "octmode" consists of integer vectors with that class attribute, used merely to ensure that they are printed in octal notation, specifically for Unix-like file permissions such as 755.
See Also

These are auxiliary functions for file.info

---

**on.exit**

*Function Exit Code*

**Description**

`on.exit` records the expression given as its argument as needing to be executed when the current function exits (either naturally or as the result of an error). This is useful for resetting graphical parameters or performing other cleanup actions.

If no expression is provided, i.e., the call is `on.exit()`, then the current `on.exit` code is removed.

**Usage**

```r
on.exit(expr, add = FALSE)
```

**Arguments**

- `expr`: an expression to be executed.
- `add`: if TRUE, add `expr` to be executed after any previously set expressions.

**References**


**See Also**

`sys.on.exit` to see the current expression.

**Examples**

```r
opar <- par(mai = c(1,1,1,1))
on.exit(par(opar))
```
Description

Allow the user to set and examine a variety of global “options” which affect the way in which R computes and displays its results.

Usage

options(...)

getOption(x)

.Options

Arguments

any options can be defined, using name = value or by passing a list of such tagged values. However, only the ones below are used in “base R”. Further, options('name') == options()['name'], see the example.

x a character string holding an option name.

Details

Invoking options() with no arguments returns a list with the current values of the options. Note that not all options listed below are set initially. To access the value of a single option, one should use getOption("width"), e.g., rather than options("width") which is a list of length one.

.Options also always contains the options() list (as a pairlist, unsorted), for S compatibility. Assigning to it will make a local copy and not change the original.

Value

For getOption, the current value set for option x, or NULL if the option is unset.

For options(), a list of all set options sorted by name. For options(name), a list of length one containing the set value, or NULL if it is unset. For uses setting one or more options, a list with the previous values of the options changed (returned invisibly).

Options used in base R

.add.smooth: typically logical, defaulting to TRUE. Could also be set to an integer for specifying how many (simulated) smooths should be added. This is currently only used by plot.lm.

.check.bounds: logical, defaulting to FALSE. If true, a warning is produced whenever a “generalized vector” (atomic or list) is extended, by something like x <- 1:3; x[5] <- 6.

.continue: a string setting the prompt used for lines which continue over one line.
**defaultPackages**: the packages that are attached by default when R starts up. Initially set from value of the environment variable R_DEFAULT_PACKAGES, or if that is unset to c("datasets", "utils", "grDevices", "graphics", "stats", "methods"). (Set R_DEFAULT_PACKAGES to NULL or a comma-separated list of package names.) A call to options should be in your `.Rprofile` file to ensure that the change takes effect before the base package is initialized (see Startup).

digits: controls the number of digits to print when printing numeric values. It is a suggestion only. Valid values are 1...22 with default 7. See print.default.

download.file.method: Method to be used for download.file. Currently download methods "internal", "wget" and "lynx" are available. There is no default for this option, when method = "auto" is chosen: see download.file.

echo: logical. Only used in non-interactive mode, when it controls whether input is echoed. Command-line option --slave sets this initially to FALSE.

encoding: The name of an encoding, default "native.enc"). See connections.

error: either a function or an expression governing the handling of non-catastrophic errors such as those generated by stop as well as by signals and internally detected errors. If the option is a function, a call to that function, with no arguments, is generated as the expression. The default value is NULL: see stop for the behaviour in that case. The functions dump.frames and recover provide alternatives that allow post-mortem debugging.

expressions: sets a limit on the number of nested expressions that will be evaluated. Valid values are 25...500000 with default 5000. If you increase it, you may also want to start R with a larger stack; see -max-ppsize in Memory.

keep.source: When TRUE, the source code for functions (newly defined or loaded) is stored in their "source" attribute (see attr) allowing comments to be kept in the right places. The default is interactive(), i.e., TRUE for interactive use.

keep.source.pkgs: As for keep.source, for functions in packages loaded by library or require. Defaults to FALSE unless the environment variable R_KEEP_PKG_SOURCE is set to yes.

Note this does not apply to packages using lazy-loading or saved images. Whether they have kept source is determined when they are installed (and is almost certainly false).

mailer: default mailer used by bug.report(). Can be "none".

max.print: integer, defaulting to 10000. print or show methods can make use of this option, to limit the amount of information that is printed, typically to something in the order max.print lines. This is not yet used in base R.

OutDec: one-character string. The character to be used as the decimal point in output conversions, that is in printing, plotting and as.character but not deparsing.

pager: the (stand-alone) program used for displaying text files on R’s console, also used by file.show and sometimes help. Defaults to "internal", which uses a pager similar to the GUI console. Another possibility is "console" to use the console itself.

papersize: the default paper format used by postscript: set by environment variable R_PAPERSIZE when R is started: if that is unset or invalid it defaults to "a4", or "letter" in US and Canadian locales.

printcmd: the command used by postscript for printing; set by environment variable R_PRINTCMD when R is started. This should be a command that expects either input to be piped to `stdin` or to be given a single filename argument.

prompt: a string, used for R’s prompt; should usually end in a blank (" ").
**options**

save.defaults, save.image.defaults: see save.

scipen: integer. A penalty to be applied when deciding to print numeric values in fixed or exponential notation. Positive values bias towards fixed and negative towards scientific notation: fixed notation will be preferred unless it is more than scipen digits wider.

show.error.messages: a logical. Should error messages be printed? Intended for use with try or a user-installed error handler.

texi2dvi: used by the unexported function texi2dvi in namespace tools.

timeout: integer. The timeout for some Internet operations, in seconds. Default 60 seconds. See download.file and connections.

topLevelEnvironment: see topenv and sys.source.

verbose: logical. Should R report extra information on progress? Set to TRUE by the command-line option ‘-verbose’.

warn: sets the handling of warning messages. If warn is negative all warnings are ignored. If warn is zero (the default) warnings are stored until the top–level function returns. If fewer than 10 warnings were signalled they will be printed otherwise a message saying how many (max 50) were signalled. A top–level variable called last.warning is created and can be viewed through the function warnings. If warn is one, warnings are printed as they occur. If warn is two or larger all warnings are turned into errors.

warning.expression: an R code expression to be called if a warning is generated, replacing the standard message. If non-null it is called irrespective of the value of option warn.

warnings.length: sets the truncation limit for error and warning messages. A non-negative integer, with allowed values 100...8192, default 1000.

width: controls the number of characters on a line. You may want to change this if you re-size the window that R is running in. Valid values are 10...10000 with default normally 80. (The limits on valid values are in file ’Print.h’ and can be changed by re-compiling R.)

The ‘factory-fresh’ default settings of some of these options are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.smooth</td>
<td>TRUE</td>
</tr>
<tr>
<td>check.bounds</td>
<td>FALSE</td>
</tr>
<tr>
<td>continue</td>
<td>&quot;+ &quot;</td>
</tr>
<tr>
<td>digits</td>
<td>7</td>
</tr>
<tr>
<td>echo</td>
<td>TRUE</td>
</tr>
<tr>
<td>encoding</td>
<td>&quot;native.enc&quot;</td>
</tr>
<tr>
<td>error</td>
<td>NULL</td>
</tr>
<tr>
<td>expressions</td>
<td>5000</td>
</tr>
<tr>
<td>keep.source</td>
<td>interactive()</td>
</tr>
<tr>
<td>keep.source.pkgs</td>
<td>FALSE</td>
</tr>
<tr>
<td>max.print</td>
<td>1000</td>
</tr>
<tr>
<td>OutDec</td>
<td>&quot;. &quot;</td>
</tr>
<tr>
<td>prompt</td>
<td>&quot; &gt; &quot;</td>
</tr>
<tr>
<td>scipen</td>
<td>0</td>
</tr>
<tr>
<td>show.error.messages</td>
<td>TRUE</td>
</tr>
<tr>
<td>timeout</td>
<td>60</td>
</tr>
<tr>
<td>verbose</td>
<td>FALSE</td>
</tr>
<tr>
<td>warn</td>
<td>0</td>
</tr>
<tr>
<td>warnings.length</td>
<td>1000</td>
</tr>
<tr>
<td>width</td>
<td>80</td>
</tr>
</tbody>
</table>

Others are set from environment variables or are platform-dependent.
Options set in package grDevices

These will be set when package grDevices (or its name space) is loaded if not already set.

device: a character string giving the default device for that session. This defaults to the normal screen device (e.g., x11, windows or quartz) for an interactive session, and postscript in batch use or if a screen is not available.

locatorBell: logical. Should selection in locator and identify be confirmed by a bell. Default TRUE. Honoured at least on X11 and windows devices.

gamma: double. The default value of gamma for windows devices, defaulting to 1 if unset (the default).

graphics.record: logical. The default value of record for windows devices. Default FALSE.

windowsBuffered: logical. Should windows devices be double-buffered?. Default TRUE.

windowsTimeout: integer vector of length 2 representing two times in milliseconds. These control the double-buffering of windows devices when that is enabled: the first is the delay after plotting finishes (default 100) and the second is the update interval during continuous plotting (default 500).

xpinch, ypinch: double. Pixels per inch, horizontally and vertically. If unset (the default), taken from Windows.

Options set in package stats

These will be set when package stats (or its name space) is loaded if not already set.

contrasts: the default contrasts used in model fitting such as with aov or lm. A character vector of length two, the first giving the function to be used with unordered factors and the second the function to be used with ordered factors. By default the elements are named c("unordered", "ordered"), but the names are unused.

na.action: the name of a function for treating missing values (NA’s) for certain situations.

show.coef.Pvalues: logical, affecting whether P values are printed in summary tables of coefficients. See printCoefmat.

show.signif.stars: logical, should stars be printed on summary tables of coefficients? See printCoefmat.

ts.eps: the relative tolerance for certain time series (ts) computations. Default 1e−05.

ts.S.compat: logical. Used to select S compatibility for plotting time-series spectra. See the description of argument log in plot.spec.

Options set in package utils

These will be set when package utils (or its name space) is loaded if not already set.

browser: default HTML browser used by help.start () on UNIX, or a non-default browser on Windows.

de.cellwidth: integer: the cell widths (number of characters) to be used in the data editor dataentry. If this is unset (the default), 0, negative or NA, variable cell widths are used.

editor: sets the default text editor, e.g., for edit. Set from the environment variable VISUAL on UNIX.

help.try.all.packages: default for an argument of help.
**internet.info**: The minimum level of information to be printed on URL downloads etc. Default is 2, for failure causes. Set to 1 or 0 to get more information.

**pkgType**: The default type of packages to be downloaded and installed — see `install.packages`. Possible values are "win.binary" (the default) and "source".

**repos**: URLs of the repositories for use by `update.packages`. Defaults to `c(CRAN="@CRAN@")`, a value that causes some utilities to prompt for a CRAN mirror. To avoid this do set the CRAN mirror, by something like `local({r <- getOption("repos"); r["CRAN"] <- "http://my.local.cran"; options(repos=r)})`. Note that you can add more repositories (Bioconductor and Omegahat, notably) using `setRepositories()`.

**SweaveHooks, SweaveSyntax**: see Sweave.

**unzip**: the command used for unzipping help files. Defaults to "internal" when the internal unzip code is used.

**Options used on Windows only**

**warn.FPU**: logical, by default undefined. If true, a warning is produced whenever `dyn.load` repairs the control word damaged by a buggy DLL.

**References**


**Examples**

```r
options() # printing all current options
op <- options(); str(op) # nicer printing

options('width')[[1]] = options()$width # the latter needs more memory

options(digits = 20)

pi

# set the editor, and save previous value
old.o <- options(editor = "nedit")

old.o

options(check.bounds = TRUE, warn = 1)
x <- NULL; x[4] <- "yes" # gives a warning

options(digits=5)
print(1e5)

options(scipen=3); print(1e5)

options(op) # reset (all) initial options

options('digits')

## Not run:
## set contrast handling to be like S
options(contrasts = c("contr.helmert", "contr.poly"))
## End(Not run)
## Not run:
## on error, terminate the R session with error status 66
```
options(error = quote(q("no", status=66, runLast=FALSE)))
stop("test it")
## End(Not run)
## Not run:
## Set error actions for debugging:
## enter browser on error, see ?recover:
options(error = recover)
## allows to call debugger() afterwards, see ?debugger:
options(error = dump.frames)
## A possible setting for non-interactive sessions
options(error = quote({dump.frames(to.file=TRUE); q()}))
## End(Not run)

order

Ordering Permutation

Description

order returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments. sort.list is the same, using only one argument. See the examples for how to use these functions to sort data frames, etc.

Usage

order(..., na.last = TRUE, decreasing = FALSE)

sort.list(x, partial = NULL, na.last = TRUE, decreasing = FALSE, 
method = c("shell", "quick", "radix"))

Arguments

... a sequence of numeric, complex, character or logical vectors, all of the same length.
x a vector.
partial vector of indices for partial sorting.
decreasing logical. Should the sort order be increasing or decreasing?
na.last for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
method the method to be used: partial matches are allowed.

Details

In the case of ties in the first vector, values in the second are used to break the ties. If the values are still tied, values in the later arguments are used to break the tie (see the first example). The sort used is stable (except for method = "quick"), so any unresolved ties will be left in their original ordering.

The default method for sort.list is a good compromise. Method "quick" is only supported for numeric x with na.last=NA, and is not stable, but will be faster for long vectors. Method "radix" is only implemented for integer x with a range of less than 100,000. For such x it is very fast (and stable), and hence is ideal for sorting factors.

partial is supplied for compatibility with other implementations of S, but no other values are accepted and ordering is always complete.
References


See Also

`sort` and `rank`.

Examples

```r
(ii <- order(x <- c(1,1,3:1,1:4,3), y <- c(9,9:1), z <-c(2,1:9)))
## 6 5 2 1 7 4 10 8 3 9
rbind(x,y,z)[,ii] # shows the reordering (ties via 2nd & 3rd arg)

## Suppose we wanted descending order on y. A simple solution is
rbind(x,y,z)[, order(x, -y, z)]
## For character vectors we can make use of rank:
cy <- as.character(y)
rbind(x,y,z)[, order(x, -rank(y), z)]

## Sorting data frames:
dd <- transform(data.frame(x,y,z),
                 z = factor(z, labels=LETTERS[9:1]))
## Either as above {for factor 'z': using internal coding}:
dd[ order(x, -y, z) ,]
## or along 1st column, ties along 2nd, ... *arbitrary* no. {columns}:
dd[ do.call(order, dd) ,]

set.seed(1)# reproducible example:
d4 <- data.frame(x = round( rnorm(100)), y = round(10*runif(100)),
                 z = round( 8*runorm(100)), u = round(50*runif(100)))
(d4s <- d4[ do.call(order, d4) ,])
(i <- which(diff(d4s[,3]) == 0))# in 2 places, needed 3 cols to break ties:
d4s[ rbind(i,i+1), ]

## rearrange matched vectors so that the first is in ascending order
x <- c(5:1, 6:8, 12:9)
y <- (x - 5)^2
o <- order(x)
rbind(x[o], y[o])

## tests of na.last
a <- c(4, 3, 2, NA, 1)
b <- c(4, NA, 2, 7, 1)
z <- cbind(a, b)
(o <- order(a, b)); z[o, ]
(o <- order(a, b, na.last = FALSE)); z[o, ]
(o <- order(a, b, na.last = NA)); z[o, ]

## Not run:
## speed examples for long vectors: timings are immediately after gc()
x <- factor(sample(letters, 1e6, replace=TRUE))
system.time(o <- sort.list(x)) # 4 secs
stopifnot(!is.unsorted(x[o]))
system.time(o <- sort.list(x, method="quick", na.last=NA)) # 0.4 sec
stopifnot(!is.unsorted(x[o]))
```
outer

Outer Product of Arrays

Description

The outer product of the arrays X and Y is the array A with dimension c(dim(X), dim(Y)) where element A[c(arrayindex.x, arrayindex.y)] = FUN(X[arrayindex.x], Y[arrayindex.y], ...).

Usage

outer(X, Y, FUN=" * ", ...)
X %o% Y

Arguments

X A vector or array.
Y A vector or array.
FUN a function to use on the outer products, it may be a quoted string.
... optional arguments to be passed to FUN.

Details

FUN must be a function (or the name of it) which expects at least two arguments and which operates elementwise on arrays.

Where they exist, the [dim]names of X and Y will be preserved.
%o% is an alias for outer (where FUN cannot be changed from " * ").

Author(s)

Jonathan Rougier

References


See Also

% * % for usual (inner) matrix vector multiplication; kronecker which is based on outer.
Examples

```r
x <- 1:9; names(x) <- x
# Multiplication & Power Tables
x %o% x
y <- 2:8; names(y) <- paste(y,":",sep="")
outer(y, x, "+")

outer(month.abb, 1999:2003, FUN = "paste")

## three way multiplication table:
x %o% x %o% y[1:3]
```

package-version

Package versions

Description

A simple S3 class for representing package versions, and associated methods.

Usage

```r
package_version(x, strict = TRUE)
getRversion()
```

Arguments

- `x`: a character vector with package version strings.
- `strict`: a logical indicating whether invalid package versions should result in an error (default) or not.

Details

R (package) versions are sequences of at least two non-negative integers, usually (e.g., in package `DESCRIPTION` files) represented as character strings with the elements of the sequence concatenated and separated by single '.' or '-' characters.

`package_version` creates a representation from such strings which allows for coercion and testing, combination, comparison, summaries (min/max), inclusion in data frames, subscripting, and printing.

`getRversion` returns the version of the running R as an object of class "package_version".

See Also

`compareVersion`

Examples

```r
x <- package_version(c("1.2-4", "1.2-3", "2.1"))
x < "1.4-2.3"
c(min(x), max(x))
x[2, 2]
x$major
x$minor
```
Parentheses and Braces

Description

Open parenthesis, (, and open brace, {, are Primitive functions in R.
Effectively, ( is semantically equivalent to the identity function(x) x, whereas { is slightly more interesting, see examples.

Usage

( ... )

{ ... }

References


See Also

if, return, etc for other objects used in the R language itself.

Syntax for operator precedence.

Examples

f <- get("(")
e <- expression(3 + 2 * 4)
identical(f(e), e)

do <- get("{")
do(x <- 3, y <- 2*x-3, 6-x-y); x; y

Parse Expressions

Description

parse returns the parsed but unevaluated expressions in a list. Each element of the list is of mode expression.

Usage

parse(file = "", n = NULL, text = NULL, prompt = "?"))
Arguments

file a connection, or a character string giving the name of a file or a URL to read the expressions from. If file is "" and text is missing or NULL then input is taken from the console.
n the number of statements to parse. If n is negative the file is parsed in its entirety.
text character vector. The text to parse. Elements are treated as if they were lines of a file.
prompt the prompt to print when parsing from the keyboard. NULL means to use R's prompt,getOption("prompt").

Details

All versions of R accept input from a connection with end of line marked by LF (as used on Unix), CRLF (as used on DOS/Windows) or CR (as used on classic MacOS). The final line can be incomplete, that is missing the final EOL marker.

See source for the limits on the size of functions that can be parsed (by default).

References


See Also

scan, source, eval, deparse.

Examples

cat("x <- c(1,4)
 x ^ 3 -10 ; outer(1:7,5:9)\n", file="xyz.Rdmped")
# parse 3 statements from the file "xyz.Rdmped"
parse(file = "xyz.Rdmped", n = 3)
unlink("xyz.Rdmped")

paste Concatenate Strings

description

Concatenate vectors after converting to character.

Usage

paste(..., sep = " ", collapse = NULL)

Arguments

... one or more R objects, to be coerced to character vectors.
sep a character string to separate the terms.
collapse an optional character string to separate the results.
Details

`paste` converts its arguments to character strings, and concatenates them (separating them by the string given by `sep`). If the arguments are vectors, they are concatenated term-by-term to give a character vector result.

If a value is specified for `collapse`, the values in the result are then concatenated into a single string, with the elements being separated by the value of `collapse`.

Value

A character vector of the concatenated values. This will be of length zero if all the objects are, unless `collapse` is non-NULL, in which case it is a single empty string.

References


See Also

String manipulation with `as.character`, `substr`, `nchar`, `strsplit`; further, `cat` which concatenates and writes to a file, and `sprintf` for C like string construction.

Examples

```r
paste(1:12)  # same as as.character(1:12)
paste("A", 1:6, sep = ")
paste("Today is", date())
```

path.expand

Expand File Paths

Description

Expand a path name, for example by replacing a leading tilde by the user’s home directory (if defined on that platform).

Usage

`path.expand(path)`

Arguments

`path` character vector containing one or more path names.

See Also

`basename`

Examples

```r
path.expand("~/foo")
```
pmatch

Partial String Matching

Description

pmatch seeks matches for the elements of its first argument among those of its second.

Usage

pmatch(x, table, nomatch = NA, duplicates.ok = FALSE)

Arguments

x the values to be matched.

x table the values to be matched against.

nomatch the value returned at non-matching or multiply partially matching positions.

duplicates.ok should elements be in table be used more than once?

Details

The behaviour differs by the value of duplicates.ok. Consider first the case if this is true. First

exact matches are considered, and the positions of the first exact matches are recorded. Then unique

partial matches are considered, and if found recorded. (A partial match occurs if the whole of the

element of x matches the beginning of the element of table.) Finally, all remaining elements of x

are regarded as unmatched. In addition, an empty string can match nothing, not even an exact match

to an empty string. This is the appropriate behaviour for partial matching of character indices, for

example.

If duplicates.ok is FALSE, values of table once matched are excluded from the search

for subsequent matches. This behaviour is equivalent to the R algorithm for argument matching,

except for the consideration of empty strings (which in argument matching are matched after exact

and partial matching to any remaining arguments).

charmatch is similar to pmatch with duplicates.ok true, the differences being that it dif-

ferentiates between no match and an ambiguous partial match, it does match empty strings, and it

does not allow multiple exact matches.

Value

A numeric vector of integers (including NA if nomatch = NA) of the same length as x, giving

the indices of the elements in table which matched, or nomatch.

References


Brooks/Cole.


See Also

match, charmatch and match.arg, match.fun, match.call, for function argument

matching etc., grep etc for more general (regexp) matching of strings.
polyroot

Find Zeros of a Real or Complex Polynomial

Description

Find zeros of a real or complex polynomial.

Usage

polyroot(z)

Arguments

z

the vector of polynomial coefficients in increasing order.

Details

A polynomial of degree \( n - 1 \),

\[ p(x) = z_1 + z_2 x + \cdots + z_n x^{n-1} \]

is given by its coefficient vector \( z[1:n] \). polyroot returns the \( n - 1 \) complex zeros of \( p(x) \) using the Jenkins-Traub algorithm.

If the coefficient vector \( z \) has zeroes for the highest powers, these are discarded.

Value

A complex vector of length \( n - 1 \), where \( n \) is the position of the largest non-zero element of \( z \).

References


See Also

uniroot for numerical root finding of arbitrary functions; complex and the zero example in the demos directory.

Examples

polyroot(c(1, 2, 1))
round(polyroot(choose(8, 0:8)), 11) # guess what!
for (n1 in 1:4) print(polyroot(1:n1), digits = 4)
polyroot(c(1, 2, 1, 0, 0)) # same as the first
pos.to.env  Convert Positions in the Search Path to Environments

Description

Returns the environment at a specified position in the search path.

Usage

pos.to.env(x)

Arguments

x  
an integer between 1 and length(search()), the length of the search path.

Details

Several R functions for manipulating objects in environments (such as get and ls) allow specifying environments via corresponding positions in the search path. pos.to.env is a convenience function for programmers which converts these positions to corresponding environments; users will typically have no need for it.

Examples

pos.to.env(1) # R_GlobalEnv
# the next returns NULL, which is how package:base is represented.
pos.to.env(length(search()))

pretty  Pretty Breakpoints

Description

Compute a sequence of about n+1 equally spaced nice values which cover the range of the values in x. The values are chosen so that they are 1, 2 or 5 times a power of 10.

Usage

pretty(x, n = 5, min.n = n %/% 3, shrink.sml = 0.75,
      high.u.bias = 1.5, u5.bias = .5 + 1.5*high.u.bias,
      eps.correct = 0)

Arguments

x  
numeric vector

n  
integer giving the desired number of intervals. Non-integer values are rounded down.

min.n  
nonnegative integer giving the minimal number of intervals. If min.n == 0,
      pretty(.) may return a single value.
shrink.sml  positive numeric by a which a default scale is shrunk in the case when range(x) is “very small” (usually 0).

high.u.bias  non-negative numeric, typically > 1. The interval unit is determined as {1,2,5,10} times b, a power of 10. Larger high.u.bias values favor larger units.

u5.bias  non-negative numeric multiplier favoring factor 5 over 2. Default and “optimal”: u5.bias = .5 + 1.5*high.u.bias.

eps.correct  integer code, one of {0,1,2}. If non-0, an “epsilon correction” is made at the boundaries such that the result boundaries will be outside range(x); in the small case, the correction is only done if eps.correct >=2.

Details

As from R 2.0.0 pretty ignores non-finite values in x.

Let d <- max(x) - min(x) >= 0. If d is not (very close) to 0, we let c <- d/n, otherwise more or less c <- max(abs(range(x))) * shrink.sml / min.n. Then, the 10 base b is 10^[log_{10}(c)] such that b <= c < 10b.

Now determine the basic unit u as one of {1,2,5,10}b, depending on c/b \in [1,10) and the two “bias” coefficients, h = high.u.bias and f = u5.bias.

References


See Also

axTicks for the computation of pretty axis tick locations in plots, particularly on the log scale.

Examples

```r
pretty(1:15)  # 0  2  4  6  8 10 12 14 16
pretty(1:15, h=2) # 0  5 10 15
pretty(1:15, n=4) # 0  5 10 15
pretty(1:15 * 2) # 0  5 10 15 20 25 30
pretty(1:20)  # 0  5 10 15 20
pretty(1:20, n=2) # 0 10 20
pretty(1:20, n=10) # 0  2  4 ... 20
for(k in 5:11) {
  cat("k=",k,"": "); print(diff(range(pretty(100 + c(0, pi*10^-k))))))
}

##-- more bizarre, when min(x) == max(x):
pretty(pi)
add.names <- function(v) { names(v) <- paste(v); v}
str(lapply(add.names(-10:20), pretty))
str(lapply(add.names(0:20), pretty, min = 0))
sapply( add.names(0:20), pretty, min = 4)
pretty(1.234e100)
pretty(1001.1001)
```
pretty(1001.1001, shrink = .2)
for(k in -7:3)
cat("shrink=",formatC(2^k,wid=9),":",
    formatC(pretty(1001.1001, shrink = 2^k), wid=6),"\n")

---

**Primitive**

*Call a “Primitive” Internal Function*

**Description**

`.Primitive` returns an entry point to a “primitive” (internally implemented) function. The advantage of `.Primitive` over `.Internal` functions is the potential efficiency of argument passing.

**Usage**

`.Primitive(name)`

**Arguments**

- `name`: name of the \texttt{R} function.

**See Also**

`.Internal`.

**Examples**

```r
mysqrt <- .Primitive("sqrt")
# this one *must* be primitive!
get("if") # just 'if' or 'print(if)' are not syntactically ok.
```

---

**Print Values**

**Description**

`print` prints its argument and returns it *invisibly* (via \texttt{invisible(x)}). It is a generic function which means that new printing methods can be easily added for new \texttt{classes}.

**Usage**

`print(x, ...)`

```r
## S3 method for class 'factor':
print(x, quote = FALSE, max.levels = NULL,
    width = getOption("width"), ...)

## S3 method for class 'table':
print(x, digits = getOption("digits"), quote = FALSE,
    na.print = "", zero.print = "0", justify = "none", ...)
```
Arguments

- **x**: an object used to select a method.
- **...**: further arguments passed to or from other methods.
- **quote**: logical, indicating whether or not strings should be printed with surrounding quotes.
- **max.levels**: integer, indicating how many levels should be printed for a factor; if 0, no extra "Levels" line will be printed. The default, NULL, entails choosing max.levels such that the levels print on one line of width width.
- **width**: only used when max.levels is NULL, see above.
- **digits**: minimal number of significant digits, see `print.default`.
- **na.print**: character string (or NULL) indicating NA values in printed output, see `print.default`.
- **zero.print**: character specifying how zeros (0) should be printed; for sparse tables, using "." can produce stronger results.
- **justify**: character indicating if strings should left- or right-justified or left alone, passed to `format`.

Details

The default method, `print.default` has its own help page. Use `methods("print")` to get all the methods for the print generic.

`print.factor` allows some customization and is used for printing ordered factors as well.

`print.table` for printing tables allows other customization.

See `noquote` as an example of a class whose main purpose is a specific `print` method.

References


See Also

The default method `print.default`, and help for the methods above; further options, `noquote`.

For more customizable (but cumbersome) printing, see `cat`, `format` or also `write`.

Examples

```r
ts(1:20)#-- print is the "Default function" --> print.ts(.) is called
rr <- for(i in 1:3) print(1:i)
rr
## Printing of factors
attenu$station ## 117 levels --> `max.levels' depending on width

## ordered factors: levels "11 < 12 < .."
esoph$agegp[1:12]
esoph$alcgp[1:12]

## Printing of sparse (contingency) tables
set.seed(521)
```

t1 <- round(abs(rt(200, df=1.8)))
t2 <- round(abs(rt(200, df=1.4)))
table(t1,t2) # simple
print(table(t1,t2), zero.print = ".")# nicer to read

print.data.frame Printing Data Frames

Description
Print a data frame.

Usage
### S3 method for class 'data.frame':
print(x, ..., digits = NULL, quote = FALSE, right = TRUE)

Arguments
- `x` object of class data.frame.
- `...` optional arguments to print or plot methods.
- `digits` the minimum number of significant digits to be used: see print.default.
- `quote` logical, indicating whether or not entries should be printed with surrounding quotes.
- `right` logical, indicating whether or not strings should be right-aligned. The default is left-alignment.

Details
This calls format which formats the data frame column-by-column, then converts to a character matrix and dispatches to the print method for matrices.

When quote = TRUE only the entries are quoted not the row names nor the column names.

See Also
data.frame.

print.default Default Printing

Description
print.default is the default method of the generic print function which prints its argument.

Usage
### Default S3 method:
print(x, digits = NULL, quote = TRUE, na.print = NULL, print.gap = NULL, right = FALSE, ...)

Arguments

\( \text{x} \)  
the object to be printed.

\( \text{digits} \)  
a non-null value for \( \text{digits} \) specifies the minimum number of significant digits to be printed in values. The default, NULL, uses \( \text{getOption(digits)} \). (For the interpretation for complex numbers see \text{signif}.)

\( \text{quote} \)  
logical, indicating whether or not strings (characters) should be printed with surrounding quotes.

\( \text{na.print} \)  
a character string which is used to indicate \text{NA} values in printed output, or NULL (see Details)

\( \text{print.gap} \)  
a non-negative integer \( \leq 1024 \), giving the spacing between adjacent “columns” in printed vectors, matrices and arrays, or NULL meaning 1.

\( \text{right} \)  
logical, indicating whether or not strings should be right-aligned. The default is left-alignment.

...  
further arguments to be passed to or from other methods. They are ignored in this function.

Details

The default for printing \text{NAs} is to print \text{NA} (without quotes) unless this is a character \text{NA} and \text{quote} = \text{FALSE}, when \text{NA} is printed.

The same number of decimal places is used throughout a vector. This means that \text{digits} specifies the minimum number of significant digits to be used, and that at least one entry will be encoded with that minimum number. However, if all the encoded elements then have trailing zeroes, the number of decimal places is reduced until at least one element has a non-zero final digit.

Attributes are printed respecting their class(es), using the values of \text{digits} to print.default, but using the default values (for the methods called) of the other arguments.

When the \text{methods} package is attached, print will call show for \text{R} objects with formal classes if called with no optional arguments.

If a non-printable character is encountered during output, it is represented as one of the ANSI escape sequences (\text{a}, \text{b}, \text{f}, \text{n}, \text{r}, \text{t}, \text{v} and \text{0}), or failing that as a 3-digit octal code: for example the UK currency pound in the C locale (if implemented correctly) is printed as 243. Which characters are non-printable depends on the locale. (Because some versions of Windows get this wrong, all 8-bit characters are regarded as printable on Windows.)

Unicode and other multi-byte locales

In a Unicode (UTF-8) locale, characters 0x00 to 0x1F and 0x7F (the ASCII non-printing characters) are printed in the same way, via ANSI escape sequences or 3-digit octal escapes. Multi-byte non-printing characters are printed with as an escape sequence of the form \text{uxxxxx} or \text{Uxxxxxxxx} (in hexadecimal).
It is possible to have a character string in an object that is not valid UTF-8. If a byte is encountered that is not part of an encoded Unicode character it is printed in hex in the form \texttt{<xx>} and the next character is tried.

**See Also**

- The generic \texttt{print}, \texttt{options}. The \texttt{"noquote"} class and print method. 
- \texttt{encodeString}.

**Examples**

```r
pi
print(pi, digits = 16)
LETTERS[1:16]
print(LETTERS, quote = FALSE)
```

---

**prmatrix**

*Print Matrices, Old-style*

**Description**

An earlier method for printing matrices, provided for S compatibility.

**Usage**

```r
prmatrix(x, rowlab =, collab =,
quote = TRUE, right = FALSE, na.print = NULL, ...)
```

**Arguments**

- \texttt{x} numeric or character matrix.
- \texttt{rowlab, collab} (optional) character vectors giving row or column names respectively. By default, these are taken from \texttt{dimnames(x)}.
- \texttt{quote} logical; if \texttt{TRUE} and \texttt{x} is of mode \texttt{"character"}, \texttt{quotes(\textcolor{red}{")})} are used.
- \texttt{right} if \texttt{TRUE} and \texttt{x} is of mode \texttt{"character"}, the output columns are \texttt{right}-justified.
- \texttt{na.print} how \texttt{NA}s are printed. If this is non-null, its value is used to represent \texttt{NA}.
- \texttt{...} arguments for \texttt{print} methods.

**Details**

\texttt{prmatrix} is an earlier form of \texttt{print.matrix}, and is very similar to the S function of the same name.

**Value**

Invisibly returns its argument, \texttt{x}.
References


See Also

`print.default`, and other print methods.

Examples

```r
prmatrix(m6 <- diag(6), row = rep("",6), coll=rep("",6))

chm <- matrix(scan(system.file("help", "AnIndex", package = "splines"),
         what = ""), , 2, byrow = TRUE)
chm # uses print.matrix()
prmatrix(chm, collab = paste("Column",1:3), right=TRUE, quote=FALSE)
```

proc.time  

Running Time of R

Description

`proc.time` determines how much time (in seconds) the currently running R process already consumed.

Usage

`proc.time()`

Value

A numeric vector of length 5, containing the user, system, and total elapsed times for the currently running R process, and the cumulative sum of user and system times of any child processes spawned by it.

The resolution of the times will be system-specific; it is common for them to be recorded to of the order of 1/100 second, and elapsed time is rounded to the nearest 1/100.

It is most useful for “timing” the evaluation of R expressions, which can be done conveniently with `system.time`.

Note

CPU times will be returned as `NA` on Windows 9x/ME systems, but are genuine times on NT4/2000/XP systems. Times of child processes are not available and will always be given as `NA`.

References

prod

Description

prod returns the product of all the values present in its arguments.

Usage

prod(..., na.rm = FALSE)

Arguments

... numeric vectors.
na.rm logical. Should missing values be removed?

Details

If na.rm is FALSE an NA value in any of the arguments will cause a value of NA to be returned, otherwise NA values are ignored.

This is a generic function: methods can be defined for it directly or via the Summary group generic.

References


See Also

sum, cumprod, cumsum.

Examples

print(prod(1:7)) == print(gamma(8))

See Also

system.time for timing a valid R expression, gc.time for how much of the time was spent in garbage collection.

Examples

## Not run:
## a way to time an R expression: system.time is preferred
ptm <- proc.time()
for (i in 1:50) mad(runif(500))
proc.time() - ptm
## End(Not run)
prop.table  Express Table Entries as Fraction of Marginal Table

Description
This is really sweep(x, margin, margin.table(x, margin), "/") for newbies, except that if margin has length zero, then one gets x/sum(x).

Usage
prop.table(x, margin=NULL)

Arguments
x  table
margin  index, or vector of indices to generate margin for

Value
Table like x expressed relative to margin

Author(s)
Peter Dalgaard

See Also
margin.table

Examples
m <- matrix(1:4,2)
m
prop.table(m,1)

pushBack  Push Text Back on to a Connection

Description
Functions to push back text lines onto a connection, and to enquire how many lines are currently pushed back.

Usage
pushBack(data, connection, newLine = TRUE)
pushBackLength(connection)
Arguments

data a character vector.
connection A connection.
newline logical. If true, a newline is appended to each string pushed back.

Details

Several character strings can be pushed back on one or more occasions. The occasions form a stack, so the first line to be retrieved will be the first string from the last call to pushBack. Lines which are pushed back are read prior to the normal input from the connection, by the normal text-reading functions such as readLines and scan.

Pushback is only allowed for readable connections.

Not all uses of connections respect pushbacks, in particular the input connection is still wired directly, so for example parsing commands from the console and scan ("") ignore pushbacks on stdin.

Value

pushBack returns nothing.
pushBackLength returns number of lines currently pushed back.

See Also

connections, readLines.

Examples

zz <- textConnection(LETTERS)
readLines(zz, 2)
pushBack(c("aa", "bb"), zz)
pushBackLength(zz)
readLines(zz, 1)
pushBackLength(zz)
readLines(zz, 1)
readLines(zz, 1)
close(zz)

qr

The QR Decomposition of a Matrix

Description

qr computes the QR decomposition of a matrix. It provides an interface to the techniques used in the LINPACK routine DQRDC or the LAPACK routines DGEQP3 and (for complex matrices) ZGEQP3.
Usage

qr(x, tol = 1e-07, LAPACK = FALSE)
qr.coef(qr, y)
qr.qy(qr, y)
qr.qty(qr, y)
qr.resid(qr, y)
qr.fitted(qr, y, k = qr$rank)
qr.solve(a, b, tol = 1e-7)
## S3 method for class 'qr':
solve(a, b, ...)

is.qr(x)
as.qr(x)

Arguments

x a matrix whose QR decomposition is to be computed.
tol the tolerance for detecting linear dependencies in the columns of x. Only used if LAPACK is false and x is real.
qr a QR decomposition of the type computed by qr.
y, b a vector or matrix of right-hand sides of equations.
a A QR decomposition or (qr.solve only) a rectangular matrix.
k effective rank.
LAPACK logical. For real x, if true use LAPACK otherwise use LINPACK.
... further arguments passed to or from other methods

Details

The QR decomposition plays an important role in many statistical techniques. In particular it can be used to solve the equation \( Ax = b \) for given matrix \( A \), and vector \( b \). It is useful for computing regression coefficients and in applying the Newton-Raphson algorithm.

The functions qr.coef, qr.resid, and qr.fitted return the coefficients, residuals and fitted values obtained when fitting \( y \) to the matrix with QR decomposition \( qr \). qr.qy and qr.qty return \( Q \%\%^t y \) and \( t(Q) \%\%^t y \), where \( Q \) is the (complete) \( Q \) matrix.

All the above functions keep dimnames (and names) of \( x \) and \( y \) if there are.

solve.qr is the method for solve for qr objects. qr.solve solves systems of equations via the QR decomposition: if \( a \) is a QR decomposition it is the same as solve.qr, but if \( a \) is a rectangular matrix the QR decomposition is computed first. Either will handle over- and under-determined systems, providing a minimal-length solution or a least-squares fit if appropriate.

is.qr returns TRUE if \( x \) is a list with components named qr, rank and qraux and FALSE otherwise.

It is not possible to coerce objects to mode "qr". Objects either are QR decompositions or they are not.

Value

The QR decomposition of the matrix as computed by LINPACK or LAPACK. The components in the returned value correspond directly to the values returned by DQRDC/DGEQP3/ZGEQP3.
qr a matrix with the same dimensions as \( x \). The upper triangle contains the \( R \) of the decomposition and the lower triangle contains information on the \( Q \) of the decomposition (stored in compact form). Note that the storage used by DQRDC and DGEQP3 differs.

qraux a vector of length \( ncol(x) \) which contains additional information on \( Q \).

rank the rank of \( x \) as computed by the decomposition: always full rank in the LAPACK case.

pivot information on the pivoting strategy used during the decomposition.

Non-complex QR objects computed by LAPACK have the attribute "useLAPACK" with value TRUE.

Note

To compute the determinant of a matrix (do you really need it?), the QR decomposition is much more efficient than using Eigen values (eigen). See det.

Using LAPACK (including in the complex case) uses column pivoting and does not attempt to detect rank-deficient matrices.

References


See Also


det (using qr) to compute the determinant of a matrix.

Examples

hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
h9 <- hilbert(9); h9
qr(h9)$rank #--> only 7
qrh9 <- qr(h9, tol = 1e-10)
qrh9$rank #--> 9
###-- Solve linear equation system \( H \%*% x = y \):
y <- 1:9/10
x <- qr.solve(h9, y, tol = 1e-10) # or equivalently :
x <- qr.coef(qrh9, y) #-- is == but much better than
#-- solve(h9) \%*% y
h9 \%*% x # = y
Reconstruct the Q, R, or X Matrices from a QR Object

Description

Returns the original matrix from which the object was constructed or the components of the decomposition.

Usage

qr.X(qr, complete = FALSE, ncol =)
qr.Q(qr, complete = FALSE, Dvec =)
qr.R(qr, complete = FALSE)

Arguments

qr object representing a QR decomposition. This will typically have come from a previous call to qr or lsfit.

complete logical expression of length 1. Indicates whether an arbitrary orthogonal completion of the Q or X matrices is to be made, or whether the R matrix is to be completed by binding zero-value rows beneath the square upper triangle.

ncol integer in the range 1:nrow(qr$qr). The number of columns to be in the reconstructed X. The default when complete is FALSE is the first \( \min(n\text{col}(X), n\text{row}(X)) \) columns of the original X from which the qr object was constructed. The default when complete is TRUE is a square matrix with the original X in the first ncol(X) columns and an arbitrary orthogonal completion (unitary completion in the complex case) in the remaining columns.

Dvec vector (not matrix) of diagonal values. Each column of the returned Q will be multiplied by the corresponding diagonal value. Defaults to all 1s.

Value

qr.X returns X, the original matrix from which the qr object was constructed, provided ncol(X) <= nrow(X). If complete is TRUE or the argument ncol is greater than ncol(X), additional columns from an arbitrary orthogonal (unitary) completion of X are returned.

qr.Q returns part or all of Q, the order-nrow(X) orthogonal (unitary) transformation represented by qr. If complete is TRUE, Q has nrow(X) columns. If complete is FALSE, Q has ncol(X) columns. When Dvec is specified, each column of Q is multiplied by the corresponding value in Dvec.

qr.R returns R, the upper triangular matrix such that \( X = Q \times R \). The number of rows of R is nrow(X) or ncol(X), depending on whether complete is TRUE or FALSE.

See Also

qr, qr.qy.
Examples

```r
p <- ncol(x <- LifeCycleSavings[,-1]) # not the 'sr'
qrstr <- qr(x) # dim(x) == c(n,p)
qrstr $ rank # = 4 = p
Q <- qr.Q(qrstr) # dim(Q) == dim(x)
R <- qr.R(qrstr) # dim(R) == ncol(x)
X <- qr.X(qrstr) # X == x
range(X - as.matrix(x))# ~ < 6e-12
## X == Q %*% R :
Q %*% R
```

Description

The function `quit` or its alias `q` terminate the current R session.

Usage

```r
quit(save = "default", status = 0, runLast = TRUE)
q(save = "default", status = 0, runLast = TRUE)
.Last <- function(x) { ...... }
```

Arguments

- `save` a character string indicating whether the environment (workspace) should be saved, one of "no", "yes", "ask" or "default".
- `status` the (numerical) error status to be returned to the operating system, where relevant. Conventionally 0 indicates successful completion.
- `runLast` should `.Last()` be executed?

Details

`save` must be one of "no", "yes", "ask" or "default". In the first case the workspace is not saved, in the second it is saved and in the third the user is prompted and can also decide not to quit. The default is to ask in interactive use but may be overridden by command-line arguments (which must be supplied in non-interactive use).

Immediately before terminating, the function `.Last()` is executed if it exists and `runLast` is true. If in interactive use there are errors in the `.Last` function, control will be returned to the command prompt, so do test the function thoroughly.

Some error statuses are used by R itself. The default error handler for non-interactive effectively calls `q("no", 1, FALSE)` and returns error code 1. Error status 2 is used for R 'suicide', that is a catastrophic failure, and other small numbers are used by specific ports for initialization failures. It is recommended that users choose statuses of 10 or more.

Valid values of `status` are system-dependent, but 0:255 are normally valid.

References

### Description

Descriptions of the various uses of quoting in R.

### Details

Three types of quote are part of the syntax of R: single and double quotation marks and the backtick (or back quote, `'). In addition, backslash is used for quoting the following character(s) inside character constants.

### Character constants

Single and double quotes delimit character constants. They can be used interchangeably but double quotes are preferred (and character constants are printed using double quotes), so single quotes are normally only used to delimit character constants containing double quotes.

Backslash is used to start an escape sequence inside character constants. Unless specified in the following table, an escaped character is interpreted as the character itself. Single quotes need to be escaped by backslash in single-quoted strings, and double quotes in double-quoted strings.

<table>
<thead>
<tr>
<th>Escape</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n   \r</td>
<td>carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\b</td>
<td>backspace</td>
</tr>
<tr>
<td>\a</td>
<td>alert (bell)</td>
</tr>
<tr>
<td>\f</td>
<td>form feed</td>
</tr>
<tr>
<td>\v</td>
<td>vertical tab</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\xnn</td>
<td>character with given hex code (1 or 2 hex digits)</td>
</tr>
<tr>
<td>\Unnnnn</td>
<td>Unicode character with given hex code (1–8 hex digits)</td>
</tr>
</tbody>
</table>

The last two are only supported in Unicode and other multibyte locales, and the last is not supported.
on Windows. All but the Unicode escape sequences are also supported when reading character strings from a connection by `scan`.

### Names and Identifiers

Identifiers consist of a sequence of letters, digits, the period (.) and the underscore. They must not start with a digit nor underscore, nor with a period followed by a digit.

The definition of a *letter* depends on the current locale, but only ASCII digits are considered to be digits.

Such identifiers are also known as *syntactic names* and may be used directly in R code. Almost always, other names can be used provided they are quoted. The preferred quote is the backtick (`\`), and `deparse` will normally use it, but under many circumstances single or double quotes can be used (as a character constant will often be converted to a name). One place where backticks may be essential is to delimit variable names in formulae: see `formula`.

### See Also

- `Syntax` for other aspects of the syntax.
- `sQuote` for quoting English text.
- `shQuote` for quoting OS commands.
- The *R Language Definition* manual.

---

**R.home**

*Return the R Home Directory*

---

**Description**

Return the R home directory.

**Usage**

```r
R.home(component="home")
```

**Arguments**

- `component` As well as "home" which gives the R home directory, other known values are "bin", "doc", "etc" and "share" giving the paths to the corresponding parts of an R installation.

**Value**

A character string giving the current R home directory or path to a particular component. Normally the components are all subdirectories of the R home directory, but this may not be the case in a Unix-like installation.
**R.Version**  

**Version Information**

**Description**

R.Version() provides detailed information about the version of R running.  

R.version is a variable (a list) holding this information (and version is a copy of it for S compatibility), whereas R.version.string is a simple character string, useful for plotting, etc.

**Usage**

R.Version()  
R.version  
R.version.string

**Value**

R.Version returns a list with character-string components

- **platform** the platform for which R was built. A triplet of the form CPU-VENDOR-OS, as determined by the configure script. E.g, "i586-known-linux" or "i386-pc-mingw32".
- **arch** the architecture (CPU) R was built on/for.
- **os** the underlying operating system
- **system** CPU and OS, separated by a comma.
- **status** the status of the version (e.g., "Alpha")
- **major** the major version number
- **minor** the minor version number, including the patchlevel
- **year** the year the version was released
- **month** the month the version was released
- **day** the day the version was released
- **svn rev** the Subversion revision number, which should be either "unknown" or a single number. (A range of numbers or a number with M or S appended indicates inconsistencies in the sources used to build this version of R.)
- **language** always "R".

R.version and version are lists of class "simple.list" which has a print method.

**Note**

Do not use R.version$os to test the platform the code is running on: use .Platform$OS.type instead. Slightly different versions of the OS may report different values of R.version$os, as may different versions of R.

**See Also**

getRversion .Platform.
Examples

R.version$os # to check how lucky you are ...
plot(0) # any plot
mtext(R.version.string, side=1,line=4,adj=1)# a useful bottom-right note

r2dtable

Random 2-way Tables with Given Marginals

Description
Generate random 2-way tables with given marginals using Patefield’s algorithm.

Usage
r2dtable(n, r, c)

Arguments
- n: a non-negative numeric giving the number of tables to be drawn.
- r: a non-negative vector of length at least 2 giving the row totals, to be coerced to integer. Must sum to the same as c.
- c: a non-negative vector of length at least 2 giving the column totals, to be coerced to integer.

Value
A list of length n containing the generated tables as its components.

References

Examples

## Fisher's Tea Drinker data.
TeaTasting <-
matrix(c(3, 1, 1, 3),
nr = 2,
dimnames = list(Guess = c("Milk", "Tea"),
Truth = c("Milk", "Tea")))

## Simulate permutation test for independence based on the maximum
## Pearson residuals (rather than their sum).
rowTotals <- rowSums(TeaTasting)
colTotals <- colSums(TeaTasting)
NoOfCases <- sum(rowTotals)
expected <- outer(rowTotals, colTotals, "+") / NoOfCases
maxSqResid <- function(x) max((x - expected) ^ 2 / expected)
simMaxSqResid <-
sapply(r2dtable(1000, rowTotals, colTotals), maxSqResid)
sum(simMaxSqResid >= maxSqResid(TeaTasting)) / 1000

## Fisher's exact test gives p = 0.4857 ...
**Random Number Generation**

**Description**

`.Random.seed` is an integer vector, containing the random number generator (RNG) **state** for random number generation in R. It can be saved and restored, but should not be altered by the user.

RNGkind is a more friendly interface to query or set the kind of RNG in use.

RNGversion can be used to set the random generators as they were in an earlier R version (for reproducibility).

`set.seed` is the recommended way to specify seeds.

**Usage**

```r
.Random.seed <- c(rng.kind, n1, n2, ...)
save.seed <- .Random.seed
RNGkind(kind = NULL, normal.kind = NULL)
RNGversion(vstr)
set.seed(seed, kind = NULL)
```

**Arguments**

- `kind` character or `NULL`. If `kind` is a character string, set R’s RNG to the kind desired. If it is `NULL`, return the currently used RNG. Use "default" to return to the R default.

- `normal.kind` character string or `NULL`. If it is a character string, set the method of Normal generation. Use "default" to return to the R default.

- `seed` a single value, interpreted as an integer.

- `vstr` a character string containing a version number, e.g., "1.6.2"

- `rng.kind` integer code in 0:k for the above kind.

- `n1, n2, ...` integers. See the details for how many are required (which depends on `rng.kind`).

**Details**

The currently available RNG kinds are given below. `kind` is partially matched to this list. The default is "Mersenne-Twister".

"**Wichmann-Hill**" The seed, `.Random.seed[-1] == r[1:3]` is an integer vector of length 3, where each r[i] is in 1:(p[i] - 1), where p is the length 3 vector of primes, p = (30269, 30307, 30323). The Wichmann–Hill generator has a cycle length of 6.9536 x 10^{12} (= prod(p-1)/4, see Applied Statistics (1984) 33, 123 which corrects the original article).

"**Marsaglia-Multicarry**": A multiply-with-carry RNG is used, as recommended by George Marsaglia in his post to the mailing list ‘sci.stat.math’. It has a period of more than 2^{60} and has passed all tests (according to Marsaglia). The seed is two integers (all values allowed).
"Super-Duper": Marsaglia’s famous Super-Duper from the 70’s. This is the original version which does not pass the MTUPLE test of the Diehard battery. It has a period of \( \approx 4.6 \times 10^{18} \) for most initial seeds. The seed is two integers (all values allowed for the first seed: the second must be odd).

We use the implementation by Reeds et al. (1982–84).

The two seeds are the Tausworthe and congruence long integers, respectively. A one-to-one mapping to S’s \(_\text{Random}.seed[1:12] \) is possible but we will not publish one, not least as this generator is not exactly the same as that in recent versions of S-PLUS.

"Mersenne-Twister": From Matsumoto and Nishimura (1998). A twisted GFSR with period \( 2^{19937} - 1 \) and equidistribution in 623 consecutive dimensions (over the whole period). The “seed” is a 624-dimensional set of 32-bit integers plus a current position in that set.

"Knuth-TAOCP": From Knuth (1997). A GFSR using lagged Fibonacci sequences with subtraction. That is, the recurrence used is

\[
X_j = (X_{j-100} - X_{j-37}) \mod 2^{30}
\]

and the “seed” is the set of the 100 last numbers (actually recorded as 101 numbers, the last being a cyclic shift of the buffer). The period is around \( 2^{129} \).

"Knuth-TAOCP-2002": The 2002 version which not backwards compatible with the earlier version: the initialization of the GFSR from the seed was altered. \( R \) did not allow you to choose consecutive seeds, the reported ‘weakness’, and already scrambled the seeds.

"user-supplied": Use a user-supplied generator. See \(_\text{Random}.user\) for details.

normal.kind can be "Kinderman-Ramage", "Buggy Kinderman-Ramage", "Ahrens-Dieter", "Box-Muller", "Inversion" (the default), or "user-supplied". (For inversion, see the reference in \(_\text{qnorm}\).) The Kinderman-Ramage generator used in versions prior to 1.7.1 had several approximation errors and should only be used for reproduction of older results.

\(_\text{set.seed}\) uses its single integer argument to set as many seeds as are required. It is intended as a simple way to get quite different seeds by specifying small integer arguments, and also as a way to get valid seed sets for the more complicated methods (especially "Mersenne-Twister" and "Knuth-TAOCP").

Value

\(_\text{.Random.seed}\) is an \texttt{integer} vector whose first element codes the kind of RNG and normal generator. The lowest two decimal digits are in 0:(k–1) where k is the number of available RNGs. The hundreds represent the type of normal generator (starting at 0).

In the underlying C, \(_\text{.Random.seed[-1]}\) is unsigned; therefore in \( R \), \(_\text{.Random.seed[-1]}\) can be negative, due to the representation of an unsigned integer by a signed integer.

\(_\text{RNGkind}\) returns a two-element character vector of the RNG and normal kinds in use before the call, invisibly if either argument is not NULL. \(_\text{RNGversion}\) returns the same information.

\(_\text{set.seed}\) returns NULL, invisibly.

Note

Initially, there is no seed; a new one is created from the current time when one is required. Hence, different sessions will give different simulation results, by default.

\(_\text{.Random.seed}\) saves the seed set for the uniform random-number generator, at least for the system generators. It does not necessarily save the state of other generators, and in particular does
not save the state of the Box–Muller normal generator. If you want to reproduce work later, call 

As from R 1.8.0, `Random.seed` is only looked for in the user’s workspace.

All the supplied uniform generators return 32-bit integer values that are converted to doubles, so they take at most $2^{32}$ distinct values and long runs will return duplicated values.

**Author(s)**

of RNGkind: Martin Maechler. Current implementation, B. D. Ripley

**References**


Reeds, J., Hubert, S. and Abrahams, M. (1982–4) C implementation of SuperDuper, University of California at Berkeley. (Personal communication from Jim Reeds to Ross Ihaka.)


**See Also**

`runif`, `rnorm`, ....

**Examples**

```r
## the default random seed is 626 integers, so only print a few
runif(1); .Random.seed[1:6]; runif(1); .Random.seed[1:6]
## If there is no seed, a "random" new one is created:
rm(.Random.seed); runif(1); .Random.seed[1:6]
```
Random.user

User-supplied Random Number Generation

Description

Function **RNGkind** allows user-coded uniform and normal random number generators to be supplied. The details are given here.

Details

A user-specified uniform RNG is called from entry points in dynamically-loaded compiled code. The user must supply the entry point **user_unif_rand**, which takes no arguments and returns a pointer to a double. The example below will show the general pattern.

Optionally, the user can supply the entry point **user_unif_init**, which is called with an unsigned int argument when **RNGkind** (or **set.seed**) is called, and is intended to be used to initialize the user’s RNG code. The argument is intended to be used to set the “seeds”; it is the seed argument to **set.seed** or an essentially random seed if **RNGkind** is called.

If only these functions are supplied, no information about the generator’s state is recorded in **.Random.seed**. Optionally, functions **user_unif_nseed** and **user_unif_seedloc** can

### Example

```r
RNGkind("Wich") # (partial string matching on 'kind')

## This shows how 'runif(.)' works for Wichmann-Hill,
## using only R functions:

p.WH <- c(30269, 30307, 30323)
a.WH <- c(171, 172, 170)
next.WHseed <- function(i.seed = .Random.seed[-1])
  { (a.WH * i.seed) %% p.WH }
my.runif1 <- function(i.seed = .Random.seed)
  { ns <- next.WHseed(i.seed[-1]); sum(ns / p.WH) %% 1 }
rs <- .Random.seed
(WHs <- next.WHseed(rs[-1]))
u <- runif(1)
stopifnot(
  next.WHseed(rs[-1]) == .Random.seed[-1],
  all.equal(u, my.runif1(rs))
)

## ----
.Random.seed
ok <- RNGkind()
RNGkind("Super") # matches "Super-Duper"
RNGkind()
(Random.seed # new, corresponding to Super-Duper

## Reset:
RNGkind(ok[1])

## ----
sum(duplicated(runif(1e6))) # around 110
## and we would expect about almost sure duplicates beyond about
qbirthday(1-1e-6, classes=2e9) # 235,000
```
be supplied which are called with no arguments and should return pointers to the number of “seeds” and to an integer array of “seeds”. Calls to GetRNGstate and PutRNGstate will then copy this array to and from .Random.seed.

A user-specified normal RNG is specified by a single entry point user_norm_rand, which takes no arguments and returns a pointer to a double.

**Warning**

As with all compiled code, mis-specifying these functions can crash R. Do include the ‘R_ext/Random.h’ header file for type checking.

**Examples**

```c
## Not run:
## Marsaglia's congruential PRNG
#include <R_ext/Random.h>

static Int32 seed;
static double res;
static int nseed = 1;

double * user_unif_rand()
{
    seed = 69069 * seed + 1;
    res = seed * 2.32830643653869e-10;
    return &res;
}

void user_unif_init(Int32 seed_in) { seed = seed_in; }
int * user_unif_nseed() { return &nseed; }
int * user_unif_seedloc() { return (int *) &seed; }

/* ratio-of-uniforms for normal */
#include <math.h>
static double x;

double * user_norm_rand()
{
    double u, v, z;
    do {
        u = unif_rand();
        v = 0.857764 * (2. * unif_rand() - 1);
        x = v/u; z = 0.25 * x * x;
        if (z < 1. - u) break;
        if (z > 0.259/u + 0.35) continue;
    } while (z > -log(u));
    return &x;
}

## Use under Unix:
R CMD SHLIB urand.c
R
> dyn.load("urand.so")
> RNGkind("user")
> runif(10)
> .Random.seed
```
range

> RNGkind(, "user")
> rnorm(10)
> RNGkind()
[1] "user-supplied" "user-supplied"
## End(Not run)

range     Range of Values

Description

range returns a vector containing the minimum and maximum of all the given arguments.

Usage

range(..., na.rm = FALSE)
## Default S3 method:
range(..., na.rm = FALSE, finite = FALSE)

Arguments

...       any numeric objects.
n.a.rm     logical, indicating if NA's should be omitted.
finite     logical, indicating if all non-finite elements should be omitted.

Details

range is a generic function: methods can be defined for it directly or via the Summary group generic.

If na.rm is FALSE, NA and NaN values in any of the arguments will cause NA values to be returned, otherwise NA values are ignored.

If finite is TRUE, the minimum and maximum of all finite values is computed, i.e., finite=TRUE includes na.rm=TRUE.

A special situation occurs when there is no (after omission of NAs) nonempty argument left, see min.

References


See Also

min, max, Methods.
Examples

```r
(r.x <- range(rnorm(100)))
diff(r.x) # the SAMPLE range

x <- c(NA, 1:3, -1:1/0); x
range(x)
range(x, na.rm = TRUE)
range(x, finite = TRUE)
```

---

**rank**

**Sample Ranks**

**Description**

Returns the sample ranks of the values in a vector. Ties, i.e., equal values, result in ranks being averaged, by default.

**Usage**

```r
rank(x, na.last = TRUE,
     ties.method = c("average", "first", "random", "max", "min"))
```

**Arguments**

- `x` a numeric, complex, character or logical vector.
- `na.last` for controlling the treatment of *NA*s. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed; if "keep" they are kept.
- `ties.method` a character string specifying how ties are treated, see below; can be abbreviated.

**Details**

If all components are different, the ranks are well defined, with values in 1:n where n <- length(x) and we assume no NAs for the moment. Otherwise, with some values equal, called 'ties', the argument `ties.method` determines the result at the corresponding indices. The "first" method results in a permutation with increasing values at each index set of ties. The "random" method puts these in random order whereas the default, "average", replaces them by their mean, and "max" and "min" replaces them by their maximum and minimum respectively, the latter being the typical "sports" ranking.

**References**


**See Also**

`order` and `sort`. 
Examples

```
(r1 <- rank(x1 <- c(3, 1, 4, 15, 92)))
x2 <- c(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5)
names(x2) <- letters[1:11]
(r2 <- rank(x2)) # ties are averaged

## rank() is "idempotent": rank(rank(x)) == rank(x)
stopifnot(rank(r1) == r1, rank(r2) == r2)

## ranks without averaging
rank(x2, ties.method= "first") # first occurrence wins
rank(x2, ties.method= "random") # ties broken at random
rank(x2, ties.method= "random") # and again

## keep ties ties, no average
(rma <- rank(x2, ties.method= "max")) # as used classically
(rmi <- rank(x2, ties.method= "min")) # as in Sports
stopifnot(rma + rmi == round(r2 + r2))
```

### raw

**Raw Vectors**

**Description**

Creates or tests for objects of type "raw".

**Usage**

```
raw(length = 0)

as.raw(x)
```

**Arguments**

- `length` desired length.
- `x` object to be coerced.

**Details**

The raw type is intended to hold raw bytes. It is possible to extract subsequences of bytes, and to replace elements (but only by elements of a raw vector). The relational operators (see `Comparison`) work, as do the logical operators (see `Logic`) with a bitwise interpretation.

A raw vector is printed with each byte separately represented as a pair of hex digits. If you want to see a character representation (with escape sequences for non-printing characters) use `rawToChar`.

Coercion to raw treats the input values as representing a small (decimal) integers, so the input is first coerced to integer, and then values which are outside the range `[0 \ldots 255]` or are NA are set to 0 (the null byte).

**Value**

- `raw` creates a raw vector of the specified length. Each element of the vector is equal to 0. Raw vectors are used to store fixed-length sequences of bytes.
- `as.raw` attempts to coerce its argument to be of raw type. The (elementwise) answer will be 0 unless the coercion succeeds.
See Also

charToRaw, rawShift, etc.

Examples

```r
xx <- raw(2)
xx[1] <- as.raw(40) # NB, not just 40.
xx[2] <- charToRaw("A")
xx

x <- "A test string"
(y <- charToRaw(x))
is.vector(y) # TRUE
rawToChar(y)

isASCII <- function(txt) all(charToRaw(txt) <= as.raw(127))
isASCII(x) # true
isASCII("\x9c25.63") # false (in Latin-1, this is an amount in UK pounds)
```

Description

Conversion and manipulation of objects of type "raw".

Usage

```r
charToRaw(x)
rawToChar(x, multiple = FALSE)
rawShift(x, n)
rawToBits(x)
intToBits(x)
packBits(x, type = c("raw", "integer"))
```

Arguments

- `x`: object to be converted or shifted.
- `multiple`: logical: should the conversion be to a single character string or multiple individual characters?
- `n`: the number of bits to shift. Positive numbers shift right and negative numbers shift left: allowed values are \(-8 \ldots 8\).
- `type`: the result type.

Details

`packBits` accepts raw, integer or logical inputs, the last two without any NAs.
Value

charToRaw converts a length-one character string to raw bytes.
rawToChar converts raw bytes either to a single character string or a character vector of single bytes. (Note that a single character string could contain embedded nuls.)
rawToBits returns a raw vector of 8 times the length of a raw vector with entries 0 or 1.
intToBits returns a raw vector of 32 times the length of an integer vector with entries 0 or 1. In both cases the unpacking is least-significant bit first.
packbits packs its input (using only the lowest bit for raw or integer vectors) least-significant bit first to a raw or integer vector.

Examples

x <- "A test string"
(y <- charToRaw(x))
is.vector(y) # TRUE

rawToChar(y)
rawToChar(y, multiple = TRUE)
(xx <- c(y, as.raw(0), charToRaw("more")))
rawToChar(xx)

xxx <- xx
xxx[length(y)+1] <- charToRaw("&")
xxx

rawToBits(y)

rawShift(y, 1)
rawShift(y, -2)

RdUtils

Utilities for Processing Rd Files

Description

Utilities for converting files in R documentation (Rd) format to other formats or create indices from them, and for converting documentation in other formats to Rd format.

Usage

R CMD Rdconv [options] file
R CMD Rd2dvi.sh [options] files
R CMD Rd2txt [options] file
R CMD Sd2Rd [options] file

Arguments

file the path to a file to be processed.
files a list of file names specifying the R documentation sources to use, by either giving the paths to the files, or the path to a directory with the sources of a package.
options further options to control the processing, or for obtaining information about usage and version of the utility.

Details

Rdconv converts Rd format to other formats. Currently, plain text, HTML, LaTeX, S version 3 (Sd), and S version 4 (.sgml) formats are supported. It can also extract the examples for run-time testing.

Rd2dvi and Rd2txt are user-level programs for producing DVI/PDF output or pretty text output from Rd sources.

Sd2Rd converts S (version 3 or 4) documentation formats to Rd format.

Use R CMD foo --help to obtain usage information on utility foo.

Note

Conversion to S version 3/4 formats is rough: there are some .Rd constructs for which there is no natural analogue. They are intended as a starting point for hand-tuning.

See Also

The chapter “Processing Rd format” in “Writing R Extensions” (see the Manuals sub-menu of the Help menu on the console).

---

read.table Data Input

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```r
read.table(file, header = FALSE, sep = "", quote = "\\\"", dec = ".", row.names, col.names, as.is = FALSE, na.strings = "NA", colClasses = NA, nrows = -1, skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE, comment.char = "\#", allowEscapes = FALSE)
```

```r
read.csv(file, header = TRUE, sep = ",", quote="\\\"", dec=".", fill = TRUE, ...)
```

```r
read.csv2(file, header = TRUE, sep = ";", quote="\\\"", dec=".", fill = TRUE, ...)
```

```r
read.delim(file, header = TRUE, sep = "\t", quote="\\\"", dec=".", fill = TRUE, ...)
```

```r
read.delim2(file, header = TRUE, sep = "\t", quote="\\\"", dec=".", fill = TRUE, ...)
```
read.table

Arguments

file  the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory, getwd(). Tilde-expansion is performed where supported. Alternatively, file can be a connection, which will be opened if necessary, and if so closed at the end of the function call. (If stdin() is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, Ctrl-D on Unix and Ctrl-Z on Windows. Any pushback on stdin() will be cleared before return.) file can also be a complete URL.

header  a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if and only if the first row contains one fewer field than the number of columns.

sep  the field separator character. Values on each line of the file are separated by this character. If sep = "" (the default for read.table) the separator is "white space", that is one or more spaces, tabs, newlines or carriage returns.

quote  the set of quoting characters. To disable quoting altogether, use quote = "". See scan for the behaviour on quotes embedded in quotes.

dec  the character used in the file for decimal points.

row.names  a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names. If there is a header and the first row contains one fewer field than the number of columns, the first column in the input is used for the row names. Otherwise if row.names is missing, the rows are numbered. Using row.names = NULL forces row numbering.

col.names  a vector of optional names for the variables. The default is to use "V" followed by the column number.

as.is  the default behavior of read.table is to convert character variables (which are not converted to logical, numeric or complex) to factors. The variable as.is controls the conversion of columns not otherwise specified by colClasses. Its value is either a vector of logicals (values are recycled if necessary), or a vector of numeric or character indices which specify which columns should not be converted to factors.

Note: to suppress all conversions including those of numeric columns, set colClasses = "character".

Note that as.is is specified per column (not per variable) and so includes the column of row names (if any) and any columns to be skipped.

na.strings  a character vector of strings which are to be interpreted as NA values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields.

colClasses  character. A vector of classes to be assumed for the columns. Recycled as necessary, or if the character vector is named, unspecified values are taken to be NA.
Possible values are NA (when type.convert is used), "NULL" (when the column is skipped), one of the atomic vector classes (logical, integer, numeric, complex, character, raw), or "factor", "Date" or "POSIXct". Otherwise there needs to be an as method (from package methods) for conversion from "character" to the specified formal class.

Note that colClasses is specified per column (not per variable) and so includes the column of row names (if any).

nrows the maximum number of rows to read in. Negative values are ignored.

skip the number of lines of the data file to skip before beginning to read data.

check.names logical. If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.

fill logical. If TRUE then in case the rows have unequal length, blank fields are implicitly added. See Details.

strip.white logical. Used only when sep has been specified, and allows the stripping of leading and trailing white space from character fields (numeric fields are always stripped). See scan for further details, remembering that the columns may include the row names.

blank.lines.skip logical: if TRUE blank lines in the input are ignored.

comment.char character: a character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.

allowEscapes logical. Should C-style escapes such as \n be processed or read verbatim (the default)? Note that if not within quotes these could be interpreted as a delimiter (but not as a comment character). For more details see scan.

... Further arguments to read.table.

Details

A field or line is ‘blank’ if it contains nothing (except whitespace is no separator is specified) before a comment character or the end of the field or line.

If row.names is not specified and the header line has one less entry than the number of columns, the first column is taken to be the row names. This allows data frames to be read in from the format in which they are printed. If row.names is specified and does not refer to the first column, that column is discarded from such files.

The number of data columns is determined by looking at the first five lines of input (or the whole file if it has less than five lines), or from the length of col.names if it is specified and is longer. This could conceivably be wrong if fill or blank.lines.skip are true, so specify col.names if necessary.

read.csv and read.csv2 are identical to read.table except for the defaults. They are intended for reading “comma separated value” files (\'.CSV\') or (read.csv2) the variant used in countries that use a comma as decimal point and a semicolon as field separator. Similarly, read.delim and read.delim2 are for reading delimited files, defaulting to the TAB character for the delimiter. Notice that header = TRUE and fill = TRUE in these variants.

The rest of the line after a comment character is skipped; quotes are not processed in comments. Complete comment lines are allowed provided blank.lines.skip = TRUE; however, comment lines prior to the header must have the comment character in the first non-blank column.
As from R 1.9.0 quoted fields with embedded newlines are supported except after a comment character.

Value

A data frame (data.frame) containing a representation of the data in the file. Empty input is an error unless col.names is specified, when a 0-row data frame is returned: similarly giving just a header line if header = TRUE results in a 0-row data frame.

This function is the principal means of reading tabular data into R.

Note

The columns referred to in as.is and colClasses include the column of row names (if any). Less memory will be used if colClasses is specified as one of the six atomic vector classes.

Using nrow, even as a mild over-estimate, will help memory usage.

Using comment.char = "" will be appreciably faster.

read.table is not the right tool for reading large matrices, especially those with many columns: it is designed to read data frames which may have columns of very different classes. Use scan instead.

Prior to version 1.9.0, underscores were not valid in variable names, and code that relies on them being converted to dots will no longer work. The simplest workaround is to use names(d) <- gsub("_", ".", names(d)), or, avoiding the (small) risk of creating duplicate names, names(d) <- make.names(gsub("_", ".", names(d)), unique=TRUE).

References


See Also

The R Data Import/Export manual.

scan, type.convert, read.fwf for reading fixed width formatted input; write.table; data.frame.

count.fields can be useful to determine problems with reading files which result in reports of incorrect record lengths.

---

**readBin**

Transfer Binary Data To and From Connections

Description

Read binary data from a connection, or write binary data to a connection.

Usage

readBin(con, what, n = 1, size = NA, signed = TRUE, endian = .Platform$endian)

writeBin(object, con, size = NA, endian = .Platform$endian)
readBin

Arguments

con A connection object or a character string naming a file or a raw vector.

what Either an object whose mode will give the mode of the vector to be read, or a character vector of length one describing the mode: one of "numeric", "double", "integer", "int", "logical", "complex", "character", "raw".

n integer. The (maximal) number of records to be read. You can use an overestimate here, but not too large as storage is reserved for \(n\) items.

size integer. The number of bytes per element in the byte stream. The default, NA, uses the natural size. Size changing is not supported for raw and complex vectors.

signed logical. Only used for integers of sizes 1 and 2, when it determines if the quantity on file should be regarded as a signed or unsigned integer.

deian The endian-ness ("big" or "little" of the target system for the file. Using "swap" will force swapping endian-ness.

object An R object to be written to the connection.

Details

If con is a character string, the functions call file to obtain an file connection which is opened for the duration of the function call.

If the connection is open it is read/written from its current position. If it is not open, it is opened for the duration of the call and then closed again. The connection must be open or open-able in binary mode.

If readBin is called con with a raw vector, the data in the vector is used as input. If writeBin is called with con a raw vector, it is just an indication that a raw vector should be returned.

If size is specified and not the natural size of the object, each element of the vector is coerced to an appropriate type before being written or as it is read. Possible sizes are 1, 2, 4 and possibly 8 for integer or logical vectors, and 4, 8 and possibly 12/16 for numeric vectors. (Note that coercion occurs as signed types except if signed = FALSE when reading integers of sizes 1 and 2.) Changing sizes is unlikely to preserve NAs, and the extended precision sizes are unlikely to be portable across platforms.

readBin and writeBin read and write C-style zero-terminated character strings. Input strings are limited to 10000 characters. readChar and code(writeChar)

Handling R’s missing and special (Inf, -Inf and NaN) values is discussed in the R Data Import/Export manual.

Value

For readBin, a vector of appropriate mode and length the number of items read (which might be less than \(n\)).

For writeBin, a raw vector (if con is a raw vector) or invisibly NULL.

Note

Integer read/writes of size 8 will be available if either C type long is of size 8 bytes or C type long long exists and is of size 8 bytes.
Real read/writes of size \texttt{sizeof (long double)} (usually 12 or 16 bytes) will be available only if that type is available and different from \texttt{double}.

If \texttt{readBin(what = character())} is used incorrectly on a file which does not contain C-style character strings, warnings (usually many) are given. From a file or connection, the input will be broken into pieces of length 10000 with any final part being discarded.

See Also

The \textit{R Data Import/Export} manual.

\texttt{readChar} to read/write fixed-length strings.

\texttt{connections, readLines, writeLines}.

\texttt{.Machine} for the sizes of \texttt{long}, \texttt{long long} and \texttt{long double}.

Examples

\begin{verbatim}
zz <- file("testbin", "wb")
writeBin(1:10, zz)
writeBin(pi, zz, endian="swap")
writeBin(pi^2, zz, size=4)
writeBin(pi+3i, zz)
writeBin("A test of a connection", zz)
z <- paste("A very long string", 1:100, collapse=" + ")
writeBin(z, zz)
if(.Machine$sizeof.long == 8 || .Machine$sizeof.longlong == 8)
  writeBin(as.integer(5^(1:10)), zz, size = 8)
if((s <-.Machine$sizeof.longdouble) > 8)
  writeBin((pi/3)^(1:10), zz, size = s)
close(zz)

zz <- file("testbin", "rb")
readBin(zz, integer(), 4)
readBin(zz, integer(), 6)
readBin(zz, numeric(), 1, endian="swap")
readBin(zz, numeric(), size=4)
readBin(zz, numeric(), size=4, endian="swap")
readBin(zz, complex(), 1)
readBin(zz, character(), 1)
z2 <- readBin(zz, character(), 1)
if(.Machine$sizeof.long == 8 || .Machine$sizeof.longlong == 8)
  readBin(zz, integer(), 10, size = 8)
if((s <-.Machine$sizeof.longdouble) > 8)
  readBin(zz, numeric(), 10, size = s)
close(zz)
unlink("testbin")
stopifnot(z2 == z)

## signed vs unsigned ints
zz <- file("testbin", "wb")
x <- as.integer(seq(0, 255, 32))
writeBin(x, zz, size=1)
writeBin(x, zz, size=1)
x <- as.integer(seq(0, 60000, 10000))
writeBin(x, zz, size=2)
writeBin(x, zz, size=2)
\end{verbatim}
close(zz)
zz <- file("testbin", "rb")
readBin(zz, integer(), 8, size=1)
readBin(zz, integer(), 8, size=1, signed=FALSE)
readBin(zz, integer(), 7, size=2)
readBin(zz, integer(), 7, size=2, signed=FALSE)
close(zz)
unlink("testbin")

## use of raw
z <- writeBin(pi^1:5, raw(), size = 4)
readBin(z, numeric(), 5, size = 4)
z <- writeBin(c("a", "test", "of", "character"), raw())
rawToChar(z)
readBin(z, character(), 4)

readChar

Transfer Character Strings To and From Connections

Description
Transfer character strings to and from connections, without assuming they are null-terminated on the connection.

Usage
readChar(con, nchars)
writeChar(object, con, nchars = nchar(object, type="chars"), eos = "")

Arguments

- con: A connection object or a character string naming a file.
- nchars: integer, giving the lengths in characters of (unterminated) character strings to be read or written.
- object: A character vector to be written to the connection.
- eos: 'end of string': character string. The terminator to be written after each string, followed by an ASCII null; use NULL for no terminator at all.

Details
These functions complement readBin and writeBin which read and write C-style zero-terminated character strings. They are for strings of known length, and can optionally write an end-of-string mark.

If con is a character string, the functions call file to obtain an file connection which is opened for the duration of the function call.

If the connection is open it is read/written from its current position. If it is not open, it is opened for the duration of the call and then closed again. Connections can be open in either text or binary mode.
Character strings containing ASCII nul(s) will be read correctly by and appear with embedded nuls in the character vector returned.

If the character length requested for readChar is longer than the data available on the connection, what is available is returned. For writeChar if too many characters are requested the output is zero-padded, with a warning.

Missing strings are written as NA.

Value

For readChar, a character vector of length the number of items read (which might be less than length(nchars)).

For writeChar none (strictly, invisible NULL).

See Also

The R Data Import/Export manual.

connections, readLines, writeLines, readBin

Examples

```r
## test fixed-length strings
zz <- file("testchar", "wb")
x <- c("a", "this will be truncated", "abc")
nc <- c(3, 10, 3)
writeChar(x, zz, nc, eos=NULL)
writeChar(x, zz, eos="\r\n")
close(zz)

zz <- file("testchar", "rb")
readChar(zz, nc)
readChar(zz, nchar(x)+3) # need to read the terminator explicitly
close(zz)
unlink("testbin")
```

---

**Readline**

**Read a Line from the Terminal**

Description

Readline reads a line from the terminal

Usage

```r
readline(prompt = "")
```

Arguments

- `prompt` the string printed when prompting the user for input. Should usually end with a space " ".

Details

The prompt string will be truncated to a maximum allowed length, normally 256 chars (but can be changed in the source code).

Value

A character vector of length one.

See Also

readLines for reading text lines of connections, including files.

Examples

fun <- function() {
  ANSWER <- readline("Are you a satisfied R user? ")
  if (substr(ANSWER, 1, 1) == "n")
    cat("This is impossible. YOU LIED!\n")
  else
    cat("I knew it.\n")
}
fun()

readLines

Read Text Lines from a Connection

Description

Read text lines from a connection.

Usage

readLines(con = stdin(), n = -1, ok = TRUE)

Arguments

con A connection object or a character string.
n integer. The (maximal) number of lines to read. Negative values indicate that one should read up to the end of the connection.
ok logical. Is it OK to reach the end of the connection before \n lines are read? If not, an error will be generated.

Details

If the con is a character string, the functions call file to obtain an file connection which is opened for the duration of the function call.

If the connection is open it is read from its current position. If it is not open, it is opened for the duration of the call and then closed again.

If the final line is incomplete (no final EOL marker) the behaviour depends on whether the connection is blocking or not. For a blocking text-mode connection (or a non-text-mode connection) the
line will be accepted, with a warning. For a non-blocking text-mode connection the incomplete line is pushed back, silently.

Whatever mode the connection is opened in, any of LF, CRLF or CR will be accepted as the EOL marker for a line.

Value

A character vector of length the number of lines read.

See Also
connections, writeLines, readBin, scan

Examples

```r
cat("TITLE extra line", "2 3 5 7", ",", "11 13 17", file="ex.data", 
   sep="\n")
readLines("ex.data", n=-1)
unlink("ex.data") # tidy up

## difference in blocking

con <- file("test1", "r", blocking = FALSE)
readLines(con) # empty

con <- file("test1", "r", blocking = FALSE)
readLines(con) # gets both
```

real

Real Vectors

description
real creates a double precision vector of the specified length. Each element of the vector is equal to 0.

as.real attempts to coerce its argument to be of real type.

is.real returns TRUE or FALSE depending on whether its argument is of real type or not.

Usage

real(length = 0)

as.real(x, ...)

is.real(x)

Arguments

length desired length.

x object to be coerced or tested.

... further arguments passed to or from other methods.
Note

\$\mathcal{R}\$ has no single precision data type. All real numbers are stored in double precision format.

---

**Recall**

*Recursive Calling*

**Description**

Recall is used as a placeholder for the name of the function in which it is called. It allows the definition of recursive functions which still work after being renamed, see example below.

**Usage**

Recall(...)  

**Arguments**

... all the arguments to be passed.

**Note**

Recall will not work correctly when passed as a function argument, eg to the apply family of functions.

**See Also**

do.call and call.

local for another way to write anonymous recursive functions

**Examples**

## A trivial (but inefficient!) example:
fib <- function(n) if(n<=2) {if(n>=0) 1 else 0} else Recall(n-1) + Recall(n-2)  
fibonacci <- fib; rm(fib)  
## renaming wouldn't work without Recall  
fibonacci(10) # 55

---

**reg.finalizer**

*Finalization of objects*

**Description**

Registers an \$\mathcal{R}\$ function to be called upon garbage collection of object.

**Usage**

reg.finalizer(e, f)
Arguments

e Object to finalize. Must be environment or external pointer.
f Function to call on finalization. Must accept a single argument, which will be the object to finalize.

Value

NULL.

Note

The purpose of this function is mainly to allow objects that refer to external items (a temporary file, say) to perform cleanup actions when they are no longer referenced from within R. This only makes sense for objects that are never copied on assignment, hence the restriction to environments and external pointers.

See Also

gc and Memory for garbage collection and memory management.

Examples

f <- function(e) print("cleaning....")
g <- function(x){ e <- environment(); reg.finalizer(e,f) }
g()
invisible(gc()) # trigger cleanup

regex

Regular Expressions as used in R

Description

This help page documents the regular expression patterns supported by grep and related functions regexpr, gregexpr, sub and gsub, as well as by strsplit.

Details

A ‘regular expression’ is a pattern that describes a set of strings. Three types of regular expressions are used in R, extended regular expressions, used by grep(extended = TRUE) (its default), basic regular expressions, as used by grep(extended = FALSE), and Perl-like regular expressions used by grep(perl = TRUE).

Other functions which use regular expressions (often via the use of grep) include apropos, browseEnv, help.search, list.files, ls and strsplit. These will all use extended regular expressions, unless strsplit is called with argument extended = FALSE or perl = TRUE.

Patterns are described here as they would be printed by cat: do remember that backslashes need to be doubled in entering R character strings from the keyboard.
Extended Regular Expressions

This section covers the regular expressions allowed if `extended = TRUE` in `grep`, `regexpr`, `gregexpr`, `sub`, `gsub` and `strsplit`. They use the `glibc 2.3.3` implementation of the POSIX 1003.2 standard.

Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions.

The fundamental building blocks are the regular expressions that match a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any metacharacter with special meaning may be quoted by preceding it with a backslash. The metacharacters are `. ` `|` `(` `)` `[` `{` `^` `$` `*` `+` `?`.

A character class is a list of characters enclosed by `[` and `]` matches any single character in that list; if the first character of the list is the caret `^`, then it matches any character not in the list. For example, the regular expression `[0123456789]` matches any single digit, and `[^abc]` matches anything except the characters a, b or c. A range of characters may be specified by giving the first and last characters, separated by a hyphen. (Character ranges are interpreted in the collation order of the current locale.)

Certain named classes of characters are predefined. Their interpretation depends on the `locale` (see `locales`); the interpretation below is that of the POSIX locale.

`[:alnum:]` Alphanumeric characters: `[:alpha:]` and `[:digit:]`.
`[:alpha:]` Alphabetic characters: `[:lower:]` and `[:upper:]`.
`[:blank:]` Blank characters: space and tab.
`[:cntrl:]` Control characters. In ASCII, these characters have octal codes 000 through 037, and 177 (DEL). In another character set, these are the equivalent characters, if any.
`[:digit:]` Digits: 0 1 2 3 4 5 6 7 8 9.
`[:graph:]` Graphical characters: `[:alnum:]` and `[:punct:]`.
`[:lower:]` Lower-case letters in the current locale.
`[:print:]` Printable characters: `[:alnum:]`, `[:punct:]` and space.
`[:punct:]` Punctuation characters: ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~.
`[:space:]` Space characters: tab, newline, vertical tab, form feed, carriage return, and space.
`[:upper:]` Upper-case letters in the current locale.
`[:xdigit:]` Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f.

For example, `[:alnum:]` means `[0-9A-Za-z]`, except the latter depends upon the locale and the character encoding, whereas the former is independent of locale and character set. (Note that the brackets in these class names are part of the symbolic names, and must be included in addition to the brackets delimiting the bracket list.) Most metacharacters lose their special meaning inside lists. To include a literal `]`, place it first in the list. Similarly, to include a literal `^`, place it anywhere but first. Finally, to include a literal `~`, place it first or last. (Only these and `\` remain special inside character classes.)

The period `.` matches any single character. The symbol `\w` is documented to be synonym for `[:alnum:]` and `\W` is its negation. However, `\w` also matches underscore in the GNU grep code used in R.

The caret `^` and the dollar sign `$` are metacharacters that respectively match the empty string at the beginning and end of a line. The symbols `<` and `>` respectively match the empty string at the
beginning and end of a word. The symbol \b matches the empty string at the edge of a word, and \B matches the empty string provided it is not at the edge of a word.

A regular expression may be followed by one of several repetition quantifiers:

? The preceding item is optional and will be matched at most once.
* The preceding item will be matched zero or more times.
+ The preceding item will be matched one or more times.
{n} The preceding item is matched exactly n times.
{n,} The preceding item is matched n or more times.
{n,m} The preceding item is matched at least n times, but not more than m times.

Repetition is greedy, so the maximal possible number of repeats is used.

Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated subexpressions.

Two regular expressions may be joined by the infix operator |; the resulting regular expression matches any string matching either subexpression. For example, abba|cde matches either the string abba or the string cde. Note that alternation does not work inside character classes, where | has its literal meaning.

Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole subexpression may be enclosed in parentheses to override these precedence rules.

The backreference \N, where N is a single digit, matches the substring previously matched by the Nth parenthesized subexpression of the regular expression.

Before R 2.1.0 R attempted to support traditional usage by assuming that { is not special if it would be the start of an invalid interval specification. (POSIX allows this behaviour as an extension but we no longer support it.)

Basic Regular Expressions

This section covers the regular expressions allowed if extended = FALSE in grep, regexpr, gregexpr, sub, gsub and strsplit.

In basic regular expressions the metacharacters ?, +, {, |, (, and ) lose their special meaning; instead use the backslashed versions \?, \+, \, \|, \(, \), and \). Thus the metacharacters are . \[ ^ $ *.

Perl Regular Expressions

The perl = TRUE argument to grep, regexpr, gregexpr, sub, gsub and strsplit switches to the PCRE library that ‘implements regular expression pattern matching using the same syntax and semantics as Perl 5.6 or later, with just a few differences’.

For complete details please consult the man pages for PCRE, especially man pcrepattern and man pcreapi) on your system or from the sources at ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/. If PCRE support was compiled from the sources within R, the PCRE version is 4.5 as described here (version ≥ 4.0 is required even if R is configured to use the system’s PCRE library).

All the regular expressions described for extended regular expressions are accepted except \< and \\; in Perl all backslashed metacharacters are alphanumeric and backslashed symbols always are interpreted as a literal character. { is not special if it would be the start of an invalid interval specification. There can be more than 9 backreferences.
The construct (?...) is used for Perl extensions in a variety of ways depending on what immediately follows the ?. Perl-like matching can work in several modes, set by the options (?i) (caseless, equivalent to Perl's /i), (?m) (multiline, equivalent to Perl's /m), (?s) (single line, so a dot matches all characters, even new lines: equivalent to Perl's /s) and (?x) (extended, whitespace data characters are ignored unless escaped and comments are allowed: equivalent to Perl's /x). These can be concatenated, so for example, (?im) sets caseless multiline matching. It is also possible to unset these options by preceding the letter with a hyphen, and to combine setting and unsetting such as (?im-sx). These settings can be applied within patterns, and then apply to the remainder of the pattern. Additional options not in Perl include (?U) to set 'ungreedy' mode (so matching is minimal unless ? is used, when it is greedy). Initially none of these options are set.

If you want to remove the special meaning from a sequence of characters, you can do so by putting them between \Q and \E. This is different from Perl in that $ and @ are handled as literals in \Q...\E sequences in PCRE, whereas in Perl, $ and @ cause variable interpolation.

The escape sequences \d, \s and \w represent any decimal digit, space character and 'word' character (letter, digit or underscore in the current locale) respectively, and their upper-case versions represent their negation. Unlike POSIX and earlier versions of Perl and PCRE, vertical tab is not regarded as a whitespace character.

Escape sequence \a is BEL, \e is ESC, \f is LF, \n is LF and \r is CR and \t is TAB. In addition \cx is cntrl-x for any x, \ddd is the octal character ddd (for up to three digits unless interpretable as a backreference), and \xhh specifies a character in hex.

Outside a character class, \b matches a word boundary, \B is its negation, \A matches at start of a subject (even in multiline mode, unlike ^), \z matches at end of a subject or before newline at end, \Z matches at end of a subject. and \G matches at first matching position in a subject. \C matches a single byte, including a newline.

The same repetition quantifiers as extended POSIX are supported. However, if a quantifier is followed by ?, the match is 'ungreedy', that is as short as possible rather than as long as possible (unless the meanings are reversed by the (?U) option.)

The sequence (?# marks the start of a comment which continues up to the next closing parenthesis. Nested parentheses are not permitted. The characters that make up a comment play no part at all in the pattern matching.

If the extended option is set, an unescaped # character outside a character class introduces a comment that continues up to the next newline character in the pattern.

The pattern (?:...) groups characters just as parentheses do but does not make a backreference. Patterns (?=...) and (?) are zero-width positive and negative lookahead assertions: they match if an attempt to match the ... forward from the current position would succeed (or not), but use up no characters in the string being processed. Patterns (?<=...) and (?) are the lookbehind equivalents: they do not allow repetition quantifiers nor \C in ... .

Named subpatterns, atomic grouping, possessive qualifiers and conditional and recursive patterns are not covered here.

Note

Prior to R 2.1.0 the implementation used was that of GNU grep 2.4.2: as from R 2.1.0 it is that of glibc 2.3.3. The latter is more strictly compliant and rejects some extensions that used to be allowed.

The change was made both because bugs were becoming apparent in the previous code and to allow support of multibyte character sets.
remove

Author(s)

This help page is based on the documentation of GNU grep 2.4.2 (from which the C code used by R used to be taken) the pcre man page from PCRE 3.9 and the pcrepattern man page from PCRE 4.4.

See Also
grep, apropos, browseEnv, help.search, list.files, ls and strsplit.
http://www.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap09.html

remove  Remove Objects from a Specified Environment

Description

remove and rm can be used to remove objects. These can be specified successively as character strings, or in the character vector list, or through a combination of both. All objects thus specified will be removed.

If envir is NULL then the currently active environment is searched first.

If inherits is TRUE then parents of the supplied directory are searched until a variable with the given name is encountered. A warning is printed for each variable that is not found.

Usage

remove(..., list = character(0), pos = -1, envir = as.environment(pos), inherits = FALSE)
rm  (... , list = character(0), pos = -1, envir = as.environment(pos),
     inherits = FALSE)

Arguments

... the objects to be removed, supplied individually and/or as a character vector
list a character vector naming objects to be removed.
pos where to do the removal. By default, uses the current environment. See the details for other possibilities.
envir the environment to use. See the details section.
inherits should the enclosing frames of the environment be inspected?

Details

The pos argument can specify the environment from which to remove the objects in any of several ways: as an integer (the position in the search list); as the character string name of an element in the search list; or as an environment (including using sys.frame to access the currently active function calls). The envir argument is an alternative way to specify an environment, but is primarily there for back compatibility.
References


See Also

`ls`, `objects`

Examples

tmp <- 1:4
## work with tmp and cleanup
rm(tmp)

## Not run:
## remove (almost) everything in the working environment.
## You will get no warning, so don't do this unless you are really sure.
rm(list = ls())
## End(Not run)

---

**Replicate Elements of Vectors and Lists**

Description

`rep` replicates the values in `x`. It is a generic function, and the default method is described here. `rep.int` is a faster simplified version for the commonest case.

Usage

```r
rep(x, times, ...)
```

## Default S3 method:
`rep(x, times, length.out, each, ...)
```

```r
rep.int(x, times)
```

Arguments

- `x` a vector (of any mode including a list) or a pairlist or a `POSIXct` or `POSIXlt` or `date` object.
- `times` optional non-negative integer. A vector giving the number of times to repeat each element if of length `length(x)`, or to repeat the whole vector if of length 1.
- `length.out` optional integer. The desired length of the output vector.
- `each` optional integer. Each element of `x` is repeated `each` times.
- `...` further arguments to be passed to or from other methods.
Details

A least one of times, length.out and each must be specified, and normally exactly one is. If length.out is given, times is ignored. If each is specified with either of the other two, its replication is performed first, and then that implied by times or length.out.

If times consists of a single integer, the result consists of the values in x repeated this many times. If times is a vector of the same length as x, the result consists of x[1] repeated times[1] times, x[2] repeated times[2] times and so on.

length.out may be given in place of times, in which case x is repeated as many times as is necessary to create a vector of this length.

Non-integer values of times will be truncated towards zero. If times is a computed quantity it is prudent to add a small fuzz.

If x has length zero and length.out is supplied and is positive, the values are filled in using the extraction rules, that is by an NA of the appropriate class for an atomic vector (0 for raw vectors) and NULL for a list.

Value

A vector of the same class as x.

Note

If the original vector has names, these are also replicated and so will almost always contain duplicates. (In contrast, S strips the names.) Function rep.int is a simple case handled by internal code, and provided as a separate function purely for S compatibility.

References


See Also

seq, sequence.

Examples

rep(1:4, 2)
rep(1:4, each = 2) # not the same.
rep(1:4, c(2,2,2,2)) # same as second.
rep(1:4, c(2,1,2,1))
rep(1:4, each = 2, len = 4) # first 4 only.
rep(1:4, each = 2, len = 10) # 8 integers plus two recycled 1's.
rep(1:4, each = 2, times = 3) # length 24, 3 complete replications

rep(1, 40*1-.8)) # length 7 on most platforms
rep(1, 40*1-.8+1e-7) # better

## replicate a list
fred <- list(happy = 1:10, name = "squash")
rep(fred, 5)

# date-time objects
replace

Replace Values in a Vector

Description

replace replaces the values in x with indexes given in list by those given in values. If necessary, the values in values are recycled.

Usage

replace(x, list, values)

Arguments

x
vector

list
an index vector

values
replacement values

Value

A vector with the values replaced.

Note

x is unchanged: remember to assign the result.

References


rev

Reverse Elements

Description

rev provides a reversed version of its argument. It is generic function with a default method for vectors and one for dendrograms.

Note that this is no longer needed (nor efficient) for obtaining vectors sorted into descending order, since that is now rather more directly achievable by sort(x, decreasing = TRUE).

Usage

rev(x)
**Arguments**

The function `rle()` takes a vector `x` as its argument, which can be a vector or another object for which reversal is defined.

**References**


**See Also**

`seq`, `sort`.

**Examples**

```r
x <- c(1:5,5:3)
## sort into descending order; first more efficiently:
stopifnot(sort(x, decreasing = TRUE) == rev(sort(x)))
stopifnot(rev(1:7) == 7:1)#- don't need 'rev' here
```

---

**rle**

*Run Length Encoding*

**Description**

Compute the lengths and values of runs of equal values in a vector – or the reverse operation.

**Usage**

```r
rle(x)
inverse.rle(x, ...)
```

**Arguments**

- `x`: a simple vector for `rle()` or an object of class "rle" for `inverse.rle()`.
- `...`: further arguments which are ignored in R.

**Value**

`rle()` returns an object of class "rle" which is a list with components

- `lengths`: an integer vector containing the length of each run.
- `values`: a vector of the same length as `lengths` with the corresponding values.

`inverse.rle()` is the inverse function of `rle()`.
Examples

```r
x <- rev(rep(6:10, 1:5))
rle(x)
## lengths [1:5] 5 4 3 2 1
## values [1:5] 10 9 8 7 6

z <- c(TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE, TRUE, TRUE)
rle(z)
rle(as.character(z))

stopifnot(x == inverse.rle(rle(x)),
          z == inverse.rle(rle(z)))
```

Round

Rounding of Numbers

Description

`ceiling` takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `x`.

`floor` takes a single numeric argument `x` and returns a numeric vector containing the largest integers not greater than the corresponding elements of `x`.

`trunc` takes a single numeric argument `x` and returns a numeric vector containing the integers formed by truncating the values in `x` toward 0.

`round` rounds the values in its first argument to the specified number of decimal places (default 0).

`signif` rounds the values in its first argument to the specified number of significant digits.

`zapsmall` determines a `digits` argument `dr` for calling `round(x, digits = dr)` such that values “close to zero” (compared with the maximal absolute value) are “zapped”, i.e., treated as 0.

Usage

```r
ceiling(x)
floor(x)
trunc(x)

round(x, digits = 0)
signif(x, digits = 6)
zapsmall(x, digits =getOption("digits"))
```

Arguments

- `x` a numeric vector. A complex vector is allowed for `round`, `signif` and `zapsmall`.
- `digits` integer indicating the precision to be used.
Details

All but `zapsmall` are generic functions: methods can be defined for them individually or via the `Math` group generic.

Note that for rounding off a 5, the IEC 60559 standard is used, "go to the even digit". Therefore `round(0.5)` is 0 and `round(-1.5)` is -2.

For `signif` the recognized values of `digits` are 1...22. As from R 2.2.0 complex numbers are rounded to retain the specified number of digits in the larger of the components. Each element of the vector is rounded individually, unlike printing.

References


See Also

`as.integer`.

Examples

```r
round(.5 + -2:4) # IEEE rounding: -2 0 0 2 2 4 4
(x1 <- seq(-2, 4, by = .5))
round(x1)#-- IEEE rounding !
x1[trunc(x1) != floor(x1)]
x1[round(x1) != floor(x1 + .5)]
(non.int <- ceiling(x1) != floor(x1))

x2 <- pi * 100^(-1:3)
round(x2, 3)
signif(x2, 3)

print (x2 / 1000, digits=4)
zapsmall(x2 / 1000, digits=4)
zapsmall(exp(1i*0:4*pi/2))
```

---

**round.POSIXt**

Round / Truncate Data-Time Objects

Description

Round or truncate date-time objects.

Usage

```r
## S3 method for class 'POSIXt':
round(x, units = c("secs", "mins", "hours", "days"))
## S3 method for class 'POSIXt':
trunc(x, units = c("secs", "mins", "hours", "days"))
```
### S3 method for class 'Date':

```r
round(x, ...)  
```

### S3 method for class 'Date':

```r
trunc(x)
```

#### Arguments

- `x`: an object inheriting from "POSIXt" or "Date".
- `units`: one of the units listed. Can be abbreviated.
- `...`: arguments to be passed to or from other methods, notably `digits`.

#### Details

The time is rounded or truncated to the second, minute, hour or day. Timezones are only relevant to days, when midnight in the current timezone is used.

The methods for class "Date" are of little use except to remove fractional days.

#### Value

An object of class "POSIXlt".

#### See Also

- `round` for the generic function and default methods.
- `DateTimeClasses.Date`

#### Examples

```r
round(.leap.seconds + 1000, "hour")
trunc(Sys.time(), "day")
```

---

### row

#### Row Indexes

#### Description

Returns a matrix of integers indicating their row number in the matrix.

#### Usage

```r
row(x, as.factor = FALSE)
```

#### Arguments

- `x`: a matrix.
- `as.factor`: a logical value indicating whether the value should be returned as a factor rather than as numeric.

#### Value

An integer matrix with the same dimensions as `x` and whose `ij`-th element is equal to `i`. 
References


See Also

col to get columns.

Examples

```r
x <- matrix(1:12, 3, 4)
# extract the diagonal of a matrix
dx <- x[row(x) == col(x)]
dx

# create an identity 5-by-5 matrix
x <- matrix(0, nr = 5, nc = 5)
x[row(x) == col(x)] <- 1
x
```

---

**row.names**

*Get and Set Row Names for Data Frames*

Description

All data frames have a row names attribute, a character vector of length the number of rows with no duplicates nor missing values.

For convenience, these are generic functions for which users can write other methods, and there are default methods for arrays. The description here is for the `data.frame` method.

Usage

```r
row.names(x)
row.names(x) <- value
```

Arguments

- `x` object of class "data.frame", or any other class for which a method has been defined.
- `value` a vector with the same length as the number of rows of `x`, to be coerced to character. Duplicated or missing values are not allowed.

Value

- `row.names` returns a character vector.
- `row.names<-` returns a data frame with the row names changed.

Note

`row.names` is similar to `rownames` for arrays, and it has a method that calls `rownames` for an array argument.
Row and Column Names

Description

Retrieve or set the row or column names of a matrix-like object.

Usage

rownames(x, do.NULL = TRUE, prefix = "row")
rownames(x) <- value

colnames(x, do.NULL = TRUE, prefix = "col")
colnames(x) <- value

Arguments

x                      a matrix-like R object, with at least two dimensions for colnames.
do.NULL               logical. Should this create names if they are NULL?
prefix                 for created names.
value                  a valid value for that component of dimnames(x). For a matrix or array this is either NULL or a character vector of length the appropriate dimension.

Details

The extractor functions try to do something sensible for any matrix-like object x. If the object has dimnames the first component is used as the row names, and the second component (if any) is used for the col names. For a data frame, rownames and colnames are equivalent to row.names and names respectively.

If do.NULL is FALSE, a character vector (of length NROW(x) or NCOL(x)) is returned in any case, prepending prefix to simple numbers, if there are no dimnames or the corresponding component of the dimnames is NULL.

For a data frame, value for rownames should be a character vector of unique names, and for colnames a character vector of unique syntactically-valid names. (Note: uniqueness and validity are not enforced.)

See Also

dimnames, case.names, variable.names.
Examples

```r
m0 <- matrix(NA, 4, 0)
rownames(m0)

m2 <- cbind(1,1:4)
colnames(m2, do.NULL = FALSE)
colnames(m2) <- c("x","Y")
rownames(m2) <- rownames(m2, do.NULL = FALSE, prefix = "Obs.")
m2
```

Description

Compute sums across rows of a matrix-like object for each level of a grouping variable. `rowsum` is generic, with methods for matrices and data frames.

Usage

```r
rowsum(x, group, reorder = TRUE, ...)
```

Arguments

- `x` a matrix, data frame or vector of numeric data. Missing values are allowed.
- `group` a vector giving the grouping, with one element per row of `x`. Missing values will be treated as another group and a warning will be given
- `reorder` if TRUE, then the result will be in order of `sort(unique(group))`, if FALSE, it will be in the order that rows were encountered.
- `...` other arguments for future methods

Details

The default is to reorder the rows to agree with `tapply` as in the example below. Reordering should not add noticeably to the time except when there are very many distinct values of `group` and `x` has few columns.

The original function was written by Terry Therneau, but this is a new implementation using hashing that is much faster for large matrices.

To add all the rows of a matrix (ie, a single `group`) use `rowSums`, which should be even faster.

Value

a matrix or data frame containing the sums. There will be one row per unique value of `group`.

See Also

`tapply, aggregate, rowSums`
Examples

```r
x <- matrix(runif(100), ncol=5)
group <- sample(1:8, 20, TRUE)
xsum <- rowsum(x, group)
## Slower versions
xsum2 <- tapply(x, list(group[row(x)], col(x)), sum)
xsum3 <- aggregate(x, list(group), sum)
```

Description

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

Usage

```r
sample(x, size, replace = FALSE, prob = NULL)
```

Arguments

- **x**: Either a (numeric, complex, character or logical) vector of more than one element from which to choose, or a positive integer.
- **size**: non-negative integer giving the number of items to choose.
- **replace**: Should sampling be with replacement?
- **prob**: A vector of probability weights for obtaining the elements of the vector being sampled.

Details

If `x` has length 1, sampling takes place from `1:x`. Note that this convenience feature may lead to undesired behaviour when `x` is of varying length `sample(x)`. See the `resample()` example below.

By default `size` is equal to `length(x)` so that `sample(x)` generates a random permutation of the elements of `x` (or `1:x`).

The optional `prob` argument can be used to give a vector of weights for obtaining the elements of the vector being sampled. They need not sum to one, but they should be nonnegative and not all zero. If `replace` is true, Walker’s alias method (Ripley, 1987) is used when there are more than 250 reasonably probable values: this gives results incompatible with those from `R < 2.2.0`, and there will be a warning the first time this happens in a session.

If `replace` is false, these probabilities are applied sequentially, that is the probability of choosing the next item is proportional to the probabilities amongst the remaining items. The number of nonzero weights must be at least `size` in this case.

References


Examples

```r
x <- 1:12
# a random permutation
sample(x)
# bootstrap sampling -- only if length(x) > 1!
sample(x, replace = TRUE)

# 100 Bernoulli trials
sample(c(0,1), 100, replace = TRUE)

## More careful bootstrapping -- Consider this when using sample()
## programmatically (i.e., in your function or simulation)!

# sample()'s surprise -- example
x <- 1:10
  sample(x[x > 8]) # length 2
  sample(x[x > 9]) # oops -- length 10!
try(sample(x[x > 10])) # error!

## This is safer:
resample <- function(x, size, ...)
  if(length(x) <= 1) { if(!missing(size) && size == 0) x[FALSE] else x
  } else sample(x, size, ...)
resample(x[x > 8])# length 2
resample(x[x > 9])# length 1
resample(x[x > 10])# length 0
```

Description

`save` writes an external representation of R objects to the specified file. The objects can be read back from the file at a later date by using the function `load` (or `data` in some cases).

`save.image()` is just a short-cut for “save my current environment”, i.e., `save(list = ls(all=TRUE), file = ".RData")`. It is what also happens with `q("yes")`.

Usage

```r
save(..., list = character(0),
  file = stop("'file' must be specified"),
  ascii = FALSE, version = NULL, envir = parent.frame(),
  compress = FALSE)

save.image(file = ".RData", version = NULL, ascii = FALSE,
  compress = FALSE, safe = TRUE)

sys.load.image(name, quiet)
sys.save.image(name)
```
save

Arguments

... the names of the objects to be saved.

list A character vector containing the names of objects to be saved.

file a connection or the name of the file where the data will be saved. Must be a file name for workspace format version 1.

ascii if TRUE, an ASCII representation of the data is written. This is useful for transporting data between machines of different types. The default value of ascii is FALSE which leads to a more compact binary file being written.

version the workspace format version to use. NULL specifies the current default format. The version used from R 0.99.0 to R 1.3.1 was version 1. The default format as from R 1.4.0 is version 2.

envir environment to search for objects to be saved.

compress logical specifying whether saving to a named file is to use compression. Ignored when file is a connection and for workspace format version 1.

safe logical. If TRUE, a temporary file is used for creating the saved workspace. The temporary file is renamed to file if the save succeeds. This preserves an existing workspace file if the save fails, but at the cost of using extra disk space during the save.

name name of image file to save or load.

quiet logical specifying whether a message should be printed.

Details

All R platforms use the XDR representation of binary objects in binary save-d files, and these are portable across all R platforms.

Default values for the ascii, compress, safe and version arguments can be modified with the save.defaults option (used both by save and save.image). If a save.image.defaults option is set it overrides save.defaults for function save.image (which allows to have different defaults). This mechanism is experimental and subject to change.

sys.save.image is a system function that is called by q() and its GUI analogs; sys.load.image is called by the startup code. These functions should not be called directly and are subject to change.

sys.save.image closes all connections first, to ensure that it is able to open a connection to save the image. This is appropriate when called from q() and allies, but reinforces the warning that it should not be called directly.

Warning

The ... arguments only give the names of the objects to be saved: they are searched for in the environment given by the envir argument, and the actual objects given as arguments need not be those found.

Saved R objects are binary files, even those saved with ascii=TRUE, so ensure that they are transferred without conversion of end of line markers. The lines are delimited by LF on all platforms.

See Also
dput, dump, load, data.
Examples

```r
x <- runif(20)
y <- list(a = 1, b = TRUE, c = "oops")
save(x, y, file = "xy.Rdata")
save.image()
unlink("xy.Rdata")
unlink(".RData")

# set save defaults using option:
options(save.defaults=list(ascii=TRUE, safe=FALSE))
save.image()
unlink(".RData")
```

scale

Scaling and Centering of Matrix-like Objects

Description

scale is a generic function whose default method centers and/or scales the columns of a numeric matrix.

Usage

```r
scale(x, center = TRUE, scale = TRUE)
```

Arguments

- **x**: a numeric matrix (like object).
- **center**: either a logical value or a numeric vector of length equal to the number of columns of `x`.
- **scale**: either a logical value or a numeric vector of length equal to the number of columns of `x`.

Details

The value of `center` determines how column centering is performed. If `center` is a numeric vector with length equal to the number of columns of `x`, then each column of `x` has the corresponding value from `center` subtracted from it. If `center` is `TRUE` then centering is done by subtracting the column means (omitting NAs) of `x` from their corresponding columns, and if `center` is `FALSE`, no centering is done.

The value of `scale` determines how column scaling is performed (after centering). If `scale` is a numeric vector with length equal to the number of columns of `x`, then each column of `x` is divided by the corresponding value from `scale`. If `scale` is `TRUE` then scaling is done by dividing the (centered) columns of `x` by their root-mean-square, and if `scale` is `FALSE`, no scaling is done.

The root-mean-square for a column is obtained by computing the square-root of the sum-of-squares of the non-missing values in the column divided by the number of non-missing values minus one.

Value

For `scale.default`, the centered, scaled matrix. The numeric centering and scalings used (if any) are returned as attributes "scaled:center" and "scaled:scale"
322

scan

References
Brooks/Cole.
See Also
sweep which allows centering (and scaling) with arbitrary statistics.
For working with the scale of a plot, see par.
Examples
require(stats)
x <- matrix(1:10, nc=2)
(centered.x <- scale(x, scale=FALSE))
cov(centered.scaled.x <- scale(x))# all 1

scan

Read Data Values

Description
Read data into a vector or list from the console or file.
Usage
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",
quote = if(identical(sep, "\n")) "" else "'\"", dec = ".",
skip = 0, nlines = 0, na.strings = "NA",
flush = FALSE, fill = FALSE, strip.white = FALSE,
quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE,
comment.char = "", allowEscapes = FALSE)
Arguments
file

the name of a file to read data values from. If the specified file is "", then
input is taken from the keyboard (or stdin if input is redirected). (In this case
input can be terminated by a blank line or an EOF signal, Ctrl-D on Unix and
Ctrl-Z on Windows.)
Otherwise, the file name is interpreted relative to the current working directory
(given by getwd()), unless it specifies an absolute path. Tilde-expansion is
performed where supported.
Alternatively, file can be a connection, which will be opened if necessary,
and if so closed at the end of the function call. Whatever mode the connection
is opened in, any of LF, CRLF or CR will be accepted as the EOL marker for a
line and so will match sep = "
n".
file can also be a complete URL.
To read a data file not in the current encoding (for example a Latin-1 file in
a UTF-8 locale or conversely) use a file connection setting the encoding
argument.


what  the type of what gives the type of data to be read. The supported types are logical, integer, numeric, complex, character, raw and list. If what is a list, it is assumed that the lines of the data file are records each containing length(what) items (“fields”) and the list components should have elements which are one of the first six types listed or NULL, see section ‘Details’ below.

nmax  the maximum number of data values to be read, or if what is a list, the maximum number of records to be read. If omitted or not positive (and nlines is not set to a positive value), scan will read to the end of file.

n  the maximum number of data values to be read, defaulting to no limit.

sep  by default, scan expects to read white-space delimited input fields. Alternatively, sep can be used to specify a character which delimits fields. A field is always delimited by an end-of-line marker unless it is quoted.

If specified this should be the empty character string (the default) or NULL or a character string containing just one single-byte character.

quote  the set of quoting characters as a single character string or NULL. In a multibyte locale the quoting characters must be ASCII (single-byte).

dec  decimal point character. This should be a character string containing just one single-byte character. (NULL and a zero-length character vector are also accepted, and taken as the default.)

skip  the number of lines of the input file to skip before beginning to read data values.

nlines  if positive, the maximum number of lines of data to be read.

na.strings  character vector. Elements of this vector are to be interpreted as missing (NA) values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields.

flush  logical: if TRUE, scan will flush to the end of the line after reading the last of the fields requested. This allows putting comments after the last field, but precludes putting more that one record on a line.

fill  logical: if TRUE, scan will implicitly add empty fields to any lines with fewer fields than implied by what.

strip.white  vector of logical value(s) corresponding to items in the what argument. It is used only when sep has been specified, and allows the stripping of leading and trailing white space from character fields (numeric fields are always stripped).

If strip.white is of length 1, it applies to all fields; otherwise, if strip.white[i] is TRUE and the i-th field is of mode character (because what[i] is) then the leading and trailing white space from field i is stripped.

quiet  logical: if FALSE (default), scan() will print a line, saying how many items have been read.

blank.lines.skip  logical: if TRUE blank lines in the input are ignored, except when counting skip and nlines.

multi.line  logical. Only used if what is a list. If FALSE, all of a record must appear on one line (but more than one record can appear on a single line). Note that using fill = TRUE implies that a record will terminated at the end of a line.

comment.char  character: a character vector of length one containing a single character or an empty string. Use "\n" to turn off the interpretation of comments altogether (the default).
allowEscapes logical. Should C-style escapes such as
\n be processed (the default) or read verbatim? Note that if not within quotes
these could be interpreted as a delimiter (but not as a comment character).
The escapes which are interpreted are the control characters
a,
b,
f,
r,
t,
v and octal and hexadecimal represenatons like
040 and
0x2A. Any other escaped character is treated as itself, including backslash.

Details
The value of what can be a list of types, in which case scan returns a list of vectors with the
types given by the types of the elements in what. This provides a way of reading columnar data. If
any of the types is NULL, the corresponding field is skipped (but a NULL component appears in the
result).
The type of what or its components can be one of the six atomic vector types or NULL (see
is.atomic).
‘White space’ is defined for the purposes of this function as one or more contiguous characters from
the set space, horizontal tab, carriage return and line feed. It does not include form feed, vertical
tab or other non-ASCII space characters.
Empty numeric fields are always regarded as missing values. Empty character fields are scanned
as empty character vectors, unless na.strings contains "" when they are regarded as missing
values.
If sep is the default (""), the character \ in a quoted string escapes the following character, so
quotes may be included in the string by escaping them.
If sep is non-default, the fields may be quoted in the style of ‘.csv’ files where separators inside
quotes (" or ") are ignored and quotes may be put inside strings by doubling them. However, if
sep = "\n" it is assumed by default that one wants to read entire lines verbatim.
Quoting is only interpreted in character fields, and as from R 1.8.0 in NULL fields (which might be
skipping character fields).
Note that since sep is a separator and not a terminator, reading a file by scan("foo",
sep="\n", blank.lines.skip=FALSE) will give an empty final line if the file ends in
a linefeed and not if it does not. This might not be what you expected; see also readLines.
If comment.char occurs (except inside a quoted character field), it signals that the rest of the
line should be regarded as a comment and be discarded. Lines beginning with a comment character
(possibly after white space with the default separator) are treated as blank lines.
As from R 2.1.0, scan attempts to share storage with character strings that have already been read
in the call. If an upper bound on the number of character strings cannot be deduced from nmax or
n, sharing is used for the first 10000 unique strings which are read in.

Value
if what is a list, a list of the same length and same names (as any) as what.
Otherwise, a vector of the type of what.
Note

The default for multi.line differs from S. To read one record per line, use flush = TRUE and multi.line = FALSE. (Note that quoted character strings can still include embedded new lines.)

If number of items is not specified, the internal mechanism re-allocates memory in powers of two and so could use up to three times as much memory as needed. (It needs both old and new copies.)

If you can, specify either n or nmax whenever inputting a large vector, and nmax or nlines when inputting a large list.

Using scan on an open connection to read partial lines can lose chars: use an explicit separator to avoid this.

References


See Also

read.table for more user-friendly reading of data matrices; readLines to read a file a line at a time. write.

Quotes for the details of C-style escape sequences.

Examples

```r
cat("TITLE extra line", "2 3 5 7", "11 13 17", file="ex.data", sep="\n")
pp <- scan("ex.data", skip = 1, quiet= TRUE)
scan("ex.data", skip = 1)
scan("ex.data", skip = 1, nlines=1)# only 1 line after the skipped one
scan("ex.data", what = list("","","")) # flush is F -> read "7"
scan("ex.data", what = list("","",""), flush = TRUE)
unlink("ex.data") # tidy up
```

search

Give Search Path for R Objects

Description

Gives a list of attached packages (see library), and R objects, usually data.frames.

Usage

search()
searchpaths()

Value

A character vector, starting with ".GlobalEnv", and ending with "package:base" which is R's base package required always.

searchpaths gives a similar character vector, with the entries for packages being the path to the package used to load the code.
seek

Functions to Reposition Connections

Description

Functions to re-position connections.

Usage

```
seek(con, ...)  
## S3 method for class 'connection':
seek(con, where = NA, origin = "start", rw = "", ...)  
```

```
isSeekable(con)  
truncate(con, ...)  
```

Arguments

- **con**
  
a connection.

- **where**
  
numeric. A file position (relative to the origin specified by `origin`), or NA.

- **rw**
  
character. Empty or "read" or "write", partial matches allowed.

- **origin**
  
character. One of "start", "current", "end": see Details.

- **...**
  
further arguments passed to or from other methods.
Details

`seek` with `where = NA` returns the current byte offset of a connection (from the beginning), and with a non-missing `where` argument the connection is re-positioned (if possible) to the specified position. `isSeekable` returns whether the connection in principle supports `seek`: currently only (possibly gz-compressed) file connections do. `gzfile` connections do not support `origin = "end"`; the file position they use is that of the uncompressed file.

`where` is stored as a real but should represent an integer: non-integer values are likely to be truncated. Note that the possible values can exceed the largest representable number in an R `integer` on 64-bit OSes, and on some 32-bit OSes.

Clipboard connections can seek too.

We have found so many errors in the Windows implementation of file positioning that users are advised to use it only at their own risk, and asked not to waste the R developers’ time with bug reports on Windows’ deficiencies.

File connections can be open for both writing/appending, in which case R keeps separate positions for reading and writing. Which `seek` refers to can be set by its `rw` argument: the default is the last mode (reading or writing) which was used. Most files are only opened for reading or writing and so default to that state. If a file is open for reading and writing but has not been used, the default is to give the reading position (0).

The initial file position for reading is always at the beginning. The initial position for writing is at the beginning of the file for modes "r+" and "r+b", otherwise at the end of the file. Some platforms only allow writing at the end of the file in the append modes. (The reported write position for a file opened in an append mode will typically be unreliable until the file has been written to.)

`truncate` truncates a file opened for writing at its current position. It works only for file connections, and is not implemented on all platforms: on others (including Windows) it will not work for large (> 2Gb) files.

Value

`seek` returns the current position (before any move), as a (numeric) byte offset from the origin, if relevant, or 0 if not. Note that the position can exceed the largest representable number in an R `integer` on 64-bit OSes, and on some 32-bit OSes.

`truncate` returns `NULL`: it stops with an error if it fails (or is not implemented).

`isSeekable` returns a logical value, whether the connection supports `seek`.

See Also

`connections`
Usage

from:to
   a:b

seq(from, to)
seq(from, to, by=)
seq(from, to, length.out=)
seq(along.with=)
seq(from)

Arguments

from     starting value of sequence.
to      (maximal) end value of the sequence.
by     increment of the sequence.
length.out desired length of the sequence.
along.with take the length from the length of this argument.
a,b    factors of same length.

Details

The binary operator : has two meanings: for factors a:b is equivalent to interaction(a, b) (except for labelling by la:lb not la.lb). For numeric arguments a:b is equivalent to seq(from=a, to=b).

The interpretation of the unnamed arguments of seq is not standard, and it is recommended always to name the arguments when programming.

Function seq is generic, and only the default method is described here.

The operator : and the seq(from, to) form generate the sequence from, from+1, ..., to.

The second form generates from, from+by,..., up to the sequence value less than or equal to to.

The third generates a sequence of length.out equally spaced values from from to to.

The fourth form generates the sequence 1, 2, ..., length(along.with).

The last generates the sequence 1, 2, ..., length(from) (as if argument along had been specified), unless the argument is numeric of length 1 when it is interpreted as 1:from (even for seq(0) for compatibility with S).

If from and to are factors of the same length, then from : to returns the “cross” of the two.

Very small sequences (with from - to of the order of 10^-14 times the larger of the ends) will return from.

Value

Currently, the default method returns a result of storage mode "integer" if from is (numerically equal to an) integer and, e.g., only to is specified, or also if only length or only along.with is specified. Note: this may change in the future and programmers should not rely on it.
seq.Date

Generate Regular Sequences of Dates

Description

The method for `seq` for date-time classes.

Usage

```r
## S3 method for class 'Date':
seq(from, to, by, length.out = NULL, along.with = NULL, ...)
```

Arguments

- `from`: starting date. Required
- `to`: end date. Optional.
- `by`: increment of the sequence. Optional. See Details.
- `length.out`: integer, optional. desired length of the sequence.
- `along.with`: take the length from the length of this argument.
- `...`: arguments passed to or from other methods.
seq.POSIXt

Generate Regular Sequences of Dates

Description

The method for seq for date-time classes.

Usage

## S3 method for class 'POSIXt':
seq(from, to, by, length.out = NULL, along.with = NULL, ...)

Arguments

from starting date. Required.
to end date. Optional.
by increment of the sequence. Optional. See Details.
length.out integer, optional. desired length of the sequence.
along.with take the length from the length of this argument.
... arguments passed to or from other methods.

Details

by can be specified in several ways.

- A number, taken to be in days.
- A object of class difftime
- A character string, containing one of "day", "week", "month" or "year". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s".

Value

A vector of class "Date".

See Also

Date

Examples

## first days of years
seq(as.Date("1910/1/1"), as.Date("1999/1/1"), "years")

## by month
seq(as.Date("2000/1/1"), by="month", length=12)

## quarters
seq(as.Date("2000/1/1"), as.Date("2003/1/1"), by="3 months")

## find all 7th of the month between two dates, the last being a 7th.
st <- as.Date("1998-12-17")
en <- as.Date("2000-1-7")
ll <- seq.Date(en, st, by="-1 month")
rev(ll[ll > st & ll < en])
Details

by can be specified in several ways.

- A number, taken to be in seconds.
- A object of class `difftime`
- A character string, containing one of "sec", "min", "hour", "day", "DSTday", "week", "month" or "year". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s".

The difference between "day" and "DSTday" is that the former ignores changes to/from daylight savings time and the latter takes the same clock time each day. ("week" ignores DST, but "7 DSTdays") can be used as an alternative. "month" and "year" allow for DST.)

The timezone of the result is taken from `from`: remember than GMT does not have daylight savings time.

Value

A vector of class "POSIXct".

See Also

DateTimeClasses

Examples

```r
## first days of years
seq(ISOdate(1910,1,1), ISOdate(1999,1,1), "years")
## by month
seq(ISOdate(2000,1,1), by = "month", length = 12)
## quarters
seq(ISOdate(1990,1,1), ISOdate(2000,1,1), by = "3 months")
## days vs DSTdays: use c() to lose the timezone.
seq(c(ISOdate(2000,3,20)), by = "day", length = 10)
seq(c(ISOdate(2000,3,20)), by = "DSTday", length = 10)
seq(c(ISOdate(2000,3,20)), by = "7 DSTdays", length = 4)
```

---

**sequence**  
Create A Vector of Sequences

Description

For each element of `nvec` the sequence `seq(nvec[i])` is created. These are appended and the result returned.

Usage

`sequence(nvec)`

Arguments

nvec  
an integer vector each element of which specifies the upper bound of a sequence.


See Also

gl, seq, rep.

Examples

sequence(c(3, 2)) # the concatenated sequences 1:3 and 1:2.
#> [1] 1 2 3 1 2

sets

Set Operations

Description

Performs set union, intersection, (asymmetric!) difference, equality and membership on two vectors.

Usage

union(x, y)
intersect(x, y)
setdiff(x, y)
setequal(x, y)
is.element(el, set)

Arguments

x, y, el, set

vectors (of the same mode) containing a sequence of items (conceptually) with no duplicated values.

Details

Each of union, intersect and setdiff will remove any duplicated values in the arguments.

is.element(x, y) is identical to x %in% y.

Value

A vector of the same mode as x or y for setdiff and intersect, respectively, and of a common mode for union.

A logical scalar for setequal and a logical of the same length as x for is.element.

See Also

%in%
Examples

```r
(x <- c(sort(sample(1:20, 9)), NA))
(y <- c(sort(sample(3:23, 7)), NA))
union(x, y)
intersect(x, y)
setdiff(x, y)
setdiff(y, x)
setequal(x, y)

## True for all possible x & y :
setequal(union(x, y),
         c(setdiff(x, y), intersect(x, y), setdiff(y, x)))
```

```r
is.element(x, y)  # length 10
is.element(y, x)  # length 8
```

---

**shell**

*Invoke a System Command, using a Shell*

**Description**

shell runs the command specified by cmd, usually under a shell.

**Usage**

```r
shell(cmd, shell, flag="/c", intern=FALSE, wait=TRUE,
     translate=FALSE, mustWork=FALSE, ...)
```

**Arguments**

- **cmd**
  - the system command to be invoked, as a string.
- **shell**
  - a string giving the name of the shell to be used, or NULL (no shell). If missing, a suitable shell is chosen: see ‘Details’.
- **flag**
  - the switch to run a command under the shell. If the shell is `bash` or `tcsh` the default is changed to "-c".
- **intern**
  - a logical, indicates whether to make the output of the command an R object.
- **wait**
  - should the R interpreter wait for the command to finish? The default is to wait, and the interpreter will always wait if `intern = TRUE`.
- **translate**
  - If TRUE, "/" in `cmd` is translated to "\".
- **mustWork**
  - a logical; if TRUE failure to run the command will give an R error.
- **...**
  - additional arguments to `system`.

**Details**

If no shell is specified, the environment variables R_SHELL, SHELL and COMSPEC are tried in turn: COMSPEC should always succeed. Using `shell=NULL` invokes the command `cmd` directly, in which case an extension of `.exe` is assumed. It is possible to use batch files directly if their extension is given: Windows (rather than R) then chooses a shell.

See `system` for fuller details: shell is a more user-friendly wrapper for system. To make use of Windows file associations, use `shell.exec`.
Value

If \texttt{intern=TRUE}, a character vector giving the output of the command, one line per character string, or an error message if the command could not be run.

If \texttt{intern=FALSE}, the return value is an error code, given the invisible attribute (so needs to be printed explicitly). If the command could not be run for any reason, the value is \(-1\) and an \texttt{R} warning is generated. Otherwise if \texttt{wait=FALSE} the value is the error code returned by the command, and if \texttt{wait=TRUE} it is the zero (the conventional success value).

If \texttt{intern=FALSE} and \texttt{wait=TRUE} (the defaults) the text output from a command that is a console application will appear in the \texttt{R} console (\texttt{Rgui}) or the window running \texttt{R} (\texttt{Rterm}).

See Also

\texttt{system,shell.exec}

\begin{description}
  \item[shell.exec] \textit{Open a File using Windows File Associations}
  \item[Description] Opens the specified file using the application specified in the Windows file associations.
  \item[Usage] \texttt{shell.exec(file)}
  \item[Arguments] \texttt{file} file to be opened.
  \item[Details] It is likely that \texttt{file} needs to be a complete path, e.g. "\texttt{c:/R/results.html}" or "\texttt{c:\R\results.html}" as most applications interpret files relative to their working directory.
  \item[Value] No value.
  \item[Author(s)] B. D. Ripley
  \item[See Also] \texttt{system,shell}
\end{description}
showConnections

Display Connections

Description

Display aspects of connections.

Usage

showConnections(all = FALSE)
getConnection(what)
closeAllConnections()

stdin()
stdout()
stderr()

Arguments

all
logical: if true all connections, including closed ones and the standard ones are
displayed. If false only open user-created connections are included.

what
integer: a row number of the table given by showConnections.

Details

stdin(), stdout() and stderr() are standard connections corresponding to input, output
and error on the console respectively (and not necessarily to file streams). They are text-mode
connections of class "terminal" which cannot be opened or closed, and are read-only, write-
only and write-only respectively. The stdout() and stderr() connections can be re-directed
by sink.

showConnections returns a matrix of information. If a connection object has been lost or
forgotten, getConnection will take a row number from the table and return a connection object
for that connection, which can be used to close the connection, for example.

closeAllConnections closes (and destroys) all open user connections, restoring all sink
diversions as it does so.

Value

stdin(), stdout() and stderr() return connection objects.

showConnections returns a character matrix of information with a row for each connection, by
default only for open non-standard connections.

getConnection returns a connection object, or NULL.

See Also

connections
Examples

```r
showConnections(all = TRUE)

textConnection(letters)
# oops, I forgot to record that one
showConnections()
# class  description  mode  text  isopen  can read  can write
#3  "letters"  "textConnection"  "r"  "text"  "opened"  "yes"  "no"
## Not run: close(getConnection(3))

showConnections()
```

---

**shQuote**

*Quote Strings for Use in OS Shells*

**Description**

Quote a string to be passed to an operating system shell.

**Usage**

```r
shQuote(string, type = c("sh", "csh", "cmd"))
```

**Arguments**

- `string` - a character vector, usually of length one.
- `type` - character: the type of shell. Partial matching is supported. "cmd" refers to the Windows NT shell, and is the default under Windows.

**Details**

The default type of quoting supported under Unix-alikes is that for the Bourne shell `sh`. If the string does not contain single quotes, we can just surround it with single quotes. Otherwise, the string is surrounded in double quotes, which suppresses all special meanings of metacharacters except dollar, backquote and backslash, so these (and of course double quote) are preceded by backslash. This type of quoting is also appropriate for `ksh`, `zsh` and `bash`.

The other type of quoting is for the C-shell (`csh` and `tcsh`). Once again, if the string does not contain single quotes, we can just surround it with single quotes. If it does contain single quotes, we can use double quotes provided it does not contain dollar or backquote (and we need to escape backslash, exclamation mark and double quote). As a last resort, we need to split the string into pieces not containing single quotes and surround each with single quotes, and the single quotes with double quotes.

The Windows shell supports only double quoting. All this implementation does is to surround by double quotes and escape internal double quotes by a backslash. As Windows path names cannot contain double quotes, this makes `shQuote` safe for use with `system` and with `shell` if the default shell is used.

It will usually be safe to use `shQuote` to quote arguments of a command, but because `system` does not use a shell, interpretation of quoted arguments is done by the run-time code of the executable.
References


http://www.mhuffman.com/notes/dos/bash_cmd.htm

See Also

Quotes for quoting R code.
sQuote for quoting English text.

Examples

test <- "abc$def`gh`i\j"
cat(sQuote(test), "\n")
## Not run: system(paste("echo", sQuote(test)))
test <- "don't do it!"
cat(sQuote(test), "\n")

test <- "use the '-c' switch\like this"
cat(sQuote(test), "\n")
## Not run: system(paste("echo", sQuote(test)))
cat(sQuote(test, type="csh"), "\n")

## Windows-only example.
perlc当地 <- 'print "Hello World\n";
## Not run: shell(paste("perl -e", sQuote(perlc当地, type="cmd")))

---

**sign**

*Nlargen Function*

**Description**

sign returns a vector with the signs of the corresponding elements of \( x \) (the sign of a real number is 1, 0, or \(-1\) if the number is positive, zero, or negative, respectively).

Note that sign does not operate on complex vectors.

**Usage**

sign(x)

**Arguments**

\( x \) a numeric vector

**Details**

This is a generic function: methods can be defined for it directly or via the \texttt{Math} group generic.

**See Also**

abs
sink

Send R Output to a File

Description

sink diverts R output to a connection.
sink.number() reports how many diversions are in use.
sink.number(type = "message") reports the number of the connection currently being used for error messages.

Usage

sink(file = NULL, append = FALSE, type = c("output", "message"),
     split = FALSE)
sink.number(type = c("output", "message"))

Arguments

file a connection or a character string naming the file to write to, or NULL to stop sink-ing.
append logical. If TRUE, output will be appended to file; otherwise, it will overwrite the contents of file.
type character. Either the output stream or the messages stream.
split logical: if TRUE, output will be sent to the new sink and to the current output stream, like the Unix program tee.

Details

sink diverts R output to a connection. If file is a character string, a file connection with that name will be established for the duration of the diversion.
Normal R output is diverted by the default type = "output". Only prompts and warning/error messages continue to appear on the terminal. The latter can diverted by type = "message" (see below).
sink() or sink(file=NULL) ends the last diversion (of the specified type). There is a stack of diversions for normal output, so output reverts to the previous diversion (if there was one). The stack is of up to 21 connections (20 diversions).
If file is a connection if will be opened if necessary.
Sink-ing the messages stream should be done only with great care. For that stream file must be an already open connection, and there is no stack of connections.

Examples

sign(pi) # == 1
sign(-2:3)# -1 -1 0 1 1 1
Value

sink returns NULL.
For sink.number() the number (0, 1, 2, ...) of diversions of output in place.
For sink.number("message") the connection number used for messages, 2 if no diversion has been used.

Warning

Don't use a connection that is open for sink for any other purpose. The software will stop you closing one such inadvertently.
Do not sink the messages stream unless you understand the source code implementing it and hence the pitfalls.

References


See Also
capture.output

Examples

sink("sink-examp.txt")
i <- 1:10
outer(i, i, "*")
sink()
unlink("sink-examp.txt")

## Not run:
## capture all the output to a file.
zz <- file("all.Rout", open="wt")
sink(zz)
sink(zz, type="message")
try(log("a"))

## back to the console
sink(type="message")
sink()
try(log("a"))

## End(Not run)

slice.index
Slice Indexes in an Array

Description

Returns a matrix of integers indicating the number of their slice in a given array.

Usage

slice.index(x, MARGIN)
Arguments

\( x \) an array. If \( x \) has no dimension attribute, it is considered a one-dimensional array.

\( \text{MARGIN} \) an integer giving the dimension number to slice by.

Value

An integer array \( y \) with dimensions corresponding to those of \( x \) such that all elements of slice number \( i \) with respect to dimension \( \text{MARGIN} \) have value \( i \).

See Also

\( \text{row} \) and \( \text{col} \) for determining row and column indexes; in fact, these are special cases of \( \text{slice.index} \) corresponding to \( \text{MARGIN} \) equal to 1 and 2, respectively when \( x \) is a matrix.

Examples

\[
x <- \text{array}(1:24, c(2, 3, 4))\n\]
\[
\text{slice.index}(x, 2)\n\]

---

**slotOp**  
*Extract Slots*

Description

Extract the contents of a slot in an object with a formal class structure.

Usage

\( \text{object@name} \)

Arguments

\( \text{object} \) An object from a formally defined class.

\( \text{name} \) The character-string name of the slot.

Details

These operators support the formal classes of package \texttt{methods}. See \texttt{slot} for further details. Currently there is no checking that the object is an instance of a class.

See Also

\texttt{Extract, slot}
socketSelect

Wait on Socket Connections

Description

Waits for the first of several socket connections to become available.

Usage

socketSelect(socklist, write = FALSE, timeout = NULL)

Arguments

socklist list of open socket connections
write logical. If TRUE wait for corresponding socket to become available for writing; otherwise wait for it to become available for reading.
timeout numeric or NULL. Time in seconds to wait for a socket to become available; NULL means wait indefinitely.

Details

The values in write are recycled if necessary to make up a logical vector the same length as socklist. Socket connections can appear more than once in socklist; this can be useful if you want to determine whether a socket is available for reading or writing.

Value

Logical the same length as socklist indicating whether the corresponding socket connection is available for output or input, depending on the corresponding value of write.

Examples

## Not run:
## test whether socket connection s is available for writing or reading
socketSelect(list(s,s),c(TRUE,FALSE),timeout=0)
## End(Not run)

solve

Solve a System of Equations

Description

This generic function solves the equation a %*% x = b for x, where b can be either a vector or a matrix.

Usage

solve(a, b, ...)

## Default S3 method:
solve(a, b, tol, LINPACK = FALSE, ...)
**solve**

Arguments

- \(a\): a square numeric or complex matrix containing the coefficients of the linear system.
- \(b\): a numeric or complex vector or matrix giving the right-hand side(s) of the linear system. If missing, \(b\) is taken to be an identity matrix and \texttt{solve} will return the inverse of \(a\).
- \(\texttt{tol}\): the tolerance for detecting linear dependencies in the columns of \(a\). If LINPACK is TRUE the default is \(1e^{-7}\), otherwise it is \(\text{.Machine}$\text{double.eps}\). Future versions of R may use a tighter tolerance. Not presently used with complex matrices \(a\).
- \(\texttt{LINPACK}\): logical. Should LINPACK be used (for compatibility with R < 1.7.0)? Otherwise LAPACK is used.
- \(\ldots\): further arguments passed to or from other methods

Details

- \(a\) or \(b\) can be complex, but this uses double complex arithmetic which might not be available on all platforms and LAPACK will always be used.
- The row and column names of the result are taken from the column names of \(a\) and of \(b\) respectively. As from R 1.7.0 if \(b\) is missing the column names of the result are the row names of \(a\). No check is made that the column names of \(a\) and the row names of \(b\) are equal.
- For back-compatibility \(a\) can be a (real) QR decomposition, although \texttt{qr.solve} should be called in that case. \texttt{qr.solve} can handle non-square systems.

References


See Also

- \texttt{solve.qr} for the \texttt{qr} method,
- \texttt{chol2inv} for inverting from the Choleski factor \texttt{backsolve}, \texttt{qr.solve}.

Examples

```r
hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
h8 <- hilbert(8); h8
sh8 <- solve(h8)
round(sh8 %*% h8, 3)

A <- hilbert(4)
A[] <- as.complex(A)
## might not be supported on all platforms
try(solve(A))
```
sort

Sorting or Ordering Vectors

Description

Sort (or order) a vector or factor (partially) into ascending (or descending) order. For ordering along more than one variable, e.g., for sorting data frames, see order.

Usage

sort(x, partial = NULL, na.last = NA, decreasing = FALSE, method = c("shell", "quick"), index.return = FALSE)

is.unsorted(x, na.rm = FALSE)

Arguments

x       a numeric, complex, character or logical vector, or a factor.
partial  a vector of indices for partial sorting.
na.last  for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
decreasing  logical. Should the sort be increasing or decreasing? Not available for partial sorting.
method    character specifying the algorithm used.
index.return  logical indicating if the ordering index vector should be returned as well; this is only available for a few cases, the default na.last = NA and full sorting of non-factors.
na.rm     logical. Should missing values be removed?

Details

If partial is not NULL, it is taken to contain indices of elements of x which are to be placed in their correct positions by partial sorting. After the sort, the values specified in partial are in their correct position in the sorted array. Any values smaller than these values are guaranteed to have a smaller index in the sorted array and any values which are greater are guaranteed to have a bigger index in the sorted array. This is included for efficiency, and many of the options are not available for partial sorting.

The sort order for character vectors will depend on the collating sequence of the locale in use: see Comparison.

is.unsorted returns a logical indicating if x is sorted increasingly, i.e., is.unsorted(x) is true if any(x != sort(x)) (and there are no NAs).

Method "shell" uses Shellsort (an $O(n^{4/3})$ variant from Sedgewick (1996)). If x has names a stable sort is used, so ties are not reordered. (This only matters if names are present.)

Method "quick" uses Singleton’s Quicksort implementation and is only available when x is numeric (double or integer) and partial is NULL. It is normally somewhat faster than Shellsort (perhaps twice as fast on vectors of length a million) but has poor performance in the rare worst case. (Peto’s modification using a pseudo-random midpoint is used to make the worst case rarer.) This is not a stable sort, and ties may be reordered.
Value

For sort the sorted vector unless index.return is true, when the result is a list with components named x and ix containing the sorted numbers and the ordering index vector. In the latter case, if method == "quick" ties may be reversed in the ordering, unlike sort.list, as quicksort is not stable.

References


See Also

order for sorting on or reordering multiple variables.

rank.

Examples

```r
require(stats)
x <- swiss$Education[1:25]
x; sort(x); sort(x, partial = c(10, 15))
median # shows you another example for 'partial'

## illustrate 'stable' sorting (of ties):
sort(c(10:3,2:12), method = "sh", index=TRUE) # is stable
## $x: 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 12
## $ix: 9 8 10 7 11 6 12 5 13 4 14 3 15 2 16 1 17 18 19
sort(c(10:3,2:12), method = "qu", index=TRUE) # is not
## $x: 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 12
## $ix: 9 10 8 7 11 6 12 5 13 4 14 3 15 16 2 17 1 18 19
## ^^^^^

## Not run: ## Small speed comparison simulation:
N <- 2000
Sim <- 20
rep <- 50 # << adjust to your CPU
cl <- c2 <- numeric(Sim)
for(is in 1:Sim){
x <- rnorm(N)
c1[is] <- system.time(for(i in 1:rep) sort(x, method = "shell"), gcFirst = TRUE)[1]
c2[is] <- system.time(for(i in 1:rep) sort(x, method = "quick"), gcFirst = TRUE)[1]
stopifnot(sort(x, meth = "s") == sort(x, meth = "q"))
}
100 * rbind(ShellSort = c1, QuickSort = c2)
cat("Speedup factor of quick sort():\n")
summary({qq <- c1 / c2; qq[is.finite(qq)]})

## A larger test
x <- rnorm(1e6)
```

```
```r
system.time(x1 <- sort(x, method = "shell"), gcFirst = TRUE)
system.time(x2 <- sort(x, method = "quick"), gcFirst = TRUE)
stopifnot(identical(x1, x2))
```

---

**source**

Read R Code from a File or a Connection

**Description**

`source` causes R to accept its input from the named file or URL (the name must be quoted) or connection. Input is read and parsed by from that file until the end of the file is reached, then the parsed expressions are evaluated sequentially in the chosen environment.

**Usage**

```r
source(file, local = FALSE, echo = verbose, print.eval = echo,
       verbose = getOption("verbose"),
       prompt.echo = getOption("prompt"),
       max.deparse.length = 150, chdir = FALSE,
       encoding = getOption("encoding"))
```

**Arguments**

- `file`: a connection or a character string giving the pathname of the file or URL to read from.
- `local`: if `local` is `FALSE`, the statements scanned are evaluated in the user’s workspace (the global environment), otherwise in the environment calling `source`.
- `echo`: logical; if TRUE, each expression is printed after parsing, before evaluation.
- `print.eval`: logical; if TRUE, the result of `eval(i)` is printed for each expression `i`; defaults to `echo`.
- `verbose`: if TRUE, more diagnostics (than just `echo = TRUE`) are printed during parsing and evaluation of input, including extra info for each expression.
- `prompt.echo`: character; gives the prompt to be used if `echo = TRUE`.
- `max.deparse.length`: integer; is used only if `echo` is `TRUE` and gives the maximal length of the "echo" of a single expression.
- `chdir`: logical; if TRUE and `file` is a pathname, the R working directory is temporarily changed to the directory containing `file` for evaluating.
- `encoding`: character string. The encoding to be assumed when `file` is a character string: see `file`. A possible value is "unknown"; see the Details.
Details

All versions of R accept input from a connection with end of line marked by LF (as used on Unix), CRLF (as used on DOS/Windows) or CR (as used on classic MacOS). The final line can be incomplete, that is missing the final EOL marker.

If options("keep.source") is true (the default), the source of functions is kept so they can be listed exactly as input. This imposes a limit of 128K chars on the function size and a nesting limit of 265. Use option(keep.source = FALSE) when these limits might take effect: if exceeded they generate an error.

This paragraph applies if file is a filename (rather than a connection). If encoding = "unknown", an attempt is made to guess the encoding. The result of localeToCharset() is used a guide. If encoding has two or more elements, they are tried in turn until the file/URL can be read without error in the trial encoding.

References


See Also

demo which uses source; eval, parse and scan; options("keep.source").

Special Functions of Mathematics

Description

Special mathematical functions related to the beta and gamma functions.

Usage

beta(a, b)
lbeta(a, b)
gamma(x)
lgamma(x)
psigamma(x, deriv = 0)
digamma(x)
trigamma(x)
choose(n, k)
lchoose(n, k)
factorial(x)
lfactorial(x)

Arguments

a, b, x, n numeric vectors.
k, deriv integer vectors.
Special

Details

The functions `beta` and `lbeta` return the beta function and the natural logarithm of the beta function,

\[ B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a + b)}. \]

The formal definition is

\[ B(a, b) = \int_0^1 t^{a-1}(1-t)^{b-1}dt \]

(Abramowitz and Stegun (6.2.1), page 258).

The functions `gamma` and `lgamma` return the gamma function \( \Gamma(x) \) and the natural logarithm of the absolute value of the gamma function. The gamma function is defined by (Abramowitz and Stegun (6.1.1), page 255)

\[ \Gamma(x) = \int_0^\infty t^{a-1}e^{-t}dt \]

factorial(x) is \( x! \) and identical to `gamma(x+1)` and `lfactorial` is `lgamma(x+1)`.

The functions `digamma` and `trigamma` return the first and second derivatives of the logarithm of the gamma function. `psigamma(x, deriv)(deriv >= 0)` is more generally computing the deriv-th derivative of \( \psi(x) \).

\[ \text{digamma}(x) = \psi(x) = \frac{d}{dx} \ln \Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)} \]

The functions `choose` and `lchoose` return binomial coefficients and their logarithms. Note that `choose(n, k)` is defined for all real numbers \( n \) and integer \( k \). For \( k \geq 1 \) as \( n(n-1)\cdots(n-k+1)/k! \), as 1 for \( k = 0 \) and as 0 for negative \( k \).

`choose(*, k)` uses direct arithmetic (instead of \([l]gamma\) calls) for small \( k \), for speed and accuracy reasons.

The `gamma`, `lgamma`, `digamma` and `trigamma` functions are generic: methods can be defined for them individually or via the `Math` group generic.

References


See Also

`Arithmetic` for simple, `sqrt` for miscellaneous mathematical functions and `Bessel` for the real Bessel functions.

For the incomplete gamma function see `pgamma`.

Examples

```r
choose(5, 2)
for (n in 0:10) print(choose(n, k = 0:n))

factorial(100)
lfactorial(10000)
```
## gamma has 1st order poles at 0, -1, -2, ...

```r
x <- sort(c(seq(-3,4, length=201), outer(0:-3, (-1:1) * 1e-6, "+")))
plot(x, gamma(x), ylim=c(-20,20), col="red", type="l", lwd=2,
     main=expression(Gamma(x)))
abline(h=0, v=-3:0, lty=3, col="midnightblue")
```

```r
x <- seq(.1, 4, length = 201); dx <- diff(x)[1]
par(mfrow = c(2, 3))
for (ch in c("", "l","di","tri","tetra","penta")) {
  is.deriv <- nchar(ch) >= 2
  nm <- paste(ch, "gamma", sep = "")
  if (is.deriv) {
    dy <- diff(y) / dx # finite difference
    der <- which(ch == c("di","tri","tetra","penta")) - 1
    nm2 <- paste("psigamma( * , deriv = ", der, ")",sep='')
    nm <- if(der >= 2) nm2 else paste(nm, nm2, sep = " ==
          " )
    y <- psigamma(x, deriv=der)
  } else {
    y <- get(nm)(x)
  }
  plot(x, y, type = "l", main = nm, col = "red")
  abline(h = 0, col = "lightgray")
  if (is.deriv) lines(x[-1], dy, col = "blue", lty = 2)
}
```

## "Extended" Pascal triangle:

```r
fN <- function(n) formatC(n, wid=2)
for (n in -4:10) cat(fN(n),":", fN(choose(n, k= -2:max(3,n+2))), 
                     "\n")
```

## R code version of choose() [simplistic; warning for k < 0]:

```r
mychoose <- function(r,k)
  ifelse(k <= 0, (k==0),
        sapply(k, function(k) prod(r:(r-k+1))) / factorial(k))
```

## Binomial theorem for n=1/2 ;

```r
k <- -1:6
cbind(k=k, choose(1/2, k), mychoose(1/2, k))
```

## sqrt(1+x) = (1+x)^{(1/2)} = sum_(k=0)^Inf choose(1/2, k) * x^k :

```r
k <- 0:10 # 10 is sufficient for ~ 9 digit precision:
sqrt(1.25)
sum(choose(1/2, k)* .25^k)
```

---

### split

#### Divide into Groups

**Description**

split divides the data in the vector `x` into the groups defined by `f`. The assignment forms replace values corresponding to such a division. Unsplit reverses the effect of split.
Usage

\[
\text{split}(x, f, \text{drop} = \text{FALSE}, \ldots) \\
\text{split}(x, f, \text{drop} = \text{FALSE}, \ldots) \leftarrow \text{value} \\
\text{unsplit}(\text{value}, f, \text{drop} = \text{FALSE})
\]

Arguments

\begin{itemize}
  \item \textit{x} vector or data frame containing values to be divided into groups.
  \item \textit{f} a “factor” in the sense that \textit{as.factor(f)} defines the grouping, or a list of such factors in which case their interaction is used for the grouping.
  \item \textit{drop} logical indicating if levels that do not occur should be dropped (if \textit{f} is a \textit{factor} or a list).
  \item \textit{value} a list of vectors or data frames compatible with a splitting of \textit{x}. Recycling applies if the lengths do not match.
  \item \ldots further potential arguments passed to methods.
\end{itemize}

Details

\text{split} and \text{split<-.} are generic functions with default and \text{data.frame} methods. \text{f} is recycled as necessary and if the length of \textit{x} is not a multiple of the length of \textit{f} a warning is printed. \text{unsplit} works only with lists of vectors. The \text{data.frame} method can also be used to split a matrix into a list of matrices, and the assignment form likewise, provided they are invoked explicitly.

Any missing values in \textit{f} are dropped together with the corresponding values of \textit{x}.

Value

The value returned from \text{split} is a list of vectors containing the values for the groups. The components of the list are named by the \textit{used} factor levels given by \textit{f}. (If \textit{f} is longer than \textit{x} then some of the components will be of zero length.)

The assignment forms return their right hand side. \text{unsplit} returns a vector for which \text{split}(x, f) equals \text{value}

References


See Also

\texttt{cut}

Examples

\begin{verbatim}
require(stats)
n <- 10; nn <- 100
f <- factor(round(n * runif(n * nn)))
x <- rnorm(n * nn) + sqrt(as.numeric(f))
xg <- split(x, f)
boxplot(xg, col = "lavender", notch = TRUE, varwidth = TRUE)
sapply(xg, length)
sapply(xg, mean)
\end{verbatim}
```r
## Calculate z-scores by group
z <- unsplit(lapply(split(x, g), scale), g)
tapply(z, g, mean)

# or
z <- x
split(z, g) <- lapply(split(x, g), scale)
tapply(z, g, sd)

## Split a matrix into a list by columns
ma <- cbind(x = 1:10, y = (-4:5)^2)
split(ma, col(ma))

split(1:10, 1:2)
```

---

### sprintf

**Use C-style String Formatting Commands**

**Description**

A wrapper for the C function `sprintf`, that returns a character vector containing a formatted combination of text and variable values.

**Usage**

```r
sprintf(fmt, ...)
ggettextf(fmt, ..., domain = NULL)
```

**Arguments**

- `fmt` a format string.
- `...` values to be passed into `fmt`. Only logical, integer, real and character vectors are supported, but some coercion will be done: see the Details section.
- `domain` see `gettext`.

**Details**

`sprintf` is a wrapper for the system `sprintf` C-library function. Attempts are made to check that the mode of the values passed match the format supplied, and R’s special values (`NA`, `Inf`, `-Inf` and `NaN`) are handled correctly.

`gettextf` is a convenience function which provides C-style string formatting with possible translation of the format string.

The arguments (including `fmt`) are recycled if possible a whole number of times to the length of the longest, and then the formatting is done in parallel.

The following is abstracted from Kernighan and Ritchie (see References). The string `fmt` contains normal characters, which are passed through to the output string, and also special characters that operate on the arguments provided through `...`. Special characters start with a `%` and end with one of the letters in the set `difeGgsX%`. These letters denote the following types:
**sprintf**

```
d, i, x, X Integer value, x and X being hexadecimal (using the same case for a-f as the code). Numeric variables with exactly integer values will be coerced to integer.

f Double precision value, in decimal notation of the form "[-]m.ddd". The number of decimal places is specified by the precision: the default is 6; a precision of 0 suppresses the decimal point.

e, E Double precision value, in decimal notation of the form [-]m.ddde[+-]xx or [-]m.dddE[+-]xx.

g, G Double precision value, in %e or %E format if the exponent is less than -4 or greater than or equal to the precision, and %f format otherwise.

s Character string.

% Literal % (none of the formatting characters given below are permitted in this case).

as.character is used for non-character arguments with s and as.double for non-double arguments with f, e, E, g, G. NB: the length is determined before conversion, so do not rely on the internal coercion if this would change the length.

In addition, between the initial % and the terminating conversion character there may be, in any order:

m.n Two numbers separated by a period, denoting the field width (m) and the precision (n)

- Left adjustment of converted argument in its field

+ Always print number with sign

a space Prefix a space if the first number is not a sign

0 For numbers, pad to the field width with leading zeros

Further, as from R 2.1.0, immediately after % may come 1$ to 99$ to refer to the numbered argument: this allows arguments to referenced out of order and is mainly intended for translators of error messages. If this is done it is best if all formats are numbered: if not the unnumbered ones process the arguments in order. See the examples.

As from R 2.2.0, a field width or precision (but not both) may be indicated by an asterisk *. In this case an argument specifies the desired number. A negative field width is taken as a '-' flag followed by a positive field width. A negative precision is taken as if the precision were omitted. The *1$ to *99$ notation for arguments referenced out of order is also supported.

The result has a length limit, probably 8192 bytes, and attempts to exceed this may result in an error, or truncation with a warning.

**Value**

A character vector of length that of the longest input. Character NAs are converted to "NA".

**Author(s)**

Original code by Jonathan Rougier, ⟨J.C.Rougier@durham.ac.uk⟩.

**References**

See Also

`formatC` for a way of formatting vectors of numbers in a similar fashion.
`paste` for another way of creating a vector combining text and values.
`gettext` for the mechanisms for the automated translation of text.

Examples

```r
## be careful with the format: most things in R are floats
## only integer-valued reals get coerced to integer.

sprintf("%s is %f feet tall\n", "Sven", 7.1)  # OK
try(sprintf("%s is %i feet tall\n", "Sven", 7.1)) # not OK
try(sprintf("%s is %i feet tall\n", "Sven", 7))  # OK

## use a literal % :

sprintf("%.0f%% said yes (out of a sample of size %.0f)", 66.666, 3)

## various formats of pi :

sprintf("%f", pi)
sprintf("%.3f", pi)
sprintf("%1.0f", pi)
sprintf("%5.1f", pi)
sprintf("%05.1f", pi)
sprintf("%+f", pi)
sprintf("% f", pi)
sprintf("%-10f", pi)  # left justified
sprintf("%e", pi)
sprintf("%E", pi)
sprintf("%g", pi)
sprintf("%g", 1e6 * pi)  # -> exponential
sprintf("%9g", 1e6 * pi) # -> "fixed"
sprintf("%G", 1e-6 * pi)

## no truncation:

sprintf("%1.f",101)

## re-use one argument three times, show difference between %x and %X

xx <- sprintf("%1$d %1$x %1$X", 0:15)
xx <- matrix(xx, dimnames=list(rep("", 16), "%d%x%X"))
noquote(format(xx, justify="right"))

## More sophisticated:

sprintf("min 10-char string '%10s'",
        c("a", "ABC", "and an even longer one"))

n <- 1:18
sprintf(paste("e with %2d digits = %.",n,"g",sep=""), n, exp(1))

## Using arguments out of order

sprintf("second %2$1.0f, first %1$5.2f, third %3$1.0f", pi, 2, 3)

## Using asterisk for width or precision

sprintf("precision %.3f", 3, pi, 8, pi)
```

sQuote

## Asterisk and argument re-use, e example reiterated:
sprintf("e with %1$2d digits = %2$. * 1$g", n, exp(1))

## re-cycle arguments
sprintf("%s %d", "test", 1:3)

---

### Description

Single or double quote text by combining with appropriate single or double left and right quotation marks.

### Usage

sQuote(x)  
dQuote(x)

### Arguments

x

an \texttt{R} object, to be coerced to a character vector.

### Details

The purpose of the functions is to provide a simple means of markup for quoting text to be used in the \texttt{R} output, e.g., in warnings or error messages.

The choice of the appropriate quotation marks depends on both the locale and the available character sets. Older Unix/X11 fonts displayed the grave accent (0x60) and the apostrophe (0x27) in a way that they could also be used as matching open and close single quotation marks. Using modern fonts, or non-Unix systems, these characters no longer produce matching glyphs. Unicode provides left and right single quotation mark characters (U+2018 and U+2019); if Unicode cannot be assumed, it seems reasonable to use the apostrophe as an undirectional single quotation mark.

Similarly, Unicode has left and right double quotation mark characters (U+201C and U+201D); if only ASCII's typewriter characteristics can be employed, than the ASCII quotation mark (0x22) should be used as both the left and right double quotation mark.

By default, \texttt{sQuote} and \texttt{dQuote} provide undirectional ASCII quotation style. In a UTF-8 locale (see \texttt{l10n\_info}), the Unicode directional quotes are used.

### References


### See Also

Quotes for quoting \texttt{R} code.  
\texttt{shQuote} for quoting OS commands.

### Examples

\texttt{paste("argument", sQuote("x"), "must be non-zero")}
Stack or Unstack Vectors from a Data Frame or List

Description

Stacking vectors concatenates multiple vectors into a single vector along with a factor indicating where each observation originated. Unstacking reverses this operation.

Usage

```r
stack(x, ...)  
## Default S3 method:  
stack(x, ...)  
## S3 method for class 'data.frame':  
stack(x, select, ...)  

unstack(x, ...)  
## Default S3 method:  
unstack(x, form, ...)  
## S3 method for class 'data.frame':  
unstack(x, form = formula(x), ...)  
```

Arguments

- `x` object to be stacked or unstacked
- `select` expression, indicating variables to select from a data frame
- `form` a two-sided formula whose left side evaluates to the vector to be unstacked and whose right side evaluates to the indicator of the groups to create. Defaults to `formula(x)` in `unstack.data.frame`.
- `...` further arguments passed to or from other methods.

Details

The `stack` function is used to transform data available as separate columns in a data frame or list into a single column that can be used in an analysis of variance model or other linear model. The `unstack` function reverses this operation.

Value

- `unstack` produces a list of columns according to the formula `form`. If all the columns have the same length, the resulting list is coerced to a data frame.
- `stack` produces a data frame with two columns
  - `values` the result of concatenating the selected vectors in `x`
  - `ind` a factor indicating from which vector in `x` the observation originated

Author(s)

Douglas Bates
See Also

lm, reshape

Examples

```r
require(stats)
formula(PlantGrowth) # check the default formula
pg <- unstack(PlantGrowth) # unstack according to this formula
pg
stack(pg) # now put it back together
stack(pg, select = -ctrl) # omitting one vector
```

Description

In R, the startup mechanism is as follows.

Unless '---no-environ' was given on the command line, R searches for user and site files to process for setting environment variables. The name of the site file is the one pointed to by the environment variable R_ENVIRON; if this is unset or empty, '$R_HOME/etc/Renviron.site' is used (if it exists, which it does not in a “factory-fresh” installation). The user files searched for are '.Renviron' in the current or in the user’s home directory (in that order). See Details for how the files are read.

Then R searches for the site-wide startup profile unless the command line option '---no-site-file' was given. The name of this file is taken from the value of the R_PROFILE environment variable. If this variable is unset, the default is '$R_HOME/etc/Rprofile.site', which is used if it exists (which it does not in a “factory-fresh” installation). This code is loaded into package base. Users need to be careful not to unintentionally overwrite objects in base, and it is normally advisable to use local if code needs to be executed: see the examples.

Then, unless '---no-init-file' was given, R searches for a file called '.Rprofile' in the current directory or in the user’s home directory (in that order) and sources it into the user workspace.

It then loads a saved image of the user workspace from '.RData' if there is one (unless '---no-restore-data' was specified, or '---no-restore', on the command line).

Next, if a function .First is found on the search path, it is executed as .First(). Finally, function .First.sys() in the base package is run. This calls require to attach the default packages specified by options("defaultPackages").

A function .First (and .Last) can be defined in appropriate '.Rprofile' or 'Rprofile.site' files or have been saved in '.RData'. If you want a different set of packages than the default ones when you start, insert a call to options in the '.Rprofile' or 'Rprofile.site' file. For example, options(defaultPackages = character()) will attach no extra packages on startup. Alternatively, set R_DEFAULT_PACKAGES=NULL as an environment variable before running R. Using options(defaultPackages = "") or R_DEFAULT_PACKAGES="" enforces the R system default.

The commands history is read from the file specified by the environment variable R_HISTFILE (default '.Rhistory') unless '---no-restore-history' was specified (or '---no-restore'). The command-line flag '---vanilla' implies '---no-site-file', '---no-init-file', '---no-restore' and '---no-environ'.
Usage

.First <- function() { ...... }
.Rprofile <startup file>

Details

Note that there are two sorts of files used in startup: environment files which contain lists of environment variables to be set, and profile files which contain R code.

Lines in a site or user environment file should be either comment lines starting with #, or lines of the form name=value. The latter sets the environmental variable name to value, overriding an existing value. If value contains an expression of the form ${foo-bar}, the value is that of the environmental variable foo if that exists and is set to a non-empty value, otherwise bar. (If it is of the form ${foo}, the default is ""). This construction can be nested, so bar can be of the same form (as in ${foo-${bar-blah}}). Note that the braces are essential: $HOME will not be interpreted.

Leading and trailing white space in value are stripped. value is processed in a similar way to a Unix shell. In particular the outermost level of (single or double) quotes is stripped, and backslashes are removed except inside quotes.

Historical notes

Prior to R version 1.4.0, the environment files searched were ‘.Renviron’ in the current directory, the file pointed to by R_ENVIRON if set, and ‘.Renviron’ in the user’s home directory. What is now ‘Rprofile.site’ was called ‘Rprofile’.

Prior to R version 1.2.1, ‘.Rprofile’ was sourced after ‘.RData’ was loaded, although the documented order was as here.

The format for site and user environment files was changed in version 1.2.0. Older files are quite likely to work but may generate warnings on startup if they contained unnecessary export statements.

Values in environment files were not processed prior to version 1.4.0.

See Also

.Last for final actions before termination. commandArgs for accessing the command line arguments.
An Introduction to R for more command-line options: those affecting memory management are covered in the help file for Memory.

For profiling code, see Rprof.

Examples

## Not run:
# Example ~/.Renviron on Unix
R_LIBS=~/.R/library
PAGER=~usr/local/bin/less

# Example .Renviron on Windows
R_LIBS=C:/R/library
MY_TCLTK=yes
TCL_LIBRARY=c:/packages/Tcl/lib/tcl8.4
# Example of .Rprofile
options(width=65, digits=5)
options(show.signif.stars=FALSE)
ps.options(horizontal=FALSE)
set.seed(1234)
.First <- function() cat("\n Welcome to R!\n\n")
.Last <- function() cat("\n Goodbye!\n\n")

# Example of Rprofile.site
local()
    old <-getOption("defaultPackages")
    options(defaultPackages = c(old, "MASS"))
})

## if .Renviron contains
FOOBAR="coo\bar"doh\ex"abc"def'

## then we get
> cat(Sys.getenv("FOOBAR"), "\n")
coo\bardoh\exabc"def'
## End(Not run)

---

**stop**

*Stop Function Execution*

**Description**

`stop` stops execution of the current expression and executes an error action.

`geterrmessage` gives the last error message.

**Usage**

```r
stop(..., call. = TRUE, domain = NULL)
geterrmessage()
```

**Arguments**

- `...` character vectors (which are pasted together with no separator), a condition object, or NULL.
- `call.` logical, indicating if the call should become part of the error message.
- `domain` see `gettext`. If NA, messages will not be translated.

**Details**

The error action is controlled by error handlers established within the executing code and by the current default error handler set by `options(error=)`. The error is first signaled as if using `signalCondition()`. If there are no handlers or if all handlers return, then the error message is printed (if `options("show.error.messages")` is true) and the default error handler is used. The default behaviour (the NULL error handler) in interactive use is to return to the top level prompt or the top level browser, and in non-interactive use to (effectively) call `q("no", status=1, runLast=FALSE)`. The default handler stores the error message in a buffer; it can
be retrieved by `geterrmessage()`.

Errors will be truncated to `getOption("warning.length")` characters, default 1000.

An attempt is made to coerce other types of inputs to character vectors.

Value

`geterrmessage` gives the last error message, as a character string ending in "\n".

References


See Also

`warning`, `try` to catch errors and retry, and `options` for setting error handlers. `stopifnot` for validity testing. `tryCatch` and `withCallingHandlers` can be used to establish custom handlers while executing an expression.

`gettext` for the mechanisms for the automated translation of messages.

Examples

```r
options(error = expression(NULL))
# don't stop on stop(.)  << Use with CARE! >>

iter <- 12
if(iter > 10) stop("too many iterations")

tst1 <- function(...) stop("dummy error")
tst1(1:10, long, calling, expression)

tst2 <- function(...) stop("dummy error", call. = FALSE)
tst2(1:10, longcalling, expression, but.not.seen.in.Error)

options(error = NULL)# revert to default
```

---

**stopifnot**

Ensure the `Truth` of R Expressions

Description

If any of the expressions in ... are not all `TRUE`, `stop` is called, producing an error message indicating the first element of ... which was not true.

Usage

```r
stopifnot(...)```

Arguments

... any number of (logical) R expressions which should evaluate to `TRUE`. 
Details

stopifnot(A, B) is conceptually equivalent to
\{ if(!all(A)) stop(...) ;
  if(!all(B)) stop(...) \}.

Value

(NULL if all statements in \ldots are TRUE.)

See Also

stop, warning.

Examples

stopifnot(1 == 1, all.equal(pi, 3.14159265), 1 < 2) # all TRUE
m <- matrix(c(1,3,3,1), 2,2)
stopifnot(m == t(m), diag(m) == rep(1,2)) # all(.) |=> TRUE
op <- options(error = expression(NULL))
# "disable stop(.)" << Use with CARE! >>
stopifnot(all.equal(pi, 3.141593), 2 < 2, all(1:10 < 12), "a" < "b")
stopifnot(all.equal(pi, 3.1415927), 2 < 2, all(1:10 < 12), "a" < "b")
options(op)# revert to previous error handler

strptime

Date-time Conversion Functions to and from Character

Description

Functions to convert between character representations and objects of classes "POSIXlt" and "POSIXct" representing calendar dates and times.

Usage

## S3 method for class 'POSIXct':
format(x, format = "", tz = "", usetz = FALSE, ...)
## S3 method for class 'POSIXlt':
format(x, format = "", usetz = FALSE, ...)

## S3 method for class 'POSIXt':
as.character(x, ...)
strptime(x, format="", usetz = FALSE, ...)
strptime(x, format)

ISOdatetime(year, month, day, hour, min, sec, tz = "")
ISOdate(year, month, day, hour = 12, min = 0, sec = 0, tz = "GMT")
**Arguments**

- **x**: An object to be converted.
- **tz**: A timezone specification to be used for the conversion. System-specific, but """" is the current time zone, and "GMT"" is UTC.
- **format**: A character string. The default is "%Y-%m-%d %H:%M:%S" if any component has a time component which is not midnight, and "%Y-%m-%d" otherwise.
- **...**: Further arguments to be passed from or to other methods.
- **usetz**: logical. Should the timezone be appended to the output? This is used in printing time, and as a workaround for problems with using "%Z" on most Linux systems.
- **year, month, day**: numerical values to specify a day.
- **hour, min, sec**: numerical values for a time within a day.

**Details**

`strptime` is an alias for `format.POSIXlt`, and `format.POSIXct` first converts to class "POSIXlt" by calling `as.POSIXlt`. Note that only that conversion depends on the time zone.

The usual vector re-cycling rules are applied to `x` and `format` so the answer will be of length that of the longer of the vectors.

Locale-specific conversions to and from character strings are used where appropriate and available. This affects the names of the days and months, the AM/PM indicator (if used) and the separators in formats such as `%x` and `%X`.

The details of the formats are system-specific, but the following are defined by the ISO C / POSIX standard for `strftime` and are likely to be widely available. Any character in the format string other than the % escape sequences is interpreted literally (and `%%` gives %).

- `%a` Abbreviated weekday name in the current locale.
- `%A` Full weekday name in the current locale.
- `%b` Abbreviated month name in the current locale.
- `%B` Full month name in the current locale.
- `%c` Date and time, locale-specific.
- `%d` Day of the month as decimal number (01–31).
- `%H` Hours as decimal number (00–23).
- `%I` Hours as decimal number (01–12).
- `%j` Day of year as decimal number (001–366).
- `%m` Month as decimal number (01–12).
- `%M` Minute as decimal number (00–59).
- `%p` AM/PM indicator in the locale. Used in conjunction with `%I` and not with `%H`.
- `%S` Second as decimal number (00–61), allowing for up to two leap-seconds.
- `%U` Week of the year as decimal number (00–53) using the first Sunday as day 1 of week 1.
- `%w` Weekday as decimal number (0–6, Sunday is 0).
- `%W` Week of the year as decimal number (00–53) using the first Monday as day 1 of week 1.
- `%x` Date, locale-specific.
strptime

%X  Time, locale-specific.
%y  Year without century (00–99). If you use this on input, which century you get is system-specific. So don’t! Often values up to 69 (or 68) are prefixed by 20 and 70–99 by 19.
%Y  Year with century.
%z  (output only.) Offset from Greenwich, so -0800 is 8 hours west of Greenwich.
%Z  (output only.) Time zone as a character string (empty if not available).

Where leading zeros are shown they will be used on output but are optional on input.
Also defined in the current standards but less widely implemented (e.g. not for output on Windows) are

%F  Equivalent to %Y-%m-%d (the ISO 8601 date format).
%g  The last two digits of the week-based year (see %V).
%G  The week-based year (see %V) as a decimal number.
%u  Weekday as a decimal number (1–7, Monday is 1).
%V  Week of the year as decimal number (00–53). If the week (starting on Monday) containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1.

Other format specifiers in common use include

%D  Locale-specific date format such as %m/%d/%y.
%k  The 24-hour clock time with single digits preceded by a blank.
%l  The 12-hour clock time with single digits preceded by a blank.
%n  Newline on output, arbitrary whitespace on input.
%x  The 12-hour clock time (using the locale’s AM or PM).
%r  Equivalent to %H:%M.
%t  Newline on output, arbitrary whitespace on input.
%T  Equivalent to %H:%M:%S.

There are also %O[dhImMSUVwWy] which may emit numbers in an alternative local-dependent format (e.g. roman numerals), and %E[cCYxX] which can use an alternative ‘era’ (e.g. a different religious calendar). Which of these are supported is OS-dependent.

ISOdatetime and ISOdate are convenience wrappers for strptime, that differ only in their defaults and that ISOdate sets a timezone. (For dates without times is would be better to use the "Date" class.)

Value

The format methods and strftime return character vectors representing the time.
strptime turns character representations into an object of class "POSIXlt".
ISOdatetime and ISOdate return an object of class "POSIXct".
Note

The default formats follow the rules of the ISO 8601 international standard which expresses a day as "2001-02-28" and a time as "14:01:02" using leading zeroes as here. The ISO form uses no space to separate dates and times.

If the date string does not specify the date completely, the returned answer may be system-specific. The most common behaviour is to assume that unspecified seconds, minutes or hours are zero, and a missing year, month or day is the current one. If it specifies a date incorrectly, reliable implementations will give an error and the date is reported as NA. Unfortunately some common implementations (such as 'glibc') are unreliable and guess at the intended meaning.

If the timezone specified is invalid on your system, what happens is system-specific but it will probably be ignored.

OS facilities will probably not print years before 1CE (aka 1AD) correctly.

References


See Also

*DateTimeClasses* for details of the date-time classes; *locales* to query or set a locale.

Your system's help pages on *strftime* and *strptime* to see how to specify their formats. Windows users will find no help page for *strptime*: code based on 'glibc' is used (with corrections), so all the format specifiers described here are supported, but with no alternative number representation nor era available in any locale.

Examples

```r
## locale-specific version of date()
format(Sys.time(), "%a %b %d %X %Y %Z")

## read in date info in format 'ddmmmyyyy'
## This will give NA(s) in some locales; setting the C locale
## as in the commented lines will overcome this on most systems.
## lct <- Sys.getlocale("LC_TIME"); Sys.setlocale("LC_TIME", "C")
## setlocale("LC_TIME", "C")
x <- c("1jan1960", "2jan1960", "31mar1960", "30jul1960")
z <- strptime(x, "%d%mnyY")
## Sys.setlocale("LC_TIME", lct)
## z

## read in date/time info in format 'm/d/y h:m:s'
dates <- c("02/27/92", "02/27/92", "01/14/92", "02/28/92", "02/01/92")
x <- paste(dates, times)
z <- strptime(x, "%m/%d/%y %H:%M:%S")
```

**strsplit**

*Split the Elements of a Character Vector*

**Description**

Split the elements of a character vector `x` into substrings according to the presence of substring `split` within them.

**Usage**

```r
strsplit(x, split, extended = TRUE, fixed = FALSE, perl = FALSE)
```

**Arguments**

- `x`: character vector, each element of which is to be split.
- `split`: character vector containing regular expression(s) (unless `fixed = TRUE`) to use as “split”. If empty matches occur, in particular if `split` has length 0, `x` is split into single characters. If `split` has length greater than 1, it is re-cycled along `x`.
- `extended`: logical. If `TRUE`, extended regular expression matching is used, and if `FALSE` basic regular expressions are used.
- `fixed`: logical. If `TRUE` match string exactly, otherwise use regular expressions.
- `perl`: logical. Should perl-compatible regexps be used? Has priority over `extended`.

**Details**

Arguments `x` and `split` will be coerced to character, so you will see uses with `split = NULL` to mean `split = character(0)`, including in the examples below.

Note that splitting into single characters can be done via `split=character(0)` or `split=""`; the two are equivalent as from R 1.9.0.

A missing value of `split` does not split the the corresponding element(s) of `x` at all.

**Value**

A list of length `length(x)` the i-th element of which contains the vector of splits of `x[i]`.

**Warning**

The standard regular expression code has been reported to be very slow when applied to extremely long character strings (tens of thousands of characters or more): the code used when `perl=TRUE` seems much faster and more reliable for such usages.

The `perl = TRUE` option is only implemented for singlebyte and UTF-8 encodings, and will warn if used in a non-UTF-8 multibyte locale.

**See Also**

- `paste` for the reverse, `grep` and `sub` for string search and manipulation; further `nchar`, `substr`.
- `regular expression` for the details of the pattern specification.
Examples

noquote(strsplit("A text I want to display with spaces", NULL)[[1]])

x <- c("as = "asfief", qu = "qwerty", "yuiop[", "b", "stuff.blah.yech")
# split x on the letter e
strsplit(x,"e")

unlist(strsplit("a.b.c", "."))
## [1] "a" "b" "c"
## or
unlist(strsplit("a.b.c", ".", fixed = TRUE))

## a useful function: rev() for strings
strReverse <- function(x)
  sapply(lapply(strsplit(x, NULL), rev), paste, collapse="")

strReverse(c("abc", "Statistics"))

## get the first names of the members of R-core
a <- readLines(file.path(R.home(),"AUTHORS"))[-(1:8)]
a <- a[(0:2)-length(a)]
(a <- sub(" . *","", a))
# and reverse them
strReverse(a)

---

**strtrim**

*Trim Character Strings to Specified Widths*

**Description**

Trim character strings to specified display widths.

**Usage**

strtrim(x, width)

**Arguments**

- **x**: a character vector, or an object which can be coerced to a character vector.
- **width**: Positive integer values: recycled to the length of x.

**Details**

‘Width’ is interpreted as the display width in a monospaced font. What happens with non-printable characters (such as backspace, tab) is implementation-dependent and may depend on the locale (e.g. they may be included in the count or they may be omitted).

Using this function rather than `substr` is important when there might be double-width characters in character vectors.
**structure**

**Value**

A character vector of the same length as \( x \).

**Examples**

```r
strtrim(c("abcdef", "abcdef", "abcdef"), c(1,5,10))
```

---

**strwrap**

**Wrap Character Strings to Format Paragraphs**

**Description**

Each character string in the input is first split into paragraphs (on lines containing whitespace only). The paragraphs are then formatted by breaking lines at word boundaries. The target columns for wrapping lines and the indentation of the first and all subsequent lines of a paragraph can be controlled independently.

**Usage**

```r
strwrap(x, width = 0.9 * getOption("width"), indent = 0, exdent = 0,
        prefix = "", simplify = TRUE)
```
Arguments

- **x** a character vector
- **width** a positive integer giving the target column for wrapping lines in the output.
- **indent** a non-negative integer giving the indentation of the first line in a paragraph.
- **exdent** a non-negative integer specifying the indentation of subsequent lines in paragraphs.
- **prefix** a character string to be used as prefix for each line.
- **simplify** a logical. If `TRUE`, the result is a single character vector of line text; otherwise, it is a list of the same length as `x` the elements of which are character vectors of line text obtained from the corresponding element of `x`. (Hence, the result in the former case is obtained by unlisting that of the latter.)

Details

Whitespace in the input is destroyed. Double spaces after periods (thought as representing sentence ends) are preserved. Currently, possible sentence ends at line breaks are not considered specially.

Indentation is relative to the number of characters in the prefix string.

Examples

```r
## Read in file 'THANKS'.
x <- paste(readLines(file.path(R.home(), "THANKS")), collapse = "\n")
## Split into paragraphs and remove the first three ones
x <- unlist(strsplit(x, "\n[ \t\n]* \n\n")[-(1:3)])
## Join the rest
x <- paste(x, collapse = "\n\n")
## Now for some fun:
writeLines(strwrap(x, width = 60))
writeLines(strwrap(x, width = 60, indent = 5))
writeLines(strwrap(x, width = 60, exdent = 5))
writeLines(strwrap(x, prefix = "THANKS> "))
## Note that messages are wrapped AT the target column indicated by
## 'width' (and not beyond it).
## From an R-devel posting by J. Hosking <jh910@juno.com>.
x <- paste(sapply(sample(10, 100, rep=TRUE),
  function(x) substring("aaaaaaaaaa", 1, x)), collapse = " ")
sapply(10:40,
  function(m)
    c(target = m, actual = max(nchar(strwrap(x, m)))))
```

subset Subsetting Vectors, Matrices and Data Frames

Description

Return subsets of vectors, matrices or data frames which meet conditions.
Usage

subset(x, ...)

## Default S3 method:
subset(x, subset, ...)

## S3 method for class 'matrix':
subset(x, subset, select, drop = FALSE, ...)

## S3 method for class 'data.frame':
subset(x, subset, select, drop = FALSE, ...)

Arguments

x          object to be subsetted.
subset     logical expression indicating elements or rows to keep: missing values are taken
           as false.
select     expression, indicating columns to select from a data frame.
drop       passed on to [ indexing operator.
...         further arguments to be passed to or from other methods.

Details

This is a generic function, with methods supplied for matrices, data frames and vectors (including
lists). Packages and users can add further methods.

For ordinary vectors, the result is simply x[subset & !is.na(subset)].

For data frames, the subset argument works on the rows. Note that subset will be evaluated
in the data frame, so columns can be referred to (by name) as variables in the expression (see the
examples).

The select argument exists only for the methods for data frames and matrices. It works by first
replacing column names in the selection expression with the corresponding column numbers in the
data frame and then using the resulting integer vector to index the columns. This allows the use of
the standard indexing conventions so that for example ranges of columns can be specified easily, or
single columns can be dropped (see the examples).

The drop argument is passed on to the indexing method for matrices and data frames: note that
the default for matrices is different from that for indexing.

Value

An object similar to x contain just the selected elements (for a vector), rows and columns (for a
matrix or data frame), and so on.

Author(s)

Peter Dalgaard and Brian Ripley

See Also

[,.transform
Examples

subset(airquality, Temp > 80, select = c(Ozone, Temp))
subset(airquality, Day == 1, select = -Temp)
subset(airquality, select = Ozone:Wind)

with(airquality, subset(Ozone, Temp > 80))

## sometimes requiring a logical 'subset' argument is a nuisance
nm <- rownames(state.x77)
start_with_M <- nm %in% grep("^M", nm, value=TRUE)
subset(state.x77, start_with_M, Illiteracy:Murder)

---

substitute

Substituting and Quoting Expressions

Description

substitute returns the parse tree for the (unevaluated) expression expr, substituting any variables bound in env.

quote simply returns its argument. The argument is not evaluated and can be any R expression.

Usage

substitute(expr, env)
quote(expr)

Arguments

expr Any syntactically valid R expression
env An environment or a list object. Defaults to the current evaluation environment.

Details

The typical use of substitute is to create informative labels for data sets and plots. The myplot example below shows a simple use of this facility. It uses the functions deparse and substitute to create labels for a plot which are character string versions of the actual arguments to the function myplot.

Substitution takes place by examining each component of the parse tree as follows: If it is not a bound symbol in env, it is unchanged. If it is a promise object, i.e., a formal argument to a function or explicitly created using delayedAssign(), the expression slot of the promise replaces the symbol. If it is an ordinary variable, its value is substituted, unless env is .GlobalEnv in which case the symbol is left unchanged.

Value

The mode of the result is generally "call" but may in principle be any type. In particular, single-variable expressions have mode "name" and constants have the appropriate base mode.
Note

Substitute works on a purely lexical basis. There is no guarantee that the resulting expression makes any sense.

Substituting and quoting often causes confusion when the argument is `expression(...)`. The result is a call to the `expression` constructor function and needs to be evaluated with `eval` to give the actual expression object.

References


See Also

`missing` for argument “missingness”, `bquote` for partial substitution, `sQuote` and `dQuote` for adding quotation marks to strings.

Examples

```r
(s.e <- substitute(expression(a + b), list(a = 1))) #> expression(1 + b)
(s.s <- substitute(a + b, list(a = 1))) #> 1 + b

c(mode(s.e), typeof(s.e)) # "call", "language"
c(mode(s.s), typeof(s.s)) # (the same)

# but:
(e.s.e <- eval(s.e)) #> expression(1 + b)
c(mode(e.s.e), typeof(e.s.e)) # "expression", "expression"

substitute(x <= x + 1, list(x=1)) # nonsense

myplot <- function(x, y)
  plot(x, y, xlab=deparse(substitute(x)),
       ylab=deparse(substitute(y)))

## Simple examples about lazy evaluation, etc:

f1 <- function(x, y = x) {
  x <- x + 1; y }
s1 <- function(x, y = substitute(x)) {
  x <- x + 1; y }
s2 <- function(x, y) {
  if(missing(y)) y <- substitute(x); x <- x + 1; y }
a <- 10
f1(a)# 11
s1(a)# 11
s2(a)# a
typeof(s2(a))# "symbol"
```

substr

Substrings of a Character Vector

Description

Extract or replace substrings in a character vector.
Usage

substr(x, start, stop)
substring(text, first, last = 1000000)
substr(x, start, stop) <- value
substring(text, first, last = 1000000) <- value

Arguments

x, text a character vector
start, first integer. The first element to be replaced.
stop, last integer. The last element to be replaced.
value a character vector, recycled if necessary.

Details

substring is compatible with S, with first and last instead of start and stop. For vector arguments, it expands the arguments cyclically to the length of the longest provided none are of zero length.
When extracting, if start is larger than the string length then "" is returned.
For the replacement functions, if start is larger than the string length then no replacement is done.
If the portion to be replaced is longer than the replacement string, then only the portion the length of the string is replaced.

Value

For substr, a character vector of the same length as x.
For substring, a character vector of length the longest of the arguments.

Note

The S4 version of substring<- ignores last; this version does not.
These functions are often used with nchar to truncate a display. That does not really work (you want to limit the width, not the number of characters, so it would be better to use strtrim), but at least make sure you use nchar(type="c").

References


See Also

strsplit, paste, nchar.

Examples

substr("abcdef",2,4)
substring("abcdef",1:6,1:6)
## strsplit is more efficient ...
substr(rep("abcdef",4),1:4,4:5)
x <- c("asfef", "qwerty", "yuiop", "b", "stuff.blah.yech")
sum

substr(x, 2, 5)
substring(x, 2, 4:6)
substring(x, 2) <- c("..", "+++")
x

---

sum Sum of Vector Elements

Description
sum returns the sum of all the values present in its arguments. If na.rm is FALSE an NA value in any of the arguments will cause a value of NA to be returned, otherwise NA values are ignored.

Usage
sum(..., na.rm = FALSE)

Arguments
...
  numeric or complex vectors.
na.rm
  logical. Should missing values be removed?

Details
This is a generic function: methods can be defined for it directly or via the Summary group generic.

Value
The sum. If all of ... are of type integer or logical, then so is the sum, and in that case the result will be NA (with a warning) if integer overflow occurs.
NB: the sum of an empty set is zero, by definition.

References

---

summary Object Summaries

Description
summary is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.
Usage

summary(object, ...)

## Default S3 method:
summary(object, ..., digits = max(3, getOption("digits")-3))

## S3 method for class "data.frame":
summary(object, maxsum = 7,
digits = max(3, getOption("digits")-3), ...)

## S3 method for class "factor":
summary(object, maxsum = 100, ...)

## S3 method for class "matrix":
summary(object, ...)

Arguments

object an object for which a summary is desired.
maxsum integer, indicating how many levels should be shown for factors.
digits integer, used for number formatting with signif() (for summary.default) or format() (for summary.data.frame).
... additional arguments affecting the summary produced.

Details

For factors, the frequency of the first maxsum - 1 most frequent levels is shown, where the less frequent levels are summarized in "(Others)" (resulting in maxsum frequencies).

The functions summary.lm and summary.glm are examples of particular methods which summarise the results produced by lm and glm.

Value

The form of the value returned by summary depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

References


See Also

anova, summary.glm, summary.lm.

Examples

summary(attenu, digits = 4) #-> summary.data.frame(...), default precision
summary(attenu$station, maxsum = 20) #-> summary.factor(...)

lst <- unclass(attenu$station) > 20 # logical with NAs
## summary.default() for logicals -- different from *.factor:
summary(lst)
summary(as.factor(lst))
Singular Value Decomposition of a Matrix

Description

Compute the singular-value decomposition of a rectangular matrix.

Usage

```r
svd(x, nu = min(n, p), nv = min(n, p), LINPACK = FALSE)
```

```r
La.svd(x, nu = min(n, p), nv = min(n, p),
       method = c("dgesdd", "dgesvd"))
```

Arguments

- `x` a matrix whose SVD decomposition is to be computed.
- `nu` the number of left singular vectors to be computed. This must be one of 0, `nrow(x)` and `ncol(x)`, except for method = "dgesdd".
- `nv` the number of right singular vectors to be computed. This must be one of 0 and `ncol(x)`.
- `LINPACK` logical. Should LINPACK be used (for compatibility with R < 1.7.0)?
- `method` The LAPACK routine to use in the real case.

Details

The singular value decomposition plays an important role in many statistical techniques. `svd` and `La.svd` provide two slightly different interfaces. The main functions used are the LAPACK routines DGESDD and ZGESVD; `svd(LINPACK=TRUE)` provides an interface to the LINPACK routine DSVDC, purely for backwards compatibility.

`La.svd` provides an interface to both the LAPACK routines DGESVD and DGESDD. The latter is usually substantially faster if singular vectors are required: see [http://www.cs.berkeley.edu/~demmel/DOE2000/Report0100.html](http://www.cs.berkeley.edu/~demmel/DOE2000/Report0100.html). Most benefit is seen with an optimized BLAS system.

Computing the singular vectors is the slow part for large matrices.

Unsuccessful results from the underlying LAPACK code will result in an error giving a positive error code: these can only be interpreted by detailed study of the FORTRAN code.

Value

The SVD decomposition of the matrix as computed by LINPACK,

\[ X = UDV', \]

where \( U \) and \( V \) are orthogonal, \( V' \) means \( V \) transposed, and \( D \) is a diagonal matrix with the singular values \( D_{ii} \). Equivalently, \( D = U'XV \), which is verified in the examples, below.

The returned value is a list with components

- `d` a vector containing the singular values of \( x \).
sweep

a matrix whose columns contain the left singular vectors of \( x \), present if \( n_u > 0 \).

\( v \) a matrix whose columns contain the right singular vectors of \( x \), present if \( n_v > 0 \).

For \( \text{La.svd} \) the return value replaces \( v \) by \( v^t \), the (conjugated if complex) transpose of \( v \).

References


See Also

eigen, qr.
capabilities to test for IEEE 754 arithmetic.

Examples

hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
X <- hilbert(9)[,1:6]
(s <- svd(X))
D <- diag(s$d)
%*% s$v %*% t(s$u) # X = U D V'
t(s$u) %*% X %*% s$v # D = U' X V

sweep

Sweep out Array Summaries

Description

Return an array obtained from an input array by sweeping out a summary statistic.

Usage

sweep(x, MARGIN, STATS, FUN="-", ...)

Arguments

x an array.
MARGIN a vector of indices giving the extents of \( x \) which correspond to \( \text{STATS} \).
STATS the summary statistic which is to be swept out.
FUN the function to be used to carry out the sweep. In the case of binary operators such as "/" etc., the function name must be quoted.
... optional arguments to FUN.
Value
An array with the same shape as \( x \), but with the summary statistics swept out.

References

See Also
*apply* on which *sweep* used to be based; *scale* for centering and scaling.

Examples
```r
require(stats) # for median
med.att <- apply(attitude, 2, median)
sweep(data.matrix(attitude), 2, med.att)# subtract the column medians
```

---

switch

Select One of a List of Alternatives

Description
switch evaluates \( \text{EXPR} \) and accordingly chooses one of the further arguments (in \( ... \)).

Usage
```
switch(EXPR, ...)
```

Arguments
- \( \text{EXPR} \) an expression evaluating to a number or a character string.
- \( ... \) the list of alternatives, given explicitly.

Details
If the value of \( \text{EXPR} \) is an integer between 1 and \( \text{nargs()} - 1 \) then the corresponding element of \( ... \) is evaluated and the result returned.

If \( \text{EXPR} \) returns a character string then that string is used to match the names of the elements in \( ... \). If there is an exact match then that element is evaluated and returned if there is one, otherwise the next element is chosen, e.g., \( \text{switch("cc", a=1, cc=, d=2)} \) evaluates to 2.

In the case of no match, if there's a further argument in *switch* that one is returned, otherwise NULL.

Warning
Beware of partial matching: an alternative \( E = \text{foo} \) will match the first argument \( \text{EXPR} \) unless that is named. See the examples for good practice in naming the first argument.
Examples

```r
require(stats)
centre <- function(x, type) {
  switch(type,
    mean = mean(x),
    median = median(x),
    trimmed = mean(x, trim = .1))
}
x <- rcauchy(10)
centre(x, "mean")
centre(x, "median")
centre(x, "trimmed")

ccc <- c("b","Q","a","A","bb")
for(ch in ccc)
cat(ch,":",switch(EXPR = ch, a=1, b=2:3), "\n")
for(ch in ccc)
cat(ch,":",switch(EXPR = ch, a=, A=1, b=2:3, "Otherwise: last"),"\n")

## Numeric EXPR don't allow 'otherwise':
for(i in c(-1:3,9)) print(switch(i, 1,2,3,4))
```

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Operator Syntax and Precedence</th>
</tr>
</thead>
</table>

References


Description

Outlines R syntax and gives the precedence of operators.

Details

The following unary and binary operators are defined. They are listed in precedence groups, from highest to lowest.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[</td>
<td>indexing</td>
</tr>
<tr>
<td>::</td>
<td>access variables in a name space</td>
</tr>
<tr>
<td>$ @</td>
<td>component / slot extraction</td>
</tr>
<tr>
<td>^</td>
<td>exponentiation (right to left)</td>
</tr>
<tr>
<td>- +</td>
<td>unary minus and plus</td>
</tr>
<tr>
<td>:</td>
<td>sequence operator</td>
</tr>
<tr>
<td>%any%</td>
<td>special operators</td>
</tr>
<tr>
<td>* /</td>
<td>multiply, divide</td>
</tr>
<tr>
<td>+ -</td>
<td>(binary) add, subtract</td>
</tr>
<tr>
<td>&lt; &gt; &lt;= &gt;= == !=</td>
<td>ordering and comparison</td>
</tr>
<tr>
<td>!</td>
<td>negation</td>
</tr>
<tr>
<td>&amp; &amp; &amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Within an expression operators of equal precedence are evaluated from left to right except where indicated.

The links in the **See Also** section covers most other aspects of the basic syntax.

**Note**

There are substantial precedence differences between R and S. In particular, in S `?` has the same precedence as (binary) `+ -` and `& && | ||` have equal precedence.

**References**


**See Also**

*Arithmetic, Comparison, Control, Extract, Logic, NumericConstants, Paren, Quotes.*

The *R Language Definition* manual.

---

**Sys.getenv**

*Get Environment Variables*

**Description**

`Sys.getenv` obtains the values of the environment variables named by `x`.

**Usage**

`Sys.getenv(x)`

**Arguments**

- `x` a character vector, or missing

**Value**

A vector of the same length as `x`, with the variable names as its `names` attribute. Each element holds the value of the environment variable named by the corresponding component of `x` (or `""` if no environment variable with that name was found).

On most platforms `Sys.getenv()` will return a named vector giving the values of all the environment variables.
Sys.info

Extract System and User Information

Description

Reports system and user information.

Usage

Sys.info()

Details

This function is not implemented on all R platforms, and returns NULL when not available. Where possible it is based on POSIX system calls.

Sys.info() returns details of the platform R is running on, whereas R.version gives details of the platform R was built on: they may well be different.

Value

A character vector with fields

- sysname: The operating system.
- release: The OS release.
- version: The OS version.
- nodename: A name by which the machine is known on the network (if any).
- machine: A concise description of the hardware.
- login: The user's login name, or "unknown" if it cannot be ascertained.
- user: The name of the real user ID, or "unknown" if it cannot be ascertained.

The information is obtained from Windows system calls. It is likely to be most complete on Windows NT/2000/XP systems.

Note

The meaning of OS “release” and “version” is highly system-dependent and there is no guarantee that the node or login or user names will be what you might reasonably expect. (In particular on some Linux distributions the login name is unknown from sessions with re-directed inputs.)
See Also


Examples

Sys.info()
## An alternative (and probably better) way to get the login name on Unix
Sys.getenv("LOGNAME")

sys.parent

Functions to Access the Function Call Stack

Description

These functions provide access to environments ("frames" in S terminology) associated with functions further up the calling stack.

Usage

sys.call(which = 0)
sys.frame(which = 0)
sys.nframe()
sys.function(which = 0)
sys.parent(n = 1)

sys.calls()
sys.frames()
sys.parents()
sys.on.exit()
sys.status()
parent.frame(n = 1)

Arguments

which the frame number if non-negative, the number of frames to go back if negative.
n the number of generations to go back. (See the Details section.)

Details

_GlobalEnv_ is given number 0 in the list of frames. Each subsequent function evaluation increases the frame stack by 1 and the call, function definition and the environment for evaluation of that function are returned by `sys.call`, `sys.function` and `sys.frame` with the appropriate index.

`sys.call`, `sys.frame` and `sys.function` accept integer values for the argument `which`. Non-negative values of `which` are frame numbers whereas negative values are counted back from the frame number of the current evaluation.

The parent frame of a function evaluation is the environment in which the function was called. It is not necessarily numbered one less than the frame number of the current evaluation, nor is it the environment within which the function was defined. `sys.parent` returns the number of the parent frame if `n` is 1 (the default), the grandparent if `n` is 2, and so on.
sys.nframe returns an integer, the number of the current frame as described in the first paragraph.

sys.calls and sys.frames give a pairlist of all the active calls and frames, repectively, and
sys.parents returns an integer vector of indices of the parent frames of each of those frames.

Notice that even though the sys.xxx functions (except sys.status) are interpreted, their contexts are not counted nor are they reported. There is no access to them.

sys.status() returns a list with components sys.calls, sys.parents and
sys.frames, the results of calls to those three functions (which this will include the call to sys.status: see the first example).

sys.on.exit() returns the expression stored for use by on.exit in the function currently being evaluated. (Note that this differs from S, which returns a list of expressions for the current frame and its parents.)

parent.frame(n) is a convenient shorthand for sys.frame(sys.parent(n)) (implemented slightly more efficiently).

Value

sys.call returns a call, sys.function a function definition, and sys.frame, sys.parent and parent.frame return an environment.

For the other functions, see the Details section.

References

Brooks/Cole. (not parent.frame.)

See Also

eval for a usage of sys.frame and parent.frame.

Examples

## Note: the first two examples will give different results
## if run by example().
ff <- function(x) gg(x)

gg <- function(y) sys.status()
str(ff(1))

gg <- function(y) {
  ggg <- function() {
    cat("current frame is", sys.nframe(), "\n")
    cat("parents are", sys.parents(), "\n")
    print(sys.function(0)) # ggg
    print(sys.function(2)) # gg
  }
  if(y > 0) gg(y-1) else ggg()
}

gg(3)

t1 <- function() {
  aa <- "here"
  t2 <- function() {
    ## in frame 2 here
    cat("current frame is", sys.nframe(), "\n")
Sys.putenv

```r
str(sys.calls()) ## list with two components t1() and t2()
cat("parents are frame numbers", sys.parents(), "\n") ## 0 1
print(ls(envir=sys.frame(-1))) ## [1] "aa" "t2"
invisible()
}
t2()
}
t1()

test.sys.on.exit <- function() {
on.exit(print(1))
ex <- sys.on.exit()
str(ex)
cat("exiting...\n")
}
test.sys.on.exit()
## gives 'language print(1)', prints 1 on exit

## An example where the parent is not the next frame up the stack
## since method dispatch uses a frame.
as.double.foo <- function(x)
{
  str(sys.calls())
  print(sys.frames())
  print(sys.parents())
  print(sys.frame(-1)); print(parent.frame())
  x
}  
t2 <- function(x) as.double(x)
a <- structure(pi, class = "foo")
t2(a)
```

---

### Description

`Sys.putenv` sets environment variables (for other processes called from within R or future calls to `Sys.getenv` from this R process).

### Usage

```r
Sys.putenv(...)```

### Arguments

... arguments in name=value form, with value coercible to a character string.

### Details

Non-standard R names must be quoted: see the Examples section.
Value
A logical vector of the same length as x, with elements being true if setting the corresponding variable succeeded.

Note
Not all systems need support Sys.putenv.

See Also
Sys.getenv, setwd for the working directory.

Examples
print(Sys.putenv("R_TEST"="testit", ABC=123))
Sys.getenv("R_TEST")

Sys.sleep Suspend Execution for a Time Interval

Description
Suspend execution of R expressions for a given number of seconds

Usage
Sys.sleep(time)

Arguments
time The time interval to suspend execution for, in seconds.

Details
Using this function allows R to be given very low priority and hence not to interfere with more important foreground tasks. A typical use is to allow a process launched from R to set itself up and read its input files before R execution is resumed.

The intention is that this function suspends execution of R expressions but wakes the process up often enough to respond to GUI events, typically every 0.5 seconds.

There is no guarantee that the process will sleep for the whole of the specified interval, and it may well take slightly longer in real time to resume execution. The resolution of the time interval is system-dependent, but will normally be down to 0.02 secs or better.

Value
Invisible NULL.

Note
This function may not be implemented on all systems.
Examples

testit <- function(x)
{
  pl <- proc.time()
  Sys.sleep(x)
  proc.time() - pl # The cpu usage should be negligible
}
testit(3.7)

Description

Parses expressions in the given file, and then successively evaluates them in the specified environment.

Usage

sys.source(file, envir = baseenv(), chdir = FALSE,
           keep.source = getOption("keep.source.pkgs"))

Arguments

  file a character string naming the file to be read from
  envir an R object specifying the environment in which the expressions are to be evaluated. May also be a list or an integer. The default value NULL corresponds to evaluation in the base environment. This is probably not what you want; you should typically supply an explicit envir argument.
  chdir logical; if TRUE, the R working directory is changed to the directory containing file for evaluating.
  keep.source logical. If TRUE, functions “keep their source” including comments, see options(keep.source = *) for more details.

Details

For large files, keep.source = FALSE may save quite a bit of memory. In order for the code being evaluated to use the correct environment (for example, in global assignments), source code in packages should call toopenv(), which will return the namespace, if any, the environment set up by sys.source, or the global environment if a saved image is being used.

See Also

source, and library which uses sys.source.
Sys.time

Get Current Date, Time and Timezone

Description

Sys.time and Sys.Date returns the system’s idea of the current date with and without time, and Sys.timezone returns the current time zone.

Usage

Sys.time()
Sys.Date()
Sys.timezone()

Details

Sys.time returns an absolute date-time value which can be converted in various time zones and may return different days.
Sys.Date returns the day in the current timezone.

Value

Sys.time returns an object of class "POSIXct" (see DateTimeClasses).
Sys.Date returns an object of class "Date" (see Date).
Sys.timezone returns an OS-specific character string, possibly an empty string.

See Also

date for the system time in a fixed-format character string; the elapsed time component of proc.time for finer resolution in changes in time.

Examples

Sys.time()
  ## locale-specific version of date()
  format(Sys.time(), "%a %b %d %X %Y")

Sys.Date()

Sys.timezone()
**system**  
Invoke a System Command

**Description**

system invokes the system command specified by command.

**Usage**

```r
system(command, intern = FALSE, wait = TRUE, input = NULL,
        show.output.on.console = FALSE,
        minimized = FALSE, invisible = FALSE)
```

**Arguments**

- `command`  
  the system command to be invoked, as a string.
- `intern`  
  a logical, indicates whether to make the output of the command an R object.
- `wait`  
  should the R interpreter wait for the command to finish? The default is to wait, and the interpreter will always wait if `intern = TRUE`.
- `input`  
  if a character vector is supplied, this is copied one string per line to a temporary file, and the standard input of `command` is redirected to the file.
- `show.output.on.console`  
  a logical, indicates whether to capture the output of the command and show it on the R console (not used by Rterm, which captures the output unless `wait` is false).
- `minimized`  
  a logical, indicates whether the command window should be initially displayed as a minimized window.
- `invisible`  
  a logical, indicates whether the command window should be visible on the screen.

**Details**

cmd is parsed as a command plus arguments separated by spaces. So if the path to the command contains spaces, it must be (double-)quoted. See the examples. (Note: a Windows path name cannot contain a double quote, so we do not need to worry about escaping embedded quotes.)

The command is run directly as a Windows command by the Windows API call `CreateProcess`: extensions of `.exe`, `.com`, `.cmd` and `.bat` are tried in turn if none is supplied. (To use DOS internal commands use `paste(Sys.getenv("COMSPEC"),"/c",cmd)`.) The search path for `command` may be system-dependent: it will include the R bin directory, the working directory and the Windows system directories before PATH.

Precisely what is seen by the user depends on whether Rgui or Rterm is being used. For Rgui a new console will always be used, so a commands window will appear for the duration of console applications unless `invisible` is true. For Rterm a separate commands window will appear for console applications only if `wait = FALSE`.

unix is a deprecated alternative, available for backwards compatibility.
Value

If `intern = TRUE`, a character vector giving the output of the command, one line per character string. If the command could not be run or gives an error a R error is generated. (Output lines of more than 8095 characters will be split.)

If `intern = FALSE`, the return value is an error code, given the invisible attribute (so needs to be printed explicitly). If the command could not be run for any reason, the value is −1 and an R warning is generated. Otherwise if `wait = FALSE` the value is the error code returned by the command, and if `wait = TRUE` it is the zero (the conventional success value).

If `intern = FALSE` and `show.output.on.console = TRUE` the text output from a command that is a console application will appear in the R console (Rgui) or the window running R (Rterm).

Warning

The command cannot be interrupted by the R process.

Do not run console applications that require user input from Rgui setting `intern = TRUE` and/or `show.output.on.console = TRUE`. They will not work, may hang and then will probably hang Rgui too.

See Also

`shell` or `shell.exec` for a less raw interface.

Examples

```r
# launch an editor, wait for it to quit
## Not run: system("notepad myfile.txt")
# launch a Windows 9x process monitor (from Win9x KernelToys)
## Not run: system("wintop", wait = FALSE)
# launch your favourite shell:
## Not run: system(Sys.getenv("COMSPEC"))
## Not run:
## note the two sets of quotes here:
system(paste("c:/Program Files/Mozilla Firefox/firefox.exe","-
-url cran.r-project.org"), wait = FALSE)
## End(Not run)
```

---

**system.file**

Find Names of R System Files

**Description**

Finds the full file names of files in packages etc.

**Usage**

```r
system.file(..., package = "base", lib.loc = NULL)
```
system.time

Arguments

... character strings, specifying subdirectory and file(s) within some package. The default, none, returns the root of the package. Wildcards are not supported.
package a character string with the name of a single package. An error occurs if more than one package name is given.
lib.loc a character vector with path names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.

Value

A character vector of positive length, containing the file names that matched ..., or the empty string, ".", if none matched. If matching the root of a package, there is no trailing separator.

As a special case, system.file() gives the root of the base package only.

See Also

list.files

Examples

system.file()  # The root of the 'base' package
system.file(package = "stats")  # The root of package 'stats'
system.file("INDEX")
system.file("help", "AnIndex", package = "splines")

\begin{tabular}{ll}
\hline
\textbf{system.time} & \textbf{CPU Time Used} \\
\hline
\end{tabular}

Description

Return CPU (and other) times that expr used.

Usage

system.time(expr, gcFirst = TRUE)
unix.time(expr, gcFirst = TRUE)

Arguments

expr Valid R expression to be “timed”
gcFirst Logical - should a garbage collection be performed immediately before the timing? Default is TRUE.
Details

system.time calls the built-in `proc.time`, evaluates `expr`, and then calls `proc.time` once more, returning the difference between the two `proc.time` calls.

`unix.time` is an alias of `system.time`, for compatibility reasons.

Timings of evaluations of the same expression can vary considerably depending on whether the evaluation triggers a garbage collection. When `gcFirst` is `TRUE` a garbage collection (`gc`) will be performed immediately before the evaluation of `expr`. This will usually produce more consistent timings.

Value

A numeric vector of length 5 containing the user cpu, system cpu, elapsed, subproc1, subproc2 times. The subproc times are the user and system cpu time used by child processes (and so are usually zero).

On Windows the subproc times are not available and so are always `NA`. The first two components are not available on Windows 9x, and so are reported as `NA`; they do return real values on Windows NT4/2000/XP.

The resolution of the times will be system-specific; it is common for them to be recorded to of the order of 1/100 second, and elapsed time is rounded to the nearest 1/100.

See Also

`proc.time`, `time` which is for time series.

Examples

```r
require(stats)
system.time(for(i in 1:100) mad(runif(1000)))
## Not run:
exT <- function(n = 1000) {
  # Purpose: Test if system.time works ok; n: loop size
  system.time(for(i in 1:n) x <- mean(rt(1000, df=4)))
}
#-- Try to interrupt one of the following (using Ctrl-C / Escape):
exT() #~ about 3 secs on a 1GHz PIII
system.time(exT()) #~ +/- same
## End(Not run)
```

---

\[ t \]

**Matrix Transpose**

Description

Given a matrix or `data.frame` \( x \), \( t(x) \) returns the transpose of \( x \).

Usage

\[ t(x) \]
Arguments

\(x\)  
a matrix or data frame, typically.

Details

A data frame is first coerced to a matrix: see \texttt{as.matrix}. When \(x\) is a vector, it is treated as “column”, i.e., the result is a 1-row matrix.

References


See Also

\texttt{aperm} for permuting the dimensions of arrays.

Examples

\begin{verbatim}
a <- matrix(1:30, 5,6)
ta <- t(a)  ##-- i.e., a[i, j] == ta[j, i] for all i,j :
for(j in seq(ncol(a)))
  if(! all(a[, j] == ta[j, ])) stop("wrong transpose")
\end{verbatim}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{Cross Tabulation and Table Creation} \\
\hline
\end{tabular}
\end{table}

Description

table uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

Usage

\begin{verbatim}
table(\ldots, exclude = c(\text{NA}, \text{NaN}), dnn = \text{list.names(\ldots)},
         \text{deparse.level = 1})
as.table(x, \ldots)
is.table(x)
\end{verbatim}

\begin{verbatim}
as.data.frame.table(x, row.names = \text{NULL}, optional = \text{FALSE},
                     responseName = "Freq")
\end{verbatim}

Arguments

\begin{verbatim}
\ldots  
\end{verbatim}

objects which can be interpreted as factors (including character strings), or a list (or data frame) whose components can be so interpreted. (For \texttt{as.table}, arguments passed to specific methods.)

\begin{verbatim}
exclude  
\end{verbatim}

values to use in the exclude argument of \texttt{factor} when interpreting non-factor objects; if specified, levels to remove from all factors in \ldots

\begin{verbatim}
dnn
\end{verbatim}

the names to be given to the dimensions in the result (the \texttt{dimnames names}).
controls how the default dnn is constructed. See details.
x an arbitrary R object, or an object inheriting from class "table" for the
as.data.frame method.
row.names a character vector giving the row names for the data frame.
optional a logical controlling whether row names are set. Currently not used.
responseName The name to be used for the column of table entries, usually counts.

Details
If the argument dnn is not supplied, the internal function list.names is called to compute the
'dimname names'. If the arguments in ... are named, those names are used. For the remaining
arguments, deparse.level = 0 gives an empty name, deparse.level = 1 uses the
supplied argument if it is a symbol, and deparse.level = 2 will deparse the argument.

Only when exclude is specified (i.e., not by default), will table drop levels of factor arguments
potentially.

Note that as.data.frame.table is also the "table" method for as.data.frame, but
the responseName argument can only be given if it is called explicitly.

Value

The internal function list.names is called to compute the
'dimname names'. If the arguments in ... are named, those names are used. For the remaining
arguments, deparse.level = 0 gives an empty name, deparse.level = 1 uses the
supplied argument if it is a symbol, and deparse.level = 2 will deparse the argument.

Only when exclude is specified (i.e., not by default), will table drop levels of factor arguments
potentially.

Note that as.data.frame.table is also the "table" method for as.data.frame, but
the responseName argument can only be given if it is called explicitly.

References
Brooks/Cole.

See Also
Use ftable for printing (and more) of multidimensional tables.

Examples
require(stats) # for rpois and xtabs
## Simple frequency distribution
table(rpois(100,5))
attach(warpbreaks)
## Check the design:
table(wool, tension)
detach()
table(state.division, state.region)

# simple two-way contingency table
with(airquality, table(cut(Temp, quantile(Temp)), Month))

a <- letters[1:3]
table(a, sample(a)) # dnn is c("a", ")
table(a, sample(a), deparse.level = 0) # dnn is c(", ")
table(a, sample(a), deparse.level = 2) # dnn is c("a", "sample(a)")

## xtabs() <-> as.data.frame.table()

UCBAdmissions ## already a contingency table
DF <- as.data.frame(UCBAdmissions)
class(tab <- xtabs(Freq ~ ., DF)) # xtabs & table
## tab *is* "the same" as the original table:
all(tab == UCBAdmissions)
all.equal(dimnames(tab), dimnames(UCBAdmissions))

a <- rep(c(NA, 1/0:3), 10)
table(a)
table(a, exclude=NULL)
b <- factor(rep(c("A","B","C"), 10))
table(b)
table(b, exclude="B")
d <- factor(rep(c("A","B","C"), 10), levels=c("A","B","C","D","E"))
table(d, exclude="B")

## NA counting:

is.na(d) <- 3:4
d <- factor(d, exclude=NULL)
d[1:7]
table(d, exclude = NULL)

---

### tabulate

**Tabulation for Vectors**

**Description**

Tabulate takes the integer-valued vector `bin` and counts the number of times each integer occurs in it.

**Usage**

`tabulate(bin, nbins = max(1, bin))`

**Arguments**

- `bin` a numeric vector (of positive integers), or a factor.
- `nbins` the number of bins to be used.

**Details**

Tabulate is used as the basis of the `table` function.

If `bin` is a factor, its internal integer representation is tabulated.

If the elements of `bin` are numeric but not integers, they are truncated to the nearest integer.
tapply

Apply a Function Over a “Ragged” Array

Description

Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

Usage

`tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)`

Arguments

- **X**: an atomic object, typically a vector.
- **INDEX**: list of factors, each of same length as X.
- **FUN**: the function to be applied. In the case of functions like +, %*%, etc., the function name must be quoted. If FUN is NULL, tapply returns a vector which can be used to subscript the multi-way array tapply normally produces. 
- **...**: optional arguments to FUN.
- **simplify**: If FALSE, tapply always returns an array of mode "list". If TRUE (the default), then if FUN always returns a scalar, tapply returns an array with the mode of the scalar.

Value

When FUN is present, tapply calls FUN for each cell that has any data in it. If FUN returns a single atomic value for each cell (e.g., functions mean or var) and when simplify is TRUE, tapply returns a multi-way array containing the values. The array has the same number of dimensions as INDEX has components; the number of levels in a dimension is the number of levels(nlevels()) in the corresponding component of INDEX.

Note that contrary to S, simplify = TRUE always returns an array, possibly 1-dimensional.

If FUN does not return a single atomic value, tapply returns an array of mode list whose components are the values of the individual calls to FUN, i.e., the result is a list with a dim attribute. Note that optional arguments to FUN supplied by the ... argument are not divided into cells. It is therefore inappropriate for FUN to expect additional arguments with the same length as X.
References


See Also

the convenience functions by and aggregate (using tapply); apply, lapply with its versions sapply and mapply.

Examples

```
require(stats)
groups <- as.factor(rbinom(32, n = 5, p = .4))
tapply(groups, groups, length) # is almost the same as
table(groups)

## contingency table from data.frame : array with named dimnames
tapply(warpbreaks$breaks, warpbreaks[,1], sum)
tapply(warpbreaks$breaks, warpbreaks[, 3, drop = FALSE], sum)

n <- 17; fac <- factor(rep(1:3, len = n), levels = 1:5)
table(fac)
tapply(1:n, fac, sum)
tapply(1:n, fac, sum, simplify = FALSE)
tapply(1:n, fac, range)
tapply(1:n, fac, quantile)

## example of ... argument: find quarterly means
tapply(presidents, cycle(presidents), mean, na.rm = TRUE)

ind <- list(c(1, 2, 2), c("A", "A", "B"))
table(ind)
tapply(1:3, ind) #-> the split vector
tapply(1:3, ind, sum)
```

taskCallback

Add or remove a top-level task callback

Description

addTaskCallback registers an R function that is to be called each time a top-level task is completed.
removeTaskCallback un-registers a function that was registered earlier via addTaskCallback.

These provide low-level access to the internal/native mechanism for managing task-completion actions. One can use taskCallbackManager at the S-language level to manage S functions that are called at the completion of each task. This is easier and more direct.

Usage

```
addTaskCallback(f, data = NULL, name = character(0))
removeTaskCallback(id)
```
Arguments

- **f**: the function that is to be invoked each time a top-level task is successfully completed. This is called with 5 or 4 arguments depending on whether `data` is specified or not, respectively. The return value should be a logical value indicating whether to keep the callback in the list of active callbacks or discard it.
- **data**: if specified, this is the 5-th argument in the call to the callback function `f`.
- **id**: a string or an integer identifying the element in the internal callback list to be removed. Integer indices are 1-based, i.e., the first element is 1. The names of currently registered handlers is available using `getTaskCallbackNames` and is also returned in a call to `addTaskCallback`.
- **name**: character: names to be used.

Details

Top-level tasks are individual expressions rather than entire lines of input. Thus an input line of the form `expression1 ; expression2` will give rise to 2 top-level tasks.

A top-level task callback is called with the expression for the top-level task, the result of the top-level task, a logical value indicating whether it was successfully completed or not (always TRUE at present), and a logical value indicating whether the result was printed or not. If the `data` argument was specified in the call to `addTaskCallback`, that value is given as the fifth argument.

The callback function should return a logical value. If the value is FALSE, the callback is removed from the task list and will not be called again by this mechanism. If the function returns TRUE, it is kept in the list and will be called on the completion of the next top-level task.

Value

- `addTaskCallback` returns an integer value giving the position in the list of task callbacks that this new callback occupies. This is only the current position of the callback. It can be used to remove the entry as long as no other values are removed from earlier positions in the list first.
- `removeTaskCallback` returns a logical value indicating whether the specified element was removed. This can fail (i.e., return FALSE) if an incorrect name or index is given that does not correspond to the name or position of an element in the list.

Note

This is an experimental feature and the interface may be changed in the future.

There is also C-level access to top-level task callbacks to allow C routines rather than R functions to be used.

See Also

- `getTaskCallbackNames`
- `taskCallbackManager`
- [http://developer.r-project.org/TaskHandlers.pdf](http://developer.r-project.org/TaskHandlers.pdf)

Examples

```r
  times <- function(total = 3, str="Task a") {
    ctr <- 0

    function(expr, value, ok, visible) {
```
taskCallbackManager

Create an R-level task callback manager

Description

This provides an entirely S-language mechanism for managing callbacks or actions that are invoked at the conclusion of each top-level task. Essentially, we register a single R function from this manager with the underlying, native task-callback mechanism and this function handles invoking the other R callbacks under the control of the manager. The manager consists of a collection of functions that access shared variables to manage the list of user-level callbacks.

Usage

```r
taskCallbackManager(handlers = list(), registered = FALSE, verbose = FALSE)
```

Arguments

- `handlers`: this can be a list of callbacks in which each element is a list with an element named "f" which is a callback function, and an optional element named "data" which is the 5-th argument to be supplied to the callback when it is invoked. Typically this argument is not specified, and one uses `add` to register callbacks after the manager is created.
registered  a logical value indicating whether the evaluate function has already been registered with the internal task callback mechanism. This is usually FALSE and the first time a callback is added via the add function, the evaluate function is automatically registered. One can control when the function is registered by specifying TRUE for this argument and calling addTaskCallback manually.

verbose  a logical value, which if TRUE, causes information to be printed to the console about certain activities this dispatch manager performs. This is useful for debugging callbacks and the handler itself.

Value
A list containing 6 functions:

add  register a callback with this manager, giving the function, an optional 5-th argument, an optional name by which the callback is stored in the list, and a register argument which controls whether the evaluate function is registered with the internal C-level dispatch mechanism if necessary.

remove  remove an element from the manager’s collection of callbacks, either by name or position/index.

evaluate  the ‘real’ callback function that is registered with the C-level dispatch mechanism and which invokes each of the R-level callbacks within this manager’s control.

suspend  a function to set the suspend state of the manager. If it is suspended, none of the callbacks will be invoked when a task is completed. One sets the state by specifying a logical value for the status argument.

register  a function to register the evaluate function with the internal C-level dispatch mechanism. This is done automatically by the add function, but can be called manually.

callbacks  returns the list of callbacks being maintained by this manager.

Note
This is an experimental feature and the interface may be changed in the future.

See Also
addTaskCallback, removeTaskCallback, getTaskCallbackNames\ http://developer.r-project.org/TaskHandlers.pdf

Examples

# create the manager
h <- taskCallbackManager()

# add a callback
h$add(function(expr, value, ok, visible) {
cat("In handler\n")
return(TRUE)
}, name = "simpleHandler")

# look at the internal callbacks.
getTaskCallbackNames()
# look at the R-level callbacks
names(h$callback())

getTaskCallbackNames()
removeTaskCallback("R-taskCallbackManager")

---

**taskCallbackNames**  
Query the names of the current internal top-level task callbacks

**Description**

This provides a way to get the names (or identifiers) for the currently registered task callbacks that are invoked at the conclusion of each top-level task. These identifiers can be used to remove a callback.

**Usage**

getTaskCallbackNames()

**Value**

A character vector giving the name for each of the registered callbacks which are invoked when a top-level task is completed successfully. Each name is the one used when registering the callbacks and returned as the in the call to `addTaskCallback`.

**Note**

One can use `taskCallbackManager` to manage user-level task callbacks, i.e., S-language functions, entirely within the S language and access the names more directly.

**See Also**

`addTaskCallback`, `removeTaskCallback`, `taskCallbackManager`  
[http://developer.r-project.org/TaskHandlers.pdf](http://developer.r-project.org/TaskHandlers.pdf)

**Examples**

```r
n <- addTaskCallback(function(expr, value, ok, visible) {
    cat("In handler\n")
    return(TRUE)
}, name = "simpleHandler")

getTaskCallbackNames()

    # now remove it by name
removeTaskCallback("simpleHandler")

h <- taskCallbackManager()
h$add(function(expr, value, ok, visible) {
    cat("In handler\n")
    return(TRUE)
}, name = "simpleHandler")

getTaskCallbackNames()
removeTaskCallback("R-taskCallbackManager")
```
tempfile

Create Names for Temporary Files

Description

tempfile returns a vector of character strings which can be used as names for temporary files.

Usage

tempfile(pattern = "file", tmpdir = tempdir())
tempdir()

Arguments

pattern a non-empty character vector giving the initial part of the name.
tmpdir a non-empty character vector giving the directory name

Details

If pattern has length greater than one then the result is of the same length giving a temporary file name for each component of pattern.

The names are very likely to be unique among calls to tempfile in an R session and across simultaneous R sessions. The filenames are guaranteed not to be currently in use.

tempdir() gives a subdirectory of the first found of the environment variables TMP, TEMP and R_USER (see Rconsole) which points to a writeable directory. If the path to the directory contains a space in any of the components, the path returned will use the shortnames version of the path.

Value

For tempfile a character vector giving the names of possible (temporary) files. Note that no files are generated by tempfile.

For tempdir, the path of the per-session temporary directory.

Both will use backslash as the path separator.

References


See Also

unlink for deleting files.

Examples

tempfile(c("ab", "a b c"))  # give file name with spaces in!
tempdir()  # working on all platforms with quite platform dependent result
Description
Input and output text connections.

Usage
textConnection(object, open = "r", local = FALSE)

Arguments
object character. A description of the connection. For an input this is an \text{R} character vector object, and for an output connection the name for the \text{R} character vector to receive the output.
open character. Either "r" (or equivalently ") for an input connection or "w" or "a" for an output connection.
local logical. Used only for output connections. If \text{TRUE}, output is assigned to a variable in the calling environment. Otherwise the global environment is used.

Details
An input text connection is opened and the character vector is copied at time the connection object is created, and \text{close} destroys the copy.

An output text connection is opened and creates an \text{R} character vector of the given name in the user’s workspace or in the calling environment, depending on the value of the \text{local} argument. This object will at all times hold the completed lines of output to the connection, and \text{isIncomplete} will indicate if there is an incomplete final line. Closing the connection will output the final line, complete or not. (A line is complete once it has been terminated by end-of-line, represented by "\n" in \text{R}.)

Opening a text connection with \text{mode} = "a" will attempt to append to an existing character vector with the given name in the user’s workspace or the calling environment. If none is found (even if an object exists of the right name but the wrong type) a new character vector will be created, with a warning.

You cannot \text{seek} on a text connection, and \text{seek} will always return zero as the position.

Value
A connection object of class "textConnection" which inherits from class "connection".

Note
As output text connections keep the character vector up to date line-by-line, they are relatively expensive to use, and it is often better to use an anonymous \text{file} () connection to collect output.

On platforms where \text{vsnprintf} does not return the needed length of output (e.g., Windows) there is a 100,000 character limit on the length of line for output connections: longer lines will be truncated with a warning.
References


See Also

`connections, showConnections, pushBack, capture.output`.

Examples

```r
zz <- textConnection(LETTERS)
readLines(zz, 2)
scan(zz, "", 4)
pushBack(c("aa", "bb"), zz)
scan(zz, "", 4)
close(zz)

zz <- textConnection("foo", "w")
writeLines(c("testit1", "testit2"), zz)
cat("testit3 ", file=zz)
isIncomplete(zz)
cat("testit4\n", file=zz)
isIncomplete(zz)
close(zz)
foo

## Not run:
# capture R output: use part of example from help(lm)
zz <- textConnection("foo", "w")
ctl <- c(4.17, 5.58, 5.18, 6.11, 4.5, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)
sink(zz)
anova(lm.D9 <- lm(weight ~ group))
cat("\nSummary of Residuals:\n\n")
summary(resid(lm.D9))
sink()
close(zz)
cat(foo, sep = "\n")
## End(Not run)
```

tilde  

**Tilde Operator**

Description

Tilde is used to separate the left- and right-hand sides in model formula.

Usage

`y ~ model`
Arguments

\$ y, \ model \quad \text{symbolic expressions.}

References


See Also

`formula`

toString

\textit{Convert an R Object to a Character String}

Description

This is a helper function for `format` to produce a single character string describing an R object.

Usage

\begin{verbatim}
toString(x, ...)
\end{verbatim}

## Default S3 method:
\begin{verbatim}
toString(x, width = NULL, ...)
\end{verbatim}

Arguments

\begin{itemize}
\item \textbf{\texttt{x}} \quad \text{The object to be converted.}
\item \textbf{\texttt{width}} \quad \text{Suggestion for the maximum field width. Values of \texttt{NULL} or 0 indicate no maximum. The minimum value accepted is 6 and smaller values are taken as 6.}
\item \textbf{\texttt{...}} \quad \text{Optional arguments passed to or from methods.}
\end{itemize}

Details

This is a generic function for which methods can be written: only the default method is described here. Most methods should honor the \texttt{width} argument to specify the maximum display width (as measured by \texttt{nchar(type = "width")} of the result.

The default method first converts \texttt{x} to character and then concatenates the elements separated by ", " if \texttt{width} is supplied and is not \texttt{NULL}, the default method returns the first \texttt{width} - 4 characters of the result with \ldots appended, if the full result would use more than \texttt{width} characters.

Value

A character vector of length 1 is returned.

Author(s)

Robert Gentleman
See Also

format

Examples

```r
x <- c("a", "b", "aaaaaaaaaaa")
toString(x)
toString(x, width=8)
```

trace

Interactive Tracing and Debugging of Calls to a Function or Method

Description

A call to `trace` allows you to insert debugging code (e.g., a call to `browser` or `recover`) at chosen places in any function. A call to `untrace` cancels the tracing. Specified methods can be traced the same way, without tracing all calls to the function. Trace code can be any R expression. Tracing can be temporarily turned on or off globally by calling `tracingState`.

Usage

```r
trace(what, tracer, exit, at, print, signature,
     where = topenv(parent.frame()), edit = FALSE)
untrace(what, signature = NULL, where = topenv(parent.frame()))
tracingState(on = NULL)
```

Arguments

- **what**: The name (quoted or not) of a function to be traced or untraced. More than one name can be given in the quoted form, and the same action will be applied to each one.
- **tracer**: Either a function or an unevaluated expression. The function will be called or the expression will be evaluated either at the beginning of the call, or before those steps in the call specified by the argument `at`. See the details section.
- **exit**: Either a function or an unevaluated expression. The function will be called or the expression will be evaluated on exiting the function. See the details section.
- **at**: optional numeric vector. If supplied, `tracer` will be called just before the corresponding step in the body of the function. See the details section.
- **print**: If TRUE (as per default), a descriptive line is printed before any trace expression is evaluated.
- **signature**: If this argument is supplied, it should be a signature for a method for function `what`. In this case, the method, and not the function itself, is traced.
- **edit**: For complicated tracing, such as tracing within a loop inside the function, you will need to insert the desired calls by editing the body of the function. If so, supply the `edit` argument either as TRUE, or as the name of the editor you want to use. Then `trace()` will call `edit` and use the version of the function after you edit it. See the details section for additional information.
where

where to look for the function to be traced; by default, the top-level environment
of the call to trace.

An important use of this argument is to trace a function when it is called from
a package with a namespace. The current namespace mechanism imports the
functions to be called (with the exception of functions in the base package).
The functions being called are not the same objects seen from the top-level (in
general, the imported packages may not even be attached). Therefore, you must
ensure that the correct versions are being traced. The way to do this is to set
argument where to a function in the namespace. The tracing computations
will then start looking in the environment of that function (which will be the
namespace of the corresponding package). (Yes, it’s subtle, but the semantics
here are central to how namespaces work in R.)

on

logical; a call to tracingState returns TRUE if tracing is globally turned on,
FALSE otherwise. An argument of one or the other of those values sets the state.
If the tracing state is FALSE, none of the trace actions will actually occur (used,
for example, by debugging functions to shut off tracing during debugging).

Details

The trace function operates by constructing a revised version of the function (or of the method, if
signature is supplied), and assigning the new object back where the original was found. If only
the what argument is given, a line of trace printing is produced for each call to the function (back
compatible with the earlier version of trace).

The object constructed by trace is from a class that extends "function" and which contains
the original, untraced version. A call to untrace re-assigns this version.

If the argument tracer or exit is the name of a function, the tracing expression will be a call
to that function, with no arguments. This is the easiest and most common case, with the functions
browser and recover the likeliest candidates; the former browses in the frame of the function
being traced, and the latter allows browsing in any of the currently active calls.

The tracer or exit argument can also be an unevaluated expression (such as returned by a call
to quote or substitute). This expression itself is inserted in the traced function, so it will
typically involve arguments or local objects in the traced function. An expression of this form is
useful if you only want to interact when certain conditions apply (and in this case you probably
want to supply print=FALSE in the call to trace also).

When the at argument is supplied, it should be a vector of integers referring to the substeps of the
body of the function (this only works if the body of the function is enclosed in { ....}). In this
case tracer is not called on entry, but instead just before evaluating each of the steps listed in at.
(Hint: you don’t want to try to count the steps in the printed version of a function; instead, look at
as.list(body(f)) to get the numbers associated with the steps in function f.)

An intrinsic limitation in the exit argument is that it won’t work if the function itself uses
on.exit, since the existing calls will override the one supplied by trace.

Tracing does not nest. Any call to trace replaces previously traced versions of that function or
method (except for edited versions as discussed below), and untrace always restores an untraced
version. (Allowing nested tracing has too many potentials for confusion and for accidentally leaving
traced versions behind.)

When the edit argument is used repeatedly with no call to untrace on the same function or
method in between, the previously edited version is retained. If you want to throw away all the
previous tracing and then edit, call untrace before the next call to trace. Editing may be
combined with automatic tracing; just supply the other arguments such as tracer, and the edit
argument as well. The edit=TRUE argument uses the default editor (see edit).
Tracing primitive functions (builtins and specials) from the base package works, but only by a special mechanism and not very informatively. Tracing a primitive causes the primitive to be replaced by a function with argument ... (only). You can get a bit of information out, but not much. A warning message is issued when trace is used on a primitive.

The practice of saving the traced version of the function back where the function came from means that tracing carries over from one session to another, if the traced function is saved in the session image. (In the next session, untrace will remove the tracing.) On the other hand, functions that were in a package, not in the global environment, are not saved in the image, so tracing expires with the session for such functions.

Tracing a method is basically just like tracing a function, with the exception that the traced version is stored by a call to setMethod rather than by direct assignment, and so is the untraced version after a call to untrace.

The version of trace described here is largely compatible with the version in S-Plus, although the two work by entirely different mechanisms. The S-Plus trace uses the session frame, with the result that tracing never carries over from one session to another (R does not have a session frame). Another relevant distinction has nothing directly to do with trace: The browser in S-Plus allows changes to be made to the frame being browsed, and the changes will persist after exiting the browser. The R browser allows changes, but they disappear when the browser exits. This may be relevant in that the S-Plus version allows you to experiment with code changes interactively, but the R version does not. (A future revision may include a “destructive” browser for R.)

**Value**

The traced function(s) name(s). The relevant consequence is the assignment that takes place.

**Note**

The version of function tracing that includes any of the arguments except for the function name requires the methods package (because it uses special classes of objects to store and restore versions of the traced functions).

If methods dispatch is not currently on, trace will load the methods namespace, but will not put the methods package on the search list.

**References**


**See Also**

browser and recover, the likeliest tracing functions; also, quote and substitute for constructing general expressions.

**Examples**

```r
## Very simple use
trace(sum)
hist(rnorm(100)) # shows about 3-4 calls to sum()
untrace(sum)

if(.isMethodsDispatchOn()) { # non-simple use needs 'methods' package
  f <- function(x, y) {
```
y <- pmax(y, .001)
x ^ y
}

## arrange to call the browser on entering and exiting
## function f
trace("f", browser, exit = browser)

## instead, conditionally assign some data, and then browse
## on exit, but only then. Don't bother me otherwise
trace("f", quote(if(any(y < 0)) yOrig <- y),
       exit = quote(if(exists("yOrig")) browser()),
       print = FALSE)

## trace a utility function, with recover so we
## can browse in the calling functions as well.
trace("as.matrix", recover)

## turn off the tracing
untrace(c("f", "as.matrix"))

## Not run:
## trace calls to the function lm() that come from the nlme package
## (The function nlme is in that package, and the package has a namespace,
## so the where= argument must be used to get the right version of lm)
trace(lm, exit = recover, where = nlme)
## End(Not run)

---

**traceback**

### Print Call Stacks

**Description**

By default, `traceback()` prints the call stack of the last uncaught error, i.e., the sequence of calls that lead to the error. This is useful when an error occurs with an unidentifiable error message. It can also be used to print arbitrary lists of deparsed calls.

**Usage**

`traceback(x = NULL)`

**Arguments**

- `x` NULL (default), or a list or pairlist of deparsed calls.
Details

The stack of the last uncaught error is stored as a list of deparsed calls in `.Traceback`, which `traceback()` prints in a user-friendly format.

Errors which are caught via `try` or `tryCatch` do not generate a traceback, so what is printed is the call sequence for the last uncaught error, and not necessarily the last error.

Value

`traceback()` returns nothing, but prints the deparsed call stack deepest call first. The calls may print on more that one line, and the first line is labelled by the frame number.

References


Examples

```r
foo <- function(x) { print(1); bar(2) }
bar <- function(x) { x + a.variable.which.does.not.exist }
## Not run:
foo(2) # gives a strange error
traceback()
## End(Not run)
## 2: bar(2)
## 1: foo(2)
bar
## Ah, this is the culprit ...
```

transform

*Transform an Object, for Example a Data Frame*

Description

`transform` is a generic function, which—at least currently—only does anything useful with data frames. `transform.default` converts its first argument to a data frame if possible and calls `transform.data.frame`.

Usage

```r
transform(x, ...)
```

Arguments

- `x` The object to be transformed
- `...` Further arguments of the form `tag=value`

Details

The `...` arguments to `transform.data.frame` are tagged vector expressions, which are evaluated in the data frame `x`. The tags are matched against `names(x)`, and for those that match, the value replace the corresponding variable in `x`, and the others are appended to `x`. 
Trig

Value

The modified value of \( x \).

Note

If some of the values are not vectors of the appropriate length, you deserve whatever you get!

Author(s)

Peter Dalgaard

See Also

subset, list, data.frame

Examples

\[
\text{transform(airquality, Ozone = \(-Ozone\)} \\
\text{transform(airquality, new = \(-Ozone, Temp = (Temp-32)/1.8\)}
\]

\[
\text{attach(airquality)} \\
\text{transform(Ozone, logOzone = log(Ozone)) } # \text{ marginally interesting ...} \\
\text{detach(airquality)}
\]

Description

These functions give the obvious trigonometric functions. They respectively compute the cosine, sine, tangent, arc-cosine, arc-sine, arc-tangent, and the two-argument arc-tangent.

Usage

\[
\text{cos}(x) \\
\text{sin}(x) \\
\text{tan}(x) \\
\text{acos}(x) \\
\text{asin}(x) \\
\text{atan}(x) \\
\text{atan2}(y, x)
\]

Arguments

\( x, y \) numeric or complex vector

Details

The arc-tangent of two arguments at\(\text{an2}(y, x)\) returns the angle between the x-axis and the vector from the origin to \((x, y)\), i.e., for positive arguments \text{atan2}(y, x) = \text{atan}(y/x).

Angles are in radians, not degrees (i.e., a right angle is \(\pi/2\)).

All except at\(\text{an2}\) are generic functions: methods can be defined for them individually or via the Math group generic.
Complex values

For the inverse trigonometric functions, branch cuts are defined as in Abramowitz and Stegun, figure 4.4, page 79. Continuity on the branch cuts is standard.

For \( \text{asin}() \) and \( \text{acos}() \), there are two cuts, both along the real axis: \((-\infty, -1]\) and \([1, \infty)\). Functions \( \text{asin}() \) and \( \text{acos}() \) are continuous from above on the interval \((-\infty, -1]\) and continuous from below on \([1, \infty)\).

For \( \text{atan}() \) there are two cuts, both along the pure imaginary axis: \((-\infty i, -1i]\) and \([1i, \infty i)\). It is continuous from the left on the interval \((-\infty i, -1i]\) and from the right on the interval \([1i, \infty i)\).

References


---

**try**

*Try an Expression Allowing Error Recovery*

**Description**

`try` is a wrapper to run an expression that might fail and allow the user's code to handle error-recovery.

**Usage**

```r
try(expr, silent = FALSE)
```

**Arguments**

- `expr` an R expression to try.
- `silent` logical: should the report of error messages be suppressed?

**Details**

`try` evaluates an expression and traps any errors that occur during the evaluation. `try` establishes a handler for errors that uses the default error handling protocol. It also establishes a `tryRestart` restart that can be used by `invokeRestart`.

**Value**

The value of the expression if `expr` is evaluated without error, but an invisible object of class "try-error" containing the error message if it fails. The normal error handling will print the same message unless `options("show.error.messages")` is false or the call includes `silent = TRUE`. 
See Also

options for setting error handlers and suppressing the printing of error messages; geterrmessage for retrieving the last error message. tryCatch provides another means of catching and handling errors.

Examples

```r
## this example will not work correctly in example(try), but
## it does work correctly if pasted in
options(show.error.messages = FALSE)
try(log("a"))
print(.Last.value)
options(show.error.messages = TRUE)

## alternatively,
print(try(log("a"), TRUE))

## run a simulation, keep only the results that worked.
set.seed(123)
x <- rnorm(50)
doit <- function(x)
{
  x <- sample(x, replace=TRUE)
  if(length(unique(x)) > 30) mean(x)
  else stop("too few unique points")
}
## alternative 1
res <- lapply(1:100, function(i) try(doit(x), TRUE))
## alternative 2
## Not run:
res <- vector("list", 100)
for(i in 1:100) res[[i]] <- try(doit(x), TRUE)
## End(Not run)
unlist(res[sapply(res, function(x) !inherits(x, "try-error"))])
```

type.convert

Type Conversion on Character Variables

Description

Convert a character vector to logical, integer, numeric, complex or factor as appropriate.

Usage

```r
type.convert(x, na.strings = "NA", as.is = FALSE, dec = ".")
```

Arguments

- `x` a character vector.
- `na.strings` a vector of strings which are to be interpreted as NA values. Blank fields are also considered to be missing values in logical, integer, numeric or complex vectors.
- `as.is` logical. See Details.
- `dec` the character to be assumed for decimal points.
**Details**

This is principally a helper function for `read.table`. Given a character vector, it attempts to convert it to logical, integer, numeric or complex, and failing that converts it to factor unless `as.is = TRUE`. The first type that can accept all the non-missing values is chosen.

Vectors which are entirely missing values are converted to logical, since `NA` is primarily logical.

**Value**

A vector of the selected class, or a factor.

**See Also**

`read.table`

---

### typeof

**The Type of an Object**

**Description**

`typeof` determines the (R internal) type or storage mode of any object

**Usage**

`typeof(x)`

**Arguments**

`x`  
any R object.

**Value**

A character string. The possible values are listed in the structure `TypeTable` in `src/main/util.c`. Current values are the vector types "logical", "integer", "double", "complex", "character", "raw" and "list", "NULL", "closure" (function), "special" and "builtin" (basic functions and operators), "environment", and others that are unlikely to be seen at user level ("symbol", "pairlist", "environment", "promise", "language", "char", ",", ",", "any", "expression", "externalptr", "bytecode" and "weakref").

**See Also**

`mode, storage.mode`

**Examples**

`typeof(2)`
`mode(2)`
Unique Elements

**Description**

`unique` returns a vector, data frame or array like `x` but with duplicate elements removed.

**Usage**

```r
unique(x, incomparables = FALSE, ...)  
## S3 method for class 'array':  
unique(x, incomparables = FALSE, MARGIN = 1, ...)
```

**Arguments**

- `x`: a vector or a data frame or an array or `NULL`.
- `incomparables`: a vector of values that cannot be compared. Currently, `FALSE` is the only possible value, meaning that all values can be compared.
- `...`: arguments for particular methods.
- `MARGIN`: the array margin to be held fixed: a single integer.

**Details**

This is a generic function with methods for vectors, data frames and arrays (including matrices).

The array method calculates for each element of the dimension specified by `MARGIN` if the remaining dimensions are identical to those for an earlier element (in row-major order). This would most commonly be used to find unique rows (the default) or columns (with `MARGIN = 2`).

**Value**

An object of the same type of `x`, but if an element is equal to one with a smaller index, it is removed. Dimensions of arrays are not dropped.

**Warning**

Using this for lists is potentially slow, especially if the elements are not atomic vectors (see `vector`) or differ only in their attributes. In the worst case it is $O(n^2)$.

**References**


**See Also**

`duplicated` which gives the indices of duplicated elements.
Examples

```r
unique(c(3:5, 11:8, 8 + 0:5))
length(unique(sample(100, 100, replace=TRUE)))
## approximately 100(1 - 1/e) = 63.21
unique(iris)
```

unlink  
**Delete Files and Directories**

Description

unlink deletes the file(s) or directories specified by `x`.

Usage

```r
unlink(x, recursive = FALSE)
```

Arguments

- `x`: a character vector with the names of the file(s) or directories to be deleted. Wildcards (normally `*` and `?`) are allowed.
- `recursive`: logical. Should directories be deleted recursively?

Details

If `recursive = FALSE` directories are not deleted, not even empty ones. `file.remove` can only remove files, but gives more detailed error information.

Value

0 for success, 1 for failure. Not deleting a non-existent file is not a failure.

Note

Prior to R version 1.2.0 the default on Unix was `recursive = TRUE`, and on Windows empty directories could be deleted.

References


See Also

`file.remove`.  

unlink
**Description**

Given a list structure `x`, `unlist` simplifies it to produce a vector which contains all the atomic components which occur in `x`.

**Usage**

```r
unlist(x, recursive = TRUE, use.names = TRUE)
```

**Arguments**

- `x` A list or vector.
- `recursive` logical. Should unlisting be applied to list components of `x`?
- `use.names` logical. Should names be preserved?

**Details**

`unlist` is generic: you can write methods to handle specific classes of objects, see `InternalMethods`.

If `recursive = FALSE`, the function will not recurse beyond the first level items in `x`.

`x` can be a vector, but then `unlist` does nothing useful, not even drop names.

By default, `unlist` tries to retain the naming information present in `x`. If `use.names = FALSE` all naming information is dropped.

Where possible the list elements are coerced to a common mode during the unlisting, and so the result often ends up as a character vector. Raw and logical vectors can be coerced to integer, integer to double, double to complex and any other atomic mode to character.

A list is a (generic) vector, and the simplified vector might still be a list (and might be unchanged). Non-vector elements of the list (for example language elements such as names, formulas and calls) are not coerced, and so a list containing one or more of these remains a list. (The effect of unlisting an `lm` fit is a list which has individual residuals as components.)

**Value**

A vector of an appropriate mode to hold the list components.

**References**


**See Also**

`c, as.list`. 
Examples

unlist(options())
unlist(options(), use.names=FALSE)

l.ex <- list(a = list(1:5, LETTERS[1:5]), b = "Z", c = NA)
unlist(l.ex, recursive = FALSE)
unlist(l.ex, recursive = TRUE)

l1 <- list(a="a", b=2, c=pi+2i)
unlist(l1) # a character vector
l2 <- list(a="a", b=as.name("b"), c=pi+2i)
unlist(l2) # remains a list

unname

Remove 'names' or 'dimnames'

Description

Remove the names or dimnames attribute of an R object.

Usage

unname(obj, force = FALSE)

Arguments

obj the R object which is wanted “nameless”.
force logical; if true, the dimnames are even removed from data.frames. This argument is currently experimental and hence might change!

Value

Object as obj but without names or dimnames.

Examples

## Answering a question on R-help (14 Oct 1999):
col3 <- 750+ 100* rt(1500, df = 3)
breaks <- factor(cut(col3,breaks=360+5*(0:155)))
z <- table(breaks)
z[1:5] # The names are larger than the data ...
barplot(unname(z), axes= FALSE)
UseMethod

Class Methods

Description

R possesses a simple generic function mechanism which can be used for an object-oriented style of programming. Method despatch takes place based on the class of the first argument to the generic function or on the object supplied as an argument to UseMethod or NextMethod.

Usage

UseMethod(generic, object)
NextMethod(generic = NULL, object = NULL, ...)

Arguments

generic a character string naming a function. Required for UseMethod.
object an object whose class will determine the method to be dispatched. Defaults to the first argument of the enclosing function.
...

further arguments to be passed to the method.

Details

An R "object" is a data object which has a class attribute. A class attribute is a character vector giving the names of the classes which the object "inherits" from. If the object does not have a class attribute, it has an implicit class. Matrices and arrays have class "matrix" or "array" followed by the class of the underlying vector. Most vectors have class the result of mode(x), expect that integer vectors have class c("integer", "numeric") and real vectors have class c("double", "numeric").

When a generic function fun is applied to an object with class attribute c("first", "second"), the system searches for a function called fun.first and, if it finds it, applied it to the object. If no such function is found a function called fun.second is tried. If no class name produces a suitable function, the function fun.default is used, if it exists, or an error results.

Function methods can be used to find out about the methods for a particular generic function or class.

Now for some obscure details that need to appear somewhere. These comments will be slightly different than those in Appendix A of the White S Book. UseMethod creates a “new” function call with arguments matched as they came in to the generic. Any local variables defined before the call to UseMethod are retained (unlike S). Any statements after the call to UseMethod will not be evaluated as UseMethod does not return. UseMethod can be called with more than two arguments; a warning will be given and additional arguments ignored. (They are not completely ignored in S.) If it is called with just one argument, the class of the first argument of the enclosing function is used as object: unlike S this is the first actual argument passed and not the current value of the object of that name.

NextMethod invokes the next method (determined by the class). It does this by creating a special call frame for that method. The arguments will be the same in number, order and name as those to the current method but their values will be promises to evaluate their name in the current method and environment. Any arguments matched to ... are handled specially. They are passed on as the
promise that was supplied as an argument to the current environment. (S does this differently!) If
they have been evaluated in the current (or a previous environment) they remain evaluated.

NextMethod should not be called except in methods called by UseMethod. In particular it will
not work inside anonymous calling functions (eg get("print.ts")(AirPassengers)).

Name spaces can register methods for generic functions. To support this, UseMethod and
NextMethod search for methods in two places: first in the environment in which the generic
function is called, and then in the registration data base for the environment in which the generic
is defined (typically a name space). So methods for a generic function need to either be available
in the environment of the call to the generic, or they must be registered. It does not matter whether
they are visible in the environment in which the generic is defined.

Warning

Prior to R 2.1.0 UseMethod accepted a call with no arguments and tried to deduce the generic from
the context. This was undocumented on the help page. It is allowed but ‘strongly discouraged’ in
S-PLUS, and is no longer allowed in R.

Note

This scheme is called S3 (S version 3). For new projects, it is recommended to use the more flexible
and robust S4 scheme provided in the methods package.

The function .isMethodsDispatchOn() returns TRUE if the S4 method dispatch has been
turned on in the evaluator. It is meant for R internal use only.

References

Chambers, J. M. (1992) Classes and methods: object-oriented programming in S. Appendix A of

See Also

methods, class, getS3method

---

**UserHooks**

*Functions to Get and Set Hooks for Load, Attach, Detach and Unload*

**Description**

These functions allow users to set actions to be taken before packages are attached/detached and
namespaces are (un)loaded.

**Usage**

getHook(hookName)

setHook(hookName, value, action = c("append", "prepend", "replace"))

packageEvent(pkgname,
               event = c("onLoad", "attach", "detach", "onUnload"))
Arguments

- **hookName**: character string: the hook name
- **pkgname**: character string: the package/namespace name. If versioned install has been used, pkgname should be the unversioned name of the package (but any version information will be stripped).
- **event**: character string: an event for the package
- **value**: A function, or for action="replace", NULL.
- **action**: The action to be taken. The names can be abbreviated.

Details

setHook provides a general mechanism for users to register hooks, a list of functions to be called from system (or user) functions. The initial set of hooks is associated with events on packages/namespaces: these hooks are named via calls to packageEvent.

To remove a hook completely, call setHook(hookName, NULL, "replace").

When an R package is attached by library, it can call initialization code via a function .First.lib, and when it is detach-ed it can tidy up via a function .Last.lib. Users can add their own initialization code via the hooks provided by these functions, functions which will be called as funname(pkgname, pkgpath) inside a try call. (The attach hook is called after .First.lib and the detach hook before .Last.lib.)

If a package has a namespace, there are two further actions, when the namespace is loaded (before being attached and after .onLoad is called ) and when it is unloaded (after being detached and before .onUnload). Note that code in these hooks is run without the package being on the search path, so objects in the package need to be referred to using the double colon operator as in the example. (Unlike .onLoad, the user hook is run after the name space has been sealed.)

Hooks are normally run in the order shown by getHook, but the "detach" and "onUnload" hooks are run in reverse order so the default for package events is to add hooks 'inside' existing ones.

Note that when an R session is finished, packages are not detached and namespaces are not unloaded, so the corresponding hooks will not be run.

The hooks are stored in the environment .userHooksEnv in the base package, with 'mangled' names.

Value

For getHook function, a list of functions (possible empty). For setHook function, no return value. For packageEvent, the derived hook name (a character string).

See Also

library, detach, loadNamespace.

Other hooks may be added later: plot.new and persp already have them.

Examples

setHook(packageEvent("grDevices", "onLoad"),
        function(...) grDevices::ps.options(horizontal=FALSE))
utf8Conversion  

**Convert to or from UTF-8-encoded Character Vectors**

**Description**

Conversion of UTF-8 encoded character vectors to and from integer vectors.

**Usage**

```r
utf8ToInt(x)
intToUtf8(x, multiple = FALSE)
```

**Arguments**

- `x`: object to be converted.
- `multiple`: logical: should the conversion be to a single character string or multiple individual characters?

**Details**

These will work in any locale, including on machines that do not otherwise support multi-byte character sets.

**Value**

- `utf8ToInt` converts a length-one character string encoded in UTF-8 to an integer vector of (numeric) UTF-8 code points.
- `intToUtf8` converts a vector of (numeric) UTF-8 code points either to a single character string or a character vector of single characters. (Note that a single character string could contain embedded nuls.)

---

**vector**  

**Vectors**

**Description**

`vector` produces a vector of the given length and mode.

- `as.vector`, a generic, attempts to coerce its argument into a vector of mode `mode` (the default is to coerce to whichever mode is most convenient). The attributes of `x` are removed.
- `is.vector` returns `TRUE` if `x` is a vector (of mode logical, integer, real, complex, character, raw or list if not specified) and `FALSE` otherwise.

**Usage**

```r
vector(mode = "logical", length = 0)
as.vector(x, mode = "any")
is.vector(x, mode = "any")
```
vector

Arguments

mode
A character string giving an atomic mode or "list", or (not for vector) "any".

length
A non-negative integer specifying the desired length.

x
An object.

Details

The atomic modes are "logical", "integer", "numeric", "complex", "character" and "raw".

is.vector returns FALSE if x has any attributes except names. (This is incompatible with S.) On the other hand, as.vector removes all attributes including names.

Note that factors are not vectors; is.vector returns FALSE and as.vector converts to character mode.

Value

For vector, a vector of the given length and mode. Logical vector elements are initialized to FALSE, numeric vector elements to 0, character vector elements to "", raw vector elements to nul bytes and list elements to NULL.

Note

as.vector and is.vector are quite distinct from the meaning of the formal class "vector" in the methods package, and hence as(x, "vector") and is(x, "vector").

References


See Also
c, is.numeric, is.list, etc.

Examples

df <- data.frame(x=1:3, y=5:7)
## Not run:
## Error:
## "as.vector(data.frame(x=1:3, y=5:7), mode="numeric")"
## End(Not run)

x <- c(a = 1, b = 2)
is.vector(x)
as.vector(x)
all.equal(x, as.vector(x)) ## FALSE

###-- All the following are TRUE:
is.list(df)
! is.vector(df)
! is.vector(df, mode="list")
Description

Generates a warning message that corresponds to its argument(s) and (optionally) the expression or function from which it was called.

Usage

warning(..., call. = TRUE, immediate. = FALSE, domain = NULL)
suppressWarnings(expr)

Arguments

... character vectors (which are pasted together with no separator), a condition object, or NULL.
call. logical, indicating if the call should become part of the warning message.
immediate. logical, indicating if the call should be output immediately, even if getOption(warn) = 0.
expr expression to evaluate.
domain see gettext. If NA, messages will not be translated.

Details

The result depends on the value of options("warn") and on handlers established in the executing code.

warning signals a warning condition by (effectively) calling signalCondition. If there are no handlers or if all handlers return, then the value of warn is used to determine the appropriate action. If warn is negative warnings are ignored; if it is zero they are stored and printed after the top–level function has completed; if it is one they are printed as they occur and if it is 2 (or larger) warnings are turned into errors. Calling warning(immediate. = TRUE) turns warn = 0 into warn = 1 for this call only.

If warn is zero (the default), a top-level variable last.warning is created. It contains the warnings which can be printed via a call to warnings.

Warnings will be truncated to getOption("warning.length") characters, default 1000.

While the warning is being processed, a muffleWarning restart is available. If this restart is invoked with invokeRestart, then warning returns immediately.

An attempt is made to coerce other types of inputs to warning to character vectors.

suppressWarnings evaluates its expression in a context that ignores all warnings.

References

See Also

\texttt{stop} for fatal errors, \texttt{message} for diagnostic messages, \texttt{warnings}, and \texttt{options} with argument \texttt{warn=}.\newline
\texttt{gettext} for the mechanisms for the automated translation of messages.

\section*{Examples}

\begin{verbatim}
testit <- function() warning("testit")
testit()  ## shows call
testit <- function() warning("problem in testit", call. = FALSE)
testit()  ## no call
suppressWarnings(warning("testit"))
\end{verbatim}

\section*{warnings}

\textit{Print Warning Messages}

\section*{Description}
\texttt{warnings} prints the top-level variable \texttt{last.warning} in a pleasing form.

\section*{Usage}
\texttt{warnings(...)}

\section*{Arguments}
\texttt{...} \hspace{1cm} arguments to be passed to \texttt{cat}.

\section*{Details}
See the description of \texttt{options("warn")} for the circumstances under which there is a \texttt{last.warning} variable and \texttt{warnings()} is used. In essence this is if \texttt{options(warn = 0)} and warning has been called at least once.

It is possible that \texttt{last.warning} refers to the last recorded warning (possibly even in a previous session with a saved workspace), and not to the last warning, for example if \texttt{options(warn)} has been changed or if a catastrophic error occurred.

\section*{References}

\section*{See Also}
\texttt{warning}. 
Examples

```r
## NB this example is intended to be pasted in, rather than run by example()
ow <- options("warn")
for(w in -1:1) {
  options(warn = w); cat("\n warn =",w,"\n")
  for(i in 1:3) { cat(i,"..\n"); m <- matrix(1:7, 3,4) }
}
warnings()
options(now) # reset
```

Description

Extract the weekday, month or quarter, or the Julian time (days since some origin). These are generic functions: the methods for the internal date-time classes are documented here.

Usage

```r
weekdays(x, abbreviate)
## S3 method for class 'POSIXt':
weekdays(x, abbreviate = FALSE)
## S3 method for class 'Date':
weekdays(x, abbreviate = FALSE)

months(x, abbreviate)
## S3 method for class 'POSIXt':
months(x, abbreviate = FALSE)
## S3 method for class 'Date':
months(x, abbreviate = FALSE)

quaters(x, abbreviate)
## S3 method for class 'POSIXt':
quaters(x, ...)  # S3 method for class 'Date':
quaters(x, ...)  # S3 method for class 'Date':

julian(x, ...)
## S3 method for class 'POSIXt':
julian(x, origin = as.POSIXct("1970-01-01", tz="GMT"), ...)
## S3 method for class 'Date':
julian(x, origin = as.Date("1970-01-01"), ...)
```

Arguments

- `x` an object inheriting from class "POSIXt" or "Date".
- `abbreviate` logical. Should the names be abbreviated?
- `origin` an length-one object inheriting from class "POSIXt" or "Date".
- `...` arguments for other methods.
which

Value

weekdays and months return a character vector of names in the locale in use.
quarters returns a character vector of "Q1" to "Q4".
julian returns the number of days (possibly fractional) since the origin, with the origin as a "origin" attribute.

Note

Other components such as the day of the month or the year are very easy to compute: just use \texttt{as.POSIXlt} and extract the relevant component.

See Also

\texttt{DateTimeClasses,Date}

Examples

\begin{verbatim}
weekdays(.leap.seconds)
months(.leap.seconds)
quarters(.leap.seconds)
\end{verbatim}

\begin{verbatim}
which                  Which indices are TRUE?
\end{verbatim}

Description

Give the TRUE indices of a logical object, allowing for array indices.

Usage

\texttt{which(x, arr.ind = FALSE)}

Arguments

\begin{itemize}
  \item \texttt{x} a logical vector or array. \texttt{NA}s are allowed and omitted (treated as if FALSE).
  \item \texttt{arr.ind} logical; should \texttt{array indices} be returned when \texttt{x} is an array?
\end{itemize}

Value

If \texttt{arr.ind == \texttt{FALSE}} (the default), an integer vector with \texttt{length} equal to \texttt{sum(x)}, i.e., to the number of TRUEs in \texttt{x}; Basically, the result is \{(1:\texttt{length(x)})[x] \}

If \texttt{arr.ind == \texttt{TRUE}} and \texttt{x} is an \texttt{array} (has a \texttt{dim} attribute), the result is a matrix who’s rows each are the indices of one element of \texttt{x}; see Examples below.

Author(s)

Werner Stahel and Peter Holzer (holzer@stat.math.ethz.ch), for the array case.
See Also

Logic, \texttt{which.min} for the index of the minimum or maximum, and \texttt{match} for the first index of an element in a vector, i.e., for a scalar \(a\), \texttt{match}(a, x) is equivalent to \texttt{min}(which(x == a)) but much more efficient.

Examples

\begin{verbatim}
which(LETTERS == "R")
which(ll <- c(TRUE, FALSE, TRUE, NA, FALSE, FALSE, TRUE)); #> 1 3 7
names(ll) <- letters[seq(ll)]
which(ll)
which((1:12)%%2 == 0) # which are even?
which(1:10 > 3, arr.ind=TRUE)

( m <- matrix(1:12,3,4) )
which(m %% 3 == 0)
which(m %% 3 == 0, arr.ind=TRUE)
rownames(m) <- paste("Case",1:3, sep="_")
which(m %% 5 == 0, arr.ind=TRUE)

dim(m) <- c(2,2,3); m
which(m %% 3 == 0, arr.ind=FALSE)
which(m %% 3 == 0, arr.ind=TRUE)

vm <- c(m)
dim(vm) <- length(vm) #-- funny thing with length(dim(...)) == 1
which(vm %% 3 == 0, arr.ind=TRUE)
\end{verbatim}

\texttt{which.min} \hspace{1cm} \texttt{Where is the Min() or Max() ?}

Description

Determines the location, i.e., index of the (first) minimum or maximum of a numeric vector.

Usage

\begin{verbatim}
which.min(x)
which.max(x)
\end{verbatim}

Arguments

\(x\) numeric vector, whose \texttt{min} or \texttt{max} is searched (NAs are allowed).

Value

an \texttt{integer} of length 1 or 0 (iff \(x\) has no non-NAs), giving the index of the first minimum or maximum respectively of \(x\).

If this extremum is unique (or empty), the result is the same (but more efficient) as \texttt{which}(x == min(x)) or \texttt{which}(x == max(x)) respectively.
Author(s)
Martin Maechler

See Also
which, max.col, max, etc.
which.is.max in package nnet differs in breaking ties at random (and having a “fuzz” in the definition of ties).

Examples

```r
x <- c(1:4, 0:5, 11)
which.min(x)
which.max(x)

## it *does* work with NA's present:
presidents[1:30]
range(presidents, na.rm = TRUE)
which.min(presidents)# 28
which.max(presidents)# 2
```

with

Evaluate an Expression in a Data Environment

Description
Evaluate an R expression in an environment constructed from data.

Usage

```r
with(data, expr, ...)
```

Arguments
data
data to use for constructing an environment. For the default method this may be an environment, a list, a data frame, or an integer as in sys.call.
expr
expression to evaluate.
...
arguments to be passed to future methods.

Details

with is a generic function that evaluates expr in a local environment constructed from data. The environment has the caller’s environment as its parent. This is useful for simplifying calls to modeling functions.

Note that assignments within expr take place in the constructed environment and not in the user’s workspace.

See Also
evalq, attach.
Examples

```r
require(stats); require(graphics)

# examples from glm:
## Not run:
library(MASS)
with(anorexia, {
  anorex.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
                  family = gaussian)
  summary(anorex.1)
})
## End(Not run)

with(data.frame(u = c(5,10,15,20,30,40,60,80,100),
                 lot1 = c(118,58,42,35,27,25,21,19,18),
                 lot2 = c(69,35,26,21,18,16,13,12,12)),
  list(summary(glm(lot1 ~ log(u), family = Gamma)),
       summary(glm(lot2 ~ log(u), family = Gamma))))

# example from boxplot:
with(ToothGrowth, {
  boxplot(len ~ dose, boxwex = 0.25, at = 1:3 - 0.2,
           subset = (supp == "VC"), col = "yellow",
           main = "Guinea Pigs' Tooth Growth",
           xlab = "Vitamin C dose mg",
           ylab = "tooth length", ylim = c(0,35))
  boxplot(len ~ dose, add = TRUE, boxwex = 0.25, at = 1:3 + 0.2,
           subset = supp == "OJ", col = "orange")
  legend(2, 9, c("Ascorbic acid", "Orange juice"),
         fill = c("yellow", "orange"))
})

# alternate form that avoids subset argument:
with(subset(ToothGrowth, supp == "VC"),
  boxplot(len ~ dose, boxwex = 0.25, at = 1:3 - 0.2,
           col = "yellow", main = "Guinea Pigs' Tooth Growth",
           xlab = "Vitamin C dose mg",
           ylab = "tooth length", ylim = c(0,35))
with(subset(ToothGrowth, supp == "OJ"),
  boxplot(len ~ dose, add = TRUE, boxwex = 0.25, at = 1:3 + 0.2,
           col = "orange")
legend(2, 9, c("Ascorbic acid", "Orange juice"),
       fill = c("yellow", "orange"))
```

write

Write Data to a File

Description

The data (usually a matrix) \( x \) are written to file \( file \). If \( x \) is a two-dimensional matrix you need to transpose it to get the columns in \( file \) the same as those in the internal representation.

Usage

```r
write(x, file = "data",
```
write.table

ncolumns = if(is.character(x)) 1 else 5,
append = FALSE, sep = " ")

Arguments

x the data to be written out.
file A connection, or a character string naming the file to write to. If " ", print to the
standard output connection.
ncolumns the number of columns to write the data in.
append if TRUE the data x is appended to file file.
sep a string used to separate columns. Using sep = " t" gives tab delimited output; default is " ".

References

Brooks/Cole.

See Also

save for writing any \texttt{R} objects, \texttt{write.table} for data frames, and \texttt{scan} for reading data.

Examples

# create a 2 by 5 matrix
x <- matrix(1:10,ncol=5)

# the file data contains x, two rows, five cols
# 1 3 5 6 9 will form the first row
write(t(x))

# Writing to the "console" 'tab-delimited'
# two rows, five cols but the first row is 1 2 3 4 5
write(x, "", sep = "\t")
unlink("data") # tidy up

write.table Data Output

Description

\texttt{write.table} prints its required argument \( x \) (after converting it to a data frame if it is not one
nor a matrix) to a file or connection.

Usage

\texttt{write.table}(x, file = "", append = FALSE, quote = TRUE, sep = " ",
eol = "\n", na = "NA", dec = ".", row.names = TRUE,
col.names = TRUE, qmethod = c("escape", "double"))

\texttt{write.csv}(...)
\texttt{write.csv2}(...)
Arguments

- **x**: the object to be written, preferably a matrix or data frame. If not, it is attempted to coerce x to a data frame.
- **file**: either a character string naming a file or a connection. """" indicates output to the console.
- **append**: logical. If TRUE, the output is appended to the file. If FALSE, any existing file of the name is destroyed.
- **quote**: a logical value or a numeric vector. If TRUE, any character or factor columns will be surrounded by double quotes. If a numeric vector, its elements are taken as the indices of the columns to quote. In both cases, row and column names are quoted if they are written. If FALSE, nothing is quoted.
- **sep**: the field separator string. Values within each row of x are separated by this string.
- **eol**: the character(s) to print at the end of each line (row).
- **na**: the string to use for missing values in the data.
- **dec**: the string to use for decimal points in numeric or complex columns: must be a single character.
- **row.names**: either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written.
- **col.names**: either a logical value indicating whether the column names of x are to be written along with x, or a character vector of column names to be written. See the section on ‘CSV files’ for the meaning of col.names=NA.
- **qmethod**: a character string specifying how to deal with embedded double quote characters when quoting strings. Must be one of "escape" (default), in which case the quote character is escaped in C style by a backslash, or "double", in which case it is doubled. You can specify just the initial letter.
- **...**: arguments to write.table: col.names, sep, dec and qmethod cannot be altered.

Details

If the table has no columns the rownames will be written only if row.names=TRUE, and vice versa.

Real and complex numbers are written to the maximal possible precision.

If a data frame has matrix-like columns these will be converted to multiple columns in the result (via as.matrix) and so a character col.names or a numeric quote should refer to the columns in the result, not the input. Such matrix-like columns are unquoted by default.

Any columns in a data frame which are lists or have a class (e.g. dates) will be converted by the appropriate as.character method: such columns are unquoted by default. On the other hand, any class information for a matrix is discarded.

The dec argument only applies to columns that are not subject to conversion to character because they have a class or are part of a matrix-like column, in particular to columns protected by I(). Use options("OutDec") to control such conversions.
CSV files

By default there is no column name for a column of row names. If `col.names = NA` and `row.names = TRUE` a blank column name is added, which is the convention for CSV files to be read by spreadsheets.

`write.csv` and `write.csv2` provide convenience wrappers for writing CSV files. They set `sep`, `dec` and `qmethod`, and `col.names` to `NA` if `row.names = TRUE` and `TRUE` otherwise.

`write.csv` uses "." for the decimal point and a comma for the separator.

`write.csv2` uses a comma for the decimal point and a semicolon for the separator, the Excel convention for CSV files in some Western European locales.

These wrappers are deliberately inflexible: they are designed to ensure that the correct conventions are used to write a valid file. Attempts to change `col.names`, `sep`, `dec` or `qmethod` are ignored, with a warning.

Note

`write.table` can be slow for data frames with large numbers (hundreds or more) of columns: this is inevitable as each column could be of a different class and so must be handled separately. If they are all of the same class, consider using a matrix instead.

See Also

The 'R Data Import/Export' manual.

`read.table`, `write`, `write.matrix` in package MASS.

Examples

```r
## Not run:
## To write a CSV file for input to Excel one might use
x <- data.frame(a = I("a \" quote"), b = pi)
write.table(x, file = "foo.csv", sep = ",", col.names = NA,
            qmethod = "double")
## and to read this file back into R one needs
read.table("foo.csv", header = TRUE, sep = ",", row.names = 1)
## NB: you do need to specify a separator if qmethod = "double".

### Alternatively
write.csv(x, file = "foo.csv")
read.csv("foo.csv")
## or without row names
write.csv(x, file = "foo.csv", row.names = FALSE)
read.csv("foo.csv")
## End(Not run)
```

writeLines

Write Lines to a Connection

Description

Write text lines to a connection.
Usage

writeLines(text, con = stdout(), sep = "\n")

Arguments

text  A character vector
con   A connection object or a character string.
sep   character. A string to be written to the connection after each line of text.

Details

If the con is a character string, the functions call file to obtain an file connection which is opened for the duration of the function call.

If the connection is open it is written from its current position. If it is not open, it is opened for the duration of the call and then closed again.

Normally writeLines is used with a text connection, and the default separator is converted to the normal separator for that platform (LF on Unix/Linux, CRLF on Windows, CR on Classic MacOS). For more control, open a binary connection and specify the precise value you want written to the file in sep. For even more control, use writeChar on a binary connection.

See Also

collections, writeChar, writeBin, readLines, cat

zip.file.extract  Extract File from a Zip Archive

Description

This will extract the file named file from the zip archive, if possible, and write it in a temporary location.

Usage

zip.file.extract(file, zipfile = "R.zip")

Arguments

file  A file name.
zipfile  The file name of a zip archive, including the ".zip" extension if required.

Details

The method used is selected by options(unzip=). All platforms support an "internal" unzip: this is the default under Windows and the fall-back under Unix if no unzip program was found during configuration and R_UNZIPCMD is not set.

The file will be extracted if it is in the archive and any required unzip utility is available. It will probably be extracted to the directory given by tempdir, overwriting an existing file of that name.
The name of the original or extracted file. Success is indicated by returning a different name.

The "internal" method is very simple, and will not set file dates.

---

### Description

.packages returns information about package availability.

### Usage

.packages(all.available = FALSE, lib.loc = NULL)

### Arguments

- all.available
  - logical; if TRUE return a character vector of all available packages in lib.loc.
- lib.loc
  - a character vector describing the location of R library trees to search through, or NULL. The default value of NULL corresponds to all libraries currently known.

### Details

.packages() returns the “base names” of the currently attached packages invisibly whereas .packages(all.available = TRUE) gives (visibly) all packages available in the library location path lib.loc. For a package to be regarded as being available it must have a ‘DESCRIPTION’ file containing a valid version field.

### Value

A character vector of package “base names”, invisible unless all.available = TRUE.

### Author(s)

R core; Guido Masarotto for the all.available=TRUE part of .packages.

### See Also

library, .libPaths.

### Examples

```
.packages()           # maybe just "base"
.packages(all = TRUE) # return all available as character vector
require(splines)      # the same
.(.packages())        # "splines", too
detach("package:splines")
```
Description

Miscellaneous internal/programming utilities.

Usage

.standard_regexps()

Details

.standard_regexps returns a list of “standard” regexps, including elements named valid_package_name and valid_package_version with the obvious meanings. The regexps are not anchored.
Chapter 2

The datasets package

\begin{verbatim}
ability.cov  Ability and Intelligence Tests
\end{verbatim}

Description

Six tests were given to 112 individuals. The covariance matrix is given in this object.

Usage

ability.cov

Details

The tests are described as

- general: a non-verbal measure of general intelligence using Cattell’s culture-fair test.
- picture: a picture-completion test
- blocks: block design
- maze: mazes
- reading: reading comprehension
- vocab: vocabulary

Bartholomew gives both covariance and correlation matrices, but these are inconsistent. Neither are in the original paper.

Source


References

Examples

```r
require(stats)
(ability.FA <- factanal(factors = 1, covmat=ability.cov))
update(ability.FA, factors=2)
update(ability.FA, factors=2, rotation="promax")
```

---

**airmiles**  
*Passenger Miles on Commercial US Airlines, 1937–1960*

**Description**

The revenue passenger miles flown by commercial airlines in the United States for each year from 1937 to 1960.

**Usage**

```r
airmiles
```

**Format**


**Source**

F.A.A. Statistical Handbook of Aviation.

**References**


**Examples**

```r
plot(airmiles, main = "airmiles data",
     xlab = "Passenger-miles flown by U.S. commercial airlines", col = 4)
```

---

**AirPassengers**  
*Monthly Airline Passenger Numbers 1949-1960*

**Description**

The classic Box & Jenkins airline data. Monthly totals of international airline passengers, 1949 to 1960.

**Usage**

```r
AirPassengers
```

**Format**

A monthly time series, in thousands.
Source


Examples

```r
## Not run:
## These are quite slow and so not run by example(AirPassengers)

## The classic 'airline model', by full ML
(fit <- arima(log10(AirPassengers), c(0, 1, 1),
             seasonal = list(order=c(0, 1,1), period=12)))
update(fit, method = "CSS")
update(fit, x=window(log10(AirPassengers), start = 1954))
tl <- pred$pred - 1.96 * pred$se
tu <- pred$pred + 1.96 * pred$se
ts.plot(AirPassengers, 10^tl, 10^tu, log = "y", lty = c(1,2,2))

## full ML fit is the same if the series is reversed, CSS fit is not
ap0 <- rev(log10(AirPassengers))
attributes(ap0) <- attributes(AirPassengers)
arima(ap0, c(0, 1, 1), seasonal = list(order=c(0, 1,1), period=12),
      method = "CSS")

## Structural Time Series
ap <- log10(AirPassengers) - 2
(fit <- StructTS(ap, type= "BSM"))
par(mfrow=c(1,2))
plot(cbind(ap, fitted(fit)), plot.type = "single")
plot(cbind(ap, tsSmooth(fit)), plot.type = "single")
## End(Not run)
```

**airquality**

*New York Air Quality Measurements*

**Description**


**Usage**

`airquality`

**Format**

A data frame with 154 observations on 6 variables.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[,1]</td>
<td>Ozone</td>
<td>numeric</td>
<td>Ozone (ppb)</td>
</tr>
<tr>
<td>[,3]</td>
<td>Wind</td>
<td>numeric</td>
<td>Wind (mph)</td>
</tr>
<tr>
<td>[,4]</td>
<td>Temp</td>
<td>numeric</td>
<td>Temperature (degrees F)</td>
</tr>
<tr>
<td>[,5]</td>
<td>Month</td>
<td>numeric</td>
<td>Month (1–12)</td>
</tr>
</tbody>
</table>
Details

Daily readings of the following air quality values for May 1, 1973 (a Tuesday) to September 30, 1973.

- **Ozone**: Mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island
- **Solar.R**: Solar radiation in Langleys in the frequency band 4000–7700 Angstroms from 0800 to 1200 hours at Central Park
- **Wind**: Average wind speed in miles per hour at 0700 and 1000 hours at LaGuardia Airport
- **Temp**: Maximum daily temperature in degrees Fahrenheit at La Guardia Airport.

Source

The data were obtained from the New York State Department of Conservation (ozone data) and the National Weather Service (meteorological data).

References


Examples

```r
pairs(airquality, panel = panel.smooth, main = "airquality data")
```

anscombe

*Anscombe's Quartet of “Identical” Simple Linear Regressions*

Description

Four \( x-y \) datasets which have the same traditional statistical properties (mean, variance, correlation, regression line, etc.), yet are quite different.

Usage

```r
anscombe
```

Format

A data frame with 11 observations on 8 variables.

```r
x1 == x2 == x3  # the integers 4:14, specially arranged
x4  # values 8 and 19
y1, y2, y3, y4  # numbers in (3, 12.5) with mean 7.5 and sdev 2.03
```

Source


References

### Examples

```r
require(stats)
summary(anscombe)

##-- now some "magic" to do the 4 regressions in a loop:
ff <- y ~ x
for(i in 1:4) {
  ff[2:3] <- lapply(paste(c("y","x"), i, sep=""), as.name)
  ## or  ff[[2]] <- as.name(paste("y", i, sep=""))
  ## ff[[3]] <- as.name(paste("x", i, sep=""))
  assign(paste("lm.",i,"sep=""), lmi <- lm(ff, data= anscombe))
  print(anova(lmi))
}

## See how close they are (numerically!)
sapply(objects(pat="lm\.[1-4]$"), function(n) coef(get(n)))
lapply(objects(pat="lm\.[1-4]$"), function(n) summary(get(n))$coef)

## Now, do what you should have done in the first place: PLOTS
op <- par(mfrow=c(2,2), mar=.1+c(4,4,1,1), oma= c(0,0,2,0))
for(i in 1:4) {
  ff[2:3] <- lapply(paste(c("y","x"), i, sep=""), as.name)
  plot(ff, data =anscombe, col="red", pch=21, bg = "orange", cex = 1.2,
       xlim=c(3,19), ylim=c(3,13))
  abline(get(paste("lm.",i,"sep="")), col="blue")
}
mtext("Anscombe's 4 Regression data sets", outer = TRUE, cex=1.5)
par(op)
```

---

#### attenu

**The Joyner–Boore Attenuation Data**

### Description

This data gives peak accelerations measured at various observation stations for 23 earthquakes in California. The data have been used by various workers to estimate the attenuating affect of distance on ground acceleration.

### Usage

```r
attenu
```

### Format

A data frame with 182 observations on 5 variables.

- `event` numeric Event Number
- `mag` numeric Moment Magnitude
- `station` factor Station Number
- `dist` numeric Station-hypocenter distance (km)
- `accel` numeric Peak acceleration (g)
Source


References


Examples

```r
## check the data class of the variables
sapply(attenu, data.class)
summary(attenu)
pairs(attenu, main = "attenu data")
coplot(accel ~ dist | as.factor(event), data = attenu, show = FALSE)
coplot(log(accel) ~ log(dist) | as.factor(event),
    data = attenu, panel = panel.smooth, show.given = FALSE)
```

The Chatterjee–Price Attitude Data

Description

From a survey of the clerical employees of a large financial organization, the data are aggregated from the questionnaires of the approximately 35 employees for each of 30 (randomly selected) departments. The numbers give the percent proportion of favourable responses to seven questions in each department.

Usage

`attitude`

Format

A dataframe with 30 observations on 7 variables. The first column are the short names from the reference, the second one the variable names in the data frame:

<table>
<thead>
<tr>
<th>Y rating</th>
<th>numeric</th>
<th>Overall rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>X[1] complaints</td>
<td>numeric</td>
<td>Handling of employee complaints</td>
</tr>
<tr>
<td>X[2] privileges</td>
<td>numeric</td>
<td>Does not allow special privileges</td>
</tr>
</tbody>
</table>
austres

<table>
<thead>
<tr>
<th>X[3]</th>
<th>learning</th>
<th>numeric</th>
<th>Opportunity to learn</th>
</tr>
</thead>
<tbody>
<tr>
<td>X[4]</td>
<td>raises</td>
<td>numeric</td>
<td>Raises based on performance</td>
</tr>
<tr>
<td>X[5]</td>
<td>critical</td>
<td>numeric</td>
<td>Too critical</td>
</tr>
<tr>
<td>X[6]</td>
<td>advance</td>
<td>numeric</td>
<td>Advancement</td>
</tr>
</tbody>
</table>

Source


Examples

```r
require(stats)
pairs(attitude, main = "attitude data")
summary(attitude)
summary(fm1 <- lm(rating ~ ., data = attitude))
opar <- par(mfrow = c(2, 2), oma = c(0, 0, 1.1, 0),
           mar = c(4.1, 4.1, 2.1, 1.1))
plot(fm1)
summary(fm2 <- lm(rating ~ complaints, data = attitude))
plot(fm2)
par(opar)
```

---

austres  
Quarterly Time Series of the Number of Australian Residents

Description

Numbers (in thousands) of Australian residents measured quarterly from March 1971 to March 1994. The object is of class "ts".

Usage

```r
austres
```

Source


---

beavers  
Body Temperature Series of Two Beavers

Description

Reynolds (1994) describes a small part of a study of the long-term temperature dynamics of beaver *Castor canadensis* in north-central Wisconsin. Body temperature was measured by telemetry every 10 minutes for four females, but data from a one period of less than a day for each of two animals is used there.
Usage

beaver1
beaver2

Format

The beaver1 data frame has 114 rows and 4 columns on body temperature measurements at 10 minute intervals.

The beaver2 data frame has 100 rows and 4 columns on body temperature measurements at 10 minute intervals.

The variables are as follows:

day  Day of observation (in days since the beginning of 1990), December 12–13 (beaver1) and November 3–4 (beaver2).
time  Time of observation, in the form 0330 for 3:30am
temp  Measured body temperature in degrees Celsius.
activ  Indicator of activity outside the retreat.

Note

The observation at 22:20 is missing in beaver1.

Source


Examples

(yl <- range(beaver1$temp, beaver2$temp))

beaver.plot <- function(bdat, ...) {
  nam <- deparse(substitute(bdat))
  attach(bdat, name=nam) # identify it by the actual name.
  # Hours since start of day:
  hours <- time %/% 100 + 24*(day - day[1]) + (time %% 100)/60
  plot (hours, temp, type = "l", ...,
       main = paste(nam, "body temperature"))
  abline(h = 37.5, col = "gray", lty = 2)
  is.act <- activ == 1
  points(hours[is.act], temp[is.act], col = 2, cex = .8)
  detach()
}

op <- par(mfrow = c(2,1), mar = c(3,3,4,2), mgp = .9* 2:0)
beaver.plot(beaver1, ylim = yl)
beaver.plot(beaver2, ylim = yl)
par(op)
**BJsales**  
*Sales Data with Leading Indicator*

**Description**

The sales time series `BJsales` and leading indicator `BJsales.lead` each contain 150 observations. The objects are of class "ts".

**Usage**

```r
BJsales  
BJsales.lead
```

**Source**


**References**


---

**BOD**  
*Biochemical Oxygen Demand*

**Description**

The `BOD` data frame has 6 rows and 2 columns giving the biochemical oxygen demand versus time in an evaluation of water quality.

**Usage**

```r
BOD
```

**Format**

This data frame contains the following columns:

- **Time** A numeric vector giving the time of the measurement (days).
- **demand** A numeric vector giving the biochemical oxygen demand (mg/l).

**Source**


Examples

```r
require(stats)
# simplest form of fitting a first-order model to these data
fm1 <- nls(demand ~ A*(1-exp(-exp(lrc)*Time)), data = BOD,
          start = c(A = 20, lrc = log(.35)))
coef(fm1)
print(fm1)
# using the plinear algorithm
fm2 <- nls(demand ~ (1-exp(-exp(lrc)*Time)), data = BOD,
          start = c(lrc = log(.35)), algorithm = "plinear", trace = TRUE)
# using a self-starting model
fm3 <- nls(demand ~ SSasympOrig(Time, A, lrc), data = BOD)
summary(fm3)
```

Description

The data give the speed of cars and the distances taken to stop. Note that the data were recorded in the 1920s.

Usage
cars

Format

A data frame with 50 observations on 2 variables.

```
[,1] speed numeric Speed (mph)
[,2] dist numeric Stopping distance (ft)
```

Source


References


Examples

```r
require(stats)
plot(cars, xlab = "Speed (mph)", ylab = "Stopping distance (ft)",
     las = 1)
lines(lowess(cars$speed, cars$dist, f = 2/3, iter = 3), col = "red")
title(main = "cars data")
plot(cars, xlab = "Speed (mph)", ylab = "Stopping distance (ft)",
     log = "xy")
title(main = "cars data (logarithmic scales)")
lines(lowess(cars$speed, cars$dist, f = 2/3, iter = 3), col = "red")
summary(fm1 <- lm(log(dist) ~ log(speed), data = cars))
```
opar <- par(mfrow = c(2, 2), oma = c(0, 0, 1.1, 0),
    mar = c(4.1, 4.1, 2.1, 1.1))
plot(fm1)
par(opar)

## An example of polynomial regression
plot(cars, xlab = "Speed (mph)", ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))
d <- seq(0, 25, len = 200)
for(degree in 1:4) {
  fm <- lm(dist ~ poly(speed, degree), data = cars)
  assign(paste("cars", degree, sep="."), fm)
  lines(d, predict(fm, data.frame(speed=d)), col = degree)
}
anova(cars.1, cars.2, cars.3, cars.4)

---

### Description

The ChickWeight data frame has 578 rows and 4 columns from an experiment on the effect of diet on early growth of chicks.

### Usage

ChickWeight

### Format

This data frame contains the following columns:

- **weight** a numeric vector giving the body weight of the chick (gm).
- **Time** a numeric vector giving the number of days since birth when the measurement was made.
- **Chick** an ordered factor with levels `18 < ... < 48` giving a unique identifier for the chick. The ordering of the levels groups chicks on the same diet together and orders them according to their final weight (lightest to heaviest) within diet.
- **Diet** a factor with levels `1,...,4` indicating which experimental diet the chick received.

### Details

The body weights of the chicks were measured at birth and every second day thereafter until day 20. They were also measured on day 21. There were four groups on chicks on different protein diets.

### Source


**chickwts**

**Chicken Weights by Feed Type**

**Description**

An experiment was conducted to measure and compare the effectiveness of various feed supplements on the growth rate of chickens.

**Usage**

chickwts

**Format**

A data frame with 71 observations on 2 variables.

- **weight** a numeric variable giving the chick weight.
- **feed** a factor giving the feed type.

**Details**

Newly hatched chicks were randomly allocated into six groups, and each group was given a different feed supplement. Their weights in grams after six weeks are given along with feed types.

**Source**


**References**


**Examples**

```r
require(stats)
boxplot(weight ~ feed, data = chickwts, col = "lightgray",
        varwidth = TRUE, notch = TRUE, main = "chickwt data",
        ylab = "Weight at six weeks (gm)")
anova(fm1 <- lm(weight ~ feed, data = chickwts))
par(opar)
```
Description

The CO2 data frame has 84 rows and 5 columns of data from an experiment on the cold tolerance of the grass species *Echinochloa crus-galli*.

Usage

CO2

Format

This data frame contains the following columns:

- **Plant**: an ordered factor with levels Qn1 < Qn2 < Qn3 < ... < Mc1 giving a unique identifier for each plant.
- **Type**: a factor with levels Quebec Mississippi giving the origin of the plant
- **Treatment**: a factor with levels nonchilled chilled
- **conc**: a numeric vector of ambient carbon dioxide concentrations (mL/L).
- **uptake**: a numeric vector of carbon dioxide uptake rates (µmol/m² sec).

Details

The CO₂ uptake of six plants from Quebec and six plants from Mississippi was measured at several levels of ambient CO₂ concentration. Half the plants of each type were chilled overnight before the experiment was conducted.

Source


Examples

```r
require(stats)
coplot(uptake ~ conc | Plant, data = CO2, show = FALSE, type = "b")
## fit the data for the first plant
fml <- nls(uptake ~ SSasymp(conc, Asym, lrc, c0),
           data = CO2, subset = Plant == 'Qn1')
summary(fml)
## fit each plant separately
fmlist <- list()
for (pp in levels(CO2$Plant)) {
  fmlist[[pp]] <- nls(uptake ~ SSasymp(conc, Asym, lrc, c0),
                      data = CO2, subset = Plant == pp)
}
## check the coefficients by plant
sapply(fmlist, coef)
```
**co2**

*Mauna Loa Atmospheric CO2 Concentration*

**Description**

Atmospheric concentrations of CO\textsubscript{2} are expressed in parts per million (ppm) and reported in the preliminary 1997 SIO manometric mole fraction scale.

**Usage**

```r
c02
```

**Format**

A time series of 468 observations; monthly from 1959 to 1997.

**Details**

The values for February, March and April of 1964 were missing and have been obtained by interpolating linearly between the values for January and May of 1964.

**Source**

Keeling, C. D. and Whorf, T. P., Scripps Institution of Oceanography (SIO), University of California, La Jolla, California USA 92039-0220.


**References**


**Examples**

```r
plot(co2, ylab = expression("Atmospheric concentration of CO^[2]\)),
     las = 1)
title(main = "co2 data set")
```

---

**datasets-package**

*The R Datasets Package*

**Description**

Base R datasets

**Details**

This package contains a variety of datasets. For a complete list, use `library(help="datasets")`. 
discoveries

Author(s)

R Development Core Team and contributors worldwide
Maintainer: R Core Team (R-core@r-project.org)

discoveries

Yearly Numbers of Important Discoveries

Description
The numbers of “great” inventions and scientific discoveries in each year from 1860 to 1959.

Usage
discoveries

Format
A time series of 100 values.

Source

References

Examples
plot(discoveries, ylab = "Number of important discoveries", las = 1)
title(main = "discoveries data set")

DNase

Elisa assay of DNase

Description
The DNase data frame has 176 rows and 3 columns of data obtained during development of an ELISA assay for the recombinant protein DNase in rat serum.

Usage
DNase
**esoph**

**Smoking, Alcohol and (O)esophageal Cancer**

**Description**

Data from a case-control study of (o)esophageal cancer in Ile-et-Vilaine, France.

**Usage**

esoph

**Format**

A data frame with records for 88 age/alcohol/tobacco combinations.

<table>
<thead>
<tr>
<th></th>
<th>&quot;agegp&quot;</th>
<th>Age group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25–34</td>
<td>1 25–34 years</td>
</tr>
<tr>
<td></td>
<td>36–44</td>
<td>2 35–44</td>
</tr>
<tr>
<td></td>
<td>45–54</td>
<td>3 45–54</td>
</tr>
<tr>
<td></td>
<td>55–64</td>
<td>4 55–64</td>
</tr>
<tr>
<td></td>
<td>65–74</td>
<td>5 65–74</td>
</tr>
<tr>
<td></td>
<td>75+</td>
<td>6 75+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>&quot;alcgp&quot;</th>
<th>Alcohol consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0–39 gm/day</td>
<td>1 0–39 gm/day</td>
</tr>
</tbody>
</table>
Conversion Rates of Euro Currencies

Description
Conversion rates between the various Euro currencies.

Usage

euro
euro.cross
euro is a named vector of length 11, euro.cross a matrix of size 11 by 11, with dimnames.

Details

The data set euro contains the value of 1 Euro in all currencies participating in the European monetary union (Austrian Schilling ATS, Belgian Franc BEF, German Mark DEM, Spanish Peseta ESP, Finnish Markka FIM, French Franc FRF, Irish Punt IEP, Italian Lira ITL, Luxembourg Franc LUF, Dutch Guilder NLG and Portuguese Escudo PTE). These conversion rates were fixed by the European Union on December 31, 1998. To convert old prices to Euro prices, divide by the respective rate and round to 2 digits.

The data set euro.cross contains conversion rates between the various Euro currencies, i.e., the result of outer(1 / euro, euro).

Examples

cbind(euro)

## These relations hold:
euro == signif(euro,6)  # [6 digit precision in Euro's definition]
all(euro.cross == outer(1/euro, euro))

## Convert 20 Euro to Belgian Franc
20 * euro["BEF"]

## Convert 20 Austrian Schilling to Euro
20 / euro["ATS"]

## Convert 20 Spanish Pesetas to Italian Lira
20 * euro.cross["ESP", "ITL"]

require(graphics)
dotchart(euro,
   main = "euro data: 1 Euro in currency unit")
dotchart(1/euro,
   main = "euro data: 1 currency unit in Euros")
dotchart(log(euro, 10),
   main = "euro data: log10(1 Euro in currency unit)")

---

eurodist

Distances Between European Cities

Description

The data give the road distances (in km) between 21 cities in Europe. The data are taken from a table in “The Cambridge Encyclopaedia”.

Usage

eurodist

Format

A dist object based on 21 objects. (You must have the stats package loaded to have the methods for this kind of object available).
faithful

Source

---

**faithful**

*Old Faithful Geyser Data*

Description
Waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

Usage
faithful

Format
A data frame with 272 observations on 2 variables.

\[
\begin{array}{lll}
\text{[.1]} & \text{eruptions} & \text{numeric} & \text{Eruption time in mins} \\
\text{[.2]} & \text{waiting} & \text{numeric} & \text{Waiting time to next eruption} \\
\end{array}
\]

Details
A closer look at `faithful$eruptions` reveals that these are heavily rounded times originally in seconds, where multiples of 5 are more frequent than expected under non-human measurement. For a “better” version of the eruptions times, see the example below.

There are many versions of this dataset around: Azzalini and Bowman (1990) use a more complete version.

Source
W. Härdle.

References

See Also
geyer in package MASS for the Azzalini-Bowman version.

Examples
```r
f.tit <- "faithful data: Eruptions of Old Faithful"
ne60 <- round(e60 <- 60 * faithful$eruptions)
all.equal(e60, ne60)  # relative diff. ~ 1/10000
table(zapsmall(abs(e60 - ne60)))  # 0, 0.02 or 0.04
```
faithful$better.eruptions <- ne60 / 60
te <- table(ne60)
te[te >= 4] # (too) many multiples of 5 !
plot(names(te), te, type="h", main = f.tit, xlab = "Eruption time (sec)")

plot(faithful[, -3], main = f.tit,
     xlab = "Eruption time (min)",
     ylab = "Waiting time to next eruption (min)"
lines(lowess(faithful$eruptions, faithful$waiting, f = 2/3, iter = 3),
     col = "red")

---

**Determination of Formaldehyde**

**Description**

These data are from a chemical experiment to prepare a standard curve for the determination of formaldehyde by the addition of chromatropic acid and concentrated sulpuric acid and the reading of the resulting purple color on a spectophotometer.

**Usage**

Formaldehyde

**Format**

A data frame with 6 observations on 2 variables.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[,1]</td>
<td>carb</td>
</tr>
<tr>
<td>[,2]</td>
<td>optden</td>
</tr>
</tbody>
</table>

**Source**


**References**


**Examples**

```r
require(stats)
plot(optden ~ carb, data = Formaldehyde,
     xlab = "Carbohydrate (ml)", ylab = "Optical Density",
     main = "Formaldehyde data", col = 4, las = 1)
abline(fm1 <- lm(optden ~ carb, data = Formaldehyde))
summary(fm1)
par <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(fm1)
par(opar)
```
Freeny's Revenue Data

Description

Freeny's data on quarterly revenue and explanatory variables.

Usage

freeny
freeny.x
freeny.y

Format

There are three `freeny` data sets.

`freeny.y` is a time series with 39 observations on quarterly revenue from (1962,2Q) to (1971,4Q).

`freeny.x` is a matrix of explanatory variables. The columns are `freeny.y` lagged 1 quarter, price index, income level, and market potential.

Finally, `freeny` is a data frame with variables `y`, `lag.quarterly.revenue`, `price.index`, `income.level`, and `market.potential` obtained from the above two data objects.

Source


References


Examples

summary(freeny)
pairs(freeny, main = "freeny data")  # gives warning: freeny$y has class "ts"
summary(fml <- lm(y ~ ., data = freeny))
opar <- par(mfrow = c(2, 2), oma = c(0, 0, 1.1, 0),
       mar = c(4.1, 4.1, 2.1, 1.1))
plot(fml)
par(opar)
HairEyeColor

Hair and Eye Color of Statistics Students

Description

Distribution of hair and eye color and sex in 592 statistics students.

Usage

HairEyeColor

Format

A 3-dimensional array resulting from cross-tabulating 592 observations on 3 variables. The variables and their levels are as follows:

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hair</td>
<td>Black, Brown, Red, Blond</td>
</tr>
<tr>
<td>2</td>
<td>Eye</td>
<td>Brown, Blue, Hazel, Green</td>
</tr>
<tr>
<td>3</td>
<td>Sex</td>
<td>Male, Female</td>
</tr>
</tbody>
</table>

Details

This data set is useful for illustrating various techniques for the analysis of contingency tables, such as the standard chi-squared test or, more generally, log-linear modelling, and graphical methods such as mosaic plots, sieve diagrams or association plots.

References


See Also

chisq.test, loglin, mosaicplot

Examples

```r
require(graphics)
## Full mosaic
mosaicplot(HairEyeColor)
## Aggregate over sex:
x <- apply(HairEyeColor, c(1, 2), sum)
x
mosaicplot(x, main = "Relation between hair and eye color")
```
Harman Example 2.3

Description
A correlation matrix of eight physical measurements on 305 girls between ages seven and seventeen.

Usage
Harman23.cor

Source

Examples
require(stats)
(Harman23.FA <- factanal(factors = 1, covmat = Harman23.cor))
for(factors in 2:4) print(update(Harman23.FA, factors = factors))

Harman Example 7.4

Description
A correlation matrix of 24 psychological tests given to 145 seventh and eight-grade children in a Chicago suburb by Holzinger and Swineford.

Usage
Harman74.cor

Source

Examples
require(stats)
(Harman74.FA <- factanal(factors = 1, covmat = Harman74.cor))
for(factors in 2:5) print(update(Harman74.FA, factors = factors))
Harman74.FA <- factanal(factors = 5, covmat = Harman74.cor, rotation="promax")
print(Harman74.FA$loadings, sort = TRUE)
**Indometh**

**Pharmacokinetics of Indomethicin**

**Description**

The `Indometh` data frame has 66 rows and 3 columns of data on the pharmacokinetics of indomethicin.

**Usage**

`Indometh`

**Format**

This data frame contains the following columns:

- **Subject**: an ordered factor with containing the subject codes. The ordering is according to increasing maximum response.
- **time**: a numeric vector of times at which blood samples were drawn (hr).
- **conc**: a numeric vector of plasma concentrations of indomethicin (mcg/ml).

**Details**

Each of the six subjects were given an intravenous injection of indomethicin.

**Source**


**Examples**

```r
require(stats)
fm1 <- nls(conc ~ SSbiexp(time, A1, lrc1, A2, lrc2),
          data = Indometh, subset = Subject == 1)
summary(fm1)
```
Infertility after Spontaneous and Induced Abortion

Description
This is a matched case-control study dating from before the availability of conditional logistic regression.

Usage

Usage

Format

1. Education 0 = 0-5 years
   1 = 6-11 years
   2 = 12+ years
2. age age in years of case
3. parity count
4. number of prior induced abortions 0 = 0
   1 = 1
   2 = 2 or more
5. case status 1 = case
   0 = control
6. number of prior spontaneous abortions 0 = 0
   1 = 1
   2 = 2 or more
7. matched set number 1-83
8. stratum number 1-63

Note
One case with two prior spontaneous abortions and two prior induced abortions is omitted.

Source


Examples

```
require(stats)
modell <- glm(case ~ spontaneous+induced, data=infert,family=binomial())
summary(modell)
## adjusted for other potential confounders:
summary(model2 <- glm(case ~ age+parity+education+spontaneous+induced,
data=infert,family=binomial()))
## Really should be analysed by conditional logistic regression
## which is in the survival package
if(require(survival)){
```
```
model3 <- clogit(case~spontaneous+induced+strata(stratum),data=infert)
print(summary(model3))
detach()# survival (conflicts)
```

---

**InsectSprays**

**Effectiveness of Insect Sprays**

**Description**

The counts of insects in agricultural experimental units treated with different insecticides.

**Usage**

InsectSprays

**Format**

A data frame with 72 observations on 2 variables.

```
[,1] count numeric Insect count
[,2] spray factor The type of spray
```

**Source**


**References**


**Examples**

```r
require(stats)
boxplot(count ~ spray, data = InsectSprays,
       xlab = "Type of spray", ylab = "Insect count",
       main = "InsectSprays data", varwidth = TRUE, col = "lightgray")
fm1 <- aov(count ~ spray, data = InsectSprays)
summary(fm1)
opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(fm1)
fm2 <- aov(sqrt(count) ~ spray, data = InsectSprays)
summary(fm2)
plot(fm2)
par(opar)
```
Description

This famous (Fisher’s or Anderson’s) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

Usage

iris
iris3

Format

iris is a data frame with 150 cases (rows) and 5 variables (columns) named Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species.

iris3 gives the same data arranged as a 3-dimensional array of size 50 by 4 by 3, as represented by S-PLUS. The first dimension gives the case number within the species subsample, the second the measurements with names Sepal L., Sepal W., Petal L., and Petal W., and the third the species.

Source


References


See Also

matplot some examples of which use iris.

Examples

dni3 <- dimnames(iris3)
ii <- data.frame(matrix(aperm(iris3, c(1,3,2)), ncol=4,
    dimnames = list(NULL, sub(" L."",".Length",
    sub(" W.",".Width", dni3[[2]])))))
    Species = gl(3, 50,
    lab=sub("S", "s", sub("V", "v", dni3[[3]])))))
all.equal(ii, iris) # TRUE
### islands

**Areas of the World’s Major Landmasses**

**Description**
The areas in thousands of square miles of the landmasses which exceed 10,000 square miles.

**Usage**

```
islands
```

**Format**

A named vector of length 48.

**Source**

**References**

**Examples**

```r
require(graphics)
dotchart(log(islands, 10),
    main = "islands data: log10(area) (log10(sq. miles))")
dotchart(log(islands[order(islands)], 10),
    main = "islands data: log10(area) (log10(sq. miles))")
```

---

### JohnsonJohnson

**Quarterly Earnings per Johnson & Johnson Share**

**Description**
Quarterly earnings (dollars) per Johnson & Johnson share 1960–80.

**Usage**

```
JohnsonJohnson
```

**Format**

A quarterly time series

**Source**
LakeHuron

Examples

```r
require(stats)
JJ <- log10(JohnsonJohnson)
plot(JJ)
(fit <- StructTS(JJ, type="BSM"))
tsdiag(fit)
sm <- tsSmooth(fit)
plot(cbind(JJ, sm[, 1], sm[, 3]-0.5), plot.type = "single",
     col = c("black", "green", "blue"));
abline(h = -0.5, col = "grey60")
monthplot(fit)
```

LakeHuron
Level of Lake Huron 1875–1972

Description

Annual measurements of the level, in feet, of Lake Huron 1875–1972.

Usage

LakeHuron

Format

A time series of length 98.

Source


lh

Luteinizing Hormone in Blood Samples

Description

A regular time series giving the luteinizing hormone in blood samples at 10 mins intervals from a human female, 48 samples.

Usage

lh

Source

LifeCycleSavings  Intercountry Life-Cycle Savings Data

Description


Usage

LifeCycleSavings

Format

A data frame with 50 observations on 5 variables.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[.1] sr</td>
<td>numeric</td>
<td>aggregate personal savings</td>
</tr>
<tr>
<td>[.2] pop15</td>
<td>numeric</td>
<td>% of population under 15</td>
</tr>
<tr>
<td>[.3] pop75</td>
<td>numeric</td>
<td>% of population over 75</td>
</tr>
<tr>
<td>[.4] dpi</td>
<td>numeric</td>
<td>real per-capita disposable income</td>
</tr>
<tr>
<td>[.5] ddpi</td>
<td>numeric</td>
<td>% growth rate of dpi</td>
</tr>
</tbody>
</table>

Details

Under the life-cycle savings hypothesis as developed by Franco Modigliani, the savings ratio (aggregate personal saving divided by disposable income) is explained by per-capita disposable income, the percentage rate of change in per-capita disposable income, and two demographic variables: the percentage of population less than 15 years old and the percentage of the population over 75 years old. The data are averaged over the decade 1960–1970 to remove the business cycle or other short-term fluctuations.

Source

The data were obtained from Belsley, Kuh and Welsch (1980). They in turn obtained the data from Sterling (1977).

References


Examples

```r
require(stats)
pairs(LifeCycleSavings, panel = panel.smooth,
     main = "LifeCycleSavings data")
fm1 <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)
summary(fm1)
```
Loblolly

Description

The Loblolly data frame has 84 rows and 3 columns of records of the growth of Loblolly pine trees.

Usage

Loblolly

Format

This data frame contains the following columns:

- **height**: a numeric vector of tree heights (ft).
- **age**: a numeric vector of tree ages (yr).
- **Seed**: an ordered factor indicating the seed source for the tree. The ordering is according to increasing maximum height.

Source


Examples

```r
require(stats)
plot(height ~ age, data = Loblolly, subset = Seed == 329,
     xlab = "Tree age (yr)", las = 1,
     ylab = "Tree height (ft)",
     main = "Loblolly data and fitted curve (Seed 329 only)"
)
fm1 <- nls(height ~ SSasymp(age, Asym, R0, lrc),
           data = Loblolly, subset = Seed == 329)
summary(fm1)
age <- seq(0, 30, len = 101)
lines(age, predict(fm1, list(age = age)))
```

longley

Longley's Economic Regression Data

Description

A macroeconomic data set which provides a well-known example for a highly collinear regression.

Usage

longley
Format

A data frame with 7 economical variables, observed yearly from 1947 to 1962 ($n = 16$).

- **GNP.deflator**: GNP implicit price deflator (1954 = 100)
- **GNP**: Gross National Product.
- **Unemployed**: number of unemployed.
- **Armed.Forces**: number of people in the armed forces.
- **Population**: ‘noninstitutionalized’ population $\geq$ 14 years of age.
- **Year**: the year (time).
- **Employed**: number of people employed.

The regression $\text{lm(Employed} \sim .)$ is known to be highly collinear.

Source


References


Examples

```r
## give the data set in the form it is used in S-PLUS:
longley.x <- data.matrix(longley[, 1:6])
longley.y <- longley[, "Employed"]
pairs(longley, main = "longley data")
summary(fm1 <- lm(Employed ~ ., data = longley))
par(opar <- par(mfrow = c(2, 2), oma = c(0, 0, 1.1, 0),
             mar = c(4.1, 4.1, 2.1, 1.1))
plot(fm1)
par(opar)
```

 lynx

*Annual Canadian Lynx trappings 1821–1934*

Description

Annual numbers of lynx trappings for 1821–1934 in Canada. Taken from Brockwell & Davis (1991), this appears to be the series considered by Campbell & Walker (1977).

Usage

`lynx`

Source

morley

References


morley Michaelson-Morley Speed of Light Data

Description

The classical data of Michaelson and Morley on the speed of light. The data consists of five experiments, each consisting of 20 consecutive ‘runs’. The response is the speed of light measurement, suitably coded.

Usage

morley

Format

A data frame contains the following components:

- **Expt** The experiment number, from 1 to 5.
- **Run** The run number within each experiment.
- **Speed** Speed-of-light measurement.

Details

The data is here viewed as a randomized block experiment with ‘experiment’ and ‘run’ as the factors. ‘run’ may also be considered a quantitative variate to account for linear (or polynomial) changes in the measurement over the course of a single experiment.

Source


Examples

```r
require(stats)
morley$Expt <- factor(morley$Expt)
morley$Run <- factor(morley$Run)
attach(morley)
plot(Expt, Speed, main = "Speed of Light Data", xlab = "Experiment No.")
fm <- aov(Speed ~ Run + Expt, data = morley)
summary(fm)
fm0 <- update(fm, . ~ . - Run)
anova(fm0, fm)
detach(morley)
```
**mtcars**  
*Motor Trend Car Road Tests*

**Description**

The data was extracted from the 1974 *Motor Trend* US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

**Usage**

`mtcars`

**Format**

A data frame with 32 observations on 11 variables.

| . 1 | mpg   | Miles/(US) gallon |
| . 2 | cyl   | Number of cylinders |
| . 3 | disp  | Displacement (cu.in.) |
| . 4 | hp    | Gross horsepower |
| . 5 | drat  | Rear axle ratio |
| . 6 | wt    | Weight (lb/1000) |
| . 7 | qsec  | 1/4 mile time |
| . 8 | vs    | V/S |
| . 9 | am    | Transmission (0 = automatic, 1 = manual) |
| .10 | gear  | Number of forward gears |
| .11 | carb  | Number of carburetors |

**Source**


**Examples**

```r
pairs(mtcars, main = "mtcars data")
coplot(mpg ~ disp | as.factor(cyl), data = mtcars,  
       panel = panel.smooth, rows = 1)
```

**nhtemp**  
*Average Yearly Temperatures in New Haven*

**Description**

The mean annual temperature in degrees Fahrenheit in New Haven, Connecticut, from 1912 to 1971.

**Usage**

`nhtemp`
Nile

Format

A time series of 60 observations.

Source


References


Examples

```r
plot(nhtemp, main = "nhtemp data",
     ylab = "Mean annual temperature in New Haven, CT (deg. F)"
)
```

---

Nile  Flow of the River Nile

Description


Usage

Nile

Format

A time series of length 100.

Source


References


Examples

```r
require(stats)
par(mfrow = c(2,2))
plot(Nile)
acf(Nile)
pacf(Nile)
ar(Nile)  # selects order 2
cpgram(ar(Nile)$resid)
par(mfrow = c(1,1))
arima(Nile, c(2, 0, 0))

## Now consider missing values, following Durbin & Koopman
NileNA <- Nile
NileNA[c(21:40, 61:80)] <- NA
arima(NileNA, c(2, 0, 0))
plot(NileNA)
pred <- predict(arima(window(NileNA, 1871, 1890), c(2,0,0)), n.ahead = 20)
lines(pred$pred, lty = 3, col = "red")
lines(pred$pred + 2*pred$se, lty=2, col="blue")
lines(pred$pred - 2*pred$se, lty=2, col="blue")
pred <- predict(arima(window(NileNA, 1871, 1930), c(2,0,0)), n.ahead = 20)
lines(pred$pred, lty = 3, col = "red")
lines(pred$pred + 2*pred$se, lty=2, col="blue")
lines(pred$pred - 2*pred$se, lty=2, col="blue")

## Structural time series models
par(mfrow = c(3, 1))
plot(Nile)

## local level model
(fit <- StructTS(Nile, type = "level"))
lines(fitted(fit), lty = 2)  # contemporaneous smoothing
lines(tsSmooth(fit), lty = 2, col = 4)  # fixed-interval smoothing
plot(residuals(fit)); abline(h = 0, lty = 3)

## local trend model
(fit2 <- StructTS(Nile, type = "trend")  # constant trend fitted
pred <- predict(fit, n.ahead = 30)
## with 50% confidence interval
ts.plot(Nile, pred$pred, pred$pred + 0.67*pred$se, pred$pred -0.67*pred$se)

## Now consider missing values
plot(NileNA)
(fit3 <- StructTS(NileNA, type = "level"))
lines(fitted(fit3), lty = 2)
lines(tsSmooth(fit3), lty = 3)
plot(residuals(fit3)); abline(h = 0, lty = 3)
```

---

**nottem**

*Average Monthly Temperatures at Nottingham, 1920–1939*

**Description**

A time series object containing average air temperatures at Nottingham Castle in degrees Fahrenheit for 20 years.
Usage

nottem

Source


Examples

## Not run:
nott <- window(nottem, end=c(1936,12))
fit <- arima(nott,order=c(1,0,0), list(order=c(2,1,0), period=12))
nott.fore <- predict(fit, n.ahead=36)
ts.plot(nott, nott.fore$pred, nott.fore$pred+2*nott.fore$se,
        nott.fore$pred-2*nott.fore$se, gpars=list(col=c(1,1,4,4)))
## End(Not run)

Orange

*Growth of orange trees*

Description

The Orange data frame has 35 rows and 3 columns of records of the growth of orange trees.

Usage

Orange

Format

This data frame contains the following columns:

Tree an ordered factor indicating the tree on which the measurement is made. The ordering is according to increasing maximum diameter.

age a numeric vector giving the age of the tree (days since 1968/12/31)

circumference a numeric vector of trunk circumferences (mm). This is probably “circumference at breast height”, a standard measurement in forestry.

Source


OrchardSprays

Examples

```r
require(stats)
coplot(circumference ~ age | Tree, data = Orange, show = FALSE)
fm1 <- nls(circumference ~ SSlogis(age, Asym, xmid, scal),
  data = Orange, subset = Tree == 3)
plot(circumference ~ age, data = Orange, subset = Tree == 3,
  xlab = "Tree age (days since 1968/12/31)",
  ylab = "Tree circumference (mm)", las = 1,
  main = "Orange tree data and fitted model (Tree 3 only)"
age <- seq(0, 1600, len = 101)
lines(age, predict(fm1, list(age = age)))
```

OrchardSprays  Potency of Orchard Sprays

Description

An experiment was conducted to assess the potency of various constituents of orchard sprays in repelling honeybees, using a Latin square design.

Usage

OrchardSprays

Format

A data frame with 64 observations on 4 variables.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rowpos</td>
<td>numeric</td>
<td>Row of the design</td>
</tr>
<tr>
<td>[.1]</td>
<td>colpos</td>
<td>numeric</td>
<td>Column of the design</td>
</tr>
<tr>
<td>[.2]</td>
<td>treatment</td>
<td>factor</td>
<td>Treatment level</td>
</tr>
<tr>
<td>[.3]</td>
<td>decrease</td>
<td>numeric</td>
<td>Response</td>
</tr>
</tbody>
</table>

Details

Individual cells of dry comb were filled with measured amounts of lime sulphur emulsion in sucrose solution. Seven different concentrations of lime sulphur ranging from a concentration of 1/100 to 1/1,562,500 in successive factors of 1/5 were used as well as a solution containing no lime sulphur.

The responses for the different solutions were obtained by releasing 100 bees into the chamber for two hours, and then measuring the decrease in volume of the solutions in the various cells.

An $8 \times 8$ Latin square design was used and the treatments were coded as follows:

- A: highest level of lime sulphur
- B: next highest level of lime sulphur
- C
- D
- E
- F
- G: lowest level of lime sulphur
- H: no lime sulphur
precip

Source

References

Examples
pairs(OrchardSprays, main = "OrchardSprays data")

PlantGrowth

Results from an Experiment on Plant Growth

Description
Results from an experiment to compare yields (as measured by dried weight of plants) obtained under a control and two different treatment conditions.

Usage
PlantGrowth

Format
A data frame of 30 cases on 2 variables.

[, 1] weight numeric
[, 2] group factor

The levels of group are ‘ctrl’, ‘trt1’, and ‘trt2’.

Source

Examples
## One factor ANOVA example from Dobson's book, cf. Table 7.4:
require(stats)
boxplot(weight ~ group, data = PlantGrowth, main = "PlantGrowth data",
        ylab = "Dried weight of plants", col = "lightgray",
        notch = TRUE, varwidth = TRUE)
anova(lm(weight ~ group, data = PlantGrowth))

precip

Annual Precipitation in US Cities
Description

The average amount of precipitation (rainfall) in inches for each of 70 United States (and Puerto Rico) cities.

Usage

precip

Format

A named vector of length 70.

Source


References


Examples

```r
require(graphics)
dotchart(precip[order(precip)], main = "precip data")
title(sub = "Average annual precipitation (in."))
```

 presidents  Quarter Approval Ratings of US Presidents

Description

The (approximately) quarterly approval rating for the President of the United states from the first quarter of 1945 to the last quarter of 1974.

Usage

presidents

Format

A time series of 120 values.

Details

The data are actually a fudged version of the approval ratings. See McNeil’s book for details.

Source

The Gallup Organisation.

References

Puromycin

Examples

```
plot(presidents, las = 1, ylab = "Approval rating (%)",
     main = "presidents data")
```

---

**pressure**

*Vapor Pressure of Mercury as a Function of Temperature*

**Description**

Data on the relation between temperature in degrees Celsius and vapor pressure of mercury in millimeters (of mercury).

**Usage**

```
preservation
```

**Format**

A data frame with 19 observations on 2 variables.

```
[, 1] temperature numeric temperature (deg C)
[, 2] pressure numeric pressure (mm)
```

**Source**


**References**


**Examples**

```
plot(preservation, xlab = "Temperature (deg C)",
     ylab = "Pressure (mm of Hg)",
     main = "pressure data: Vapor Pressure of Mercury")
plot(preservation, xlab = "Temperature (deg C)",
     ylab = "Pressure (mm of Hg)",
     log = "y",
     main = "pressure data: Vapor Pressure of Mercury")
```

---

Puromycin

*Reaction velocity of an enzymatic reaction*

**Description**

The Puromycin data frame has 23 rows and 3 columns of the reaction velocity versus substrate concentration in an enzymatic reaction involving untreated cells or cells treated with Puromycin.

**Usage**

```
Puromycin
```
**Puromycin**

**Format**

This data frame contains the following columns:

- **conc** a numeric vector of substrate concentrations (ppm)
- **rate** a numeric vector of instantaneous reaction rates (counts/min/min)
- **state** a factor with levels treated untreated

**Details**

Data on the “velocity” of an enzymatic reaction were obtained by Treloar (1974). The number of counts per minute of radioactive product from the reaction was measured as a function of substrate concentration in parts per million (ppm) and from these counts the initial rate, or “velocity,” of the reaction was calculated (counts/min/min). The experiment was conducted once with the enzyme treated with Puromycin, and once with the enzyme untreated.

**Source**


**Examples**

```R
plot(rate ~ conc, data = Puromycin, las = 1,
xlab = "Substrate concentration (ppm)",
ylab = "Reaction velocity (counts/min/min)",
pch = as.integer(Puromycin$state),
col = as.integer(Puromycin$state),
main = "Puromycin data and fitted Michaelis-Menten curves")
## simplest form of fitting the Michaelis-Menten model to these data
fm1 <- nls(rate ~ Vm * conc/(K + conc), data = Puromycin,
subset = state == "treated",
start = c(Vm = 200, K = 0.05), trace = TRUE)
fm2 <- nls(rate ~ Vm * conc/(K + conc), data = Puromycin,
subset = state == "untreated",
start = c(Vm = 160, K = 0.05), trace = TRUE)
summary(fm1)
summary(fm2)
## using partial linearity
fm3 <- nls(rate ~ conc/(K + conc), data = Puromycin,
subset = state == "treated",
start = c(K = 0.05),
algorithm = "plinear", trace = TRUE)
## using a self-starting model
fm4 <- nls(rate ~ SSmicmen(conc, Vm, K), data = Puromycin,
subset = state == "treated")
summary(fm4)
## add fitted lines to the plot
conc <- seq(0, 1.2, len = 101)
lines(conc, predict(fm1, list(conc = conc)), lty = 1, col = 1)
lines(conc, predict(fm2, list(conc = conc)), lty = 2, col = 2)
legend(0.8, 120, levels(Puromycin$state),
col = 1:2, lty = 1:2, pch = 1:2)
```
**quakes**

*Locations of Earthquakes off Fiji*

**Description**

The data set give the locations of 1000 seismic events of MB > 4.0. The events occurred in a cube near Fiji since 1964.

**Usage**

quakes

**Format**

A data frame with 1000 observations on 5 variables.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>lat</td>
<td>numeric</td>
<td>Latitude of event</td>
</tr>
<tr>
<td>.2</td>
<td>long</td>
<td>numeric</td>
<td>Longitude</td>
</tr>
<tr>
<td>.3</td>
<td>depth</td>
<td>numeric</td>
<td>Depth (km)</td>
</tr>
<tr>
<td>.4</td>
<td>mag</td>
<td>numeric</td>
<td>Richter Magnitude</td>
</tr>
<tr>
<td>.5</td>
<td>stations</td>
<td>numeric</td>
<td>Number of stations reporting</td>
</tr>
</tbody>
</table>

**Details**

There are two clear planes of seismic activity. One is a major plate junction; the other is the Tonga trench off New Zealand. These data constitute a subsample from a larger dataset of containing 5000 observations.

**Source**

This is one of the Harvard PRIM-H project data sets. They in turn obtained it from Dr. John Woodhouse, Dept. of Geophysics, Harvard University.

**Examples**

```r
pairs(quakes, main = "Fiji Earthquakes, N = 1000", cex.main=1.2, pch=".")
```

**randu**

*Random Numbers from Congruential Generator RANDU*

**Description**

400 triples of successive random numbers were taken from the VAX FORTRAN function RANDU running under VMS 1.5.

**Usage**

randu
Format

A data frame with 400 observations on 3 variables named x, y and z which give the first, second and third random number in the triple.

Details

In three dimensional displays it is evident that the triples fall on 15 parallel planes in 3-space. This can be shown theoretically to be true for all triples from the RANDU generator.

These particular 400 triples start 5 apart in the sequence, that is they are ((U[5i+1], U[5i+2], U[5i+3]), i= 0, . . . , 399), and they are rounded to 6 decimal places.

Under VMS versions 2.0 and higher, this problem has been fixed.

Source

David Donoho

Examples

```r
## Not run:
## We could re-generate the dataset by the following R code
seed <- as.double(1)
RANDU <- function() {
  seed <<- ((2^16 + 3) * seed) %% (2^31)
  seed/(2^31)
}
for(i in 1:400) {
  U <- c(RANDU(), RANDU(), RANDU(), RANDU(), RANDU())
  print(round(U[1:3], 6))
}
## End(Not run)
```

---

**rivers**

*Lengths of Major North American Rivers*

Description

This data set gives the lengths (in miles) of 141 “major” rivers in North America, as compiled by the US Geological Survey.

Usage

rivers

Format

A vector containing 141 observations.

Source

References


---

**sleep**

*Student’s Sleep Data*

Description

Data which show the effect of two soporific drugs (increase in hours of sleep) on groups consisting of 10 patients each.

Usage

```r
sleep
```

Format

A data frame with 20 observations on 2 variables.

```r
[, 1] extra     numeric increase in hours of sleep
[, 2] group    factor    patient group
```

Source

Student (1908) The probable error of the mean. *Biometrika, 6*, 20.

References


Examples

```r
require(stats)
## ANOVA
anova(lm(extra ~ group, data = sleep))
```

---

**stackloss**

*Brownlee’s Stack Loss Plant Data*

Description

Operational data of a plant for the oxidation of ammonia to nitric acid.

Usage

```r
stackloss
stack.x
stack.loss
```
state

Format

stackloss is a data frame with 21 observations on 4 variables.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>stack.loss</td>
</tr>
</tbody>
</table>

For compatibility with S-PLUS, the data sets stack.x, a matrix with the first three (independent) variables of the data frame, and stack.loss, the numeric vector giving the fourth (dependent) variable, are provided as well.

Details

“Obtained from 21 days of operation of a plant for the oxidation of ammonia (NH₃) to nitric acid (HNO₃). The nitric oxides produced are absorbed in a countercurrent absorption tower”. (Brownlee, cited by Dodge, slightly reformatted by MM.)

Air Flow represents the rate of operation of the plant. Water Temp is the temperature of cooling water circulated through coils in the absorption tower. Acid Conc. is the concentration of the acid circulating, minus 50, times 10: that is, 89 corresponds to 58.9 per cent acid. stack.loss (the dependent variable) is 10 times the percentage of the ingoing ammonia to the plant that escapes from the absorption column unabsorbed; that is, an (inverse) measure of the over-all efficiency of the plant.

Source


References


Examples

summary(lm.stack <- lm(stack.loss ~ stack.x))

state

US State Facts and Figures

Description

Data sets related to the 50 states of the United States of America.
Usage

state.abb
state.area
state.center
state.division
state.name
state.region
state.x77

Details

R currently contains the following “state” data sets. Note that all data are arranged according to alphabetical order of the state names.

state.abb: character vector of 2-letter abbreviations for the state names.
state.area: numeric vector of state areas (in square miles).
state.center: list with components named x and y giving the approximate geographic center of each state in negative longitude and latitude. Alaska and Hawaii are placed just off the West Coast.
state.division: factor giving state divisions (New England, Middle Atlantic, South Atlantic, East South Central, West South Central, East North Central, West North Central, Mountain, and Pacific).
state.name: character vector giving the full state names.
state.region: factor giving the region (Northeast, South, North Central, West) that each state belongs to.
state.x77: matrix with 50 rows and 8 columns giving the following statistics in the respective columns.

Population: population estimate as of July 1, 1975
Income: per capita income (1974)
Illiteracy: illiteracy (1970, percent of population)
Life Exp: life expectancy in years (1969–71)
Murder: murder and non-negligent manslaughter rate per 100,000 population (1976)
HS Grad: percent high-school graduates (1970)
Frost: mean number of days with minimum temperature below freezing (1931–1960) in capital or large city
Area: land area in square miles

Source


References

sunspot.month    Monthly Sunspot Data, 1749–1997

Description

Monthly numbers of sunspots.

Usage

sunspot.month

Format

The univariate time series sunspot.year and sunspot.month contain 289 and 2988 observations, respectively. The objects are of class "ts".

Note

Prior to R 2.0.0 sunspot.month and sunspot.year were copied to the user’s workspace (or elsewhere) by data(sunspot). However, as package lattice has a dataset sunspot which can be retrieved by the same command, that usage has been removed.

Source

World Data Center-C1 For Sunspot Index Royal Observatory of Belgium, Av. Circulaire, 3, B-1180 BRUSSELS http://www.oma.be/KSB-ORB/SIDC/sidc_txt.html

See Also

sunspot.month is a longer version of sunspots that runs until 1988 rather than 1983.

Examples

require(stats)
## Compare the monthly series
plot(sunspot.month, main = "sunspot.month [stats]", col = 2)
lines(sunspots) # "very barely" see something
## Now look at the difference :
all(tsp(sunspots) [c(1,3)] == tsp(sunspot.month)[c(1,3)]) ## Start & Periodicity are the same
n1 <- length(sunspots)
table(eq <- sunspots == sunspot.month[1:n1]) #> 132 are different !
i <- which(!eq)
rug(time(eq)[i])
s1 <- sunspots[i] ; s2 <- sunspot.month[i]
cbind(i = i, sunspots = s1, ss.month = s2,
      perc.diff = round(100*2*abs(s1-s2)/(s1+s2), 1))
sunspot.year  

Description
Yearly numbers of sunspots.

Usage
sunspot.year

Format
The univariate time series sunspot.year contains 289 observations, and is of class "ts".

Note
Prior to R 2.0.0 sunspot.year and sunspot.year were copied to the user's workspace (or elsewhere) by data(sunspot). However, as package lattice has a dataset sunspot which can be retrieved by the same command, that usage has been removed.

Source

sunspots  

Description
Monthly mean relative sunspot numbers from 1749 to 1983. Collected at Swiss Federal Observatory, Zurich until 1960, then Tokyo Astronomical Observatory.

Usage
sunspots

Format
A time series of monthly data from 1749 to 1983.

Source

See Also
sunspot.month has a longer (and a bit different) series.
**Examples**

```r
plot(sunspots, main = "sunspots data", xlab = "Year",
     ylab = "Monthly sunspot numbers")
```

---

**swiss**

Swiss Fertility and Socioeconomic Indicators (1888) Data

---

**Description**

Standardized fertility measure and socio-economic indicators for each of 47 French-speaking provinces of Switzerland at about 1888.

**Usage**

`swiss`

**Format**

A data frame with 47 observations on 6 variables, each of which is in percent, i.e., in $[0, 100]$.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fertility</td>
<td>$I_g$, “common standardized fertility measure”</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Agriculture</td>
<td>% of males involved in agriculture as occupation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Examination</td>
<td>% “draftees” receiving highest mark on army examination</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Education</td>
<td>% education beyond primary school for “draftees”</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Catholic</td>
<td>% catholic (as opposed to “protestant”).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Infant.Mortality</td>
<td>live births who live less than 1 year.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All variables but ‘Fertility’ give proportions of the population.

**Details**

(paraphrasing Mosteller and Tukey):

Switzerland, in 1888, was entering a period known as the “demographic transition”; i.e., its fertility was beginning to fall from the high level typical of underdeveloped countries.

The data collected are for 47 French-speaking “provinces” at about 1888.

Here, all variables are scaled to $[0, 100]$, where in the original, all but "Catholic" were scaled to $[0, 1]$.

**Note**

Files for all 182 districts in 1888 and other years have been available at [http://opr.princeton.edu/archive/eufert/switz.html](http://opr.princeton.edu/archive/eufert/switz.html) or [http://opr.princeton.edu/archive/pefp/switz.asp](http://opr.princeton.edu/archive/pefp/switz.asp).

They state that variables Examination and Education are averages for 1887, 1888 and 1889.

**Source**

Project “16P5”, pages 549–551 in


References


Examples

```r
pairs(swiss, panel = panel.smooth, main = "swiss data",
col = 3 + (swiss$Catholic > 50))
summary(lm(Fertility ~ ., data = swiss))
```

---

**Theoph**

*Pharmacokinetics of theophylline*

**Description**

The *Theoph* data frame has 132 rows and 5 columns of data from an experiment on the pharmacokinetics of theophylline.

**Usage**

`Theoph`

**Format**

This data frame contains the following columns:

- **Subject**: an ordered factor with levels 1, …, 12 identifying the subject on whom the observation was made. The ordering is by increasing maximum concentration of theophylline observed.
- **Wt**: weight of the subject (kg).
- **Dose**: dose of theophylline administered orally to the subject (mg/kg).
- **Time**: time since drug administration when the sample was drawn (hr).
- **conc**: theophylline concentration in the sample (mg/L).

**Details**

Boeckmann, Sheiner and Beal (1994) report data from a study by Dr. Robert Upton of the kinetics of the anti-asthmatic drug theophylline. Twelve subjects were given oral doses of theophylline then serum concentrations were measured at 11 time points over the next 25 hours.

These data are analyzed in Davidian and Giltinan (1995) and Pinheiro and Bates (2000) using a two-compartment open pharmacokinetic model, for which a self-starting model function, `SSfol`, is available.
Titanic

Survival of passengers on the Titanic

Source


See Also

SSfol

Examples

```r
require(stats)
coplot(conc ~ Time | Subject, data = Theoph, show = FALSE)
Theoph.4 <- subset(Theoph, Subject == 4)
fml <- nls(conc ~ SSfol(Dose, Time, lKe, lKa, lCl), data = Theoph.4)
summary(fml)
plot(conc ~ Time, data = Theoph.4,
     xlab = "Time since drug administration (hr)",
     ylab = "Theophylline concentration (mg/L)",
     main = "Observed concentrations and fitted model",
     sub = "Theophylline data - Subject 4 only",
     las = 1, col = 4)
xvals <- seq(0, par("usr"[2], len = 55)
lines(xvals, predict(fml, newdata = list(Time = xvals)),
col = 4)
```

Titanic

Survival of passengers on the Titanic

Description

This data set provides information on the fate of passengers on the fatal maiden voyage of the ocean liner ‘Titanic’, summarized according to economic status (class), sex, age and survival.

Usage

Titanic

Format

A 4-dimensional array resulting from cross-tabulating 2201 observations on 4 variables. The variables and their levels are as follows:

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Class</td>
<td>1st, 2nd, 3rd, Crew</td>
</tr>
<tr>
<td>2</td>
<td>Sex</td>
<td>Male, Female</td>
</tr>
<tr>
<td>3</td>
<td>Age</td>
<td>Child, Adult</td>
</tr>
<tr>
<td>4</td>
<td>Survived</td>
<td>No, Yes</td>
</tr>
</tbody>
</table>
Details

The sinking of the Titanic is a famous event, and new books are still being published about it. Many well-known facts—from the proportions of first-class passengers to the “women and children first” policy, and the fact that that policy was not entirely successful in saving the women and children in the third class—are reflected in the survival rates for various classes of passenger.

These data were originally collected by the British Board of Trade in their investigation of the sinking. Note that there is not complete agreement among primary sources as to the exact numbers on board, rescued, or lost.

Due in particular to the very successful film ‘Titanic’, the last years saw a rise in public interest in the Titanic. Very detailed data about the passengers is now available on the Internet, at sites such as Encyclopedia Titanica (http://www.rmpic.co.uk/eduweb/sites/phind).

Source


The source provides a data set recording class, sex, age, and survival status for each person on board of the Titanic, and is based on data originally collected by the British Board of Trade and reprinted in:


Examples

```
require(graphics)
mosaicplot(Titanic, main = "Survival on the Titanic")
## Higher survival rates in children?
apply(Titanic, c(3, 4), sum)
## Higher survival rates in females?
apply(Titanic, c(2, 4), sum)
## Use loglm() in package 'MASS' for further analysis ...
```

ToothGrowth  The Effect of Vitamin C on Tooth Growth in Guinea Pigs

Description

The response is the length of odontoblasts (teeth) in each of 10 guinea pigs at each of three dose levels of Vitamin C (0.5, 1, and 2 mg) with each of two delivery methods (orange juice or ascorbic acid).

Usage

ToothGrowth

Format

A data frame with 60 observations on 3 variables.

```r
ToothGrowth

## Column names

len numeric Tooth length
supp factor Supplement type (VC or OJ).
dose numeric Dose in milligrams.
```
Source

References

Examples
```
coplot(len ~ dose | supp, data = ToothGrowth, panel = panel.smooth, 
        xlab = "ToothGrowth data: length vs dose, given type of supplement")
```

treering | Yearly Treering Data, -6000–1979

Description
Contains normalized tree-ring widths in dimensionless units.

Usage
treering

Format
A univariate time series with 7981 observations. The object is of class "ts".
Each tree ring corresponds to one year.

Details
The data were recorded by Donald A. Graybill, 1980, from Gt Basin Bristlecone Pine 2805M, 3726-11810 in Methuselah Walk, California.

Source

References
For background on Bristlecone pines and Methuselah Walk, see [http://www.sonic.net/bristlecone/](http://www.sonic.net/bristlecone/); for some photos see [http://www.ltrr.arizona.edu/~hallman/sitephotos/meth.html](http://www.ltrr.arizona.edu/~hallman/sitephotos/meth.html)
**trees**  
*Girth, Height and Volume for Black Cherry Trees*

**Description**

This data set provides measurements of the girth, height and volume of timber in 31 felled black cherry trees. Note that girth is the diameter of the tree (in inches) measured at 4 ft 6 in above the ground.

**Usage**

trees

**Format**

A data frame with 31 observations on 3 variables.

<table>
<thead>
<tr>
<th></th>
<th>Girth</th>
<th>Height</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>numeric</td>
<td>Tree diameter in inches</td>
<td>Height in ft</td>
</tr>
</tbody>
</table>

**Source**


**References**


**Examples**

```r
pairs(trees, panel = panel.smooth, main = "trees data")
plot(Volume ~ Girth, data = trees, log = "xy")
coplot(log(Volume) ~ log(Girth) | Height, data = trees,
      panel = panel.smooth)
summary(fm1 <- lm(log(Volume) ~ log(Girth), data = trees))
summary(fm2 <- update(fm1, ~ . + log(Height), data = trees))
step(fm2)
## i.e., Volume ~ c * Height * Girth^2 seems reasonable
```

---

**UCBAAdmissions**  
*Student Admissions at UC Berkeley*

**Description**

Aggregate data on applicants to graduate school at Berkeley for the six largest departments in 1973 classified by admission and sex.

**Usage**

UCBAAdmissions
Format

A 3-dimensional array resulting from cross-tabulating 4526 observations on 3 variables. The variables and their levels are as follows:

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Admit</td>
<td>Admitted, Rejected</td>
</tr>
<tr>
<td>2</td>
<td>Gender</td>
<td>Male, Female</td>
</tr>
<tr>
<td>3</td>
<td>Dept</td>
<td>A, B, C, D, E, F</td>
</tr>
</tbody>
</table>

Details

This data set is frequently used for illustrating Simpson’s paradox, see Bickel et al. (1975). At issue is whether the data show evidence of sex bias in admission practices. There were 2691 male applicants, of whom 1198 (44.5%) were admitted, compared with 1835 female applicants of whom 557 (30.4%) were admitted. This gives a sample odds ratio of 1.83, indicating that males were almost twice as likely to be admitted. In fact, graphical methods (as in the example below) or log-linear modelling show that the apparent association between admission and sex stems from differences in the tendency of males and females to apply to the individual departments (females used to apply “more” to departments with higher rejection rates).

This data set can also be used for illustrating methods for graphical display of categorical data, such as the general-purpose mosaic plot or the “fourfold display” for 2-by-2-by-\( k \) tables. See the home page of Michael Friendly (http://www.math.yorku.ca/SCS/friendly.html) for further information.

References


Examples

```r
require(graphics)
## Data aggregated over departments
apply(UCBAdmissions, c(1, 2), sum)
mosaicplot(apply(UCBAdmissions, c(1, 2), sum),
          main = "Student admissions at UC Berkeley")
## Data for individual departments
opar <- par(mfrow = c(2, 3), oma = c(0, 0, 2, 0))
for(i in 1:6)
    mosaicplot(UCBAdmissions[,,i],
                xlab = "Admit", ylab = "Sex",
                main = paste("Department", LETTERS[i]))
mtext(expression(bold("Student admissions at UC Berkeley")),
       outer = TRUE, cex = 1.5)
par(opar)
```
**UKDriverDeaths**

**Description**

UKDriverDeaths is a time series giving the monthly totals of car drivers in Great Britain killed or seriously injured Jan 1969 to Dec 1984. Compulsory wearing of seat belts was introduced on 31 Jan 1983.

Seatbelts is more information on the same problem.

**Usage**

UKDriverDeaths  
Seatbelts

**Format**

Seatbelts is a multiple time series, with columns

- **DriversKilled** car drivers killed.
- **drivers** same as UKDriverDeaths.
- **front** front-seat passengers killed or seriously injured.
- **rear** rear-seat passengers killed or seriously injured.
- **kms** distance driven.
- **PetrolPrice** petrol price.
- **VanKilled** number of van (‘light goods vehicle’) drivers.
- **law** 0/1: was the law in effect that month?

**Source**


**References**


**Examples**

```r
require(stats)
## work with pre-seatbelt period to identify a model, use logs
work <- window(log10(UKDriverDeaths), end = 1982+11/12)
par(mfrow = c(3,1))
plot(work); acf(work); pacf(work)
par(mfrow = c(1,1))
(fit <- arima(work, c(1,0,0), seasonal = list(order= c(1,0,0))))
z <- predict(fit, n.ahead = 24)
ts.plot(log10(UKDriverDeaths), z$pred, z$pred+2*z$se, z$pred-2*z$se,
       lty = c(1,3,2,2), col = c("black", "red", "blue", "blue"))

## now see the effect of the explanatory variables
X <- Seatbelts[, c("kms", "PetrolPrice", "law")]
```
\[ X[, 1] \leftarrow \log_{10}(X[, 1]) - 4 \]
\[ \text{arima} \left( \log_{10}(\text{Seatbelts[, "drivers"]}), \text{c}(1,0,0), \right. \]
\[ \left. \text{seasonal} = \text{list} \left( \text{order} = \text{c}(1,0,0) \right), \text{xreg} = X \right) \]

---

**UKgas**

*UK Quarterly Gas Consumption*

**Description**

Quarterly UK gas consumption from 1960Q1 to 1986Q4, in millions of therms.

**Usage**

UKgas

**Format**

A quarterly time series of length 108.

**Source**


**Examples**

```r
## maybe str(UKgas) ; plot(UKgas) ...
```

---

**UKLungDeaths**

*Monthly Deaths from Lung Diseases in the UK*

**Description**

Three time series giving the monthly deaths from bronchitis, emphysema and asthma in the UK, 1974–1979, both sexes (ldeaths), males (mdeaths) and females (fdeaths).

**Usage**

ldeaths
fdeaths
mdeaths

**Source**

Examples

```r
require(stats) # for time
plot(ldeaths)
plot(mdeaths, fdeaths)
## Better labels:
yr <- floor(tt <- time(mdeaths))
plot(mdeaths, fdeaths, 
    xy.labels = paste(month.abb[12*(tt - yr)], yr-1900, sep="'"))
```

USAccDeaths

**Accidental Deaths in the US 1973–1978**

**Description**

A time series giving the monthly totals of accidental deaths in the USA. The values for the first six months of 1979 are 7798 7406 8363 8460 9217 9316.

**Usage**

USAccDeaths

**Source**


USArrests

**Violent Crime Rates by US State**

**Description**

This data set contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

**Usage**

USArrests

**Format**

A data frame with 50 observations on 4 variables.

<table>
<thead>
<tr>
<th></th>
<th>Murder</th>
<th>Assault</th>
<th>UrbanPop</th>
<th>Rape</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
</tr>
</tbody>
</table>

**Source**


USJudgeRatings

References

See Also
The `state` data sets.

Examples
```
pairs(USArrests, panel = panel.smooth, main = "USArrests data")
```

---

**USJudgeRatings**  
Lawyers’ Ratings of State Judges in the US Superior Court

Description
Lawyers’ ratings of state judges in the US Superior Court.

Usage
USJudgeRatings

Format
A data frame containing 43 observations on 12 numeric variables.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[,1]</td>
<td>CONT</td>
</tr>
<tr>
<td>[,2]</td>
<td>INTG</td>
</tr>
<tr>
<td>[,3]</td>
<td>DMNR</td>
</tr>
<tr>
<td>[,5]</td>
<td>CFMG</td>
</tr>
<tr>
<td>[,6]</td>
<td>DECI</td>
</tr>
<tr>
<td>[,7]</td>
<td>PREP</td>
</tr>
<tr>
<td>[,8]</td>
<td>FAMI</td>
</tr>
<tr>
<td>[,9]</td>
<td>ORAL</td>
</tr>
</tbody>
</table>

Source
New Haven Register, 14 January, 1977 (from John Hartigan).

Examples
```
pairs(USJudgeRatings, main = "USJudgeRatings data")
```
USPersonalExpenditure

*Personal Expenditure Data*

**Description**
This data set consists of United States personal expenditures (in billions of dollars) in the categories; food and tobacco, household operation, medical and health, personal care, and private education for the years 1940, 1945, 1950, 1955 and 1960.

**Usage**
USPersonalExpenditure

**Format**
A matrix with 5 rows and 5 columns.

**Source**

**References**

**Examples**
```r
require(stats)  # for medpolish
USPersonalExpenditure
medpolish(log10(USPersonalExpenditure))
```

----------

uspop

*Populations Recorded by the US Census*

**Description**
This data set gives the population of the United States (in millions) as recorded by the decennial census for the period 1790–1970.

**Usage**
uspop

**Format**
A time series of 19 values.
VADeaths

Death Rates in Virginia (1940)

Description

Death rates per 100 in Virginia in 1940.

Usage

VADeaths

Format

A matrix with 5 rows and 5 columns.

Details

The death rates are cross-classified by age group (rows) and population group (columns). The age groups are: 50–54, 55–59, 60–64, 65–69, 70–74 and the population groups are Rural/Male, Rural/Female, Urban/Male and Urban/Female.

This provides a rather nice 3-way analysis of variance example.

Source


References


Examples

```r
require(stats)
n <- length(dr <- c(VADeaths))
nam <- names(VADeaths)
d.VAD <- data.frame(
  Drate = dr,
  age = rep(ordered(rownames(VADeaths)), length=n),
  gender= gl(2,5,n, labels= c("M", "F")),
  site = gl(2,10, labels= c("rural", "urban")))
coplot(Drate ~ as.numeric(age) | gender * site, data = d.VAD,
  panel = panel.smooth, xlab = "VADeaths data - Given: gender")
summary(aov.VAD <- aov(Drate ~ .^2, data = d.VAD))
opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
```
volcano

Description
Maunga Whau (Mt Eden) is one of about 50 volcanos in the Auckland volcanic field. This data set gives topographic information for Maunga Whau on a 10m by 10m grid.

Usage
volcano

Format
A matrix with 87 rows and 61 columns, rows corresponding to grid lines running east to west and columns to grid lines running south to north.

Source
Digitized from a topographic map by Ross Ihaka. These data should not be regarded as accurate.

See Also
filled.contour for a nice plot.

Examples
require(graphics)
filled.contour(volcano, color = terrain.colors, asp = 1)
title(main = "volcano data: filled contour map")

warpbreaks

Description
This data set gives the number of warp breaks per loom, where a loom corresponds to a fixed length of yarn.

Usage
warpbreaks

Format
A data frame with 54 observations on 3 variables.

[,1] breaks numeric The number of breaks
[1,2] wool factor The type of wool (A or B)
[1,3] tension factor The level of tension (L, M, H)
There are measurements on 9 looms for each of the six types of warp (AL, AM, AH, BL, BM, BH).

Source

References

See Also
*xtabs* for ways to display these data as a table.

Examples
```r
summary(warpbreaks)
opar <- par(mfrow = c(1,2), oma = c(0, 0, 1.1, 0))
plot(breaks ~ tension, data = warpbreaks, col = "lightgray",
     varwidth = TRUE, subset = wool == "A", main = "Wool A")
plot(breaks ~ tension, data = warpbreaks, col = "lightgray",
     varwidth = TRUE, subset = wool == "B", main = "Wool B")
mtext("warpbreaks data", side = 3, outer = TRUE)
par(opar)
summary(fm1 <- lm(breaks ~ wool * tension, data = warpbreaks))
anova(fm1)
```

### women

*Average Heights and Weights for American Women*

Description
This data set gives the average heights and weights for American women aged 30–39.

Usage

```r
women
```

Format
A data frame with 15 observations on 2 variables.

```r
[,1] height numeric Height (in)
[,2] weight numeric Weight (lbs)
```

Details
The data set appears to have been taken from the American Society of Actuaries *Build and Blood Pressure Study* for some (unknown to us) earlier year.

The World Almanac notes: “The figures represent weights in ordinary indoor clothing and shoes, and heights with shoes.”
Description
The number of telephones in various regions of the world (in thousands).

Usage
phones

Format
A matrix with 7 rows and 8 columns. The columns of the matrix give the figures for a given region, and the rows the figures for a year.
The regions are: North America, Europe, Asia, South America, Oceania, Africa, Central America.

Warning
Prior to R 2.0.0 this dataset was called phones and conflicted with a dataset in package MASS.

Source

References

Examples
```
matplot(rownames(WorldPhones), WorldPhones, type = "b", log = "y",
       xlab = "Year", ylab = "Number of telephones (1000's)")
legend(1951.5, 80000, colnames(WorldPhones), col = 1:6, lty = 1:5,
pch = rep(21, 7))
title(main = "World phones data: log scale for response")
```
WWWusage

Internet Usage per Minute

Description

A time series of the numbers of users connected to the Internet through a server every minute.

Usage

WWWusage

Format

A time series of length 100.

Source


References


Examples

```r
work <- diff(WWWusage)
par(mfrow = c(2,1)); plot(WWWusage); plot(work)
## Not run:
require(stats)
aics <- matrix(, 6, 6, dimnames=list(p=0:5, q=0:5))
for(q in 1:5) aics[1, 1+q] <- arima(WWWusage, c(0,1,q), optim.control = list(maxit = 500))$aic
for(p in 1:5)
  for(q in 0:5) aics[1+p, 1+q] <- arima(WWWusage, c(p,1,q), optim.control = list(maxit = 500))$aic
round(aics - min(aics, na.rm=TRUE), 2)
## End(Not run)
```
Chapter 3

The grDevices package

boxplot.stats  
Box Plot Statistics

Description

This function is typically called by another function to gather the statistics necessary for producing box plots, but may be invoked separately.

Usage

boxplot.stats(x, coef = 1.5, do.conf = TRUE, do.out = TRUE)

Arguments

x  
a numeric vector for which the boxplot will be constructed (NAs and NaNs are allowed and omitted).
coef  
this determines how far the plot “whiskers” extend out from the box. If coef is positive, the whiskers extend to the most extreme data point which is no more than coef times the length of the box away from the box. A value of zero causes the whiskers to extend to the data extremes (and no outliers be returned).
do.conf, do.out  
logicals; if FALSE, the conf or out component respectively will be empty in the result.

Details

The two “hinges” are versions of the first and third quartile, i.e., close to quantile(x, c(1, 3)/4). The hinges equal the quartiles for odd n (where n <- length(x)) and differ for even n. Where the quartiles only equal observations for n %% 4 == 1 (n == 1 mod 4), the hinges do so additionally for n %% 4 == 2 (n == 2 mod 4), and are in the middle of two observations otherwise.

The notches (if requested) extend to +/-1.58 IQR/sqrt(n). This seems to be based on same calculations as the formula with 1.57 in Chambers et al. (1983, p. 62), given in McGill et al. (1978, p. 16). They are based on asymptotic normality of the median and roughly equal sample sizes for the two medians being compared, and are said to be rather insensitive to the underlying distributions of the samples. The idea appears to be to give roughly a 95% confidence interval for the difference in two medians.
Value

List with named components as follows:

- **stats**: a vector of length 5, containing the extreme of the lower whisker, the lower “hinge”, the median, the upper “hinge” and the extreme of the upper whisker.
- **n**: the number of non-NA observations in the sample.
- **conf**: the lower and upper extremes of the “notch” (if(do.conf)). See the details.
- **out**: the values of any data points which lie beyond the extremes of the whiskers (if(do.out)).

Note that $stats$ and $conf$ are sorted in increasing order, unlike S, and that $n$ and $out$ include any +/− Inf values.

References


See Also

- `fivenum`, `boxplot`, `bxp`.

Examples

```r
x <- c(1:100, 1000)
(b1 <- boxplot.stats(x))
(b2 <- boxplot.stats(x, do.conf=FALSE, do.out=FALSE))
stopifnot(b1 $ stats == b2 $ stats) # do.out=F is still robust
boxplot.stats(x, coef = 3, do.conf=FALSE)
## no outlier treatment:
boxplot.stats(x, coef = 0)

boxplot.stats(c(x, NA)) # slight change : n is 101
(r <- boxplot.stats(c(x, -1:1/0)))
stopifnot(r$out == c(1000, -Inf, Inf))
```
bringToTop  Assign Focus to a Window

Description

bringToTop brings the specified screen device’s window to the front of the window stack (and gives it focus). With argument -1, it brings the console to the top.

If stay = TRUE, the window is designated as a topmost window, i.e. it will stay on top of any regular window. stay may only be used when Rgui is run in SDI mode. This corresponds to the “Stay on top” popup menu item in Rgui.

Usage

bringToTop(which = dev.cur(), stay = FALSE)

Arguments

which  a device number, or -1.
stay   whether to make the window stay on top.

See Also

windows

check.options  Set Options with Consistency Checks

Description

Utility function for setting options with some consistency checks. The attributes of the new settings in new are checked for consistency with the model (often default) list in name.opt.

Usage

check.options(new, name.opt, reset = FALSE, assign.opt = FALSE, envir = .GlobalEnv, check.attributes = c("mode", "length"), override.check = FALSE)

Arguments

new         a named list
name.opt    character with the name of R object containing the “model” (default) list.
reset       logical; if TRUE, reset the options from name.opt. If there is more than one R object with name opt, remove the first one in the search() path.
assign.opt  logical; if TRUE, assign the ...
envir       the environment used for get and assign.
check.attributes
character containing the attributes which check.options should check.

override.check
logical vector of length length(new) (or 1 which entails recycling). For those new[i] where override.check[i] == TRUE, the checks are overridden and the changes made anyway.

Value
A list of components with the same names as the one called name.opt. The values of the components are changed from the new list, as long as these pass the checks (when these are not overridden according to override.check).

Author(s)
Martin Maechler

See Also
ps.options which uses check.options.

Examples
(L1 <- list(a=1:3, b=pi, ch="CH"))
check.options(list(a=0:2), name.opt = "L1")
check.options(NULL, reset = TRUE, name.opt = "L1")

chull

Compute Convex Hull of a Set of Points

Description
Computes the subset of points which lie on the convex hull of the set of points specified.

Usage
chull(x, y = NULL)

Arguments
x, y coordinate vectors of points. This can be specified as two vectors x and y, a 2-column matrix x, a list x with two components, etc, see xy.coords.

Details
xy.coords is used to interpret the specification of the points. The algorithm is that given by Eddy (1977).
‘Peeling’ as used in the S function chull can be implemented by calling chull recursively.

Value
An integer vector giving the indices of the points lying on the convex hull, in clockwise order.
References


See Also

*xy.coords*, *polygon*

Examples

```r
X <- matrix(rnorm(2000), ncol = 2)
chull(X)
## Not run:
# Example usage from graphics package
plot(X, cex = 0.5)
hpts <- chull(X)
hpts <- c(hpts, hpts[1])
lines(X[hpts, ])
## End(Not run)
```

---

**cm**

Unit transformation

### Description

Translates from inches to cm (centimeters).

### Usage

```r
cm(x)
```

### Arguments

- `x` numeric vector

### Examples

```r
cm(1)# = 2.54
```
col2rgb  

Color to RGB Conversion

Description

“Any R color” to RGB (red/green/blue) conversion.

Usage

col2rgb(col, alpha = FALSE)

Arguments

col vector of any of the three kind of R colors, i.e., either a color name (an element of colors()), a hexadecimal string of the form "#rrggbb", or an integer i meaning palette()[i].

alpha logical value indicating whether alpha channel values should be returned.

Details

For integer colors, 0 is shorthand for the current par("bg"), and NA means “nothing” which effectively does not draw the corresponding item.

For character colors, "NA" is equivalent to NA above.

Value

an integer matrix with three or four rows and number of columns the length (and names if any) as col.

Author(s)

Martin Maechler

See Also

rgb, colors, palette, etc.

Examples

col2rgb("peachpuff")
col2rgb(c(blu = "royalblue", reddish = "tomato")) # names kept

col2rgb(1:8)# the ones from the palette() :

col2rgb(paste("gold", 1:4, sep=""))

col2rgb("#08a0ff")

## all three kind of colors mixed :
col2rgb(c(red="red", palette= 1:3, hex="#abcdef"))

##-- NON-INTRODUCTORY examples --
grC <- col2rgb(paste("gray",0:100,sep=""))
These functions return functions that interpolate a set of given colors to create new color palettes (like `topo.colors`) and color ramps, functions that map the interval \([0, 1]\) to colors (like `grey`).

**Usage**

```r
colorRamp(colors, bias = 1, space = c("rgb", "Lab"),
          interpolate = c("linear", "spline"))
colorRampPalette(colors, ...)
```

**Arguments**

- `colors`  Colors to interpolate
- `bias`    A positive number. Higher values give more widely spaced colors at the high end.
- `space`   Interpolation in RGB or CIE Lab color spaces
- `interpolate` Use spline or linear interpolation.
- `...`     arguments to pass to `colorRamp`.
The CIE Lab color space is approximately perceptually uniform, and so gives smoother and more uniform color ramps. On the other hand, palettes that vary from one hue to another via white may have a more symmetrical appearance in RGB space.

The conversion formulas in this function do not appear to be completely accurate and the color ramp may not reach the extreme values in Lab space. Future changes in the R color model may change the colors produced with `space = "Lab"`.

**Value**

colorRamp returns a function that maps values between 0 and 1 to colors. colorRampPalette returns a function that takes an integer argument and returns that number of colors interpolating the given sequence (similar to `heat.colors` or `terrain.colors`.

**See Also**

Good starting points for interpolation are the "sequential" and "diverging" ColorBrewer palettes in the RColorBrewer package

**Examples**

```r
## Here space="rgb" gives palettes that vary only in saturation,
## as intended.
## With space="Lab" the steps are more uniform, but the hues
## are slightly purple.
filled.contour(volcano,
             color = colorRampPalette(c("red", "white", "blue")),
             asp = 1)
filled.contour(volcano,
             color = colorRampPalette(c("red", "white", "blue"),
                                      space = "Lab"),
             asp = 1)

## Interpolating a 'sequential' ColorBrewer palette
YlOrBr <- c("#FFFFD4", "#FED98E", "#FE9929", "#D95F0E", "#993404")
filled.contour(volcano,
             color = colorRampPalette(YlOrBr, space = "Lab"),
             asp = 1)
filled.contour(volcano,
             color = colorRampPalette(YlOrBr, space = "Lab",
                                      bias = 0.5),
             asp = 1)

## space="Lab" helps when colors don't form a natural sequence
m <- outer(1:20,1:20,function(x,y) sin(sqrt(x*y)/3))
rgb.palette <- colorRampPalette(c("red", "orange", "blue"),
                                 space = "rgb")
Lab.palette <- colorRampPalette(c("red", "orange", "blue"),
                                 space = "Lab")
filled.contour(m, col = rgb.palette(20))
filled.contour(m, col = Lab.palette(20))
```
colors

Color Names

Description

Returns the built-in color names which R knows about.

Usage

colors()
colours()

details

These color names can be used with a col= specification in graphics functions. An even wider variety of colors can be created with primitives rgb and hsv or the derived rainbow, heat.colors, etc.

Value

A character vector containing all the built-in color names.

See Also

palette for setting the “palette” of colors for par(col=<num>); rgb, hsv, gray, rainbow for a nice example; and heat.colors, topo.colors for images.

col2rgb for translating to RGB numbers and extended examples.

Examples

c1 <- colors()
length(c1); c1[1:20]

contourLines

Calculate Contour Lines

Description

Calculate contour lines for a given set of data.

Usage

contourLines(x = seq(0, 1, len = nrow(z)),
y = seq(0, 1, len = ncol(z)),
z, nlevels = 10,
levels = pretty(range(z, na.rm=TRUE), nlevels))
convertColor

Arguments

\(x, y\) locations of grid lines at which the values in \(z\) are measured. These must be in ascending order. By default, equally spaced values from 0 to 1 are used. If \(x\) is a list, its components \(x_x\) and \(x_y\) are used for \(x\) and \(y\), respectively. If the list has component \(z\) this is used for \(z\).

\(z\) a matrix containing the values to be plotted (NA as are allowed). Note that \(x\) can be used instead of \(z\) for convenience.

\(n\) number of contour levels desired if \(levels\) is not supplied.

\(levels\) numeric vector of levels at which to draw contour lines.

Details

contourLines draws nothing, but returns a set of contour lines. There is currently no documentation about the algorithm. The source code is in \$R_HOME/src/main/plot3d.c\).

Value

A list of contours. Each contour is a list with elements:

- \(level\) The contour level.
- \(x\) The \(x\)-coordinates of the contour.
- \(y\) The \(y\)-coordinates of the contour.

See Also

contour.

Examples

\[
x \leftarrow 10 \cdot 1: \text{row}(\text{volcano})
y \leftarrow 10 \cdot 1: \text{col}(\text{volcano})
\text{contourLines}(x, y, \text{volcano})
\]

Description

Convert colours between standard colour space representations. This function is experimental.

Usage

convertColor(color, from, to, from.ref.white, to.ref.white, scale.in=1, scale.out=1, clip=TRUE)
Arguments

- **color**
  A matrix whose rows specify colors

- **from, to**
  Input and output color spaces. See Details below

- **from.ref.white, to.ref.white**
  Reference whites or NULL if these are built in to the definition, as for RGB spaces. D65 is the default, see Details for others

- **scale.in, scale.out**
  Input is divided by scale.in, output is multiplied by scale.out. Use NULL to suppress scaling when input or output is not numeric.

- **clip**
  If TRUE, truncate RGB output to [0,1], FALSE return out-of-range RGB, NA set out of range colors to NaN.

Details

Color spaces are specified by objects of class colorConverter, created by colorConverter or make.rgb. Built-in color spaces may be referenced by strings: "XYZ", "sRGB", "Apple RGB", "CIE RGB", "Lab", "Luv". The converters for these colour spaces are in the object colorspace.

The "sRGB" color space is that used by standard PC monitors. "Apple RGB" is used by Apple monitors. "Lab" and "Luv" are approximately perceptually uniform spaces standardized by the Commission Internationale d’Eclairage. XYZ is a 1931 CIE standard capable of representing all visible colors (and then some), but not in a perceptually uniform way.

The Lab and Luv spaces describe colors of objects, and so require the specification of a reference “white light” color. Illuminant D65 is a standard indirect daylight, Illuminant D50 is close to direct sunlight, and Illuminant A is the light from a standard incandescent bulb. Other standard CIE illuminants supported are B, C, E and D55. RGB colour spaces are defined relative to a particular reference white, and can be only approximately translated to other reference whites. The Bradford chromatic adaptation algorithm is used for this.

The RGB color spaces are specific to a particular class of display. An RGB space cannot represent all colors, and the clip option controls what is done to out-of-range colors.

Value

A 3-row matrix whose columns specify the colors.

References

For all the conversion equations [http://www.brucelindbloom.com/](http://www.brucelindbloom.com/)

For the white points [http://www.efg2.com/Lab/Graphics/Colors/Chromaticity.htm](http://www.efg2.com/Lab/Graphics/Colors/Chromaticity.htm)

See Also

col2rgb and colors for ways to specify colors in graphics.

make.rgb for specifying other colour spaces.
Examples

```r
par(mfrow=c(2,2))
## The displayable colors from four planes of Lab space
ab <- expand.grid(a=(-10:15)*10,b=(-15:10)*10)
Lab <- cbind(L=20,ab)
srgb <- convertColor(Lab,from="Lab",to="sRGB",clip=NA)
clipped <- attr(na.omit(srgb),"na.action")
vals <- rgb(srgb[,1],srgb[,2],srgb[,3])
image((-10:15)*10,(-15:10)*10,matrix(1:(26*26),ncol=26),col=vals,
xlab="a",ylab="b",main="Lab: L=20")

Lab <- cbind(L=40,ab)
srgb <- convertColor(Lab,from="Lab",to="sRGB",clip=NA)
clipped <- attr(na.omit(srgb),"na.action")
vals <- rgb(srgb[,1],srgb[,2],srgb[,3])
image((-10:15)*10,(-15:10)*10,matrix(1:(26*26),ncol=26),col=vals,
xlab="a",ylab="b",main="Lab: L=40")

Lab <- cbind(L=60,ab)
srgb <- convertColor(Lab,from="Lab",to="sRGB",clip=NA)
clipped <- attr(na.omit(srgb),"na.action")
vals <- rgb(srgb[,1],srgb[,2],srgb[,3])
image((-10:15)*10,(-15:10)*10,matrix(1:(26*26),ncol=26),col=vals,
xlab="a",ylab="b",main="Lab: L=60")

Lab <- cbind(L=80,ab)
srgb <- convertColor(Lab,from="Lab",to="sRGB",clip=NA)
clipped <- attr(na.omit(srgb),"na.action")
vals <- rgb(srgb[,1],srgb[,2],srgb[,3])
image((-10:15)*10,(-15:10)*10,matrix(1:(26*26),ncol=26),col=vals,
xlab="a",ylab="b",main="Lab: L=80")

(cols <- t(col2rgb(palette())))
(lab <- convertColor(cols,from="sRGB",to="Lab",scale.in=255))
round(convertColor(lab,from="Lab",to="sRGB",scale.out=255))
```

---

**dev.interactive**

*Test if an Interactive Graphics Device is in Use*

**Description**

Test if an interactive graphics device is in use.

**Usage**

```r
dev.interactive()
```
Details

The X11 (Unix), windows (Windows) and quartz (MacOS X) are regarded as interactive, together with GTK and gnome (used with the GNOME GUI; GTK is available in package gtkDevice and gnome is expected to be in a package gnomeDevice).

Value

`dev.interactive()` returns a logical, TRUE iff an interactive (screen) device is in use.

See Also

`Devices` for the available devices on your platform.

---

**dev.xxx Control Multiple Devices**

Description

These functions provide control over multiple graphics devices.

Only one device is the active device. This is the device in which all graphics operations occur.

Devices are associated with a name (e.g., "X11" or "postscript") and a number; the "null device" is always device 1.

`dev.off` shuts down the specified (by default the current) device. `graphics.off()` shuts down all open graphics devices.

`dev.set` makes the specified device the active device.

Usage

```r
dev.cur()
dev.list()
dev.next(which = dev.cur())
dev.prev(which = dev.cur())
dev.off(which = dev.cur())
dev.set(which = dev.next())
graphics.off()
```

Arguments

`which` An integer specifying a device number

Value

`dev.cur` returns the number and name of the active device, or 1, the null device, if none is active.

`dev.list` returns the numbers of all open devices, except device 1, the null device. This is a numeric vector with a names attribute giving the names, or NULL is there is no open device.

`dev.next` and `dev.prev` return the number and name of the next / previous device in the list of devices. The list is regarded as a circular list, and "null device" will be included only if there are no open devices.

`dev.off` returns the name and number of the new active device (after the specified device has been shut down).

`dev.set` returns the name and number of the new active device.
See Also

Devices, such as postscript, etc.
layout and its links for setting up plotting regions on the current device.

Examples

```r
## Not run:
## Unix-specific example
x11()
plot(1:10)
x11()
plot(rnorm(10))
dev.set(dev.prev())
abline(0,1)# through the 1:10 points
dev.set(dev.next())
abline(h=0, col="gray")# for the residual plot
dev.set(dev.prev())
dev.off(); dev.off()#- close the two X devices
## End(Not run)
```

dev2

---

Copy Graphics Between Multiple Devices

Description

dev.copy copies the graphics contents of the current device to the device specified by which or to a new device which has been created by the function specified by device (it is an error to specify both which and device). (If recording is off on the current device, there are no contents to copy: this will result in no plot or an empty plot.) The device copied to becomes the current device.

dev.print copies the graphics contents of the current device to a new device which has been created by the function specified by device and then shuts the new device.

dev.copy2eps is similar to dev.print but produces an EPSF output file, in portrait orientation (horizontal = FALSE)

dev.control allows the user to control the recording of graphics operations in a device. If displaylist is "inhibit" ("enable") then recording is turned off (on). It is only safe to change this at the beginning of a plot (just before or just after a new page). Initially recording is on for screen devices, and off for print devices.

Usage

```r
dev.copy(device, ..., which = dev.next())
dev.print(device = postscript, ...)
dev.copy2eps(...)
dev.control(displaylist = c("inhibit", "enable"))
```
Arguments

- **device**: A device function (e.g., `x11`, `postscript`, ...)
- **...**: Arguments to the device function above. For `dev.print`, this includes which and by default any `postscript` arguments.
- **which**: A device number specifying the device to copy to.
- **displaylist**: A character string: the only valid values are "inhibit" and "enable".

Details

For `dev.copy2eps`, width and height are taken from the current device unless otherwise specified. If just one of width and height is specified, the other is adjusted to preserve the aspect ratio of the device being copied. The default file name is `Rplot.eps`.

The default for `dev.print` is to produce and print a postscript copy. This will not work unless `options("printcmd")` is set suitably and you have a PostScript printer: see `postscript` for how to set this up. Windows users may prefer to use `dev.print(win.print)`.

`dev.print` is most useful for producing a postscript print (its default) when the following applies.

- Unless `file` is specified, the plot will be printed.
- Unless width, height and `pointsize` are specified the plot dimensions will be taken from the current device, shrunk if necessary to fit on the paper. (`pointsize` is rescaled if the plot is shrunk.)
- If `horizontal` is not specified and the plot can be printed at full size by switching its value this is done instead of shrinking the plot region.

If `dev.print` is used with a specified device (even `postscript`) it sets the width and height in the same way as `dev.copy2eps`.

Value

- `dev.copy` returns the name and number of the device which has been copied to.
- `dev.print` and `dev.copy2eps` return the name and number of the device which has been copied from.

Note

Most devices (including all screen devices) have a display list which records all of the graphics operations that occur in the device. `dev.copy` copies graphics contents by copying the display list from one device to another device. Also, automatic redrawing of graphics contents following the resizing of a device depends on the contents of the display list.

After the command `dev.control("inhibit")`, graphics operations are not recorded in the display list so that `dev.copy` and `dev.print` will not copy anything and the contents of a device will not be redrawn automatically if the device is resized.

The recording of graphics operations is relatively expensive in terms of memory so the command `dev.control("inhibit")` can be useful if memory usage is an issue.

See Also

- `dev.cur` and other `dev.xxx` functions

Examples

```r
## Not run:
x11()
plot(rnorm(10), main="Plot 1")
```
dev.copy(device=x11)
mtext("Copy 1", 3)
dev.print(width=6, height=6, horizontal=FALSE) # prints it
dev.off(dev.prev())
dev.off()
## End(Not run)

dev2bitmap  Graphics Device for Bitmap Files via GhostScript

Description

bitmap generates a graphics file. dev2bitmap copies the current graphics device to a file in a graphics format.

Usage

bitmap(file, type = "png256", height = 6, width = 6, res = 72, pointsize, ...)
dev2bitmap(file, type = "png256", height = 6, width = 6, res = 72, pointsize, ...)

Arguments

file  The output file name, with an appropriate extension.

 type  The type of bitmap. the default is "png256".

 height  The plot height, in inches.

 width  The plot width, in inches.

 res  Resolution, in dots per inch.

 pointsize  The pointsize to be used for text: defaults to something reasonable given the width and height

 ...  Other parameters passed to postscript.

Details

dev2bitmap works by copying the current device to a postscript device, and post-processing the output file using ghostscript. bitmap works in the same way using a postscript device and postprocessing the output as “printing”.

You will need a version of ghostscript (5.10 and later have been tested): the full path to the executable can be set by the environment variable R_GSCMD.

Note: despite the name of the functions they can produce PDF via `type = "pdfwrite"`, and the PDF produced is not bitmapped.

For formats which contain a single image, a file specification like `Rplots%03d.png` can be used: this is interpreted by GhostScript.

For `dev2bitmap` if just one of `width` and `height` is specified, the other is chosen to preserve aspect ratio of the device being copied.

Value

None.

Note

Use of `bitmap` will leave a temporary file (with file name starting `Rbit`). Use of the `pdf`, `bmp`, `png` and `jpeg` devices is preferable to using these functions.

See Also

`postscript`, `png` and `jpeg` and on Windows `bmp`.

`pdf` generate PDF directly.

To display an array of data, see `image`.

---

**Devices**

**List of Graphical Devices**

Description

The following graphics devices are currently available:

- **windows** The graphics driver for Windows (on screen, to printer and to Windows metafile).
- **postscript** Writes PostScript graphics commands to a file
- **pdf** Write PDF graphics commands to a file
- **pictex** Writes LaTeX/PicTeX graphics commands to a file
- **png** PNG bitmap device
- **jpeg** JPEG bitmap device
- **bmp** BMP bitmap device
- **xfig** Device for XFIG graphics file format
- **bitmap** bitmap pseudo-device via GhostScript (if available).

Details

If no device is open, using a high-level graphics function will cause a device to be opened. Which device is given by `options("device")` which is initially set as the most appropriate for each platform: a screen device for most interactive use and `postscript` otherwise. The exception is interactive use under Unix if no screen device is known to be available, when `postscript()` is used for most systems; `pdf()` for Mac OS X.
extendrange

Description

Extends a numerical range by a small percentage, i.e., fraction, on both sides.

Usage

extendrange(x, r = range(x, na.rm = TRUE), f = 0.05)

Arguments

x numeric vector; not used if r is specified.

r numeric vector of length 2; defaults to the range of x.

f number specifying the fraction by which the range should be extended.

Value

numeric vector of length 2, “r extended”, namely simply \( r + c(-f, f) \times \text{diff}(r) \).

See Also

range, pretty which can be considered a sophisticated extension of extendrange.

Examples

```r
x <- 1:5
(r <- range(x))  # 1 5
extendrange(x)   # 0.8 5.2
extendrange(x, f= 0.01) # 0.96 5.04
## Use 'r' if you have it already:
stopifnot(identical(extendrange(r=r),
                         extendrange(x)))
```
**getGraphicsEvent**

*Wait for a mouse or keyboard event from a graphics window*

**Description**

This function waits for input from a graphics window in the form of a mouse or keyboard event.

**Usage**

```r
getGraphicsEvent(prompt = "Waiting for input", 
onMouseDown = NULL, onMouseMove = NULL, onMouseUp = NULL, 
onKeybd = NULL)
```

**Arguments**

- **prompt** prompt to be displayed to the user
- **onMouseDown** a function to respond to mouse clicks
- **onMouseMove** a function to respond to mouse movement
- **onMouseUp** a function to respond to mouse button releases
- **onKeybd** a function to respond to key presses

**Details**

This function allows user input from some graphics devices (currently only the Windows screen display). When called, event handlers may be installed to respond to events involving the mouse or keyboard.

The mouse event handlers should be functions with header `function(buttons, x, y)`.

- `x` and `y` will be passed to mouse event handlers in device independent coordinates (i.e. the lower left corner of the window is (0, 0), the upper right is (1, 1)). The `buttons` argument will be a vector listing the buttons that are pressed at the time of the event, with 0 for left, 1 for middle, and 2 for right.

The keyboard event handler should be a function with header `function(key)`.

- A single element character vector will be passed to this handler, corresponding to the key press. Shift and other modifier keys will have been processed, so `shift-a` will be passed as "A".
- The following special keys may also be passed to the handler:
  - Control keys, passed as "Ctrl-A", etc.
  - Navigation keys, passed as one of "Left", "Up", "Right", "Down", "PgUp", "PgDn", "End", "Home"
  - Edit keys, passed as one of "Ins", "Del"
  - Function keys, passed as one of "F1", "F2", ...

The event handlers are standard R functions, and will be executed in an environment as though they had been called directly from `getGraphicsEvent`.

Events will be processed until

- one of the event handlers returns a non-NULL value which will be returned as the value of `getGraphicsEvent`, or
- the user interrupts the function from the console.
518

gray

Value
A non-NULL value returned from one of the event handlers.
Author(s)
Duncan Murdoch
Examples
## Not run:
mousedown <- function(buttons, x, y) {
cat("Buttons ", paste(buttons, collapse=" "), " at ", x, y, "\n")
points(x, y)
if (x > 0.85 && y > 0.85) "Done"
else NULL
}
mousemove <- function(buttons, x, y) {
points(x, y)
NULL
}
keybd <- function(key) {
cat("Key <", key, ">\n", sep = "")
}
plot(0:1, 0:1, type='n')
getGraphicsEvent("Click on upper right to quit",
onMouseDown = mousedown,
onMouseMove = mousemove,
onKeybd = keybd)
## End(Not run)

gray

Gray Level Specification

Description
Create a vector of colors from a vector of gray levels.
Usage
gray(level)
grey(level)
Arguments
level

a vector of desired gray levels between 0 and 1; zero indicates "black" and
one indicates "white".


Details

The values returned by `gray` can be used with a `col=` specification in graphics functions or in `par`.

grey is an alias for `gray`.

Value

A vector of “colors” of the same length as `level`.

See Also

`rainbow`, `hsv`, `rgb`.

Examples

```r
gray(0:8 / 8)
```

---

### Description

Create a vector of `n` gamma-corrected gray colors.

### Usage

```r
gray.colors(n, start = 0.3, end = 0.9, gamma = 2.2)
grey.colors(n, start = 0.3, end = 0.9, gamma = 2.2)
```

### Arguments

- `n` the number of gray colors (`≥ 1`) to be in the palette.
- `start` starting gray level in the palette (should be between 0 and 1 where zero indicates "black" and one indicates "white").
- `end` ending gray level in the palette.
- `gamma` the gamma correction.

### Details

The function `gray.colors` chooses a series of `n` gamma-corrected gray levels between `start` and `end`: `(start^n, ..., end^n)(1/γ)`. The returned palette contains the corresponding gray colors. This palette is used in `barplot.default`.

grey.colors is an alias for `gray.colors`.

### Value

A vector of `n` gray colors.

### See Also

`gray`, `rainbow`, `palette`.
hcl

HCL Color Specification

Description

Create a vector of colors from vectors specifying hue, chroma and luminance.

Usage

```r
hcl(h = 0, c = 35, l = 85, alpha, fixup = TRUE)
```

Arguments

- **h**: The hue of the color specified as an angle in the range [0,360]. 0 yields red, 120 yields green 240 yields blue, etc.
- **c**: The chroma of the color. The upper bound for chroma depends on hue and luminance.
- **l**: A value in the range [0,100] giving the luminance of the colour. For a given combination of hue and chroma, only a subset of this range is possible.
- **alpha**: numeric value in the range [0,1] for alpha transparency channel (0 means transparent and 1 means opaque).
- **fixup**: a logical value which indicates whether the resulting RGB values should be corrected to ensure that a real color results. If `fixup` is `FALSE` RGB components lying outside the range [0,1] will result in an NA value.

Details

This function corresponds to polar coordinates in the CIE-LUV color space. Steps of equal size in this space correspond to approximately equal perceptual changes in color. Thus, `hcl` can be thought of as a perceptually based version of `hsv`.

The function is primarily intended as a way of computing colors for filling areas in plots where area corresponds to a numerical value (pie charts, bar charts, mosaic plots, histograms, etc). Choosing colors which have equal chroma and luminance provides a way of minimising the irradiation illusion which would otherwise produce a misleading impression of how large the areas are.

The default values of chroma and luminance make it possible to generate a full range of hues and have a relatively pleasant pastel appearance.

The RGB values produced by this function correspond to the sRGB color space used on most PC computer displays. There are other packages which provide more general color space facilities.

Value

A vector of character strings which can be used as color specifications by R graphics functions.


**Note**

At present there is no guarantee that the colours rendered by R graphics devices will correspond to their sRGB description. It is planned to adopt sRGB as the standard R color description in future.

**Author(s)**

Ross Ihaka

**References**


**See Also**

hsv, rgb.

**Examples**

```r
# The Foley and Van Dam PhD Data.
csd <- matrix(c(4, 2, 4, 6, 4, 3, 1, 4, 4, 7, 7, 1,
                0, 7, 3, 2, 4, 5, 3, 2, 5, 4, 2, 2,
                3, 1, 3, 0, 4, 4, 6, 7, 1, 10, 8, 7,
                1, 5, 3, 2, 1, 5, 2, 1, 4, 1, 4, 3,
                0, 3, 0, 6, 2, 1, 5, 5), nr=4)

csp =
function(colors)
  barplot(csd, col = colors, ylim = c(0, 30),
          names = 72:85, xlab = "Year", ylab = "Students",
          legend = c("Winter", "Spring", "Summer", "Fall"),
          main = "Computer Science PhD Graduates", las = 1)

# The Original (Metaphorical) Colors (Ouch!)
csp(c("blue", "green", "yellow", "orange"))

# A Color Tetrad (Maximal Color Differences)
csp(hcl(h = c(30, 120, 210, 300)))

# Same, but lighter and less colorful
# Turn of automatic correction to make sure
# that we have defined real colors.
csp(hcl(h = c(30, 120, 210, 300),
      c = 20, l = 90, fixup = FALSE))

# Analogous Colors
# Good for those with red/green color confusion
csp(hcl(h = seq(60, 240, by = 60)))

# Metaphorical Colors
csp(hcl(h = seq(210, 60, length = 4)))

# Cool Colors
csp(hcl(h = seq(120, 0, length = 4) + 150))
```
# Warm Colors
cspdh(hcl(h = seq(120, 0, length = 4) - 30))

# Single Color
hist(rnorm(1000), col = hcl(240))

Hershey Vector Fonts in R

Description

If the family graphical parameter (see `par`) has been set to one of the Hershey fonts (see below) Hershey vector fonts are used to render text.

These fonts have two advantages:

1. Vector fonts describe each character in terms of a set of points; R renders the character by joining up the points with straight lines. This intimate knowledge of the outline of each character means that R can arbitrarily transform the characters, which can mean that the vector fonts look better for rotated and 3d text.
2. This implementation was adapted from the GNU libplot library which provides support for non-ASCII and non-English fonts. This means that it is possible, for example, to produce weird plotting symbols and Japanese characters.

For historical reasons, when using the `text` function, Hershey fonts may also be selected via the vfont argument, which is a character vector of length 2 (see below for valid values).

Drawback:
You cannot use mathematical expressions (plotmath) with Hershey fonts.

Usage

Hershey

Details

The Hershey characters are organised into a set of fonts. A particular font is selected by specifying one of the following font families via `par(family)` and specifying the desired font face (plain, bold, italic, bold-italic) via `par(font)`.

<table>
<thead>
<tr>
<th>family</th>
<th>faces available</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;HersheySerif&quot;</td>
<td>plain, bold, italic, bold-italic</td>
</tr>
<tr>
<td>&quot;HersheySans&quot;</td>
<td>plain, bold, italic, bold-italic</td>
</tr>
<tr>
<td>&quot;HersheyScript&quot;</td>
<td>plain, bold</td>
</tr>
<tr>
<td>&quot;HersheyGothicEnglish&quot;</td>
<td>plain</td>
</tr>
<tr>
<td>&quot;HersheyGothicGerman&quot;</td>
<td>plain</td>
</tr>
<tr>
<td>&quot;HersheyGothicItalian&quot;</td>
<td>plain</td>
</tr>
<tr>
<td>&quot;HersheySymbol&quot;</td>
<td>plain, bold, italic, bold-italic</td>
</tr>
<tr>
<td>&quot;HersheySansSymbol&quot;</td>
<td>plain, italic</td>
</tr>
</tbody>
</table>

In the old, vfont specification for the text function, the Hershey font is specified by a typeface (e.g., serif or sans serif) and a fontindex or “style” (e.g., plain or italic). The first
element of vfont specifies the typeface and the second element specifies the fontindex. The first
table produced by demo(Hershey) shows the character a produced by each of the different fonts.
The available typeface and fontindex values are available as list components of the variable
Hershey. The allowed pairs for (typeface, fontindex) are:

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Fontindex</th>
</tr>
</thead>
<tbody>
<tr>
<td>serif</td>
<td>plain</td>
</tr>
<tr>
<td>serif</td>
<td>italic</td>
</tr>
<tr>
<td>serif</td>
<td>bold</td>
</tr>
<tr>
<td>serif</td>
<td>bold italic</td>
</tr>
<tr>
<td>serif</td>
<td>cyrillic</td>
</tr>
<tr>
<td>serif</td>
<td>oblique cyrillic</td>
</tr>
<tr>
<td>serif</td>
<td>EUC</td>
</tr>
<tr>
<td>sans serif</td>
<td>plain</td>
</tr>
<tr>
<td>sans serif</td>
<td>italic</td>
</tr>
<tr>
<td>sans serif</td>
<td>bold</td>
</tr>
<tr>
<td>sans serif</td>
<td>bold italic</td>
</tr>
<tr>
<td>script</td>
<td>plain</td>
</tr>
<tr>
<td>script</td>
<td>italic</td>
</tr>
<tr>
<td>script</td>
<td>bold</td>
</tr>
<tr>
<td>gothic english</td>
<td>plain</td>
</tr>
<tr>
<td>gothic german</td>
<td>plain</td>
</tr>
<tr>
<td>gothic italian</td>
<td>plain</td>
</tr>
<tr>
<td>serif symbol</td>
<td>plain</td>
</tr>
<tr>
<td>serif symbol</td>
<td>italic</td>
</tr>
<tr>
<td>serif symbol</td>
<td>bold</td>
</tr>
<tr>
<td>serif symbol</td>
<td>bold italic</td>
</tr>
<tr>
<td>sans serif symbol</td>
<td>plain</td>
</tr>
<tr>
<td>sans serif symbol</td>
<td>italic</td>
</tr>
</tbody>
</table>

and the indices of these are available as Hershey$allowed.

**Escape sequences:** The string to be drawn can include escape sequences, which all begin with a
\ . When R encounters a \, rather than drawing the \, it treats the subsequent character(s) as
ea coded description of what to draw.

One useful escape sequence (in the current context) is of the form: \123. The three digits
following the \ specify an octal code for a character. For example, the octal code for p is 160
so the strings "p" and "\160" are equivalent. This is useful for producing characters when
there is not an appropriate key on your keyboard.

The other useful escape sequences all begin with \\. These are described below. Remember
that backslashes have to be doubled in R character strings, so they need to be entered with
four backslashes.

**Symbols:** an entire string of Greek symbols can be produced by selecting the Serif Symbol or Sans
Serif Symbol typeface. To allow Greek symbols to be embedded in a string which uses a non-
symbol typeface, there are a set of symbol escape sequences of the form \\ab. For example,
the escape sequence \*a produces a Greek alpha. The second table in demo(Hershey)
shows all of the symbol escape sequences and the symbols that they produce.

**ISO Latin-1:** further escape sequences of the form \\ab are provided for producing ISO Latin-1
characters. Another option is to use the appropriate octal code. The (non-ASCII) ISO Latin-1
characters are in the range 241...377. For example, \366 produces the character o with an
umlaut. The third table in demo(Hershey) shows all of the ISO Latin-1 escape sequences.
These characters can be used directly in a Latin-1 or UTF-8 locale. (In the latter, non-Latin-1
characters are replaced by a dot.)
Special Characters: a set of characters are provided which do not fall into any standard font. These can only be accessed by escape sequence. For example, \LI produces the zodiac sign for Libra, and \JU produces the astronomical sign for Jupiter. The fourth table in demo(Hershey) shows all of the special character escape sequences.

Cyrillic Characters: cyrillic characters are implemented according to the K018-R encoding. On a US keyboard, these can be produced using the Serif typeface and Cyrillic (or Oblique Cyrillic) fontindex and specifying an octal code in the range 300 to 337 for lower case characters or 340 to 377 for upper case characters. The fifth table in demo(Hershey) shows the octal codes for the available cyrillic characters.

Japanese Characters: 83 Hiragana, 86 Katakana, and 603 Kanji characters are implemented according to the EUC (Extended Unix Code) encoding. Each character is identified by a unique hexadecimal code. The Hiragana characters are in the range 0x2421 to 0x2473, Katakana are in the range 0x2521 to 0x2576, and Kanji are (scattered about) in the range 0x3021 to 0x6d55. When using the Serif typeface and EUC fontindex, these characters can be produced by a pair of octal codes. Given the hexadecimal code (e.g., 0x2421), take the first two digits and add 0x80 and do the same to the second two digits (e.g., 0x21 and 0x24 become 0xa4 and 0xa1), then convert both to octal (e.g., 0x4a and 0x1a become 244 and 241). For example, the first Hiragana character is produced by \244\241. It is also possible to use the hexadecimal code directly. This works for all non-EUC fonts by specifying an escape sequence of the form \#J1234. For example, the first Hiragana character is produced by \#J2421. The Kanji characters may be specified in a third way, using the so-called "Nelson Index", by specifying an escape sequence of the form \#N1234. For example, the Kanji for "one" is produced by \#N0001. demo(Japanese) shows the available Japanese characters.

Raw Hershey Glyphs: all of the characters in the Hershey fonts are stored in a large array. Some characters are not accessible in any of the Hershey fonts. These characters can only be accessed via an escape sequence of the form \#H1234. For example, the fleur-de-lys is produced by \#H0746. The sixth and seventh tables of demo(Hershey) shows all of the available raw glyphs.

References

http://www.gnu.org/software/plotutils/plotutils.html

See Also
demo(Hershey), par, text, contour.

Japanese for the Japanese characters in the Hershey fonts.

Examples

Hershey

## for tables of examples, see demo(Hershey)
**Description**

Create a vector of colors from vectors specifying hue, saturation and value.

**Usage**

\[
\text{hsv}(h = 1, s = 1, v = 1, \text{gamma} = 1, \text{alpha})
\]

**Arguments**

- \(h,s,v\): numeric vectors of values in the range \([0,1]\) for “hue”, “saturation” and “value” to be combined to form a vector of colors. Values in shorter arguments are recycled.
- \(\text{gamma}\): a “gamma correction” exponent, \(\gamma\)
- \(\text{alpha}\): numeric value in the range \([0,1]\) for alpha transparency channel (0 means transparent and 1 means opaque).

**Value**

This function creates a vector of “colors” corresponding to the given values in HSV space. The values returned by \text{hsv} can be used with a \text{col=} specification in graphics functions or in \text{par}.

**Gamma correction**

For each color, \((r,g,b)\) in RGB space (with all values in \([0,1]\)), the final color corresponds to \((r^\gamma,g^\gamma,b^\gamma)\).

**See Also**

\text{rainbow, rgb, gray}.

**Examples**

\[
\text{hsv(0.5,0.5,0.5)}
\]

## Look at gamma effect:
\[
n <- 20; y <- \sin(3*\pi*(1:n)-1/2)/n
\]
\[
op <- \text{par(mfrow=c(3,2),mar=rep(1.5,4))}
\]
\[
\text{for(gamma in c(0.4, 0.6, 0.8, 1, 1.2, 1.5))}
\]
\[
\text{plot(y, axes = FALSE, frame.plot = TRUE,}
\]
\[
\text{xlab = "", ylab = "", pch = 21, cex = 30,}
\]
\[
\text{bg = rainbow(n, start=0.85, end=1, gamma = gamma),}
\]
\[
\text{main = paste("Red tones; gamma=",format(gamma)))}
\]
\[
\text{par(op)}
\]
Japanese characters in R

Description

The implementation of Hershey vector fonts provides a large number of Japanese characters (Hiragana, Katakana, and Kanji).

Details

Without keyboard support for typing Japanese characters, the only way to produce these characters is to use special escape sequences: see Hershey.

For example, the Hiragana character for the sound "ka" is produced by `\"J242b` and the Katakana character for this sound is produced by `\"J252b`. The Kanji ideograph for "one" is produced by `\"J306c` or `\"N0001`.

The output from `demo(Japanese)` shows tables of the escape sequences for the available Japanese characters.

References

http://www.gnu.org/software/plotutils/plotutils.html

See Also

demo(Japanese), Hershey, text, contour

Examples

```r
plot(1:9, type="n", axes=FALSE, frame=TRUE, ylab="",
     main= "example(Japanese)", xlab= "using Hershey fonts")
p <- c("serif", "plain")
Vf <- c("serif", "plain")
text(4, 2, "\\\"J2438\\\"J2421\\\"J2451\\\"J2473", vfont = Vf)
text(4, 4, "\\\"J2538\\\"J2521\\\"J2551\\\"J2573", vfont = Vf)
text(4, 6, "\\\"J467c\\\"J4b5c", vfont = Vf)
text(4, 8, "Japan", vfont = Vf)
p <- c("serif")
text(8, 2, "Hiragana")
text(8, 4, "Katakana")
text(8, 6, "Kanji")
text(8, 8, "English")
```

Create colour spaces

Description

These functions specify colour spaces for use in `convertColor`.
Usage

make.rgb(red, green, blue, name = NULL, white = "D65", gamma = 2.2)
colorConverter(toXYZ, fromXYZ, name, white=NULL)

Arguments

red, green, blue
Chromaticity (xy or xyY) of RGB primaries

name
Name for the colour space

white
Character string specifying the reference white (see Details)

gamma
Display gamma (nonlinearity). A positive number or the string "sRGB"

fromXYZ
Function to convert from XYZ tristimulus coordinates to this space

toXYZ
Function to convert from this space to XYZ tristimulus coordinates.

Details

An RGB colour space is defined by the chromaticities of the red, green and blue primaries. These
are given as vectors of length 2 or 3 in xyY coordinates (the Y component is not used and may be
omitted). The chromaticities are defined relative to a reference white, which must be one of the CIE
standard illuminants: "A", "B", "C", "D50", "D55", "D60", "E" (usually "D65").

The display gamma is most commonly 2.2, though 1.8 is used for Apple RGB. The sRGB standard
specifies a more complicated function that is close to a gamma of 2.2; gamma="sRGB" uses this
function.

Colour spaces other than RGB can be specified directly by giving conversions to and from XYZ
tristimulus coordinates. The functions should take two arguments. The first is a vector giving the
coordinates for one colour. The second argument is the reference white. If a specific reference
white is included in the definition of the colour space (as for the RGB spaces) this second argument
should be ignored and may be ....

Value

An object of class colorConverter

References

Conversion algorithms from http://www.brucelindbloom.com

See Also

cvtColor

Examples

(pal <- make.rgb(red = c(0.6400,0.3300),
green = c(0.2900,0.6000),
blue = c(0.1500,0.0600),
name = "PAL/SECAM RGB"))

## converter for sRGB in #rrggb format
hexcolor <- colorConverter(toXYZ = function(hex,...) {
  rgb <- t(col2rgb(hex))/255
colorspaces$sRGB$toXYZ(rgb,...) },
nclass = function(xyz,...) {
  rgb <- colorspaces$sRGB$fromXYZ(xyz,...)
  rgb <- round(rgb,5)
  if (min(rgb) < 0 || max(rgb) > 1)
    as.character(NA)
  else
    rgb(rgb[1],rgb[2],rgb[3])},
white = "D65", name = "#rrggbb")

(cols <- t(col2rgb(palette())))
(luv <- convertColor(cols,from="sRGB", to="Luv", scale.in=255))
(hex <- convertColor(luv, from="Luv", to=hexcolor, scale.out=NULL))

## must make hex a matrix before using it
(cc <- round(convertColor(as.matrix(hex), from= hexcolor, to= "sRGB", scale.in=NULL, scale.out=255)))
stopifnot(cc == cols)

---

### nclass

**Compute the Number of Classes for a Histogram**

#### Description

Compute the number of classes for a histogram.

#### Usage

```r
nclass.Sturges(x)
nclass.scott(x)
nclass.FD(x)
```

#### Arguments

- `x` A data vector.

#### Details

- `nclass.Sturges` uses Sturges’ formula, implicitly basing bin sizes on the range of the data.
- `nclass.scott` uses Scott’s choice for a normal distribution based on the estimate of the standard error.
- `nclass.FD` uses the Freedman-Diaconis choice based on the inter-quartile range.

#### Value

The suggested number of classes.
palette

References


See Also

hist

palette

Set or View the Graphics Palette

Description

View or manipulate the color palette which is used when a `col=` has a numeric index.

Usage

```r
palette(value)
```

Arguments

value  
an optional character vector.

Details

If `value` has length 1, it is taken to be the name of a built in color palette. If `value` has length greater than 1 it is assumed to contain a description of the colors which are to make up the new palette (either by name or by RGB levels).

If `value` is omitted or has length 0, no change is made the current palette.

Currently, the only built-in palette is "default".

Value

The palette which was in effect. This is `invisible` unless the argument is omitted.

See Also

colors for the vector of built-in “named” colors; hsv, gray, rainbow, terrain.colors,... to construct colors;
col2rgb for translating colors to RGB 3-vectors.
Palettes

Examples

```r
palette()  # obtain the current palette
palette(rainbow(6))  # six color rainbow

(palette(gray(seq(0,.9,len=25)))) # gray scales; print old palette
matplot(outer(1:100,1:30), type='l', lty=1,lwd=2, col=1:30,
    main = "Gray Scales Palette",
    sub = "palette(gray(seq(0,.9,len=25)))")
palette("default")  # reset back to the default
```

Palettes

<table>
<thead>
<tr>
<th>Color Palettes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

Create a vector of n “contiguous” colors.

Usage

```r
rainbow(n, s = 1, v = 1, start = 0, end = max(1,n - 1)/n, gamma = 1)
heat.colors(n)
terrain.colors(n)
topo.colors(n)
cm.colors(n)
```

Arguments

- **n**: the number of colors (≥ 1) to be in the palette.
- **s, v**: the “saturation” and “value” to be used to complete the HSV color descriptions.
- **start**: the (corrected) hue in [0,1] at which the rainbow begins.
- **end**: the (corrected) hue in [0,1] at which the rainbow ends.
- **gamma**: the gamma correction, see argument gamma in `hsv`.

Details

Conceptually, all of these functions actually use (parts of) a line cut out of the 3-dimensional color space, parametrized by `hsv(h, s, v, gamma)`, where gamma = 1 for the `foo.colors` function, and hence, equispaced hues in RGB space tend to cluster at the red, green and blue primaries.

Some applications such as contouring require a palette of colors which do not “wrap around” to give a final color close to the starting one.

With `rainbow`, the parameters `start` and `end` can be used to specify particular subranges of hues. The following values can be used when generating such a subrange: red=0, yellow=$\frac{1}{6}$, green=$\frac{2}{6}$, cyan=$\frac{3}{6}$, blue=$\frac{4}{6}$ and magenta=$\frac{5}{6}$.

Value

A character vector, `cv`, of color names. This can be used either to create a user–defined color palette for subsequent graphics by `palette(cv)`, a `col=` specification in graphics functions or in `par`.
pdf

PDF Graphics Device

Description

pdf starts the graphics device driver for producing PDF graphics.

Usage

pdf(file = ifelse(onefile, "Rplots.pdf", "Rplot%03d.pdf"),
    width = 6, height = 6, onefile = TRUE, family = "Helvetica",
    title = "R Graphics Output", fonts = NULL, version = "1.1",
    paper, encoding, bg, fg, pointsize)

Arguments

file a character string giving the name of the file. For use with onefile=FALSE
give a printf format such as "Rplot%03d.pdf" (the default in that case).

width, height the width and height of the graphics region in inches.

onefile logical: if true (the default) allow multiple figures in one file. If false, generate
a file name containing the page number.
family  the font family to be used, one of "AvantGarde", "Bookman", "Courier", "Helvetica", "Helvetica-Narrow", "NewCenturySchoolbook", "Palatino" or "Times". Note the other specifications allowed for postscript are not available.

title  title string to embed in the file.
paper  the size of paper in the printer. The choices are "a4", "letter", "legal" and "executive" (and these can be capitalized). The default is "special", which means that the width and height specify the paper size. A further choice is "default"; if this is selected, the papersize is taken from the option "papersize" if that is set and to "a4" if it is unset or empty.

encoding  the name of an encoding file. Defaults to "WinAnsi.enc" in the ‘R_HOME/afm’ directory, which is used if the path does not contain a path separator. An extension ".enc" can be omitted.

pointsize  the default point size to be used.
bg  the default background color to be used.
fg  the default foreground color to be used.

fonts  a character vector specifying device-independent R graphics font family names for fonts which will be included in the PDF file.

version  a string describing the PDF version that will be used to produce output.

Details

pdf() opens the file file and the PDF commands needed to plot any graphics requested are sent to that file.

The file argument is interpreted as a C integer format as used by sprintf, with integer argument the page number. The default gives files ‘Rplot001.pdf’, ‘Rplot999.pdf’, ‘Rplot1000.pdf’, ….

The family argument can be used to specify a PDF-specific font family as the initial/default font for the device.

If a device-independent R graphics font family is specified (e.g., via par(family=) in the graphics package), the PDF device makes use of the PostScript font mappings to convert the R graphics font family to a PDF-specific font family description. R does not embed fonts in the PDF file though, so it is only possible straightforward to use mappings to the font families that are assumed to be available in a PDF viewer: "Times" or "Times New Roman", "Helvetica" or "Arial", "Courier", "Symbol", and "ZapfDingbats". Other fonts may be specified, but it is the user’s responsibility to ensure that these fonts are available on the system and third-party software, e.g., ghostscript, may be required to embed the fonts so that the PDF can be included in other documents (e.g., LaTeX).

See postscript for details of encodings, as the internal code is shared between the drivers. The native PDF encoding is given in file ‘PDFDoc.enc’.

pdf writes uncompressed PDF. It is primarily intended for producing PDF graphics for inclusion in other documents, and PDF-includers such as pdftex are usually able to handle compression.

At present the PDF is fairly simple, with each page being represented as a single stream. The R graphics model does not distinguish graphics objects at the level of the driver interface.

The version argument modifies the sort of PDF code that gets produced. At the moment this only concerns the production of transparent output. The version must be greater than 1.4 for transparent output to be produced. Specifying a lower version number may be useful if you want to produce PDF output that can be viewed on older PDF viewers.
Line widths as controlled by `par(lwd=)` are in multiples of 1/96inch. Multiples less than 1 are allowed. `pch="."` with `cex = 1` corresponds to a square of side 1/72 inch.

**Note**

Acrobat Reader does not use the fonts specified but rather emulates them from multiple-master fonts. This can be seen in imprecise centering of characters, for example the multiply and divide signs in Helvetica.

**See Also**

`postscriptFonts, Devices, postscript`

**Examples**

```r
## Not run:
## Test function for encodings
TestChars <- function(encoding="ISOLatin1", ...) {
  pdf(encoding=encoding, ...)
  par(pty="s")
  plot(c(-1,16), c(-1,16), type="n", xlab="", ylab="", xaxs="i", yaxs="i")
  title(paste("Centred chars in encoding", encoding))
  grid(17, 17, lty=1)
  for(i in c(32:255)) {
    x <- i %% 16
    y <- i %/% 16
    points(x, y, pch=i)
  }
  dev.off()
}
## there will be many warnings.
TestChars("ISOLatin2")
## doesn't view properly in US-spec Acrobat 5.05, but gs7.04 works.
## Lots of characters are not centred.
## End(Not run)
```

---

**pictex**

*A PicTeX Graphics Driver*

**Description**

This function produces graphics suitable for inclusion in TeX and LaTeX documents.

**Usage**

```r
pictex(file = "Rplots.tex", width = 5, height = 4, debug = FALSE, 
   bg = "white", fg = "black")
```
Arguments

- **file**: the file where output will appear.
- **width**: The width of the plot in inches.
- **height**: the height of the plot in inches.
- **debug**: should debugging information be printed.
- **bg**: the background color for the plot. Ignored.
- **fg**: the foreground color for the plot. Ignored.

Details

This driver does not have any font metric information, so the use of **plotmath** is not supported. Multiple plots will be placed as separate environments in the output file. Line widths are ignored except when setting the spacing of line textures. **pch="."** corresponds to a square of side 1pt. This device does not support colour (nor does the PicTeX package), and all colour settings are ignored.

Author(s)

This driver was provided by Valerio Aimale ⟨valerio@svpop.com.dist.unige.it⟩ of the Department of Internal Medicine, University of Genoa, Italy.

References


See Also

`postscript.Devices`

Examples

```r
pictex()
plot(1:11,(-5:5)^2, type='b', main="Simple Example Plot")
dev.off()
##---------------------
## Not run:
%\% LaTeX Example
\documentclass{article}
\usepackage{pictex}
\begin{document}
%...
\begin{figure}[h]
\centerline{\input{Rplots.tex}}
\caption{}
\end{figure}
%...
\end{document}
```
Mathematical Annotation in R

Description

If the `text` argument to one of the text-drawing functions (`text`, `mtext`, `axis`) in R is an expression, the argument is interpreted as a mathematical expression and the output will be formatted according to TeX-like rules. Expressions can also be used for titles, subtitles and x- and y-axis labels (but not for axis labels on `persp` plots).

Details

A mathematical expression must obey the normal rules of syntax for any R expression, but it is interpreted according to very different rules than for normal R expressions. It is possible to produce many different mathematical symbols, generate sub- or superscripts, produce fractions, etc.

The output from `demo(plotmath)` includes several tables which show the available features. In these tables, the columns of grey text show sample R expressions, and the columns of black text show the resulting output.

The available features are also described in the tables below:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x + y</td>
<td>x plus y</td>
</tr>
<tr>
<td>x - y</td>
<td>x minus y</td>
</tr>
<tr>
<td>x*y</td>
<td>juxtapose x and y</td>
</tr>
<tr>
<td>x/y</td>
<td>x forwardslash y</td>
</tr>
<tr>
<td>x %=~% y</td>
<td>x plus or minus y</td>
</tr>
<tr>
<td>x %//% y</td>
<td>x divided by y</td>
</tr>
<tr>
<td>x %*% y</td>
<td>x times y</td>
</tr>
<tr>
<td>x[i]</td>
<td>x subscript i</td>
</tr>
<tr>
<td>x^2</td>
<td>x superscript 2</td>
</tr>
<tr>
<td>paste(x, y, z)</td>
<td>juxtapose x, y, and z</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>square root of x</td>
</tr>
<tr>
<td>sqrt(x, y)</td>
<td>yth root of x</td>
</tr>
<tr>
<td>x == y</td>
<td>x equals y</td>
</tr>
<tr>
<td>x != y</td>
<td>x is not equal to y</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>x is less than y</td>
</tr>
<tr>
<td>x &lt;= y</td>
<td>x is less than or equal to y</td>
</tr>
<tr>
<td>x &gt; y</td>
<td>x is greater than y</td>
</tr>
<tr>
<td>x &gt;= y</td>
<td>x is greater than or equal to y</td>
</tr>
<tr>
<td>x %=~% y</td>
<td>x is approximately equal to y</td>
</tr>
<tr>
<td>x %=~% y</td>
<td>x and y are congruent</td>
</tr>
<tr>
<td>x %=~% y</td>
<td>x is defined as y</td>
</tr>
</tbody>
</table>
x \%prop\% y  
x is proportional to y

plain(x)  
draw x in normal font

bold(x)  
draw x in bold font

italic(x)  
draw x in italic font

bolditalic(x)  
draw x in bolditalic font

list(x, y, z)  
comma-separated list

...  
ellipsis (height varies)

cdots  
ellipsis (vertically centred)

ldots  
ellipsis (at baseline)

x \%subset\% y  
x is a proper subset of y

x \%notsubset\% y  
x is not a subset of y

x \%supset\% y  
x is a proper superset of y

x \%notsupset\% y  
x is not a superset of y

x \%in\% y  
x is an element of y

x \%notin\% y  
x is not an element of y

hat(x)  
x with a circumflex

tilde(x)  
x with a tilde

dot(x)  
x with a dot

ring(x)  
x with a ring

bar(xy)  
xy with bar

widehat(xy)  
xy with a wide circumflex

widetilde(xy)  
xy with a wide tilde

x \%<->\% y  
x double-arrow y

x \%->\% y  
x right-arrow y

x \%<\% y  
x left-arrow y

x \%up\% y  
x up-arrow y

x \%down\% y  
x down-arrow y

x \%<=>\% y  
x is equivalent to y

x \%=>\% y  
x implies y

x \%<>\% y  
y implies x

x \%dblup\% y  
x double-up-arrow y

x \%dbldown\% y  
x double-down-arrow y

alpha--omega  
Greek symbols

Alpha–Omega  
uppercase Greek symbols

infinity  
infinity symbol

partialdiff  
partial differential symbol

32*degree  
32 degrees

60*minute  
60 minutes of angle

30*second  
30 seconds of angle

displaystyle(x)  
draw x in normal size (extra spacing)

textstyle(x)  
draw x in normal size

scriptstyle(x)  
draw x in small size

scriptscriptstyle(x)  
draw x in very small size

underline(x)  
draw x underlined

x ~~ y  
put extra space between x and y

x + phantom(0) + y  
leave gap for "0", but don’t draw it

x + over(1, phantom(0))  
leave vertical gap for "0" (don’t draw)

frac(x, y)  
x over y

over(x, y)  
x over y

atop(x, y)  
x over y (no horizontal bar)

sum(x[i], i==1, n)  
sum x[i] for i equals 1 to n
prod(plain(P)(X==x), x)  product of P(X=x) for all values of x
integral(f(x) * dx, a, b)  definite integral of f(x) wrt x
union(A[i], i==1, n)  union of A[i] for i equals 1 to n
intersect(A[i], i==1, n)  intersection of A[i]
lim(f(x), x %->% 0)  limit of f(x) as x tends to 0
min(g(x), x > 0)  minimum of g(x) for x greater than 0
inf(S)  infimum of S
sup(S)  supremum of S
x^y + z  normal operator precedence
x^{y + z}  visible grouping of operands
 invisible grouping of operands
group("","list(a, b),"]")  specify left and right delimiters
bgroup("","atop(x,y),"")  use scalable delimiters
group(lceil, x, rceil)  special delimiters

References

See Also
demo(plotmath). axis, mtext, text, title, substitute quote, bquote

Examples
x <- seq(-4, 4, len = 101)
y <- cbind(sin(x), cos(x))
matplot(x, y, type = "l", xaxt = "n",
main = expression(paste(plain(sin) * phi, " and ",
plain(cos) * phi)),
ylab = expression("sin" * phi, "cos" * phi), # only 1st is taken
xlab = expression(paste("Phase Angle ", phi)),
col.main = "blue")
axis(1, at = c(-pi, -pi/2, 0, pi/2, pi),
lab = expression(-pi, -pi/2, 0, pi/2, pi))

## How to combine "math" and numeric variables :
plot(1:10, type="n", xlab="", ylab="", main = "plot math & numbers")
theta <- 1.23 ; mtext(bquote(hat(\.theta) == .(theta)))
for(i in 2:9)
  text(i,i+1, substitute(list(x_i, eta) == group("","list(x,y),")",
  list(x=i, y=i+1)))

plot(1:10, 1:10)
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y) ",
cex = .8)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
text(4, 6.4, "expression(bar(x) == sum(frac(x[i], n), i==1, n)) ",
cex = .8)
text(8, 5, expression(paste(frac(1, sigma*sqrt(2*pi)), " ",
  plain(e)^{frac(-(x-mu)^2, 2*sigma^2)}))),
cex = 1.2)
BMP, JPEG and PNG graphics devices

Description

A graphics device for BMP, JPEG or PNG format bitmap files.

Usage

```r
bmp(filename = "Rplot%03d.bmp", width = 480, height = 480,
    pointsize = 12, bg = "white", res = NA)
jpeg(filename = "Rplot%03d.jpg", width = 480, height = 480,
    pointsize = 12, quality = 75, bg = "white", res = NA)
png(filename = "Rplot%03d.png", width = 480, height = 480,
    pointsize = 12, bg = "white", res = NA)
```

Arguments

- **filename**: the name of the output file, up to 511 characters. The page number is substituted if an integer format is included in the character string. (The result must be less than 600 characters long.)
- **width**: the width of the device in pixels.
- **height**: the height of the device in pixels.
- **pointsize**: the default pointsize of plotted text, interpreted at 72 dpi, so one point is approximately one pixel.
- **bg**: the initial background colour: can be overridden by setting `par("bg")`.
- **quality**: the ‘quality’ of the JPEG image, as a percentage. Smaller values will give more compression but also more degradation of the image.
- **res**: The nominal resolution in dpi which will be recorded in the bitmap file, if a positive integer.

Details

Plots in PNG and JPEG format can easily be converted to many other bitmap formats, and both can be displayed in most modern web browsers. The PNG format is lossless and is best for line diagrams and blocks of solid colour. The JPEG format is lossy, but may be useful for image plots, for example. The BMP format is standard on Windows, and supported elsewhere.

`png` supports transparent backgrounds on 16-bit (‘High Color’) or better screens: use `bg = "transparent"`. Not all PNG viewers render files with transparency correctly.

Windows imposes limits on the size of bitmaps: these are not documented in the SDK and may depend on the version of Windows. It seems that `width` and `height` are each limited to $2^{15} - 1$ and there is a 16Mb limit on the total amount of memory in Windows 95/98/ME.

By default no resolution is recorded in the file. Readers will often assume nominal resolution of 72dpi when none is recorded. As resolutions in PNG files are recorded in pixels/metre, the dpi value will be changed slightly.

Both `bmp` and `png` will use a palette if there are less than 256 colours on the page, and record a 24-bit RGB file otherwise.
**postscript**

**Value**

A plot device is opened: nothing is returned to the R interpreter.

**Warning**

If you plot more than one page on one of these devices and do not include something like `%d` for the sequence number in `file`, the file will contain the last page plotted.

**Note**

These devices effectively plot on a hidden screen and then copy the image to the required format. This means that they have the same colour handling as the actual screen device, and work best if that is set to a 24-bit or 32-bit colour mode.

**See Also**

`Devices`, `dev.print`, `bitmap`

**Examples**

```r
## copy current plot to a (large) PNG file
## Not run: dev.print(png, file="myplot.png", width=1024, height=768)

png(file="myplot.png", bg="transparent")
plot(1:10)
rect(1, 5, 3, 7, col="white")
dev.off()

jpeg(file="myplot.jpeg")
example(rect)
dev.off()
## End(Not run)
```

---

**postscript**  
PostScript Graphics

**Description**

postscript starts the graphics device driver for producing PostScript graphics. The auxiliary function ps.options can be used to set and view (if called without arguments) default values for the arguments to postscript.

**Usage**

```r
postscript(file = ifelse(onefile, "Rplots.ps", "Rplot%03d.ps"),
          onefile = TRUE,
          paper, family, encoding, bg, fg,
          width, height, horizontal, pointsize,
          pagecentre, print.it, command,
          title = "R Graphics Output", fonts = NULL)

ps.options(paper, horizontal, width, height, family, encoding,
          ...)  
```
pointsize, bg, fg,
onefile = TRUE, print.it = FALSE, append = FALSE,
reset = FALSE, override.check = FALSE)

Arguments

file
a character string giving the name of the file. If it is "", the output is piped to
the command given by the argument command.
For use with onefile=FALSE give a printf format such as
"Rplot%03d.ps" (the default in that case).

paper
the size of paper in the printer. The choices are "a4", "letter", "legal"
and "executive" (and these can be capitalized). Also, "special" can be
used, when the width and height specify the paper size. A further choice
is "default", which is the default. If this is selected, the papersize is taken
from the option "papersize" if that is set and to "a4" if it is unset or empty.

horizontal
the orientation of the printed image, a logical. Defaults to true, that is landscape
orientation on paper sizes with width less than height.

width, height
the width and height of the graphics region in inches. The default is to use the
entire page less a 0.25 inch border on each side.

family
the font family to be used. EITHER a single character string OR a character
vector of length four or five. See the section ‘Families’.

encoding
the name of an encoding file. Defaults to "WinAnsi.enc" in the ‘R_HOME/afm’
directory, which is used if the path does not contain a path separator. An exten-
sion "enc" can be omitted.

pointsize
the default point size to be used.

bg
the default background color to be used. If "transparent" (or an equivalent
specification), no background is painted.

fg
the default foreground color to be used.

onefile
logical: if true (the default) allow multiple figures in one file. If false, gen-
erate a file name containing the page number and use an EPSF header and no
DocumentMedia comment.

pagecentre
logical: should the device region be centred on the page: defaults to true.

print.it
logical: should the file be printed when the device is closed? (This only applies
if file is a real file name.)

command
the command to be used for "printing". Defaults to option "printcmd"; this
can also be selected as "default".

append
logical; currently disregarded; just there for compatibility reasons.

reset, override.check
logical arguments passed to check.options. See the Examples.

title
a character vector specifying R graphics (device-independent) font family names
for fonts which must be included in the PostScript file.
Details

`postscript` opens the file `file` and the PostScript commands needed to plot any graphics requested are stored in that file. This file can then be printed on a suitable device to obtain hard copy.

A postscript plot can be printed via `postscript` in two ways.

1. Setting `print.it = TRUE` causes the command given in argument `command` to be called with argument "file" when the device is closed. Note that the plot file is not deleted unless `command` arranges to delete it.
2. `file=""` or `file="|cmd"` can be used to print using a pipe on systems that support `popen`. Failure to open the command will probably be reported to the terminal but not to `popen`, in which case close the device by `dev.off` immediately.

Only the first of these will work on Windows, and the default "printcmd" is empty and will give an error if `print.it=TRUE` is used. Suitable commands to spool a PostScript file to a printer can be found in ‘RedMon’ suite available from http://www.cs.wisc.edu/~ghost/index.html. The command will be run in a minimized window. GSView 4.x provides ‘gsprint.exe’ which may be even more convenient (it requires GhostScript version 6.0 or later).

The `file` argument is interpreted as a C integer format as used by `sprintf`, with integer argument the page number. The default gives files ‘Rplot001.ps’, ‘Rplot999.ps’, ‘Rplot1000.ps’, ....

The postscript produced by `R` is EPS (Encapsulated PostScript) compatible, and can be included into other documents, e.g., into LaTeX, using `\includegraphics{<filename>}`. For use in this way you will probably want to set `horizontal = FALSE`, `oneline = FALSE`, `paper = "special"`.

Most of the PostScript prologue used is taken from the `R` character vector `.ps.prolog`. This is marked in the output, and can be changed by changing that vector. (This is only advisable for PostScript experts: the standard version is in namespace:grDevices.)

`ps.options` needs to be called before calling `postscript`, and the default values it sets can be overridden by supplying arguments to `postscript`.

A PostScript device has a default font, which can be set by the user via `family`. If other fonts are to be used when drawing to the PostScript device, these must be declared when the device is created via `fonts`; the font family names for this argument are device-independent R graphics font family names (see the documentation for `postscriptFonts`).

Line widths as controlled by `par(lwd=)` are in multiples of 1/96 inch. Multiples less than 1 are allowed. `pch="."` with `cex = 1` corresponds to a square of side 1/72 inch.

Families

Font families may be specified in several ways. The `family` argument specifies an initial/default font family for the device. This is a PostScript-specific font family specification (see below). The `fonts` argument specifies a set of device-independent font families that are mapped to PostScript-specific fonts via a font database (see `postscriptFonts`).

The argument `family` specifies the initial/default font family to be used. In normal use it is one of "AvantGarde", "Bookman", "Courier", "Helvetica", "Helvetica-Narrow", "NewCenturySchoolbook", "Palatino" or "Times", and refers to the standard Adobe PostScript fonts of those names which are included (or cloned) in all common PostScript devices.

Many PostScript emulators (including those based on ghostscript) use the URW equivalents of these fonts, which are "URWGothic", "URWBookman", "NimbusMon", "NimbusSan", "NimbusSanCond", "CenturySch", "URWPalladio" and "NimbusRom" respectively.
If your PostScript device is using URW fonts, you will obtain access to more characters and more appropriate metrics by using these names. To make these easier to remember, "URWHelvetica" == "NimbusSan" and "URWTimes" == "NimbusRom" are also supported.

It is also possible to specify family = "ComputerModern". This is intended to use with the Type 1 versions of the TeX CM fonts. It will normally be possible to include such output in TeX or LaTeX provided it is processed with dvips -Ppfb -j0 or the equivalent on your system. (-j0 turns off font subsetting.)

If the second form of argument "family" is used, it should be a character vector of four or five paths to Adobe Font Metric files for the regular, bold, italic, bold italic and (optionally) symbol fonts to be used. If these paths do not contain the file separator, they are taken to refer to files in the R directory ‘R_HOME/afm’. Thus the default Helvetica family can be specified by family = c("hv______.afm", "hvb______.afm", "hvo______.afm", "hvbo______.afm", "sy______.afm"). It is the user’s responsibility to check that suitable fonts are made available, and that they contain the needed characters when re-encoded. The fontnames used are taken from the FontName fields of the afm files. The software including the PostScript plot file should either embed the font outlines (usually from ‘.pfb’ or ‘.pfa’ files) or use DSC comments to instruct the print spooler to do so.

The .afm files for the first four fonts do not need to be in the correct encoding, but that for the symbol font must be.

When family = "ComputerModern" is used, the italic/bold-italic fonts used are slanted fonts (cmsl10 and cmbxsl110). To use text italic fonts instead, use family = c("CM_regular_10.afm", "CM_boldx_10.afm", "cmti10.afm", "cmbxti10.afm", "CM_symbol_10.afm").

**Encodings**

Encodings describe which glyphs are used to display the character codes (in the range 0–255). By default R uses ISOLatin1 encoding, and the examples for text are in that encoding. However, the encoding used on machines running R may well be different, and by using the encoding argument the glyphs can be matched to encoding in use.

None of this will matter if only ASCII characters (codes 32–126) are used as all the encodings agree over that range. Some encodings are supersets of ISOLatin1, too. However, if accented and special characters do not come out as you expect, you may need to change the encoding. Three other encodings are supplied with R: "WinAnsi.enc" and "MacRoman.enc" correspond to the encodings normally used on Windows and MacOS (at least by Adobe), and "PDFDoc.enc" is the first 256 characters of the Unicode encoding, the standard for PDF.

If you change the encoding, it is your responsibility to ensure that the PostScript font contains the glyphs used. One issue here is the Euro symbol which is in the WinAnsi and MacRoman encodings but may well not be in the PostScript fonts. (It is in the URW variants; it is not in the supplied Adobe Font Metric files.)

There is one exception. Character 45 ("-") is always set as minus (its value in Adobe ISOLatin1) even though it is hyphen in the other encodings. Hyphen is available as character 173 (octal 0255) in ISOLatin1.

Computer Modern fonts are encoded rather differently and should be used with encoding = "TeXtext.enc", taking care that the symbols < > \ _ { } are not available in those fonts.

**Author(s)**

Support for Computer Modern fonts is based on a contribution by Brian D’Urso (durso@hussle.harvard.edu).
References


See Also

`postscriptFonts`, `Devices`, `check.options` which is called from both `ps.options` and `postscript`.

Examples

```r
## Not run:
# open the file "foo.ps" for graphics output
postscript("foo.ps")
# produce the desired graph(s)
dev.off() # turn off the postscript device
options(printcmd='redpr -P"\markov\lw"')
postscript(file=tempfile("Rps."), print.it=TRUE)
# produce the desired graph(s)
dev.off() # send plot file to the printer
## alternative using GSView 4.x
options(printcmd='/GhostGum/gsview/gsprint -query')

# for URW PostScript devices
postscript("foo.ps", family = "NimbusSan")

## for inclusion in Computer Modern TeX documents, perhaps
postscript("cm_test.eps", width = 4.0, height = 3.0,
           horizontal = FALSE, onefile = FALSE, paper = "special",
           family = "ComputerModern", encoding = "TeXtext.enc")
## The resultant postscript file can be used by dvips -Ppfb -j0.

## To test out encodings, you can use
TestChars <- function(encoding="ISOLatin1", family="URWHelvetica")
{
  postscript(encoding=encoding, family=family)
  par(pty="s")
  plot(c(-1,16), c(-1,16), type="n", xlab="", ylab="", xaxs="i", yaxs="i")
  title(paste("Centred chars in encoding", encoding))
  grid(17, 17, lty=1)
  for(i in c(32:255)) {
    x <- i %% 16
    y <- i %/% 16
    points(x, y, pch=i)
  }
  dev.off()
}
## there will be many warnings. We use URW to get a complete enough
## set of font metrics.
TestChars()
TestChars("ISOLatin2")
TestChars("WinAnsi")
## End(Not run)

ps.options(bg = "pink")
```
### ---- error checking of arguments: ----

```r
ps.options(width=0:12, onefile=0, bg=pi)
# override the check for 'onefile', but not the others:
utils::str(ps.options(width=0:12, onefile=1, bg=pi,
    override.check = c(FALSE,TRUE,FALSE)))
```

---

**postscriptFonts**

*PostScript Fonts*

### Description

These functions handle the translation of a device-independent R graphics font family name to a PostScript font description.

### Usage

```r
postscriptFont(family, metrics, encoding = "default")
postscriptFonts(...)
```

### Arguments

- **family**: a character string giving the name of an Adobe Type 1 font family.
- **metrics**: a vector of four or five strings giving paths to the afm (font metric) files for the Type 1 font.
- **encoding**: the name of an encoding file. Defaults to "WinAnsi.enc" in the `R_HOME/afm` directory, which is used if the path does not contain a path separator. An extension ".enc" can be omitted.
- **...**: either character strings naming mappings to display, or new (named) mappings to define.

### Details

A PostScript device is created with a default font (see the documentation for `postscript`), but it is also possible to specify a font family when drawing to the device (for example, see the documentation for `gpar` in the grid package).

The font family sent to the device is a simple string name, which must be mapped to something more specific to PostScript fonts. A list of mappings is maintained and can be modified by the user. The `postscriptFonts` function can be used to list existing translations and to define new mappings. The `postscriptFont` function can be used to create a new mapping.

The argument `family` specifies the font family to be used. Default mappings are provided for four device-independent family names: "sans" for a sans-serif font, "serif" for a serif font, "mono" for a monospaced font, and "symbol" for a symbol font.

Mappings for a number of standard Adobe fonts (and URW equivalents) are also provided: "AvantGarde", "Bookman", "Courier", "Helvetica", "Helvetica-Narrow", "NewCenturySchoolbook", "Palatino" and "Times": "URWGothic", "URWBookman", "NimbusMon", "NimbusSan", "NimbusSanCond", "CenturySch", "URWPalladio" and "NimbusRom".
There is also a mapping for "ComputerModern".
The specification of font metrics and encodings is described in the help for the postscript function.
The fonts are not embedded in the resulting PostScript file, so software including the PostScript plot file should either embed the font outlines (usually from `.pfb` or `.pfa` files) or use DSC comments to instruct the print spooler to do so.

Value

postscriptFont returns a PostScript font description.
postscriptFonts returns one or more font mappings.

Author(s)

Support for Computer Modern fonts is based on a contribution by Brian D’Urso (durso@hussle.harvard.edu).

See Also

postscript

Examples

postscriptFonts()
CMitalic <- postscriptFont("ComputerModern",
c("CM_regular_10.afm", "CM_boldx_10.afm",
"cmti10.afm", "cmbxti10.afm",
"CM_symbol_10.afm"))
postscriptFonts(CMitalic=CMitalic)

recordGraphics  Record graphics operations

Description

Records arbitrary code on the graphics engine display list. Useful for encapsulating calculations with graphical output that depends on the calculations. Intended only for expert use.

Usage

recordGraphics(expr, list, env)

Arguments

expr  object of mode expression or call or an “unevaluated expression”.
list  a list defining the environment in which expr is to be evaluated.
env   An environment specifying where R looks for objects not found in envir.
recordPlot

Details

The code in `expr` is evaluated in an environment constructed from `list`, with `env` as the parent of that environment.

All three arguments are saved on the graphics engine display list so that on a device resize or copying between devices, the original evaluation environment can be recreated and the code can be reevaluated to reproduce the graphical output.

Value

The value from evaluating `expr`.

Warning

This function is not intended for general use. Incorrect or improper use of this function could lead to unintended and/or undesirable results.

An example of acceptable use is querying the current state of a graphics device or graphics system setting and then calling a graphics function.

An example of improper use would be calling the `assign` function to performing assignments in the global environment.

See Also

`eval`

Examples

```r
plot(1:10)
# This rectangle remains 1inch wide when the device is resized
recordGraphics(
  {
    rect(4, 2,
        4 + diff(par("usr")[1:2])/par("pin")[1], 3)
  },
  list(),
  getNamespace("graphics"))
```

---

**recordPlot**  
*Record and Replay Plots*

Description

Functions to save the current plot in an \texttt{R} variable, and to replay it.

Usage

```r
recordPlot()
replayPlot(x)
```

Arguments

- \texttt{x}  
  A saved plot.
**Details**

These functions record and replay the displaylist of the current graphics device. The returned object is of class "recordedplot", and replayPlot acts as a print method for that class.

**Value**

- `recordPlot` returns an object of class "recordedplot".
- `replayPlot` has no return value.

**WARNING**

The format of recorded plots may change between R versions. Recorded plots should **not** be used as a permanent storage format for R plots.

R will always attempt to replay a recorded plot, but if the plot was recorded with a different R version then bad things may happen.

---

**rgb**

**RGB Color Specification**

**Description**

This function creates “colors” corresponding to the given intensities (between 0 and max) of the red, green and blue primaries.

An alpha transparency value can also be specified (0 means fully transparent and max means opaque). If `alpha` is not specified, an opaque colour is generated.

The names argument may be used to provide names for the colors.

The values returned by these functions can be used with a `col=` specification in graphics functions or in `par`.

**Usage**

```r
gb(red, green, blue, alpha, names = NULL, maxColorValue = 1)
```

**Arguments**

- `red, blue, green, alpha` vectors of same length with values in \([0, M]\) where \(M = \text{maxColorValue}\). When this is 255, the `red, blue, green, and alpha` values are coerced to integers in `0:255` and the result is computed most efficiently.

- `names` character. The names for the resulting vector.

- `maxColorValue` number giving the maximum of the color values range, see above.

**See Also**

- `col2rgb` the “inverse” for translating R colors to RGB vectors; `rainbow, hsv, gray`. 
Examples

```r
go <- 1, 0, seq(0, 1, length=11))
stopifnot(rgb(u01, u01, u01) == gray(u01))
reds <- rgb((0:15)/15, g=0, b=0, names=paste("red", 0:15, sep=".")
reds
rgb(0, 0:12, 0, max = 255)# integer input
```

rgb2hsv

**RGB to HSV Conversion**

**Description**

`rgb2hsv` transforms colors from RGB space (red/green/blue) into HSV space (hue/saturation/value).

**Usage**

```r
rgb2hsv(r, g = NULL, b = NULL, gamma = 1, maxColorValue = 255)
```

**Arguments**

- `r` vector of "red" values in \([0, M]\), \(M = \text{maxColorValue}\) or 3-row rgb matrix.
- `g` vector of "green" values, or `NULL` when `r` is a matrix.
- `b` vector of "blue" values, or `NULL` when `r` is a matrix.
- `gamma` a "gamma correction" (supposedly applied to the r,g,b values previously), see `hsv(\ldots, \text{gamma})`.
- `maxColorValue` number giving the maximum of the RGB color values range. The default 255 corresponds to the typical 0:255 RGB coding as in `col2rgb()`.

**Details**

Value (brightness) gives the amount of light in the color.  
Hue describes the dominant waveleth.  
Saturation is the amount of Hue mixed into the color.

**Value**

A matrix with a column for each color. The three rows of the matrix indicate hue, saturation and value and are named "h", "s", and "v" accordingly.

**Author(s)**

R interface by Wolfram Fischer (wolfram@fischer-zim.ch);  
C code mainly by Nicholas Lewin-Koh (nikko@hailmail.net).

**See Also**

`hsv, col2rgb, rgb`
Examples

```r
## These (saturated, bright ones) only differ by hue
(rc <- col2rgb(c("red", "yellow", "green", "cyan", "blue", "magenta")))
(hc <- rgb2hsv(rc))
6 * hc["h",] # the hues are equispaced
```

```r
(rgb3 <- floor(256 * matrix(runif(3*12), 3,12)))
(hsv3 <- rgb2hsv(rgb3))
```

## Consistency :
```r
stopifnot(rgb3 == col2rgb(hsv(h=hsv3[1,], s=hsv3[2,], v=hsv3[3,])),
  all.equal(hsv3, rgb2hsv(rgb3/255, maxC = 1)))
```

## A (simplified) pure R version -- originally by Wolfram Fischer --
## showing the exact algorithm:
```r
rgb2hsvR <- function(rgb, gamma = 1, maxColorValue = 255)
{
  if(!is.numeric(rgb)) stop("rgb matrix must be numeric")
  d <- dim(rgb)
  if(d[1] != 3) stop("rgb matrix must have 3 rows")
  n <- d[2]
  if(n == 0) return(cbind(c(h=1,s=1,v=1))[,0])
  rgb <- rgb/maxColorValue
  if(gamma != 1) rgb <- rgb ^ (1/gamma)

  ## get the max and min
  v <- apply( rgb, 2, max)
  s <- apply( rgb, 2, min)
  D <- v - s # range

  ## set hue to zero for undefined values (gray has no hue)
  h <- numeric(n)
  notgray <- ( s != v )

  ## blue hue
  idx <- (v == rgb[3,] & notgray )
  if (any (idx))
    h[idx] <- 2/3 + 1/6 * (rgb[1,idx] - rgb[2,idx]) / D[idx]

  ## green hue
  idx <- (v == rgb[2,] & notgray )
  if (any (idx))
    h[idx] <- 1/3 + 1/6 * (rgb[3,idx] - rgb[1,idx]) / D[idx]

  ## red hue
  idx <- (v == rgb[1,] & notgray )
  if (any (idx))

  ## correct for negative red
  idx <- (h < 0)
  h[idx] <- 1+h[idx]

  ## set the saturation
  s[! notgray] <- 0;
  s[notgray] <- 1 - s[notgray] / v[notgray]

  rbind( h=h, s=s, v=v )
}
```
## confirm the equivalence:
all.equal(rgb2hsv (rgb3),
          rgb2hsvR(rgb3), tol=1e-14) # TRUE

---

**savePlot**

*Save Windows Plot to a File*

**Description**

Saves the current plot on a windows device to a file.

**Usage**

```r
type = c("wmf", "emf", "png", "jpeg", "jpg", "bmp", "ps", "eps", "pdf"),
device = dev.cur())
```

**Arguments**

- `filename`: The filename under which to save the plot, without the extension.
- `type`: The type of plot, Windows metafile, PNG, JPEG, BMP (Windows bitmap format), PostScript or PDF.
- `device`: A device number of a windows device, by default the current device.

**Details**

This is equivalent to selecting the ‘Save as’ menu item on the ‘File’ menu of a windows device.

Using `filename` as "clipboard" or "" with `type = "wmf"` will copy to the clipboard.

Types "eps" and "ps" are the same thing apart from the extension. Similarly "wmf"/"emf" and "jpeg"/"jpg".

**Value**

None, but a plot file will be created.

**See Also**

- `windows`, `dev.print`
trans3d  

3D to 2D Transformation for Perspective Plots

Description

Projection of 3-dimensional to 2-dimensional points using a 4x4 viewing transformation matrix. Mainly for adding to perspective plots such as `persp`.

Usage

```
trans3d(x, y, z, pmat)
```

Arguments

- `x, y, z` numeric vectors of equal length, specifying points in 3D space.
- `pmat` a 4x4 viewing transformation matrix, suitable for projecting the 3D coordinates \((x, y, z)\) into the 2D plane using homogenous 4D coordinates \((x, y, z, \ell)\); such matrices are returned by `persp`.

Value

a list with two components

- `x, y` the projected 2d coordinates of the 3d input \((x, y, z)\).

See Also

`persp`

Examples

```r
## See help(persp) (after attaching the 'graphics' package)
##
## windows
```

windows

Windows graphics devices

Description

A graphics device is opened. For `windows`, `win.graph`, `x11` and `X11` this is a graphics window on the current Windows display: the multiple names are for compatibility with other systems. `win.metafile` prints to a file and `win.print` to the Windows print system.
Usage

windows(width = 7, height = 7, pointsize = 12,
       record = getOption("graphics.record"),
       rescale = c("R", "fit", "fixed"), xpinch, ypinch,
       bg = "transparent", canvas = "white",
       gamma = getOption("gamma"), xpos = NA, ypos = NA,
       buffered = getOption("windowsBuffered"))

win.graph(width = 7, height = 7, pointsize = 12)
x11(width = 7, height = 7, pointsize = 12)
X11(width = 7, height = 7, pointsize = 12)

win.metafile(filename = "", width = 7, height = 7, pointsize = 12)
win.print(width = 7, height = 7, pointsize = 12, printer = "")

Arguments

width the (nominal) width of the plotting window in inches.
height the (nominal) height of the plotting window in inches.
pointsize the default pointsize of plotted text. The allowed range is [6, 48]: values outside that range are reset to 12.
record logical: sets the initial state of the flag for recording plots.
rescale controls the action for resizing plots.
xpinch, ypinch double. Pixels per inch, horizontally and vertically.
bg color. The default background color.
canvas color. The color of the canvas which is visible when the background color is transparent.
gamma the gamma correction factor. This value is used to ensure that the colors displayed are linearly related to RGB values. By default this is taken from options("gamma"), or is 1 (no correction) if that is unset. It sets par("gamma") for the device.
xpos, ypos integer. Position of the top left of the window, in pixels. Negative values are taken from the opposite edge of the monitor. Missing values meant take the default from the ‘Rconsole’ file, which in turn defaults to xpos=-25, ypos=0: this puts the right edge of the window 25 pixels from the right edge of the monitor.
buffered logical. Should the screen output be double-buffered?
filename the name of the output file: it will be an enhanced Windows metafile, usually given extension .emf or .wmf. Up to 511 characters are allowed. The page number is substituted if an integer format is included in the character string. (The result must be less than 600 characters long.) The default, "", means the clipboard.
printer The name of a printer as known to Windows. The default causes a dialog box to come for the user to choose a printer.
Details

All these devices are implemented as windows devices, the display parameter selects which is actually used.

The size of a window is computed from information provided about the display: it depends on the system being configured accurately. By default a screen device asks Windows for the number of pixels per inch. This can be overridden (it is often wrong) by specifying xpinch and ypinch or the corresponding options "xpinch" and "ypinch".

A graphics window is not allowed to be specified at more than 85% of the screen width or height: the width and height are rescaled proportionally. The window can be resized to a larger size. In MDI mode, the limit is 85% of the MDI client region.

If the filename is omitted for a win.metafile device, the output is copied to the clipboard when the device is closed.

If a screen device is re-sized, the default behaviour is to redraw the plot(s) as if the new size had been specified originally. Using "fit" will rescale the existing plot(s) to fit the new device region, preserving the aspect ratio. Using "fixed" will leave the plot size unchanged, adding scrollbars if part of the plot is obscured.

A graphics window will never be created at more than 85% of the screen width or height, but can be resized to a larger size. For the first two rescale options the width and height are rescaled proportionally if necessary, and if rescale = "fit" the plot(s) are rescaled accordingly. If rescale = "fixed" the initially displayed portion is selected within these constraints, separately for width and height.

Using strwidth or strheight after a window has been rescaled (when using "fit") gives dimensions in the original units, but only approximately as they are derived from the metrics of the rescaled fonts (which are in integer sizes)

The displayed region may be bigger than the ‘paper’ size, and areas outside the ‘paper’ are coloured in the Windows application background colour. Graphics parameters such as "din" refer to the scaled plot if rescaling is in effect.

The different colours need to be distinguished carefully. The area outside the device region is coloured in the Windows application background colour. The device region is coloured in the canvas colour. This is over-painted by the background colour of a plot when a new page is called for, but that background colour can be transparent (and is by default). The difference between setting the canvas colour and the background colour is that when the device is copied the background colour is copied but the canvas colour is not.

Recorded plot histories are of class "SavedPlots". They have a print method, and a subset method. As the individual plots are of class "recordedplot" they can be replayed by printing them: see recordPlot. The active plot history is stored in variable .SavedPlots in package base.

When a screen device is double-buffered (the default) the screen is updated 100ms after last plotting call or every 500ms during continuous plotting. These times can be altered by setting options("windowsTimeout") to a vector of two integers before opening the device.

The font used for text drawn in a Windows device may be controlled in two ways. The file RHOME\etc\Rdevga can be used to specify mappings for par(font=) (in the graphics package). Alternatively, a device-independent R graphics font family can be specified (e.g., via par(family=) in the graphics package) and this will be mapped via the Windows font database (see windowsFonts).

Line widths as controlled by par(lwd=) are in multiples of 1/96inch. Multiples less than 1 are allowed, down to one pixel width.

pch="." with cex = 1 corresponds to a rectangle of sides the larger of one pixel and 0.01 inch.
For `win.metafile` only one plot is allowed per file, and Windows seems to disallow reusing the file. So the only way to allow multiple plots is to use a parametrized filename as in the example.

Value

A plot device is opened: nothing is returned to the R interpreter.

See Also

windowsFonts, savePlot, bringToTop, Devices, postscript

Examples

```r
## Not run:
## A series of plots written to a sequence of metafiles
win.metafile("8plot%02d.wmf", pointsize = 10)
## End(Not run)
```

windowsFonts Windows Fonts

Description

These functions handle the translation of a device-independent R graphics font family name to a windows font description.

Usage

```r
windowsFont(family)
windowsFonts(...)```

Arguments

- **family** a character vector containing the font family name ("TT" as the first two characters indicates a TrueType font).
- **...** either character strings naming mappings to display, or new (named) mappings to define.

Details

A windows device is created with a default font (see the documentation for windows), but it is also possible to specify a font family when drawing to the device (for example, see the documentation for gpar in the grid package).

The font family sent to the device is a simple string name, which must be mapped to something more specific to windows fonts. A list of mappings is maintained and can be modified by the user.

The `windowsFonts` function can be used to list existing mappings and to define new mappings. The `windowsFont` function can be used to create a new mapping.

Default mappings are provided for four device-independent font family names: "sabs" for a sans-serif font, "serif" for a serif font, "mono" for a monospaced font, and "symbol" for a symbol font.

These mappings will only be used if the current font face is 1 (plain), 2 (bold), 3 (italic), or 4 (bolditalic).
xfig

See Also
windows

Examples

windowsFonts()
windowsFonts("mono")

xfig

XFig Graphics Device

Description

xfig starts the graphics device driver for producing XFig (version 3.2) graphics.

The auxiliary function ps.options can be used to set and view (if called without arguments) default values for the arguments to xfig and postscript.

Usage

xfig(file = ifelse(onefile, "Rplots.fig", "Rplot%03d.fig"),
    onefile = FALSE, ...)

Arguments

file a character string giving the name of the file. For use with onefile = FALSE give a printf format such as "Rplot%d.fig" (the default in that case).

onefile logical: if true allow multiple figures in one file. If false, assume only one page per file and generate a file number containing the page number.

... further arguments to ps.options accepted by xfig():

paper the size of paper in the printer. The choices are "A4", "Letter" and "Legal" (and these can be lowercase). A further choice is "default", which is the default. If this is selected, the papersize is taken from the option "papersize" if that is set and to "A4" if it is unset or empty.

horizontal the orientation of the printed image, a logical.Defaults to true, that is landscape orientation.

width, height the width and height of the graphics region in inches. The default is to use the entire page less a 0.25 inch border.

family the font family to be used. This must be one of "AvantGarde", "Bookman", "Courier", "Helvetica", "Helvetica-Narrow", "NewCenturySchoolbook", "Palatino" or "Times".

pointsize the default point size to be used.

bg the default background color to be used.

fg the default foreground color to be used.

pagecentre logical: should the device region be centred on the page: defaults to TRUE.
Details

Although xfig can produce multiple plots in one file, the XFig format does not say how to separate or view them. So onefile = FALSE is the default.

The file argument is interpreted as a C integer format as used by sprintf, with integer argument the page number. The default gives files ‘Rplot001.fig’, ‘Rplot999.fig’, ‘Rplot1000.fig’, ....

Line widths as controlled by par(lwd=) are in multiples of 5/6*1/72 inch. Multiples less than 1 are allowed. pch="." with cex = 1 corresponds to a square of side 1/72 inch.

Note

Only some line textures (0 <= lty < 4) are used. Eventually this will be partially remedied, but the XFig file format does not allow as general line textures as the R model. Unimplemented line textures are displayed as dash-double-dotted.

There is a limit of 512 colours (plus white and black) per file.

See Also

Devices.postscript.ps.options.

xy.coords

Extracting Plotting Structures

Description

xy.coords is used by many functions to obtain x and y coordinates for plotting. The use of this common mechanism across all R functions produces a measure of consistency.

Usage

xy.coords(x, y, xlab = NULL, ylab = NULL, log = NULL, recycle = FALSE)

Arguments

x, y the x and y coordinates of a set of points. Alternatively, a single argument x can be provided.

xlab, ylab names for the x and y variables to be extracted.

log character, "x", "y" or both, as for plot. Sets negative values to NA and gives a warning.

recycle logical; if TRUE, recycle (rep) the shorter of x or y if their lengths differ.

Details

An attempt is made to interpret the arguments x and y in a way suitable for plotting.

If y is missing and x is a

formula: of the form yvar ~ xvar. xvar and yvar are used as x and y variables.

list: containing components x and y, these are used to define plotting coordinates.

time series: the x values are taken to be time(x) and the y values to be the time series.
matrix with two columns: the first is assumed to contain the x values and the second the y values.

In any other case, the x argument is coerced to a vector and returned as y component where the resulting x is just the index vector 1:n. In this case, the resulting xlab component is set to "Index".

If x (after transformation as above) inherits from class "POSIXt" it is coerced to class "POSIXct".

Value

A list with the components

x numeric (i.e., "double") vector of abscissa values.
y numeric vector of the same length as x.
xlab character(1) or NULL, the 'label' of x.
ylab character(1) or NULL, the 'label' of y.

See Also

plot.default, lines, points and lowess are examples of functions which use this mechanism.

Examples

xy.coords(stats::fft(c(1:10)), NULL)

with(cars, xy.coords(dist ~ speed, NULL)$xlab ) # = "speed"

xy.coords(1:3, 1:2, recycle=TRUE)
xy.coords(-2:10,NULL, log="y")
##> warning: 3 y values <=0 omitted ..

xyz.coords

Extracting Plotting Structures

Description

Utility for obtaining consistent x, y and z coordinates and labels for three dimensional (3D) plots.

Usage

xyz.coords(x, y, z, xlab = NULL, ylab = NULL, zlab = NULL, log = NULL, recycle = FALSE)
xyz.coords

Arguments

x, y, z  
the x, y and z coordinates of a set of points. Alternatively, a single argument x  
can be provided. In this case, an attempt is made to interpret the argument in a  
way suitable for plotting.

If the argument is a formula zvar ~ xvar + yvar, xvar, yvar and zvar  
are used as x, y and z variables; if the argument is a list containing components  
x, y and z, these are assumed to define plotting coordinates; if the argument is  
a matrix with three columns, the first is assumed to contain the x values, etc.
Alternatively, two arguments x and y can be provided. One may be real, the  
other complex; in any other case, the arguments are coerced to vectors and the  
values plotted against their indices.

xlab, ylab, zlab  
names for the x, y and z variables to be extracted.

log  
character, "x", "y", "z" or combinations. Sets negative values to NA and gives  
a warning.

recycle  
logical; if TRUE, recycle (rep) the shorter ones of x, y or z if their lengths  
differ.

Value

A list with the components

x  numeric (i.e., double) vector of abscissa values.

y  numeric vector of the same length as x.

z  numeric vector of the same length as x.

xlab  character(1) or NULL, the axis label of x.

ylab  character(1) or NULL, the axis label of y.

zlab  character(1) or NULL, the axis label of z.

Author(s)

Uwe Ligges and Martin Maechler

See Also

xyz.coords for 2D.

Examples

xyz.coords(data.frame(10*1:9, -4), y = NULL, z = NULL)

xyz.coords(1:6, stats::fft(1:6), z = NULL, xlab = "X", ylab = "Y")

y <- 2 * (x2 <- 10 + (x1 <- 1:10))

xyz.coords(y ~ x1 + x2, y = NULL, z = NULL)

xyz.coords(data.frame(x = -1:9, y = 2:12, z = 3:13), y = NULL, z = NULL,

log = "xy")

##> Warning message: 2 x values <= 0 omitted ...
Chapter 4

The graphics package

abline  Add a Straight Line to a Plot

Description

This function adds one or more straight lines through the current plot.

Usage

abline(a, b, untf = FALSE, ...)  
abline(h=, untf = FALSE, ...)  
abline(v=, untf = FALSE, ...)  
abline(coef=, untf = FALSE, ...)  
abline(reg=, untf = FALSE, ...)

Arguments

a,b  the intercept and slope.  
untf  logical asking to untransform. See Details.  
h  the y-value for a horizontal line.  
v  the x-value for a vertical line.  
coef  a vector of length two giving the intercept and slope.  
reg  an object with a coef component. See Details.  
...  graphical parameters.

Details

The first form specifies the line in intercept/slope form (alternatively a can be specified on its own and is taken to contain the slope and intercept in vector form).

The h= and v= forms draw horizontal and vertical lines at the specified coordinates.

The coef form specifies the line by a vector containing the slope and intercept.

reg is a regression object which contains reg$coef. If it is of length 1 then the value is taken to be the slope of a line through the origin, otherwise, the first 2 values are taken to be the intercept and slope.

559
If `untf` is true, and one or both axes are log-transformed, then a curve is drawn corresponding to a line in original coordinates, otherwise a line is drawn in the transformed coordinate system. The `h` and `v` parameters always refer to original coordinates.

The graphical parameters `col` and `lty` can be specified as arguments to `abline`; see `par` for details.

References


See Also

`lines` and `segments` for connected and arbitrary lines given by their endpoints. `par`.

Examples

```r
z <- lm(dist ~ speed, data = cars)
plot(cars)
abline(z)
```

---

**arrows** Add Arrows to a Plot

Description

Draw arrows between pairs of points.

Usage

```r
arrows(x0, y0, x1, y1, length = 0.25, angle = 30, code = 2,
col = par("fg"), lty = NULL, lwd = par("lwd"), xpd = NULL)
```

Arguments

- `x0, y0` coordinates of points from which to draw.
- `x1, y1` coordinates of points to which to draw.
- `length` length of the edges of the arrow head (in inches).
- `angle` angle from the shaft of the arrow to the edge of the arrow head.
- `code` integer code, determining kind of arrows to be drawn.
- `col, lty, lwd, xpd` usual graphical parameters as in `par`.
assocplot

Details

For each \( i \), an arrow is drawn between the point \((x_0[i], y_0[i])\) and the point \((x_1[i], y_1[i])\).

If \( \text{code}=1 \) an arrowhead is drawn at \((x_0[i], y_0[i])\) and if \( \text{code}=2 \) an arrowhead is drawn at \((x_1[i], y_1[i])\). If \( \text{code}=3 \) a head is drawn at both ends of the arrow. Unless \( \text{length} = 0 \), when no head is drawn.

The graphical parameters \( \text{col} \) and \( \text{lty} \) can be used to specify a color and line texture for the line segments which make up the arrows (\( \text{col} \) may be a vector).

The direction of a zero-length arrow is indeterminate, and hence so is the direction of the arrowheads. To allow for rounding error, arrowheads are omitted (with a warning) on any arrow of length less than 1/1000 inch.

Note

The first four arguments in the comparable S function are named \( x_1, y_1, x_2, y_2 \).

References


See Also

segments to draw segments.

Examples

```r
x <- runif(12); y <- rnorm(12)
i <- order(x, y); x <- x[i]; y <- y[i]
plot(x, y, main="arrows(\ldots) and segments(\ldots)")
## draw arrows from point to point :
s <- seq(length(x)-1)# one shorter than data
arrows(x[s], y[s], x[s+1], y[s+1], col= 1:3)
s <- s[-length(s)]
segments(x[s], y[s], x[s+2], y[s+2], col='pink')
```

assocplot

Association Plots

Description

Produce a Cohen-Friendly association plot indicating deviations from independence of rows and columns in a 2-dimensional contingency table.

Usage

```r
assocplot(x, col = c("black", "red"), space = 0.3,
         main = NULL, xlab = NULL, ylab = NULL)
```
Arguments

- **x**: a two-dimensional contingency table in matrix form.
- **col**: a character vector of length two giving the colors used for drawing positive and negative Pearson residuals, respectively.
- **space**: the amount of space (as a fraction of the average rectangle width and height) left between each rectangle.
- **main**: overall title for the plot.
- **xlab**: a label for the x axis. Defaults to the name of the row variable in x if non-NULL.
- **ylab**: a label for the y axis. Defaults to the column names of the column variable in x if non-NULL.

Details

For a two-way contingency table, the signed contribution to Pearson’s \( \chi^2 \) for cell \( i,j \) is \( d_{ij} = \frac{(f_{ij} - e_{ij})}{\sqrt{e_{ij}}} \), where \( f_{ij} \) and \( e_{ij} \) are the observed and expected counts corresponding to the cell. In the Cohen-Friendly association plot, each cell is represented by a rectangle that has (signed) height proportional to \( d_{ij} \) and width proportional to \( \sqrt{e_{ij}} \), so that the area of the box is proportional to the difference in observed and expected frequencies. The rectangles in each row are positioned relative to a baseline indicating independence (\( d_{ij} = 0 \)). If the observed frequency of a cell is greater than the expected one, the box rises above the baseline and is shaded in the color specified by the first element of **col**, which defaults to black; otherwise, the box falls below the baseline and is shaded in the color specified by the second element of **col**, which defaults to red.

References


See Also

- `mosaicplot`, `chisq.test`.

Examples

```r
## Aggregate over sex:
x <- margin.table(HairEyeColor, c(1, 2))
x
assocplot(x, main = "Relation between hair and eye color")
```

Description

**axis**

Add an Axis to a Plot

Adds an axis to the current plot, allowing the specification of the side, position, labels, and other options.
**Usage**

```r
axis(side, at = NULL, labels = TRUE, tick = TRUE, line = NA,
     pos = NA, outer = FALSE, font = NA, vfont = NULL,
     lty = "solid", lwd = 1, col = NULL, hadj = NA, padj = NA,
     ...)```

**Arguments**

- `side`: an integer specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.
- `at`: the points at which tick-marks are to be drawn. Non-finite (infinite, NaN or NA) values are omitted. By default, when `NULL`, tickmark locations are computed, see Details below.
- `labels`: this can either be a logical value specifying whether (numerical) annotations are to be made at the tickmarks, or a vector of character strings or expressions to be placed at the tickpoints. If this is specified as strings or expressions, `at` should be supplied and they should be the same length.
- `tick`: a logical value specifying whether tickmarks should be drawn.
- `line`: the number of lines into the margin which the axis will be drawn. This overrides the value of the graphical parameter `mgp[3]`. The relative placing of tickmarks and tick labels is unchanged.
- `pos`: the coordinate at which the axis line is to be drawn: this overrides the values of both `line` and `mgp[3]`.
- `outer`: a logical value indicating whether the axis should be drawn in the outer plot margin, rather than the standard plot margin.
- `font`: font for text.
- `vfont`: vector font for text.
- `lty, lwd`: line type, width for the axis line and the tick marks.
- `col`: color for the axis line and the tick marks. The default `NULL` means to use `par("fg")`.
- `hadj`: adjustment (see `par("adj")`) for all labels parallel ("horizontal") to the reading direction. If this is not a finite value, the default is used (centring for strings parallel to the axis, justification of the end nearest the axis otherwise).
- `padj`: adjustment for each tick label perpendicular to the reading direction. For labels parallel to the axes, `padj=0` means right or top alignment, and `padj=1` means left or bottom alignment. This can be a vector given a value for each string, and will be recycled as necessary.
  
  If `padj` is not a finite value (the default), the value of `par("las")` determines the adjustment. For strings plotted perpendicular to the axis the default is to centre the string.
- `...`: other graphical parameters may also be passed as arguments to this function, particularly, `cex.axis`, `col.axis` and `font.axis` for axis annotation, `mgp` for positioning, `tck` or `tcl` for tick mark length and direction, `las` for vertical/horizontal label orientation, or `fg` instead of `col`, see `par` on these.
Details

The axis line is drawn from the lowest to the highest value of `at`, but will be clipped at the plot region. Only ticks which are drawn from points within the plot region (up to a tolerance for rounding error) are plotted, but the ticks and their labels may well extend outside the plot region.

When `at = NULL`, pretty tick mark locations are computed internally (the same way `axTicks(side)` would) from `par("usr", "lab")` and `par("xlog")` (or "ylog"). Note that these may change if an on-screen plot is resized (for example, if `plot` argument `asp` is set).

Several of the graphics parameters affect the way axes are drawn. The vertical (for sides 1 and 3) positions of the axis and the tick labels are controlled by `mgp`, the size and direction of the ticks is controlled by `tck` and `tcl` and the appearance of the tick labels by `cex.axis`, `col.axis` and `font.axis` with orientation controlled by `las` (but not `srt`, unlike S which uses `srt` if `at` is supplied and `las` if it is not). Note that `adj` is not supported. See `par` for details.

Value

The numeric locations on the axis scale at which tick marks were drawn when the plot was first drawn (see Details).

This function is usually invoked for its side effect, which is to add an axis to an already existing plot.

References


See Also

`axTicks` returns the axis tick locations corresponding to `at=NULL`; `pretty` is more flexible for computing pretty tick coordinates and does not depend on (nor adapt to) the coordinate system in use.

Several graphics parameters affecting the appearance are documented in `par`.

Examples

```r
plot(1:4, rnorm(4), axes=FALSE)
axis(1, 1:4, LETTERS[1:4])
axis(2)
box() # to make it look "as usual"

plot(1:7, rnorm(7), main = "axis() examples",
     type = "s", xaxt = "n", frame = FALSE, col = "red")
axis(1, 1:7, LETTERS[1:7], col.axis = "blue")
# unusual options:
axis(4, col = "violet", col.axis="dark violet", lwd = 2)
axis(3, col = "gold", lty = 2, lwd = 0.5)
```
Description

Functions to plot objects of classes "POSIXlt", "POSIXct" and "Date" representing calendar dates and times.

Usage

axis.POSIXct(side, x, at, format, labels = TRUE, ...)  
axis.Date(side, x, at, format, labels = TRUE, ...)

## S3 method for class 'POSIXct':  
plot(x, y, xlab = "", ...)  
## S3 method for class 'POSIXlt':  
plot(x, y, xlab = "", ...)  
## S3 method for class 'Date':  
plot(x, y, xlab = "", ...)

Arguments

x, at A date-time object.
y numeric values to be plotted against x.
xlab a character string giving the label for the x axis.
side See axis.
format See strftime.
labels Either a logical value specifying whether annotations are to be made at the tickmarks, or a vector of character strings to be placed at the tickpoints.
... Further arguments to be passed from or to other methods, typically graphical parameters or arguments of plot.default.

Details

The functions plot against an x-axis of date-times. axis.POSIXct and axis.Date work quite hard to choose suitable time units (years, months, days, hours, minutes or seconds) and a sensible output format, but this can be overridden by supplying a format specification.

If at is supplied it specifies the locations of the ticks and labels whereas if x is specified a suitable grid of labels is chosen. Printing of tick labels can be suppressed by using labels = FALSE.

Value

The locations on the axis scale at which tick marks were drawn.

See Also

DateTimeClasses, Dates for details of the classes.
Examples

```r
attach(beaver1)
time <- strptime(paste(1990, day, time %/% 100, time %% 100),
"%Y %j %H %M")
plot(time, temp, type="l") # axis at 4-hour intervals.
# now label every hour on the time axis
plot(time, temp, type="l", xaxt="n")
r <- as.POSIXct(round(range(time), "hours"))
axis.POSIXct(1, at=seq(r[1], r[2], by="hour"), format="%H")
rm(time)
detach(beaver1)

plot(.leap.seconds, 1:22, type="n", yaxt="n",
     xlab="leap seconds", ylab="", bty="n")
rug(.leap.seconds)
## or as dates
lps <- as.Date(.leap.seconds)
plot(lps, 1:22, type="n", yaxt="n", xlab="leap seconds", ylab="", bty="n")
rug(lps)
## 100 random dates in a 10-week period
random.dates <- as.Date("2001/1/1") + 70*sort(runif(100))
plot(random.dates, 1:100)
# or for a better axis labelling
plot(random.dates, 1:100, xaxt="n")
axis.Date(1, at=seq(as.Date("2001/1/1"), max(random.dates)+6, "weeks"))
axis.Date(1, at=seq(as.Date("2001/1/1"), max(random.dates)+6, "days"),
          labels = FALSE, tcl = -0.2)
```

axTicks

Compute Axis Tickmark Locations

AxTicks

Compute pretty tickmark locations, the same way as R does internally. This is only non-trivial when log coordinates are active. By default, gives the at values which axis(side) would use.

Usage

```r
axTicks(side, axp = NULL, usr = NULL, log = NULL)
```

Arguments

- `side` integer in 1:4, as for `axis`.
- `axp` numeric vector of length three, defaulting to `par("zaxp")` where “Z” is “x” or “y” depending on the side argument.
- `usr` numeric vector of length four, defaulting to `par("usr")` giving horizontal (‘x’) and vertical (‘y’) user coordinate limits.
- `log` logical indicating if log coordinates are active; defaults to `par("zlog")` where ‘Z’ is as for the `axp` argument above.
Details

The `axp`, `usr`, and `log` arguments must be consistent as their default values (the `par(..) results) are. If you specify all three (as non-NULL), the graphics environment is not used at all. Note that the meaning of `axp` alters very much when `log` is `TRUE`, see the documentation on `par(xaxp=..)`.

`axTicks()` can be regarded as an R implementation of the C function `CreateAtVector()` in ‘`...../src/main/plot.c`’ which is called by `axis(side,*)` when no argument at is specified.

Value

numeric vector of coordinate values at which axis tickmarks can be drawn. By default, when only the first argument is specified, these values should be identical to those that `axis(side)` would use or has used.

See Also

`axis`, `par`, `pretty` uses the same algorithm (but independently of the graphics environment) and has more options. However it is not available for `log = TRUE`.

Examples

```r
plot(1:7, 10*21:27)
axTicks(1)
axTicks(2)
stopifnot(identical(axTicks(1), axTicks(3)),
          identical(axTicks(2), axTicks(4)))
```

```r
## Show how axTicks() and axis() correspond :
op <- par(mfrow = c(3,1))
for(x in 9999*c(1,2,8)) {
  plot(x,9, log = "x")
  cat(formatC(par("xaxp"),wid=5),";","T <- axTicks(1),"\n")
  rug(T, col="red")
}
par(op)
```

Description

Creates a bar plot with vertical or horizontal bars.

Usage

```r
## Default S3 method:
barplot(height, width = 1, space = NULL, 
        names.arg = NULL, legend.text = NULL, beside = FALSE, 
        horiz = FALSE, density = NULL, angle = 45, 
        col = NULL, border = par("fg"), 
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL, 
        xlim = NULL, ylim = NULL, xpd = TRUE, log = "",
```
barplot

axes = TRUE, axisnames = TRUE,
cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
add = FALSE, ...

Arguments

height either a vector or matrix of values describing the bars which make up the plot.
If height is a vector, the plot consists of a sequence of rectangular bars with
heights given by the values in the vector. If height is a matrix and beside is
FALSE then each bar of the plot corresponds to a column of height, with the
values in the column giving the heights of stacked "sub-bars" making up the bar.
If height is a matrix and beside is TRUE, then the values in each column
are juxtaposed rather than stacked.

width optional vector of bar widths. Re-cycled to length the number of bars drawn.
Specifying a single value will no visible effect unless xlim is specified.

space the amount of space (as a fraction of the average bar width) left before each bar.
May be given as a single number or one number per bar. If height is a matrix
and beside is TRUE, space may be specified by two numbers, where the first
is the space between bars in the same group, and the second the space between
the groups. If not given explicitly, it defaults to c(0,1) if height is a matrix
and beside is TRUE, and to 0.2 otherwise.
	names.arg a vector of names to be plotted below each bar or group of bars. If this argument
is omitted, then the names are taken from the names attribute of height if
this is a vector, or the column names if it is a matrix.

legend.text a vector of text used to construct a legend for the plot, or a logical indicating
whether a legend should be included. This is only useful when height is
a matrix. In that case given legend labels should correspond to the rows of
height; if legend.text is true, the row names of height will be used as
labels if they are non-null.

beside a logical value. If FALSE, the columns of height are portrayed as stacked
bars, and if TRUE the columns are portrayed as juxtaposed bars.

horiz a logical value. If FALSE, the bars are drawn vertically with the first bar to the
left. If TRUE, the bars are drawn horizontally with the first at the bottom.

density a vector giving the density of shading lines, in lines per inch, for the bars or bar
components. The default value of NULL means that no shading lines are drawn.
Non-positive values of density also inhibit the drawing of shading lines.

angle the slope of shading lines, given as an angle in degrees (counter-clockwise), for
the bars or bar components.

col a vector of colors for the bars or bar components. By default, grey is used if
height is a vector, and a gamma-corrected grey palette if height is a matrix.

border the color to be used for the border of the bars.

main,sub overall and sub title for the plot.

xlab a label for the x axis.

ylab a label for the y axis.
xlim limits for the x axis.

ylim limits for the y axis.
xpd logical. Should bars be allowed to go outside region?
log  string specifying if axis scales should be logarithmic; see plot.default.
axes  logical. If TRUE, a vertical (or horizontal, if horiz is true) axis is drawn.
axisnames  logical. If TRUE, and if there are names.arg (see above), the other axis is drawn (with lty=0) and labeled.
cex.axis  expansion factor for numeric axis labels.
cex.names  expansion factor for axis names (bar labels).
inside  logical. If TRUE, the lines which divide adjacent (non-stacked!) bars will be drawn. Only applies when space = 0 (which it partly is when beside = TRUE).
plot  logical. If FALSE, nothing is plotted.
axis.lty  the graphics parameter lty applied to the axis and tick marks of the categorical (default horizontal) axis. Note that by default the axis is suppressed.
offset  a vector indicating how much the bars should be shifted relative to the x axis.
add  logical specifying if bars should be added to an already existing plot; defaults to FALSE.
...  further graphical parameters (par) are passed to plot.window(), title() and axis.

Details

This is a generic function, it currently only has a default method. A formula interface may be added eventually.

Value

A numeric vector (or matrix, when beside = TRUE), say mp, giving the coordinates of all the bar midpoints drawn, useful for adding to the graph.

If beside is true, use colMeans(mp) for the midpoints of each group of bars, see example.

Note

Prior to R 1.6.0, barplot behaved as if axis.lty = 1, unintentionally.

References


See Also

plot(..., type="h"), dotchart, hist.

Examples

tN <- table(Ni <- rpois(100, lambda=5))
r <- barplot(tN, col=rainbow(20))
#- type = "h" plotting *is* 'bar'plot
lines(r, tN, type='h', col='red', lwd=2)

barplot(tN, space = 1.5, axisnames=FALSE,
        sub = "barplot(..., space= 1.5, axisnames = FALSE")
}
barplot(VADeaths, plot = FALSE)
barplot(VADeaths, plot = FALSE, beside = TRUE)

mp <- barplot(VADeaths) # default
tot <- colMeans(VADeaths)
text(mp, tot + 3, format(tot), xpd = TRUE, col = "blue")
barplot(VADeaths, beside = TRUE,
  col = c("lightblue", "mistyrose", "lightcyan",
         "lavender", "cornsilk"),
  legend = rownames(VADeaths), ylim = c(0, 100))
title(main = "Death Rates in Virginia", font.main = 4)

hh <- t(VADeaths)[, 5:1]
mybarcol <- "gray20"
mp <- barplot(hh, beside = TRUE,
  col = c("lightblue", "mistyrose",
          "lightcyan", "lavender"),
  legend = colnames(VADeaths), ylim = c(0, 100),
  main = "Death Rates in Virginia", font.main = 4,
  sub = "Faked upper 2*sigma error bars", col.sub = mybarcol,
  cex.names = 1.5)
segments(mp, hh, mp + 2*sqrt(1000*hh/100), col = mybarcol, lwd = 1.5)
stopifnot(dim(mp) == dim(hh)) # corresponding matrices
mtext(side = 1, at = colMeans(mp), line = -2,
      text = paste("Mean", formatC(colMeans(hh))), col = "red")

# Bar shading example
barplot(VADeaths, angle = 15+10*1:5, density = 20, col = "black",
        legend = rownames(VADeaths))
title(main = list("Death Rates in Virginia", font = 4))

# border:
barplot(VADeaths, border = "dark blue")

# log scales (not much sense here):
barplot(tN, col=heat.colors(12), log = "y")
barplot(tN, col=gray.colors(20), log = "xy")

---

**box**

**Draw a Box around a Plot**

**Description**

This function draws a box around the current plot in the given color and linetype. The *bty* parameter determines the type of box drawn. See `par` for details.

**Usage**

```r
box(which = "plot", lty = "solid", ...)```

**Arguments**

*which* character, one of "plot", "figure", "inner" and "outer".
Description

Produce box-and-whisker plot(s) of the given (grouped) values.

Usage

boxplot(x, ...)

## S3 method for class 'formula':
boxplot(formula, data = NULL, ..., subset, na.action = NULL)

## Default S3 method:
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,
notch = FALSE, outline = TRUE, names, plot = TRUE,
border = par("fg"), col = NULL, log = "",
pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5),
horizontal = FALSE, add = FALSE, at = NULL)

Arguments

formula a formula, such as y ~ grp, where y is a numeric vector of data values to be
split into groups according to the grouping variable grp (usually a factor).
data a data.frame (or list) from which the variables in formula should be taken.
subset an optional vector specifying a subset of observations to be used for plotting.
na.action a function which indicates what should happen when the data contain NAs. The
default is to ignore missing values in either the response or the group.
x for specifying data from which the boxplots are to be produced. Either a numeric
vector, or a single list containing such vectors. Additional unnamed arguments
specify further data as separate vectors (each corresponding to a component
boxplot). NAs are allowed in the data.
... For the formula method, arguments to the default method and graphical parameters.

For the default method, unnamed arguments are additional data vectors (unless \( x \) is a list when they are ignored), and named arguments are graphical parameters in addition to the ones given by argument \( \text{pars} \).

\textbf{range} \quad \text{this determines how far the plot whiskers extend out from the box. If range is positive, the whiskers extend to the most extreme data point which is no more than range times the interquartile range from the box. A value of zero causes the whiskers to extend to the data extremes.}

\textbf{width} \quad \text{a vector giving the relative widths of the boxes making up the plot.}

\textbf{varwidth} \quad \text{if \text{varwidth} is TRUE, the boxes are drawn with widths proportional to the square-roots of the number of observations in the groups.}

\textbf{notch} \quad \text{if \text{notch} is TRUE, a notch is drawn in each side of the boxes. If the notches of two plots do not overlap this is 'strong evidence' that the two medians differ (Chambers et al., 1983, p. 62). See \text{boxplot.stats} for the calculations used.}

\textbf{outline} \quad \text{if \text{outline} is not true, the outliers are not drawn (as points whereas S+ uses lines).}

\textbf{names} \quad \text{group labels which will be printed under each boxplot.}

\textbf{boxwex} \quad \text{a scale factor to be applied to all boxes. When there are only a few groups, the appearance of the plot can be improved by making the boxes narrower.}

\textbf{staplewex} \quad \text{staple line width expansion, proportional to box width.}

\textbf{outwex} \quad \text{outlier line width expansion, proportional to box width.}

\textbf{plot} \quad \text{if TRUE (the default) then a boxplot is produced. If not, the summaries which the boxplots are based on are returned.}

\textbf{border} \quad \text{an optional vector of colors for the outlines of the boxplots. The values in \text{border} are recycled if the length of \text{border} is less than the number of plots.}

\textbf{col} \quad \text{if \text{col} is non-null it is assumed to contain colors to be used to colour the bodies of the box plots.}

\textbf{log} \quad \text{character indicating if \( x \) or \( y \) or both coordinates should be plotted in log scale.}

\textbf{pars} \quad \text{a list of (potentially many) more graphical parameters, e.g., boxwex or \text{outpch}; these are passed to \text{bxp} (if \text{plot} is true); for details, see there.}

\textbf{horizontal} \quad \text{logical indicating if the boxplots should be horizontal; default FALSE means vertical boxes.}

\textbf{add} \quad \text{logical, if true \textit{add} boxplot to current plot.}

\textbf{at} \quad \text{numeric vector giving the locations where the boxplots should be drawn, particularly when \text{add} = \text{TRUE}; defaults to \( 1:n \) where \( n \) is the number of boxes.}

\textbf{Details}

The generic function \text{boxplot} currently has a default method (\text{boxplot.default}) and a formula interface (\text{boxplot.formula}).

If multiple groups are supplied either as multiple arguments or via a formula, parallel boxplots will be plotted, in the order of the arguments or the order of the levels of the factor (see \text{factor}).

Missing values are ignored when forming boxplots.
Value

List with the following components:

- **stats**: a matrix, each column contains the extreme of the lower whisker, the lower hinge, the median, the upper hinge and the extreme of the upper whisker for one group/plot.
- **n**: a vector with the number of observations in each group.
- **conf**: a matrix where each column contains the lower and upper extremes of the notch.
- **out**: the values of any data points which lie beyond the extremes of the whiskers.
- **group**: a vector of the same length as **out** whose elements indicate which group the outlier belongs to.
- **names**: a vector of names for the groups.

References


See also boxplot.stats.

See Also

- **boxplot.stats** which does the computation, **bxp** for the plotting and more examples; and **stripchart** for an alternative (with small data sets).

Examples

```r
## boxplot on a formula:
boxplot(count ~ spray, data = InsectSprays, col = "lightgray")
# *add* notches (somewhat funny here):
boxplot(count ~ spray, data = InsectSprays,
       notch = TRUE, add = TRUE, col = "blue")

boxplot(decrease ~ treatment, data = OrchardSprays,
        log = "y", col = "bisque")

rb <- boxplot(decrease ~ treatment, data = OrchardSprays, col="bisque")

mn.t <- tapply(OrchardSprays$decrease, OrchardSprays$treatment, mean)
sd.t <- tapply(OrchardSprays$decrease, OrchardSprays$treatment, sd)
x1 <- 0.3 + seq(rb$n)
points(x1, mn.t, col = "orange", pch = 18)
arrows(x1, mn.t - sd.t, x1, mn.t + sd.t,
       code = 3, col = "pink", angle = 75, length = .1)

## boxplot on a matrix:
mat <- cbind(Uni05 = (1:100)/21, Norm = rnorm(100),
             T5 = rt(100, df = 5), Gam2 = rgamma(100, shape = 2))
boxplot(data.frame(mat), main = "boxplot(data.frame(mat), main = ...")
par(las=1)# all axis labels horizontal
boxplot(data.frame(mat), main = "boxplot(*, horizontal = TRUE)",
```
## Using 'at = ' and adding boxplots -- example idea by Roger Bivand:

```r
boxplot(len ~ dose, data = ToothGrowth, 
  boxwex = 0.25, at = 1:3 - 0.2, 
  subset = supp == "VC", col = "yellow", 
  main = "Guinea Pigs' Tooth Growth", 
  xlab = "Vitamin C dose mg", 
  ylab = "tooth length", ylim = c(0, 35), yaxs = "i")

boxplot(len ~ dose, data = ToothGrowth, add = TRUE, 
  boxwex = 0.25, at = 1:3 + 0.2, 
  subset = supp == "OJ", col = "orange")

legend(2, 9, c("Ascorbic acid", "Orange juice"), 
  fill = c("yellow", "orange"))
```

## more examples in help(bxp)

---

### bxp

**Box Plots from Summaries**

**Description**

bxp draws box plots based on the given summaries in z. It is usually called from within boxplot, but can be invoked directly.

**Usage**

```r
bxp(z, notch = FALSE, width = NULL, varwidth = FALSE, outline = TRUE, 
notch.frac = 0.5, log = "", border = par("fg"), col = par("bg"), 
pars = NULL, frame.plot = axes, horizontal = FALSE, 
add = FALSE, at = NULL, show.names = NULL, ...)
```

**Arguments**

- `z`: a list containing data summaries to be used in constructing the plots. These are usually the result of a call to boxplot, but can be generated in any fashion.
- `notch`: if notch is TRUE, a notch is drawn in each side of the boxes. If the notches of two plots do not overlap then the medians are significantly different at the 5 percent level.
- `width`: a vector giving the relative widths of the boxes making up the plot.
- `varwidth`: if varwidth is TRUE, the boxes are drawn with widths proportional to the square-roots of the number of observations in the groups.
- `outline`: if outline is not true, the outliers are not drawn.
- `notch.frac`: numeric in (0,1). When notch=TRUE, the fraction of the box width that the notches should use.
- `border`: character or numeric (vector), the color of the box borders. Is recycled for multiple boxes. Is used as default for the boxcol, medcol, whiskcol, staplecol, and outcol options (see below).
- `col`: character or numeric; the color within the box; recycled for multiple boxes. Is only used as default for boxfill and will be deprecated.
character, indicating if any axis should be drawn in logarithmic scale, as in `plot.default`.

frame.plot logical, indicating if a “frame” (box) should be drawn; defaults to TRUE, unless `axes = FALSE` is specified.

horizontal logical indicating if the boxplots should be horizontal; default FALSE means vertical boxes.

add logical, if true add boxplot to current plot.

at numeric vector giving the locations where the boxplots should be drawn, particularly when `add = TRUE`; defaults to `1:n` where `n` is the number of boxes.

show.names Set to TRUE or FALSE to override the defaults on whether an x-axis label is printed for each group.

pars,... graphical parameters can be passed as arguments to this function, either as a list (pars) or normally(...), see the following.

Currently, `yaxs` and `ylim` are used ‘along the boxplot’, i.e., vertically, when `horizontal` is false. `xaxt`, `yaxt`, `las`, `cex.axis`, and `col.axis` are passed to `axis`, and `main`, `cex.main`, `col.main`, `sub`, `cex.sub`, `col.sub`, `xlab`, `ylab`, `cex.lab`, and `col.lab` are passed to `title`.

The following arguments (or pars components) allow further customization of the boxplot graphics. Their defaults are typically determined from the non-prefixed version (e.g., boxlty from lty), either from the specified argument or pars component or the corresponding par one.

`boxwex`: a scale factor to be applied to all boxes. When there are only a few groups, the appearance of the plot can be improved by making the boxes narrower.

`staplewex, outwex`: staple and outlier line width expansion, proportional to box width.

`boxlty, boxlwd, boxcol, boxfill`: box outline type, width, color, and fill color.

`medlty, medlwd, medpch, medcex, medcol, medbg`: median line type, line width, point character, point size expansion, color, and background color. The default `medpch= NA` suppresses the point, and `medlty= "blank"` does so for the line. Note that (since R 2.1.0) `medlwd` defaults to 3x the “default” `lwd`.

`whisklty, whisklwd, whiskcol`: whisker line type, width, and color.

`staplelty, staplelwd, staplecol`: staple (= end of whisker) line type, width, and color.

`outlty, outlwd, outpch, outcex, outcol, outbg`: outlier line type, line width, point character, point size expansion, color, and background color. The default `outlty= "blank"` suppresses the lines and `outpch=NA` suppresses points.

Value

An invisible vector, actually identical to the at argument, with the coordinates ("x" if horizontal is false, "y" otherwise) of box centers, useful for adding to the plot.

Author(s)

The R Core development team and Arni Magnusson (arnima@u.washington.edu) who has provided most changes for the box*, med*, whisk*, staple*, and out* arguments.
References


Examples

set.seed(753)
(bx.p <- boxplot(split(rt(100, 4), gl(5,20))))
op <- par(mfrow= c(2,2))
bxp(bx.p, xaxt = "n")
 bxp(bx.p, notch = TRUE, axes = FALSE, pch = 4, boxfill=1:5)
 bxp(bx.p, notch = TRUE, boxfill= "lightblue", frame= FALSE, outl= FALSE,
     main = "bxp(*, frame= FALSE, outl= FALSE)"
 bxp(bx.p, notch = TRUE, boxfill= "lightblue", border= 2:6, ylim = c(-4,4),
     pch = 22, bg = "green", log = "x", main = ",.. log='x', ylim='*")
par(op)

## single group -- no label
boxplot (weight ~ group, data = PlantGrowth, subset = group=="ctrl")
## with label
bx <- boxplot(weight ~ group, data = PlantGrowth,
               subset = group=="ctrl", plot = FALSE)

bxp(bx,show.names=TRUE)
par(op)

z <- split(rnorm(1000), rpois(1000,2.2))
boxplot(z, whisklty=3, main="boxplot(z, whisklty = 3")

## Colour support similar to plot.default:
op <- par(mfrow=1:2, bg="light gray", fg="midnight blue")
boxplot(z, col.axis="skyblue3", main="boxplot(*, col.axis=..,main=..)")
plot(z[,1], col.axis="skyblue3", main="plot(*, col.axis=..,main=..)")
mtext("par(bg="light gray", fg="midnight blue")",
   outer = TRUE, line = -1.2)
par(op)

## Mimic S-Plus:
splus <- list(boxwex=0.4, staplewex=1, outwex=1, boxfill="grey40",
              medlwd=3, medcol="white", whisklty=3, outlty=1, outpch=NA)
boxplot(z, pars=splus)

## Recycled and "sweeping" parameters
op <- par(mfrow=c(1,2))
boxplot(z, border=1:5, lty = 3, medlty = 1, medlwd = 2.5)
boxplot(z, boxfill=1:3, pch=1:5, lwd = 1.5, medcol="white")
par(op)

## too many possibilities
boxplot(z, boxfill= "light gray", outpch = 21:25, outlty = 2,
         bg = "pink", lwd = 2, medcol = "dark blue", medcex = 2, medpch=20)

---

cdplot

Conditional Density Plots
cdplot

Description
Computes and plots conditional densities describing how the conditional distribution of a categorical variable y changes over a numerical variable x.

Usage

```r
CDPLOT(x, ...) 
## Default S3 method: 
CDPLOT(x, y, 
plot = TRUE, tol.ylab = 0.05, 
bw = "nrd0", n = 512, from = NULL, to = NULL, 
col = NULL, border = 1, main = "", xlab = NULL, ylab = NULL, 
yaxlabels = NULL, xlim = NULL, ylim = c(0, 1), ...) 
## S3 method for class 'formula': 
CDPLOT(formula, data = list(), 
plot = TRUE, tol.ylab = 0.05, 
bw = "nrd0", n = 512, from = NULL, to = NULL, 
col = NULL, border = 1, main = "", xlab = NULL, ylab = NULL, 
yaxlabels = NULL, xlim = NULL, ylim = c(0, 1), ..., 
subset = NULL)
```

Arguments

- **x**: an object, the default method expects either a single numerical variable.
- **y**: a "factor" interpreted to be the dependent variable
- **formula**: a "formula" of type y ~ x with a single dependent "factor" and a single numerical explanatory variable.
- **data**: an optional data frame.
- **plot**: logical. Should the computed conditional densities be plotted?
- **tol.ylab**: convenience tolerance parameter for y-axis annotation. If the distance between two labels drops under this threshold, they are plotted equidistantly.
- **bw**, **n**, **from**, **to**, ... arguments passed to `density`
- **col**: a vector of fill colors of the same length as `levels(y)`. The default is to call `gray.colors`.
- **border**: border color of shaded polygons.
- **main**, **xlab**, **ylab**: character strings for annotation
- **yaxlabels**: character vector for annotation of y axis, defaults to `levels(y)`.
- **xlim**, **ylim**: the range of x and y values with sensible defaults.
- **subset**: an optional vector specifying a subset of observations to be used for plotting.

Details

`cdplot` computes the conditional densities of x given the levels of y weighted by the marginal distribution of y. The densities are derived cumulatively over the levels of y.

This visualization technique is similar to spinograms (see `spineplot`) and plots P(y|x) against x. The conditional probabilities are not derived by descretization (as in the spinogram), but using a smoothing approach via `density`. 
Note that the estimates of the conditional densities are more reliable for high-density regions of $x$. Conversely, they are less reliable in regions with only few $x$ observations.

**Value**

The conditional density functions (cumulative over the levels of $y$) are returned invisibly.

**Author(s)**

Achim Zeileis (Achim.Zeileis@R-project.org)

**References**


**See Also**

spineplot, density

**Examples**

```r
## NASA space shuttle o-ring failures
fail <- factor(c(2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1),
               levels = 1:2, labels = c("no", "yes"))
temperature <- c(53, 57, 58, 63, 66, 67, 67, 67, 68, 69, 70, 70, 70, 70, 72, 73, 75, 75, 76, 76, 78, 79, 81)

## CD plot
cdplot(fail ~ temperature)
cdplot(fail ~ temperature, bw = 2)
cdplot(fail ~ temperature, bw = "SJ")

## compare with spinogram
(spineplot(fail ~ temperature, breaks = 3))

## scatter plot with conditional density
cdens <- cdplot(fail ~ temperature, plot = FALSE)
plot(I(as.numeric(fail) - 1) ~ jitter(temperature, factor = 2),
    xlab = "Temperature", ylab = "Conditional failure probability")
lines(53:81, 1 - cdens[[1]](53:81), col = 2)
```

**contour**

**Display Contours**

Create a contour plot, or add contour lines to an existing plot.
Usage

contour(x, ...)

## Default S3 method:
contour(x = seq(0, 1, len = nrow(z)),
        y = seq(0, 1, len = ncol(z)),
        z,
        nlevels = 10, levels = pretty(zlim, nlevels), labels = NULL,
        xlim = range(x, finite = TRUE),
        ylim = range(y, finite = TRUE),
        zlim = range(z, finite = TRUE),
        labcex = 0.6, drawlabels = TRUE, method = "flattest",
        vfont = c("sans serif", "plain"),
        axes = TRUE, frame.plot = axes,
        col = par("fg"), lty = par("lty"), lwd = par("lwd"),
        add = FALSE, ...)

Arguments

x, y  
locations of grid lines at which the values in z are measured. These must be in
ascending order. By default, equally spaced values from 0 to 1 are used. If x is
a list, its components x$x and x$y are used for x and y, respectively. If the
list has component z this is used for z.

z  
a matrix containing the values to be plotted (NAs are allowed). Note that x can
be used instead of z for convenience.
nlevels  
number of contour levels desired if levels is not supplied.

levels  
numeric vector of levels at which to draw contour lines.

labels  
a vector giving the labels for the contour lines. If NULL then the levels are used
as labels.

labcex  
cex for contour labelling. This is an absolute size, not a multiple of
par("cex").

drawlabels  
logical. Contours are labelled if TRUE.

method  
character string specifying where the labels will be located. Possible values are
"simple", "edge" and "flattest" (the default). See the Details section.

vfont  
if a character vector of length 2 is specified, then Hershey vector fonts are used
for the contour labels. The first element of the vector selects a typeface and the
second element selects a fontindex (see text for more information).

xlim, ylim, zlim  
x-, y- and z-limits for the plot.

axes, frame.plot  
logical indicating whether axes or a box should be drawn, see plot.default.

col  
color for the lines drawn.

lty  
line type for the lines drawn.

lwd  
line width for the lines drawn.

add  
logical. If TRUE, add to a current plot.

...  
additional graphical parameters (see par). The plot aspect ratio asp (see
plot.window) and the arguments to title may also be supplied.
contour is a generic function with only a default method in base R.

The methods for positioning the labels on contours are "simple" (draw at the edge of the plot, overlaying the contour line), "edge" (draw at the edge of the plot, embedded in the contour line, with no labels overlapping) and "flattest" (draw on the flattest section of the contour, embedded in the contour line, with no labels overlapping). The second and third may not draw a label on every contour line.

For information about vector fonts, see the help for text and Hershey.

Notice that contour interprets the z matrix as a table of \( f(x[i], y[j]) \) values, so that the x axis corresponds to row number and the y axis to column number, with column 1 at the bottom, i.e. a 90 degree clockwise rotation of the conventional textual layout.

contourLines returns a list of contours. Each contour is a list with elements:

- level
- x
- y

Value

References


See Also

contourLines, filled.contour for “color-filled” contours, image and the graphics demo which can be invoked as demo(graphics).

Examples

```r
x <- -6:16
op <- par(mfrow = c(2, 2))
contour(outer(x, x), method = "edge", vfont = c("sans serif", "plain"))
z <- outer(x, sqrt(abs(x)), FUN = "/")
image(x, x, z)
contour(x, x, z, col = "pink", add = TRUE, method = "edge",
        vfont = c("sans serif", "plain"))
contour(x, x, z, ylim = c(1, 6), method = "simple", labcex = 1)
contour(x, x, z, ylim = c(-6, 6), nlev = 20, lty = 2, method = "simple")
par(op)
## Persian Rug Art:
x <- y <- seq(-4*pi, 4*pi, len = 27)
r <- sqrt(outer(x^2, y^2, "+"))
opar <- par(mfrow = c(2, 2), mar = rep(0, 4))
for(f in pi^(0:3))
    contour(cos(r^2)*exp(-r/f),
             drawlabels = FALSE, axes = FALSE, frame = TRUE)
rx <- range(x <- 10+1:nrow(volcano))
ry <- range(y <- 10+1:ncol(volcano))
```
coplot

Conditioning Plots

description

This function produces two variants of the conditioning plots discussed in the reference below.

usage

coplot(formula, data, given.values, panel = points, rows, columns,
show.given = TRUE, col = par("fg"), pch = par("pch"),
bar.bg = c(num = gray(0.8), fac = gray(0.95)),
lab = c(x.name, paste("Given :", a.name)),
ylab = c(y.name, paste("Given :", b.name)),
subscripts = FALSE,
axlabels = function(f) abbreviate(levels(f)),
number = 6, overlap = 0.5, xim, ylim, ...)
co.intervals(x, number = 6, overlap = 0.5)

arguments

formula a formula describing the form of conditioning plot. A formula of the form
y ~ x | a indicates that plots of y versus x should be produced conditional on
the variable a. A formula of the form y ~ x | a * b indicates that plots of y
versus x should be produced conditional on the two variables a and b.
All three or four variables may be either numeric or factors. When x or y are
factors, the result is almost as if as.numeric() was applied, whereas for
factor a or b, the conditioning (and its graphics if show.given is true) are
adapted.
data

A data frame containing values for any variables in the formula. By default the environment where `coplot` was called from is used.

given.values

A value or list of two values which determine how the conditioning on a and b is to take place.

When there is no b (i.e., conditioning only on a), usually this is a matrix with two columns each row of which gives an interval, to be conditioned on, but is can also be a single vector of numbers or a set of factor levels (if the variable being conditioned on is a factor). In this case (no b), the result of `co.intervals` can be used directly as `given.values` argument.

panel

A function `x, y, col, pch, ...` which gives the action to be carried out in each panel of the display. The default is `points`.

rows

The panels of the plot are laid out in a rows by columns array. `rows` gives the number of rows in the array.

columns

The number of columns in the panel layout array.

show.given

Logical (possibly of length 2 for 2 conditioning variables): should conditioning plots be shown for the corresponding conditioning variables (default TRUE)

col

A vector of colors to be used to plot the points. If too short, the values are recycled.

pch

A vector of plotting symbols or characters. If too short, the values are recycled.

bar.bg

A named vector with components "num" and "fac" giving the background colors for the (shingle) bars, for numeric and factor conditioning variables respectively.

xlab

Character; labels to use for the x axis and the first conditioning variable. If only one label is given, it is used for the x axis and the default label is used for the conditioning variable.

ylab

Character; labels to use for the y axis and any second conditioning variable.

subscripts

Logical: if true the panel function is given an additional (third) argument subscripts giving the subscripts of the data passed to that panel.

axlabels

Function for creating axis (tick) labels when x or y are factors.

number

Integer; the number of conditioning intervals, for a and b, possibly of length 2. It is only used if the corresponding conditioning variable is not a factor.

overlap

Numeric < 1; the fraction of overlap of the conditioning variables, possibly of length 2 for x and y direction. When overlap < 0, there will be gaps between the data slices.

xlim

The range for the x axis.

ylim

The range for the y axis.

... Additional arguments to the panel function.

x A numeric vector.

Details

In the case of a single conditioning variable a, when both rows and columns are unspecified, a "close to square" layout is chosen with `columns >= rows`.

In the case of multiple rows, the order of the panel plots is from the bottom and from the left (corresponding to increasing a, typically).

A panel function should not attempt to start a new plot, but just plot within a given coordinate system: thus `plot` and `boxplot` are not panel functions.

As from R 2.0.0 the rendering of arguments xlab and ylab is not controlled by `par` arguments `cex.lab` and `font.lab` even though they are plotted by `mtext` rather than `title`.
Value

co.intervals(., number, .) returns a (number × 2) matrix, say ci, where ci[k,] is the range of x values for the k-th interval.

References


See Also

pairs, panel.smooth, points.

Examples

## Tonga Trench Earthquakes
coplot(lat ~ long | depth, data = quakes)
given.depth <- co.intervals(quakes$depth, number=4, overlap=.1)
coplot(lat ~ long | depth, data = quakes, given.v=given.depth, rows=1)

## Conditioning on 2 variables:
ll.dm <- lat ~ long | depth * mag
coplot(ll.dm, data = quakes)
coplot(ll.dm, data = quakes, number=c(4,7), show.given=c(TRUE,FALSE))
coplot(ll.dm, data = quakes, number=c(3,7),
       overlap=c(-.5,.1)) # negative overlap DROPS values

## given two factors
Index <- seq(length=nrow(warpbreaks)) # to get nicer default labels
coplot(breaks ~ Index | wool * tension, data = warpbreaks, show.given = 0:1)
coplot(breaks ~ Index | wool * tension, data = warpbreaks, 
       col = "red", bg = "pink", pch = 21, bar.bg = c(fac = "light blue"))

## Example with empty panels:
attach(data.frame(state.x77)) # don't need 'data' arg. below
coplot(Life.Exp ~ Income | Illiteracy * state.region, number = 3, 
       panel = function(x, y, ...) panel.smooth(x, y, span = .8, ...))

## y ~ factor -- not really sensical, but 'show off':
coplot(Life.Exp ~ state.region | Income * state.division, 
       panel = panel.smooth)
detach() # data.frame(state.x77)

curve  

Draw Function Plots

Description

Draws a curve corresponding to the given function or expression (in x) over the interval [from, to].
Usage

curve(expr, from, to, n = 101, add = FALSE, type = "l",
       ylab = NULL, log = NULL, xlim = NULL, ...)  
## S3 method for class 'function':
plot(x, from = 0, to = 1, xlim = NULL, ...)

Arguments

expr    an expression written as a function of x, or alternatively the name of a function which will be plotted.
x      a 'vectorizing' numeric \texttt{R} function.
from, to  the range over which the function will be plotted.
n      integer; the number of x values at which to evaluate.
add    logical; if \texttt{TRUE} add to already existing plot.
xlim numeric of length 2; if specified, it serves as default for \texttt{c(from, to)}.
type, ylab, log, ...
      graphical parameters can also be specified as arguments. \texttt{plot.function} passes all these to \texttt{curve}.

Details

The evaluation of \texttt{expr} is at \texttt{n} points equally spaced over the range \texttt{[from, to]}, possibly adapted to log scale. The points determined in this way are then joined with straight lines. \texttt{x(t)} or \texttt{expr} (with \texttt{x} inside) must return a numeric of the same length as the argument \texttt{t} or \texttt{x}.

If \texttt{add = TRUE}, \texttt{c(from, to)} default to \texttt{xlim} which defaults to the current x-limits. Further, \texttt{log} is taken from the current plot when \texttt{add} is true.

This used to be a quick hack which now seems to serve a useful purpose, but can give bad results for functions which are not smooth.

For “expensive” expressions, you should use smarter tools.

See Also

\texttt{splinefun} for spline interpolation, \texttt{lines}.

Examples

```r
op <- par(mfrow=c(2,2))
curve(x^3-3*x, -2, 2)
curve(x^2-2, add = TRUE, col = "violet")
plot(cos, xlim = c(-pi,3*pi), n = 1001, col = "blue")
chippy <- function(x) sin(cos(x)*exp(-x/2))
curve(chippy, -8, 7, n=2001)
curve(chippy, -8, -5)
for(ll in c("","x","y","xy"))
  curve(log(1+x), 1,100, log=ll, sub=paste("log= ",ll,"",sep=""))
par(op)
```
Description

Draw a Cleveland dot plot.

Usage

```
dotchart(x, labels = NULL, groups = NULL, gdata = NULL,
cex = par("cex"), pch = 21, gpch = 21, bg = par("bg"),
color = par("fg"), gcolor = par("fg"), lcolor = "gray",
xlim = range(x[is.finite(x)]),
main = NULL, xlab = NULL, ylab = NULL, ...)
```

Arguments

- `x`: either a vector or matrix of numeric values (NAs are allowed). If `x` is a matrix, the overall plot consists of juxtaposed dotplots for each row.
- `labels`: a vector of labels for each point. For vectors the default is to use `names(x)` and for matrices the row labels `dimnames(x)[[1]]`.
- `groups`: an optional factor indicating how the elements of `x` are grouped. If `x` is a matrix, groups will default to the columns of `x`.
- `gdata`: data values for the groups. This is typically a summary such as the median or mean of each group.
- `cex`: the character size to be used. Setting `cex` to a value smaller than one can be a useful way of avoiding label overlap. Unlike many other graphics functions, this sets the actual size, not a multiplet of `par("cex")`.
- `pch`: the plotting character or symbol to be used.
- `gpch`: the plotting character or symbol to be used for group values.
- `bg`: the background color of plotting characters or symbols to be used; use `par(bg=*)` to set the background color of the whole plot.
- `color`: the color(s) to be used for points an labels.
- `gcolor`: the single color to be used for group labels and values.
- `lcolor`: the color(s) to be used for the horizontal lines.
- `xlim`: horizontal range for the plot, see `plot.window`, e.g.
- `main`: overall title for the plot, see `title`.
- `xlab`, `ylab`: axis annotations as in `title`.
- `...`: graphical parameters can also be specified as arguments.

Value

This function is invoked for its side effect, which is to produce two variants of dotplots as described in Cleveland (1985).

Dot plots are a reasonable substitute for bar plots.
References


Examples

dotchart(VADeaths, main = "Death Rates in Virginia - 1940")
op <- par(xaxs="i")# 0 -- 100%
dotchart(t(VADeaths), xlim = c(0,100),
        main = "Death Rates in Virginia - 1940")
par(op)

filled.contour

Level (Contour) Plots

Description

This function produces a contour plot with the areas between the contours filled in solid color (Cleveland calls this a level plot). A key showing how the colors map to z values is shown to the right of the plot.

Usage

```
filled.contour(x = seq(0, 1, len = nrow(z)),
y = seq(0, 1, len = ncol(z)),
z,
xlim = range(x, finite=TRUE),
ylim = range(y, finite=TRUE),
zlim = range(z, finite=TRUE),
levels = pretty(zlim, nlevels), nlevels = 20,
color.palette = cm.colors,
col = color.palette(length(levels) - 1),
plot.title, plot.axes, key.title, key.axes,
asp = NA, xaxs = "i", yaxs = "i", las = 1,
axes = TRUE, frame.plot = axes, ...)
```

Arguments

- `x, y`: locations of grid lines at which the values in `z` are measured. These must be in ascending order. By default, equally spaced values from 0 to 1 are used. If `x` is a list, its components `x$x` and `x$y` are used for `x` and `y`, respectively. If the list has component `z` this is used for `z`.
- `z`: a matrix containing the values to be plotted (NAs are allowed). Note that `x` can be used instead of `z` for convenience.
- `xlim`: x limits for the plot.
- `ylim`: y limits for the plot.
- `zlim`: z limits for the plot.
levels        a set of levels which are used to partition the range of \( z \). Must be strictly increasing (and finite). Areas with \( z \) values between consecutive levels are painted with the same color.

nlevels      if levels is not specified, the range of \( z \), values is divided into approximately this many levels.

color.palette a color palette function to be used to assign colors in the plot.

col          an explicit set of colors to be used in the plot. This argument overrides any palette function specification.

plot.title   statements which add titles to the main plot.

plot.axes    statements which draw axes (and a box) on the main plot. This overrides the default axes.

key.title    statements which add titles for the plot key.

key.axes     statements which draw axes on the plot key. This overrides the default axis.

asp          the \( y/x \) aspect ratio, see plot.window.

xaxs         the x axis style. The default is to use internal labeling.

yaxs         the y axis style. The default is to use internal labeling.

las           the style of labeling to be used. The default is to use horizontal labeling.

axes, frame.plot logicals indicating if axes and a box should be drawn, as in plot.default.

...          additional graphical parameters, currently only passed to title().

Note

This function currently uses the layout function and so is restricted to a full page display. As an alternative consider the levelplot function from the lattice package which works in multipanel displays.

The output produced by filled.contour is actually a combination of two plots; one is the filled contour and one is the legend. Two separate coordinate systems are set up for these two plots, but they are only used internally - once the function has returned these coordinate systems are lost. If you want to annotate the main contour plot, for example to add points, you can specify graphics commands in the plot.axes argument. An example is given below.

Author(s)

Ross Ihaka.

References


See Also

contour, image, palette; levelplot from package lattice.
Examples

filled.contour(volcano, color = terrain.colors, asp = 1)# simple
x <- 10*1:nrow(volcano)
y <- 10*1:ncol(volcano)
filled.contour(x, y, volcano, color = terrain.colors,
              plot.title = title(main = "The Topography of Maunga Whau",
                             xlab = "Meters North", ylab = "Meters West"),
              plot.axes = { axis(1, seq(100, 800, by = 100))
                           axis(2, seq(100, 600, by = 100)) },
              key.title = title(main="Height
                   (meters)",
              key.axes = axis(4, seq(90, 190, by = 10)))# maybe also asp=1
mtext(paste("filled.contour(.) from", R.version.string),
      side = 1, line = 4, adj = 1, cex = .66)

# Annotating a filled contour plot
a <- expand.grid(1:20, 1:20)
b <- matrix(a[,1] + a[,2], 20)
filled.contour(x = 1:20, y = 1:20, z = b,
               plot.axes={ axis(1); axis(2); points(10,10) })

## Persian Rug Art:
x <- y <- seq(-4*pi, 4*pi, len = 27)
r <- sqrt(outer(x^2, y^2, "+"))
filled.contour(cos(r^2)*exp(-r/(2*pi)), axes = FALSE)
## rather, the key *should* be labeled:
filled.contour(cos(r^2)*exp(-r/(2*pi)), frame.plot = FALSE, plot.axes = {})
std  a character string specifying how to standardize the table. Must be one of "margins", "ind.max", or "all.max", and can be abbreviated by the initial letter. If set to "margins", each 2 by 2 table is standardized to equate the margins specified by margin while preserving the odds ratio. If "ind.max" or "all.max", the tables are either individually or simultaneously standardized to a maximal cell frequency of 1.

margin a numeric vector with the margins to equate. Must be one of 1, 2, or c(1, 2) (the default), which corresponds to standardizing the row, column, or both margins in each 2 by 2 table. Only used if std equals "margins".

space the amount of space (as a fraction of the maximal radius of the quarter circles) used for the row and column labels.

main  character string for the fourfold title.

mfrow a numeric vector of the form c(nr, nc), indicating that the displays for the 2 by 2 tables should be arranged in an nr by nc layout, filled by rows.

mfcol a numeric vector of the form c(nr, nc), indicating that the displays for the 2 by 2 tables should be arranged in an nr by nc layout, filled by columns.

Details

The fourfold display is designed for the display of 2 by 2 by k tables. Following suitable standardization, the cell frequencies $f_{ij}$ of each 2 by 2 table are shown as a quarter circle whose radius is proportional to $\sqrt{f_{ij}}$ so that its area is proportional to the cell frequency. An association (odds ratio different from 1) between the binary row and column variables is indicated by the tendency of diagonally opposite cells in one direction to differ in size from those in the other direction; color is used to show this direction. Confidence rings for the odds ratio allow a visual test of the null of no association; the rings for adjacent quadrants overlap iff the observed counts are consistent with the null hypothesis.

Typically, the number k corresponds to the number of levels of a stratifying variable, and it is of interest to see whether the association is homogeneous across strata. The fourfold display visualizes the pattern of association. Note that the confidence rings for the individual odds ratios are not adjusted for multiple testing.

References


See Also

mosaicplot

Examples

```r
## Use the Berkeley admission data as in Friendly (1995).
x <- aperm(UCBAdmissions, c(2, 1, 3))
dimnames(x)[[2]] <- c("Yes", "No")
names(dimnames(x)) <- c("Sex", "Admit?", "Department")
stats::ftable(x)
## Fourfold display of data aggregated over departments, with
## frequencies standardized to equate the margins for admission
```
## and sex.
## Figure 1 in Friendly (1994).
fourfoldplot(margin.table(x, c(1, 2)))

## Fourfold display of x, with frequencies in each table
## standardized to equate the margins for admission and sex.
## Figure 2 in Friendly (1994).
fourfoldplot(x)

## Fourfold display of x, with frequencies in each table
## standardized to equate the margins for admission. but not
## for sex.
## Figure 3 in Friendly (1994).
fourfoldplot(x, margin = 2)

---

**frame**

Create / Start a New Plot Frame

### Description

This function (frame is an alias for plot.new) causes the completion of plotting in the current plot (if there is one) and an advance to a new graphics frame. This is used in all high-level plotting functions and also useful for skipping plots when a multi-figure region is in use.

### Usage

```r
plot.new()
frame()
```

### Details

There is a hook called "plot.new" (see setHook) called immediately after advancing the frame, which is used in the testing code to annotate the new page. The hook function(s) are called with no argument. (If the value is a character string, get is called on it from within the graphics namespace.)

### References


### See Also

`plot.window`, `plot.default`
The \textit{R Graphics Package}

\section*{Description}
R functions for base graphics

\section*{Details}
This package contains functions for base graphics. Base graphics are traditional S graphics, as opposed to the newer \textit{grid} graphics.
For a complete list of functions, use \texttt{library(help="graphics")}.

\section*{Author(s)}
R Development Core Team and contributors worldwide
Maintainer: R Core Team (R-core@r-project.org)

\section*{grid \hspace{1cm} Add Grid to a Plot}

\section*{Description}
\texttt{grid} adds an \texttt{nx} by \texttt{ny} rectangular grid to an existing plot.

\section*{Usage}
\texttt{grid(nx = NULL, ny = nx, col = "lightgray", lty = "dotted",
\hspace{2cm} lwd = NULL, equilogs = TRUE)}

\section*{Arguments}
\begin{itemize}
\item \texttt{nx, ny} \hspace{1cm} number of cells of the grid in x and y direction. When \texttt{NULL}, as per default, the grid aligns with the tick marks on the corresponding \texttt{default} axis (i.e., tickmarks as computed by \texttt{axTicks}). When \texttt{NA}, no grid lines are drawn in the corresponding direction.
\item \texttt{col} \hspace{1cm} character or (integer) numeric; color of the grid lines.
\item \texttt{lty} \hspace{1cm} character or (integer) numeric; line type of the grid lines.
\item \texttt{lwd} \hspace{1cm} non-negative numeric giving line width of the grid lines; defaults to \texttt{par("lwd")}.
\item \texttt{equilogs} \hspace{1cm} logical, only used when \texttt{log} coordinates and alignment with the axis tick marks are active. Setting \texttt{equilogs = FALSE} in that case gives \texttt{non equidistant} tick aligned grid lines.
\end{itemize}

\section*{Note}
If more fine tuning is required, use \texttt{abline(h = ., v = .)} directly.
See Also

plot, abline, lines, points.

Examples

```r
plot(1:3)
grid(NA, 5, lwd = 2) # grid only in y-direction

## maybe change the desired number of tick marks: par(lab=c(mx,my,?))
op <- par(mfcol = 1:2)
with(iris, {
  plot(Sepal.Length, Sepal.Width, col = as.integer(Species),
       xlim = c(4, 8), ylim = c(2, 4.5), panel.first = grid(),
       main = "with(iris, plot(...., panel.first = grid(), ..) )")
  plot(Sepal.Length, Sepal.Width, col = as.integer(Species),
       panel.first = grid(3, lty=1,lwd=2),
       main = "... panel.first = grid(3, lty=1,lwd=2), ..")
}
par(op)
```
hist

• a character string naming an algorithm to compute the number of cells (see Details),
• a function to compute the number of cells.

In the last three cases the number is a suggestion only.

freq logical; if TRUE, the histogram graphic is a representation of frequencies, the counts component of the result; if FALSE, probability densities, component density, are plotted (so that the histogram has a total area of one). Defaults to TRUE iff breaks are equidistant (and probability is not specified).

probability an alias for !freq, for S compatibility.

include.lowest logical; if TRUE, an x[i] equal to the breaks value will be included in the first (or last, for right = FALSE) bar. This will be ignored (with a warning) unless breaks is a vector.

right logical; if TRUE, the histograms cells are right-closed (left open) intervals.

density the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.

angle the slope of shading lines, given as an angle in degrees (counter-clockwise).

col a colour to be used to fill the bars. The default of NULL yields unfilled bars.

border the color of the border around the bars. The default is to use the standard foreground color.

main, xlab, ylab these arguments to title have useful defaults here.

xlim, ylim the range of x and y values with sensible defaults. Note that xlim is not used to define the histogram (breaks), but only for plotting (when plot = TRUE).

axes logical. If TRUE (default), axes are draw if the plot is drawn.

plot logical. If TRUE (default), a histogram is plotted, otherwise a list of breaks and counts is returned.

labels logical or character. Additionally draw labels on top of bars, if not FALSE; see plot.histogram.

nclass numeric (integer). For S(-PLUS) compatibility only, nclass is equivalent to breaks for a scalar or character argument.

... further graphical parameters to title and axis.

Details

The definition of “histogram” differs by source (with country-specific biases). R’s default with equi-spaced breaks (also the default) is to plot the counts in the cells defined by breaks. Thus the height of a rectangle is proportional to the number of points falling into the cell, as is the area provided the breaks are equally-spaced.

The default with non-equi-spaced breaks is to give a plot of area one, in which the area of the rectangles is the fraction of the data points falling in the cells.

If right = TRUE (default), the histogram cells are intervals of the form (a, b], i.e., they include their right-hand endpoint, but not their left one, with the exception of the first cell when include.lowest is TRUE.

For right = FALSE, the intervals are of the form [a, b), and include.lowest really has the meaning of “include highest”.

A numerical tolerance of $10^{-7}$ times the median bin size is applied when counting entries on the edges of bins.

The default for `breaks` is "Sturges"; see `nclass.Sturges`. Other names for which algorithms are supplied are "Scott" and "FD" / "Friedman-Diaconis" (with corresponding functions `nclass.scott` and `nclass.FD`). Case is ignored and partial matching is used. Alternatively, a function can be supplied which will compute the intended number of breaks as a function of \( x \).

**Value**

An object of class "histogram" which is a list with components:

- `breaks` the \( n+1 \) cell boundaries (= `breaks` if that was a vector).
- `counts` \( n \) integers; for each cell, the number of \( x[] \) inside.
- `density` values \( \hat{f}(x_i) \), as estimated density values. If all(`diff(breaks) == 1`), they are the relative frequencies \( \text{counts}/n \) and in general satisfy \( \sum_i \hat{f}(x_i)(b_{i+1} - b_i) = 1 \), where \( b_i = \text{breaks}[i] \).
- `intensities` same as density. Deprecated, but retained for compatibility.
- `mids` the \( n \) cell midpoints.
- `xname` a character string with the actual \( x \) argument name.
- `equidist` logical, indicating if the distances between `breaks` are all the same.

**Note**

The resulting value does not depend on the values of the arguments `freq` (or `probability`) or `plot`. This is intentionally different from S.

Prior to R 1.7.0, the element `breaks` of the result was adjusted for numerical tolerances. The nominal values are now returned even though tolerances are still used when counting.

**References**


**See Also**

`nclass.Sturges`, `stem`, `density`, `truehist` in package MASS.

Typical plots with vertical bars are not histograms. Consider `barplot` or `plot(*, type = "h")` for such bar plots.

**Examples**

```r
op <- par(mfrow=c(2, 2))
hist(islands)
utils::str(hist(islands, col="gray", labels = TRUE))

hist(sqrt(islands), br = 12, col="lightblue", border="pink")
##-- For non-equidistant breaks, counts should NOT be graphed unscaled:
  r <- hist(sqrt(islands), br = c(4*0:5, 10*3:5, 70, 100, 140), col="blue1")
text(r$mids, r$density, r$counts, adj=c(.5, -.5), col='blue3')
sapply(r[2:3], sum)
```
```r
sum(r$density * diff(r$breaks)) # == 1
lines(r, lty = 3, border = "purple") # -> lines.histogram(*)
par(op)

utils::str(hist(islands, br=12, plot= FALSE)) #=> 10 (~= 12) breaks
utils::str(hist(islands, br=c(12,20,36,80,200,1000,17000), plot = FALSE))

hist(islands, br=c(12,20,36,80,200,1000,17000), freq = TRUE,
     main = "WRONG histogram") # and warning
```

## hist.POSIXt

**Histogram of a Date or Date-Time Object**

### Description

Method for `hist` applied to date or date-time objects.

### Usage

```r
## S3 method for class 'POSIXt':
hist(x, breaks, ..., 
     xlab = deparse(substitute(x)),
     plot = TRUE, freq = FALSE,
     start.on.monday = TRUE, format)

## S3 method for class 'Date':
hist(x, breaks, ..., 
     xlab = deparse(substitute(x)),
     plot = TRUE, freq = FALSE,
     start.on.monday = TRUE, format)
```

### Arguments

- `x`: an object inheriting from class "POSIXt" or "Date".
- `breaks`: a vector of cut points or number giving the number of intervals which `x` is to be cut into or an interval specification, one of "days", "weeks", "months" or "years", plus "secs", "mins", "hours" for date-time objects.
- `...`: graphical parameters, or arguments to `hist.default` such as `include.lowest`, `right` and `labels`.
- `xlab`: a character string giving the label for the x axis, if plotted.
- `plot`: logical. If TRUE (default), a histogram is plotted, otherwise a list of breaks and counts is returned.
- `freq`: logical; if TRUE, the histogram graphic is a representation of frequencies, i.e, the counts component of the result; if FALSE, relative frequencies ("probabilities") are plotted.
- `start.on.monday`: logical. If breaks = "weeks", should the week start on Mondays or Sundays?
- `format`: for the x-axis labels. See `strptime`.
Value

An object of class "histogram": see hist.

See Also

seq.POSIXt, axis.POSIXct, hist

Examples

```r
hist(.leap.seconds, "years", freq = TRUE)
hist(.leap.seconds,
    seq(ISOdate(1970, 1, 1), ISOdate(2002, 1, 1), "5 years"))

## 100 random dates in a 10-week period
random.dates <- as.Date("2001/1/1") + 70*runif(100)
hist(random.dates, "weeks", format = "%d %b")
```

identify

Identify Points in a Scatter Plot

Description

identify reads the position of the graphics pointer when the (first) mouse button is pressed. It then searches the coordinates given in x and y for the point closest to the pointer. If this point is close enough to the pointer, its index will be returned as part of the value of the call.

Usage

```r
identify(x, ...)  
## Default S3 method:  
identify(x, y = NULL, labels = seq(along = x), pos = FALSE,  
n = length(x), plot = TRUE, atpen = FALSE, offset = 0.5,  
tolerance = 0.25, ...)  
```

Arguments

- `x, y`: coordinates of points in a scatter plot. Alternatively, any object which defines coordinates (a plotting structure, time series etc: see `xy.coords`) can be given as x, and y left undefined.
- `labels`: an optional vector, the same length as x and y, giving labels for the points.
- `pos`: if pos is TRUE, a component is added to the return value which indicates where text was plotted relative to each identified point: see Value.
- `n`: the maximum number of points to be identified.
- `plot`: logical: if plot is TRUE, the labels are printed at the points and if FALSE they are omitted.
- `atpen`: logical: if TRUE and plot = TRUE, the lower-left corners of the labels are plotted at the points clicked rather than relative to the points.
- `offset`: the distance (in character widths) which separates the label from identified points. Ignored if atpen = TRUE or pos = 0.
- `tolerance`: the maximal distance (in inches) for the pointer to be ‘close enough’ to a point.
- `...`: further arguments passed to `par` such as cex, col and font.
Details

identify is only supported on screen devices such as X11, windows and quartz. On other devices the call will do nothing.

If plot is TRUE, the point is labelled with the corresponding element of text. The labels are placed below, to the left, above or to the right of the identified point, depending on where the cursor was relative to the point.

The identification process is terminated by clicking the second button and selecting ‘Stop’ from the menu, or from the ‘Stop’ menu on the graphics window.

On most devices which support identify, successful selection of a point is indicated by a bell sound unless options(locatorBell = FALSE) has been set.

If the window is resized or hidden and then exposed before the identification process has terminated, any labels drawn by identify will disappear. These will reappear once the identification process has terminated and the window is resized or hidden and exposed again. This is because the labels drawn by identify are not recorded in the device’s display list until the identification process has terminated.

Value

If pos is FALSE, an integer vector containing the indexes of the identified points.

If pos is TRUE, a list containing a component ind, indicating which points were identified and a component pos, indicating where the labels were placed relative to the identified points (1=below, 2=left, 3=above, 4=right and 0=no offset, used if atpen = TRUE).

References


See Also

locator

---

**image**

*Display a Color Image*

Description

Creates a grid of colored or gray-scale rectangles with colors corresponding to the values in z. This can be used to display three-dimensional or spatial data aka “images”. This is a generic function.

The functions `heat.colors`, `terrain.colors` and `topo.colors` create heat-spectrum (red to white) and topographical color schemes suitable for displaying ordered data, with n giving the number of colors desired.

Usage

```r
image(x, ...) 
```

## Default S3 method:
```r
default = FALSE, xaxs = "i", yaxs = "i", xlab, ylab, 
breaks, oldstyle = FALSE, ...) 
```
Arguments

\(x, y\)  
locations of grid lines at which the values in \(z\) are measured. These must be in (strictly) ascending order. By default, equally spaced values from 0 to 1 are used. If \(x\) is a list, its components \(x_1\) and \(x_2\) are used for \(x\) and \(y\), respectively. If the list has component \(z\) this is used for \(z\).

\(z\)  
a matrix containing the values to be plotted (NAs are allowed). Note that \(x\) can be used instead of \(z\) for convenience.

\(zlim\)  
the minimum and maximum \(z\) values for which colors should be plotted. Each of the given colors will be used to color an equispaced interval of this range. The midpoints of the intervals cover the range, so that values just outside the range will be plotted.

\(xlim, ylim\)  
ranges for the plotted \(x\) and \(y\) values, defaulting to the range of the finite values of \(x\) and \(y\).

\(col\)  
a list of colors such as that generated by rainbow, heat.colors, topo.colors, terrain.colors or similar functions.

\(add\)  
logical; if TRUE, add to current plot (and disregard the following arguments). This is rarely useful because image “paints” over existing graphics.

\(xaxs, yaxs\)  
style of \(x\) and \(y\) axis. The default "i" is appropriate for images. See par.

\(xlab, ylab\)  
each a character string giving the labels for the \(x\) and \(y\) axis. Default to the ‘call names’ of \(x\) or \(y\), or to "" if these where unspecified.

\(breaks\)  
a set of breakpoints for the colours: must give one more breakpoint than colour.

\(oldstyle\)  
logical. If true the midpoints of the colour intervals are equally spaced, and \(zlim[1]\) and \(zlim[2]\) were taken to be midpoints. (This was the default prior to R 1.1.0.) The current default is to have colour intervals of equal lengths between the limits.

\(...)\  
graphical parameters for plot may also be passed as arguments to this function, as can the plot aspect ratio \(asp\) (see plot.window).

Details

The length of \(x\) should be equal to the \(nrow(z)+1\) or \(nrow(z)\). In the first case \(x\) specifies the boundaries between the cells: in the second case \(x\) specifies the midpoints of the cells. Similar reasoning applies to \(y\). It probably only makes sense to specify the midpoints of an equally-spaced grid. If you specify just one row or column and a length-one \(x\) or \(y\), the whole user area in the corresponding direction is filled.

Rectangles corresponding to missing values are not plotted (and so are transparent and (unless \(add=TRUE\)) the default background painted in \(par(*bg*)\) will show though and if that is transparent, the canvas colour will be seen).

If \(breaks\) is specified then \(zlim\) is unused and the algorithm used follows cut, so intervals are closed on the right and open on the left except for the lowest interval.

Notice that image interprets the \(z\) matrix as a table of \(f(x[i], y[j])\) values, so that the \(x\) axis corresponds to row number and the \(y\) axis to column number, with column 1 at the bottom, i.e. a 90 degree counter-clockwise rotation of the conventional printed layout of a matrix.

Note

Based on a function by Thomas Lumley (tlumley@u.washington.edu).
See Also

*filled.contour* or *heatmap* which can look nicer (but are less modular), *contour; heat.colors, topo.colors, terrain.colors, rainbow, hsv, par.*

Examples

```r
x <- y <- seq(-4*pi, 4*pi, len=27)
r <- sqrt(outer(x^2, y^2, "+"))
image(z = z <- cos(r^2)*exp(-r/6), col=gray((0:32)/32))
image(z, axes = FALSE, main = "Math can be beautiful ...",
     xlab = expression(cos(r^2) * e^{-r/6}))
contour(z, add = TRUE, drawlabels = FALSE)

# Volcano data visualized as matrix. Need to transpose and flip
image(t(volcano)[ncol(volcano):1,])

# A prettier display of the volcano
x <- 10*(1:nrow(volcano))
y <- 10*(1:ncol(volcano))
image(x, y, volcano, col = terrain.colors(100), axes = FALSE)
contour(x, y, volcano, levels = seq(90, 200, by = 5),
         add = TRUE, col = "peru")
axis(1, at = seq(100, 800, by = 100))
axis(2, at = seq(100, 600, by = 100))
box()
title(main = "Maunga Whau Volcano", font.main = 4)
```

layout

### Specifying Complex Plot Arrangements

**Description**

`layout` divides the device up into as many rows and columns as there are in matrix `mat`, with the column-widths and the row-heights specified in the respective arguments.

**Usage**

```r
layout(mat, widths = rep(1, ncol(mat)),
       heights = rep(1, nrow(mat)), respect = FALSE)
```

`layout.show(n = 1)`

`lcm(x)`

**Arguments**

- `mat` a matrix object specifying the location of the next `N` figures on the output device. Each value in the matrix must be 0 or a positive integer. If `N` is the largest positive integer in the matrix, then the integers `{1, ..., N - 1}` must also appear at least once in the matrix.

- `widths` a vector of values for the widths of columns on the device. Relative widths are specified with numeric values. Absolute widths (in centimetres) are specified with the `lcm()` function (see examples).
heights  a vector of values for the heights of rows on the device. Relative and absolute heights can be specified, see widths above.
respect  either a logical value or a matrix object. If the latter, then it must have the same dimensions as mat and each value in the matrix must be either 0 or 1.
n  number of figures to plot.
x  a dimension to be interpreted as a number of centimetres.

Details

Figure i is allocated a region composed from a subset of these rows and columns, based on the rows and columns in which i occurs in mat.

The respect argument controls whether a unit column-width is the same physical measurement on the device as a unit row-height.

There is a limit (currently 50) for the numbers of rows and columns in the layout, and also for the total number of cells (500).

layout.show(n) plots (part of) the current layout, namely the outlines of the next n figures.

lcm is a trivial function, to be used as the interface for specifying absolute dimensions for the widths and heights arguments of layout().

Value

layout returns the number of figures, N, see above.

Warnings

These functions are totally incompatible with the other mechanisms for arranging plots on a device: par(mfrow), par(mfcol) and split.screen.

Author(s)

Paul R. Murrell

References


See Also

par with arguments mfrow, mfcol, or mfg.

Examples

def.par <- par(no.readonly = TRUE) # save default, for resetting...

## divide the device into two rows and two columns
## allocate figure 1 all of row 1
## allocate figure 2 the intersection of column 2 and row 2
layout(matrix(c(1,1,0,2), 2, 2, byrow = TRUE))
## show the regions that have been allocated to each plot
layout.show(2)

## divide device into two rows and two columns
## allocate figure 1 and figure 2 as above
## respect relations between widths and heights
nf <- layout(matrix(c(1, 1, 0, 2), 2, 2, byrow=TRUE), respect=TRUE)
layout.show(nf)

## create single figure which is 5cm square
nf <- layout(matrix(1), widths=lcm(5), heights=lcm(5))
layout.show(nf)

##-- Create a scatterplot with marginal histograms -----

x <- pmin(3, pmax(-3, rnorm(50)))
y <- pmin(3, pmax(-3, rnorm(50)))
xhist <- hist(x, breaks=seq(-3, 3, 0.5), plot=FALSE)
yhist <- hist(y, breaks=seq(-3, 3, 0.5), plot=FALSE)
top <- max(c(xhist$counts, yhist$counts))
xrange <- c(-3, 3)
yrange <- c(-3, 3)
nf <- layout(matrix(c(2, 0, 1, 3), 2, 2, byrow=TRUE), c(3, 1), c(1, 3), TRUE)
layout.show(nf)

par(mar=c(3, 3, 1, 1))
plot(x, y, xlim=xrange, ylim=yrange, xlab="", ylab="")
par(mar=c(0, 3, 1, 1))
barplot(xhist$counts, axes=FALSE, ylim=c(0, top), space=0)
par(mar=c(3, 0, 1, 1))
barplot(yhist$counts, axes=FALSE, xlim=c(0, top), space=0, horiz=TRUE)

par(def.par)#- reset to default

---

### legend

Add Legends to Plots

**Description**

This function can be used to add legends to plots. Note that a call to the function `locator` can be used in place of the `x` and `y` arguments.

**Usage**

```r
legend(x, y = NULL, legend, fill = NULL, col = "black",
       lty, lwd, pch,
       angle = 45, density = NULL, bty = "o", bg = par("bg"),
       pt.bg = NA, cex = 1, pt.cex = cex, pt.lwd = lwd,
       xjust = 0, yjust = 1, x.intersp = 1, y.intersp = 1,
       adj = c(0, 0.5), text.width = NULL, text.col = par("col"),
       merge = do.lines && has.pch, trace = FALSE,
       plot = TRUE, ncol = 1, horiz = FALSE, title = NULL,
       inset = 0)
```

**Arguments**

- `x, y` the x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by `xy.coords`: See Details.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>legend</code></td>
<td>a vector of text values or an expression of length ≥ 1, or a call (as resulting from substitute) to appear in the legend.</td>
</tr>
<tr>
<td><code>fill</code></td>
<td>if specified, this argument will cause boxes filled with the specified colors (or shaded in the specified colors) to appear beside the legend text.</td>
</tr>
<tr>
<td><code>col</code></td>
<td>the color of points or lines appearing in the legend.</td>
</tr>
<tr>
<td><code>lty,lwd</code></td>
<td>the line types and widths for lines appearing in the legend. One of these two must be specified for line drawing.</td>
</tr>
<tr>
<td><code>pch</code></td>
<td>the plotting symbols appearing in the legend, either as vector of 1-character strings, or one (multi character) string. Must be specified for symbol drawing.</td>
</tr>
<tr>
<td><code>angle</code></td>
<td>angle of shading lines.</td>
</tr>
<tr>
<td><code>density</code></td>
<td>the density of shading lines, if numeric and positive. If NULL or negative or NA color filling is assumed.</td>
</tr>
<tr>
<td><code>bty</code></td>
<td>the type of box to be drawn around the legend. The allowed values are &quot;o&quot; (the default) and &quot;n&quot;.</td>
</tr>
<tr>
<td><code>bg</code></td>
<td>the background color for the legend box. (Note that this is only used if bty != &quot;n&quot;.)</td>
</tr>
<tr>
<td><code>pt.bg</code></td>
<td>the background color for the points, corresponding to its argument bg.</td>
</tr>
<tr>
<td><code>cex</code></td>
<td>character expansion factor relative to current par(&quot;cex&quot;).</td>
</tr>
<tr>
<td><code>pt.cex</code></td>
<td>expansion factor(s) for the points.</td>
</tr>
<tr>
<td><code>pt.lwd</code></td>
<td>line width for the points, defaults to the one for lines.</td>
</tr>
<tr>
<td><code>xjust</code></td>
<td>how the legend is to be justified relative to the legend x location. A value of 0 means left justified, 0.5 means centered and 1 means right justified.</td>
</tr>
<tr>
<td><code>yjust</code></td>
<td>the same as xjust for the legend y location.</td>
</tr>
<tr>
<td><code>x.intersp</code></td>
<td>character interspacing factor for horizontal (x) spacing.</td>
</tr>
<tr>
<td><code>y.intersp</code></td>
<td>the same for vertical (y) line distances.</td>
</tr>
<tr>
<td><code>adj</code></td>
<td>numeric of length 1 or 2; the string adjustment for legend text. Useful for y-adjustment when labels are plotmath expressions.</td>
</tr>
<tr>
<td><code>text.width</code></td>
<td>the width of the legend text in x (&quot;user&quot;) coordinates. Defaults to the proper value computed by strwidth(legend).</td>
</tr>
<tr>
<td><code>text.col</code></td>
<td>the color used for the legend text.</td>
</tr>
<tr>
<td><code>merge</code></td>
<td>logical; if TRUE, “merge” points and lines but not filled boxes. Defaults to TRUE if there are points and lines.</td>
</tr>
<tr>
<td><code>trace</code></td>
<td>logical; if TRUE, shows how legend does all its magical computations.</td>
</tr>
<tr>
<td><code>plot</code></td>
<td>logical. If FALSE, nothing is plotted but the sizes are returned.</td>
</tr>
<tr>
<td><code>ncol</code></td>
<td>the number of columns in which to set the legend items (default is 1, a vertical legend).</td>
</tr>
<tr>
<td><code>horiz</code></td>
<td>logical; if TRUE, set the legend horizontally rather than vertically (specifying horiz overrides the ncol specification).</td>
</tr>
<tr>
<td><code>title</code></td>
<td>a text value giving a title to be placed at the top of the legend.</td>
</tr>
<tr>
<td><code>inset</code></td>
<td>inset distance(s) from the margins as a fraction of the plot region when legend is placed by keyword.</td>
</tr>
</tbody>
</table>
Details

Arguments \( x, y, \) legend are interpreted in a non-standard way to allow the coordinates to be specified \textit{via} one or two arguments. If legend is missing and \( y \) is not numeric, it is assumed that the second argument is intended to be legend and that the first argument specifies the coordinates. The coordinates can be specified in any way which is accepted by \texttt{xy.coords}. If this gives the coordinates of one point, it is used as the top-left coordinate of the rectangle containing the legend. If it gives the coordinates of two points, these specify opposite corners of the rectangle (either pair of corners, in any order).

The location may also be specified by setting \( x \) to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location. Partial argument matching is used. The optional \texttt{inset} argument specifies how far the legend is inset from the plot margins. If a single value is given, it is used for both margins; if two values are given, the first is used for \( x \)-distance, the second for \( y \)-distance.

"Attribute" arguments such as \texttt{col}, \texttt{pch}, \texttt{lty}, etc, are recycled if necessary. \texttt{merge} is not.

Points are drawn after lines in order that they can cover the line with their background color \texttt{pt.bg}, if applicable.

See the examples for how to right-justify labels.

Value

A list with list components

\begin{itemize}
  \item \texttt{rect} a list with components
    \begin{itemize}
      \item \texttt{w, h} positive numbers giving width and height of the legend’s box.
      \item \texttt{left, top} \( x \) and \( y \) coordinates of upper left corner of the box.
    \end{itemize}
  \item \texttt{text} a list with components
    \begin{itemize}
      \item \texttt{x, y} numeric vectors of length \( \text{length(legend)} \), giving the \( x \) and \( y \) coordinates of the legend’s text(s).
    \end{itemize}
\end{itemize}

returned invisibly.

References


See Also

\texttt{plot}, \texttt{barplot} which uses \texttt{legend()}, and \texttt{text} for more examples of math expressions.

Examples

\begin{verbatim}
## Run the example in 'matplot' or the following:
leg.txt <- c("Setosa Petals", "Setosa Sepals", "Versicolor Petals", "Versicolor Sepals")
y.leg <- c(4.5, 3, 2.1, 1.4, .7)
cexv <- c(1.2, 1, 4/5, 2/3, 1/2)
matplot(c(1,8), c(0,4.5), type = "n", xlab = "Length", ylab = "Width",
       main = "Petal and Sepal Dimensions in Iris Blossoms")
for (i in seq(cexv)) {
text (1, y.leg[i]-.1, paste("cex=",formatC(cexv[i])), cex=.8, adj = 0)
}
\end{verbatim}
legend(3, y.leg[i], leg.txt, pch = "sSvV", col = c(1, 3), cex = cexv[i])

## 'merge = TRUE' for merging lines & points:
x <- seq(-pi, pi, len = 65)
plot(x, sin(x), type = "l", ylim = c(-1.2, 1.8), col = 3, lty = 2)
points(x, cos(x), pch = 3, col = 4)
lines(x, tan(x), type = "b", lty = 1, pch = 4, col = 6)
title("legend(..., lty = c(2, -1, 1), pch = c(-1,3,4), merge = TRUE)",
cex.main = 1.1)
legend(-1, 1.9, c("sin", "cos", "tan"), col = c(3,4,6), text.col = "green4",
lty = c(2, -1, 1), pch = c(-1, 3, 4), merge = TRUE, bg='gray90')

## right-justifying a set of labels: thanks to Uwe Ligges
x <- 1:5; y1 <- 1/x; y2 <- 2/x
plot(rep(x, 2), c(y1, y2), type="n", xlab="x", ylab="y")
lines(x, y1); lines(x, y2, lty=2)
temp <- legend("topright", legend = c("", " "),
    text.width = strwidth("1,000,000"),
    lty = 1:2, xjust = 1, yjust = 1,
    title = "Line Types")
text(temp$rect$left + temp$text$y,
    c("1,000", "1,000,000"), adj=c(0,.3))

##--- log scaled Examples ------------------------------
leg.txt <- c("a one", "a two")
par(mfrow = c(2,2))
for(ll in c("", "x", "y", "xy") ) {
 plot(2:10, log=ll, main=paste("log = ",ll,"", sep=""))
  abline(1,1)
  lines(2:3,3:4, col=2) #
  points(2,2, col=3) #
  rect(2,3,2, col=4)
  text(c(3,3),2:3,
       c("rect(2,3,3,2, col=4)",
       "text(c(3,3),2:3,""c(rect(...))""), adj = c(0,0.3))
  legend(list(x=2,y=8), legend = leg.txt, col=2:3, pch=1:2,
lty=1, merge=TRUE)#, trace=TRUE)
}
par(mfrow=c(1,1))

##-- Math expressions: ------------------------------
x <- seq(-pi, pi, len = 65)
plot(x, sin(x), type="l", col = 2, xlab = expression(phi),
    ylab = expression(f(phi)))
abline(h=-1:1, v=pi/2*(-6:6), col="gray90")
lines(x, cos(x), col = 3, lty = 2)
ex.cs1 <- expression(plain(sin) * phi, paste("cos", phi)))# 2 ways
utils::str(legend(-3, .9, ex.cs1, lty=1:2, plot=FALSE,
    adj = c(0,.6)))
legend(-3, .9, ex.cs1, lty=1:2, col=2:3, adj = c(0, .6))
x <- rexp(100, rate = .5)
hist(x, main = "Mean and Median of a Skewed Distribution")
abline(v = mean(x), col=2, lty=2, lwd=2)
abline(v = median(x), col=3, lty=3, lwd=2)
ex12 <- expression(bar(x) == sum(over(x[i], n), i==1, n),
    adj = c(0, .6))
Add Connected Line Segments to a Plot

Description
A generic function taking coordinates given in various ways and joining the corresponding points with line segments.

Usage
lines(x, ...)
## Default S3 method:
lines(x, y = NULL, type = "l", col = par("col"),
     lty = par("lty"), ...)  

### Arguments

- **x**, **y**: coordinate vectors of points to join.
- **type**: character indicating the type of plotting; actually any of the types as in `plot`.
- **col**: color to use. This can be vector of length greater than one, but only the first value will be used.
- **lty**: line type to use.
- **...**: Further graphical parameters (see `par`) may also be supplied as arguments, particularly, line type, *lty* and line width, *lwd*.

### Details

The coordinates can be passed to `lines` in a plotting structure (a list with *x* and *y* components), a time series, etc. See `xy.coords`.

The coordinates can contain *NA* values. If a point contains *NA* in either its *x* or *y* value, it is omitted from the plot, and lines are not drawn to or from such points. Thus missing values can be used to achieve breaks in lines.

For `type = "h", col` can be a vector and will be recycled as needed.

### References


### See Also

`points`, particularly for `type %in% c("p","b","o")`, `plot`, and the underlying “primitive” `plot.xy`.

`par` for how to specify colors.

### Examples

```r
# draw a smooth line through a scatter plot
plot(cars, main="Stopping Distance versus Speed")
lines(lowess(cars))
```

### Graphical Input

#### Description

Reads the position of the graphics cursor when the (first) mouse button is pressed.

#### Usage

```r
locator(n = 512, type = "n", ...)
```
Arguments

- **n**: the maximum number of points to locate. Valid values start at 1.
- **type**: One of "n", "p", "l" or "c". If "p" or "c" the points are plotted; if "l" or "c" they are joined by lines.
- **...**: additional graphics parameters used if type != "n" for plotting the locations.

Details

locator is only supported on screen devices such as X11, windows and quartz. On other devices the call will do nothing.

Unless the process is terminated prematurely by the user (see below) at most n positions are determined.

The identification process can be terminated by clicking the second button and selecting ‘Stop’ from the menu, or from the ‘Stop’ menu on the graphics window.

The current graphics parameters apply just as if plot.default has been called with the same value of type. The plotting of the points and lines is subject to clipping, but locations outside the current clipping rectangle will be returned.

On most devices which support locator, successful selection of a point is indicated by a bell sound unless options(locatorBell=FALSE) has been set.

If the window is resized or hidden and then exposed before the input process has terminated, any lines or points drawn by locator will disappear. These will reappear once the input process has terminated and the window is resized or hidden and exposed again. This is because the points and lines drawn by locator are not recorded in the device’s display list until the input process has terminated.

Value

A list containing x and y components which are the coordinates of the identified points in the user coordinate system, i.e., the one specified by par("usr").

References


See Also

- identify

Description

Plot the columns of one matrix against the columns of another.
Usage

```r
matplot(x, y, type = "p", lty = 1:5, lwd = 1, pch = NULL,
        col = 1:6, cex = NULL, xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
        ..., add = FALSE, verbose = getOption("verbose"))
```

```r
matpoints(x, y, type = "p", lty = 1:5, lwd = 1, pch = NULL,
         col = 1:6, ...)
```

```r
matlines(x, y, type = "l", lty = 1:5, lwd = 1, pch = NULL,
          col = 1:6, ...)
```

Arguments

- **x**, **y**: vectors or matrices of data for plotting. The number of rows should match. If one of them are missing, the other is taken as y and an x vector of 1:n is used. Missing values (NA s) are allowed.
- **type**: character string (length 1 vector) or vector of 1-character strings indicating the type of plot for each column of y, see `plot` for all possible types. The first character of type defines the first plot, the second character the second, etc. Characters in type are cycled through; e.g., "pl" alternately plots points and lines.
- **lty**, **lwd**: vector of line types and widths. The first element is for the first column, the second element for the second column, etc., even if lines are not plotted for all columns. Line types will be used cyclically until all plots are drawn.
- **pch**: character string or vector of 1-characters or integers for plotting characters, see `points`. The first character is the plotting-character for the first plot, the second for the second, etc. The default is the digits (1 through 9, 0) then the letters.
- **col**: vector of colors. Colors are used cyclically.
- **cex**: vector of character expansion sizes, used cyclically. This works as a multiple of `par("cex")`. NULL is equivalent to 1.0.
- **xlab**, **ylab**: titles for x and y axes, as in `plot`.
- **xlim**, **ylim**: ranges of x and y axes, as in `plot`.
- **...**: Graphical parameters (see `par`) and any further arguments of `plot`, typically `plot.default`, may also be supplied as arguments to this function. Hence, the high-level graphics control arguments described under `par` and the arguments to `title` may be supplied to this function.
- **add**: logical. If TRUE, plots are added to current one, using `points` and `lines`.
- **verbose**: logical. If TRUE, write one line of what is done.

Details

Points involving missing values are not plotted.

The first column of x is plotted against the first column of y, the second column of x against the second column of y, etc. If one matrix has fewer columns, plotting will cycle back through the columns again. (In particular, either x or y may be a vector, against which all columns of the other argument will be plotted.)

The first element of `col`, `cex`, `lty`, `lwd` is used to plot the axes as well as the first line. Because plotting symbols are drawn with lines and because these functions may be changing the line style, you should probably specify `lty=1` when using plotting symbols.
Side Effects

Function `matplot` generates a new plot; `matpoints` and `matlines` add to the current one.

References


See Also

`plot.points, lines, matrix.plot`

Examples

```r
matplot((-4:5)^2, main = "Quadratic") # almost identical to plot(*)
sines <- outer(1:20, 1:4, function(x, y) sin(x / 20 * pi * y))
matplot(sines, pch = 1:4, type = "o", col = rainbow(ncol(sines)))

x <- 0:50/50
matplot(x, outer(x, 1:8, function(x, k) sin(k * pi * x)),
        ylim = c(-2,2), type = "plobcsSh",
        main = "matplot(type = \"plobcsSh\")")
## pch & type = vector of 1-chars :
matplot(x, outer(x, 1:4, function(x, k) sin(k * pi * x)),
        pch = letters[1:4], type = c("b","p","o"))

table(iris$Species) # is data.frame with 'Species' factor
iS <- iris$Species == "setosa"
iV <- iris$Species == "versicolor"
op <- par(bg = "bisque")
matplot(c(1, 8), c(0, 4.5), type = "n", xlab = "Length", ylab = "Width",
        main = "Petal and Sepal Dimensions in Iris Blossoms")
matpoints(iris[iS,c(1,3)], iris[iS,c(2,4)], pch = "sS", col = c(2,4))
matpoints(iris[iV,c(1,3)], iris[iV,c(2,4)], pch = "vV", col = c(2,4))
legend(1, 4, c(" Setosa Petals", " Versicolor Petals", " Setosa Sepals",
               " Versicolor Sepals"),
       pch = "sSvV", col = rep(c(2,4), 2))

nam.var <- colnames(iris)[-5]
nam.spec <- as.character(iris[1+50:0:2, "Species"])
iris.S <- array(NA, dim = c(50,4,3), dimnames = list(NULL, nam.var, nam.spec))
for(i in 1:3) iris.S[,i] <- data.matrix(iris[1:50+50*(i-1), -5])

         col = rainbow(3, start = .8, end = .1),
         sub = paste(c("S", "C", "V"), dimnames(iris.S)[[3]],
                     sep = "="), collapse= "",
         main = "Fisher's Iris Data")
par(op)
```
mosaicplot

Mosaic Plots

Description

Plots a mosaic on the current graphics device.

Usage

mosaicplot(x, ...)

## Default S3 method:
mosaicplot(x, main = deparse(substitute(x)),
            sub = NULL, xlab = NULL, ylab = NULL,
            sort = NULL, off = NULL, dir = NULL,
            color = NULL, shade = FALSE, margin = NULL,
            cex.axis = 0.66, las = par("las"),
            type = c("pearson", "deviance", "FT"), ...)

## S3 method for class 'formula':
mosaicplot(formula, data = NULL, ...,
            main = deparse(substitute(data)), subset,
            na.action = stats::na.omit)

Arguments

x
  a contingency table in array form, with optional category labels specified in the
dimnames(x) attribute. The table is best created by the table() command.
main
  character string for the mosaic title.
sub
  character string for the mosaic sub-title (at bottom).
xlab,ylab
  x- and y-axis labels used for the plot; by default, the first and second element of
  names(dimnames(X)) (i.e., the name of the first and second variable in X).
sort
  vector ordering of the variables, containing a permutation of the integers
  1:length(dim(x)) (the default).
off
  vector of offsets to determine percentage spacing at each level of the mosaic
  (appropriate values are between 0 and 20, and the default is 20 times the number
  of splits for 2-dimensional tables, and 10 otherwise. Rescaled to maximally 50,
  and recycled if necessary.
dir
  vector of split directions ("v" for vertical and "h" for horizontal) for each level
  of the mosaic, one direction for each dimension of the contingency table. The
  default consists of alternating directions, beginning with a vertical split.
color
  logical or (recycling) vector of colors for color shading, used only when shade
  is FALSE, or NULL (default). By default, grey boxes are drawn. color=TRUE
  uses a gamma-corrected grey palette. color=FALSE gives empty boxes with
  no shading.
shade
  a logical indicating whether to produce extended mosaic plots, or a numeric
  vector of at most 5 distinct positive numbers giving the absolute values of the
  cut points for the residuals. By default, shade is FALSE, and simple mosaics
  are created. Using shade = TRUE cuts absolute values at 2 and 4.
mosaicplot

margin
a list of vectors with the marginal totals to be fit in the log-linear model. By
default, an independence model is fitted. See \texttt{loglin} for further information.

cex.axis
The magnification to be used for axis annotation, as a multiple of\texttt{par("cex")}.

las
numeric; the style of axis labels, see \texttt{par}.

type
a character string indicating the type of residual to be represented. Must be one
of "pearson" (giving components of Pearson's $\chi^2$), "deviance" (giving
components of the likelihood ratio $\chi^2$), or "FT" for the Freeman-Tukey residu-
als. The value of this argument can be abbreviated.

formula
a formula, such as $y \sim x$.

data
a data frame (or list), or a contingency table from which the variables in
formula should be taken.

... 
进一步的参数将被传递到方法，或者

subset
an optional vector specifying a subset of observations in the data frame to be
used for plotting.

na.action
a function which indicates what should happen when the data contains variables
to be cross-tabulated, and these variables contain NAs. The default is to omit
cases which have an NA in any variable. Since the tabulation will omit all cases
containing missing values, this will only be useful if the \texttt{na.action} function
replaces missing values.

Details

This is a generic function. It currently has a default method (\texttt{mosaicplot.default}) and a
formula interface (\texttt{mosaicplot.formula}).

Extended mosaic displays show the standardized residuals of a loglinear model of the counts from
by the color and outline of the mosaic's tiles. (Standardized residuals are often referred to a stan-
dard normal distribution.) Negative residuals are drawn in shaded of red and with broken outlines;
positive ones are drawn in blue with solid outlines.

For the formula method, if data is an object inheriting from classes "table" or "ftable", or
an array with more than 2 dimensions, it is taken as a contingency table, and hence all entries should
be nonnegative. In this case, the left-hand side of formula should be empty, and the variables on
the right-hand side should be taken from the names of the dimnames attribute of the contingency
table. A marginal table of these variables is computed, and a mosaic of this table is produced.

Otherwise, data should be a data frame or matrix, list or environment containing the variables
to be cross-tabulated. In this case, after possibly selecting a subset of the data as specified by the
subset argument, a contingency table is computed from the variables given in formula, and a
mosaic is produced from this.

See Emerson (1998) for more information and a case study with television viewer data from Nielsen
Media Research.

Missing values are not supported except via an \texttt{na.action} function when data contains vari-
ables to be cross-tabulated.

Author(s)

S-PLUS original by John Emerson \{emerson@stat.yale.edu\}. Originally modified and enhanced for
\texttt{R} by KH.
References


The home page of Michael Friendly ([http://www.math.yorku.ca/SCS/friendly.html](http://www.math.yorku.ca/SCS/friendly.html)) provides information on various aspects of graphical methods for analyzing categorical data, including mosaic plots.

See Also

`assocplot`, `loglin`.

Examples

```r
mosaicplot(Titanic, main = "Survival on the Titanic", color = TRUE)
```

```r
## Formula interface for tabulated data:
mosaicplot(~ Sex + Age + Survived, data = Titanic, color = TRUE)
```

```r
mosaicplot(HairEyeColor, shade = TRUE)
```

```r
## Independence model of hair and eye color and sex. Indicates that
## there are significantly more blue eyed blonde females than expected
## in the case of independence (and too few brown eyed blonde females).
```

```r
mosaicplot(HairEyeColor, shade = TRUE, margin = list(c(1,2), 3))
## Model of joint independence of sex from hair and eye color. Males
## are underrepresented among people with brown hair and eyes, and are
## overrepresented among people with brown hair and blue eyes, but not
## "significantly".
```

```r
## Formula interface for raw data: visualize crosstabulation of numbers
## of gears and carburettors in Motor Trend car data.
mosaicplot(~ gear + carb, data = mtcars, color = TRUE, las = 1)
```

```r
# color recycling
mosaicplot(~ gear + carb, data = mtcars, color = 2:3, las = 1)
```

---

**mtext**

*Write Text into the Margins of a Plot*

**Description**

Text is written in one of the four margins of the current figure region or one of the outer margins of the device region.

**Usage**

```r
mtext(text, side = 3, line = 0, outer = FALSE, at = NA,
      adj = NA, padj = NA, cex = NA, col = NA, font = NA, vfont = NULL, ...)```

**Arguments**

- **text**: one or more character strings or expressions.
- **side**: on which side of the plot (1=bottom, 2=left, 3=top, 4=right).
- **line**: on which margin line, starting at 0 counting outwards.
- **outer**: use outer margins if available.
- **at**: give location in user-coordinates. If `length(at)==0` (the default), the location will be determined by **adj**.
- **adj**: adjustment for each string in reading direction. For strings parallel to the axes, `adj=0` means left or bottom alignment, and `adj=1` means right or top alignment. If `adj` is not a finite value (the default), the value of `par("las")` determines the adjustment. For strings plotted parallel to the axis the default is to centre the string.
- **padj**: adjustment for each string perpendicular to the reading direction (which is controlled by **adj**). For strings parallel to the axes, `padj=0` means right or top alignment, and `padj=1` means left or bottom alignment. If `padj` is not a finite value (the default), the value of `par("las")` determines the adjustment. For strings plotted perpendicular to the axis the default is to centre the string.

Further graphical parameters (see `text` and `par`); currently supported are:

- **cex**: character expansion factor. NULL and NA are equivalent to 1.0. This is an absolute measure, not scaled by `par("cex")` or by setting `par("mfrow")` or `par("mfcol")`.
- **col**: color to use.
- **font**: font for text.
- **vfont**: vector font for text.

**Details**

The “user coordinates” in the outer margins always range from zero to one, and are not affected by the user coordinates in the figure region(s) — R is differing here from other implementations of S.

The arguments **side**, **line**, **at**, **at**, **adj**, the further graphical parameters and even **outer** can be vectors, and recycling will take place to plot as many strings as the longest of the vector arguments. Note that a vector **adj** has a different meaning from **text**.

**adj = 0.5** will centre the string, but for **outer=TRUE** on the device region rather than the plot region.

Parameter **las** will determine the orientation of the string(s). For strings plotted perpendicular to the axis the default justification is to place the end of the string nearest the axis on the specified line. (Note that this differs from S, which uses **srt** if **at** is supplied and **las** if it is not.)

Note that if the text is to be plotted perpendicular to the axis, **adj** determines the justification of the string *and* the position along the axis unless **at** is specified.

**Side Effects**

The given text is written onto the current plot.

**References**

See Also

title, text, plot, par; plotmath for details on mathematical annotation.

Examples

plot(1:10, (-4:5)^2, main="Parabola Points", xlab="xlab")
mtext("10 of them")
for(s in 1:4)
  mtext(paste("mtext(..., line=-1, (side, col, font) = ",s,";
  ", cex = ", (1+s)/2, ",")", line = -1,
  side=s, col=s, font=s, cex= (1+s)/2)
mtext("mtext(..., line=-2)", line = -2)
mtext("mtext(..., line=-2, adj=0)", line = -2, adj=0)
##--- log axis :
plot(1:10, exp(1:10), log='y', main="log='y'", xlab="xlab")
for(s in 1:4) mtext(paste("mtext(...,side=",s,")"), side=s)

pairs

Scatterplot Matrices

Description

A matrix of scatterplots is produced.

Usage

pairs(x, ...)

## S3 method for class 'formula':
pairs(formula, data = NULL, ..., subset,
  na.action = stats::na.pass)

## Default S3 method:
pairs(x, labels, panel = points, ...,
  lower.panel = panel, upper.panel = panel,
  diag.panel = NULL, text.panel = textPanel,
  label.pos = 0.5 + has.diag/3,
  cex.labels = NULL, font.labels = 1,
  rowlattop = TRUE, gap = 1)

Arguments

x

the coordinates of points given as columns of a numeric matrix. Other objects
such as data frames will if possible be converted by data.matrix.

formula

a formula, such as ~ x + y + z. Each term will give a separate variable in
the pairs plot, so terms should be numeric vectors. (A response will be inter-
preted as another variable, but not treated specially, so it is confusing to use
one.)

data

a data.frame (or list) from which the variables in formula should be taken.

subset

an optional vector specifying a subset of observations to be used for plotting.
pairs

na.action a function which indicates what should happen when the data contain NAs. The default is to pass missing values on to the panel functions, but na.action = na.omit will cause cases with missing values in any of the variables to be omitted entirely.

labels the names of the variables.

panel function(x,y,...) which is used to plot the contents of each panel of the display.

... arguments to be passed to or from methods.

Also, graphical parameters can be given as can arguments to plot such as main.par("oma") will be set appropriately unless specified.

lower.panel, upper.panel separate panel functions to be used below and above the diagonal respectively.

diag.panel optional function(x, ...) to be applied on the diagonals.

text.panel optional function(x, y, labels, cex, font, ...) to be applied on the diagonals.

label.pos y position of labels in the text panel.

cex.labels, font.labels graphics parameters for the text panel.

rowlattop logical. Should the layout be matrix-like with row 1 at the top, or graph-like with row 1 at the bottom?

gap Distance between subplots, in margin lines.

Details

The \(i,j\)th scatterplot contains \(x[,i]\) plotted against \(x[,j]\). The “scatterplot” can be customised by setting panel functions to appear as something completely different. The off-diagonal panel functions are passed the appropriate columns of \(x\) as \(x\) and \(y\): the diagonal panel function (if any) is passed a single column, and the text.panel function is passed a single \((x, y)\) location and the column name.

The graphical parameters pch and col can be used to specify a vector of plotting symbols and colors to be used in the plots.

The graphical parameter oma will be set by pairs.default unless supplied as an argument.

A panel function should not attempt to start a new plot, but just plot within a given coordinate system: thus plot and boxplot are not panel functions.

By default, missing values are passed to the panel functions and will often be ignored within a panel. However, for the formula method and na.action = na.omit, all cases which contain a missing values for any of the variables are omitted completely (including when the scales are selected). (The latter was the default behaviour prior to R 2.0.0.)

Author(s)

Enhancements for R 1.0.0 contributed by Dr. Jens Oehlschlaegel-Akiyoshi and R-core members.

References

Examples

pairs(iris[1:4], main = "Anderson's Iris Data -- 3 species",
    pch = 21, bg = c("red", "green3", "blue") [unclass(iris$Species)])

## formula method
pairs(~ Fertility + Education + Catholic, data = swiss,
    subset = Education < 20, main = "Swiss data, Education < 20")
pairs(USJudgeRatings)

## put histograms on the diagonal
panel.hist <- function(x, ...) {
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5))
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col="cyan", ...)
}
pairs(USJudgeRatings[1:5], panel=panel.smooth,
    cex = 1.5, pch = 24, bg="light blue",
    diag.panel=panel.hist, cex.labels = 2, font.labels=2)

## put (absolute) correlations on the upper panels,
## with size proportional to the correlations.
panel.cor <- function(x, y, digits=2, prefix="", cex.cor) {
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y))
  txt <- format(c(r, 0.123456789), digits=digits)[1]
  txt <- paste(prefix, txt, sep="")
  if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
  text(0.5, 0.5, txt, cex = cex * r)
}
pairs(USJudgeRatings, lower.panel=panel.smooth, upper.panel=panel.cor)

panel.smooth

Simple Panel Plot

Description

An example of a simple useful panel function to be used as argument in e.g., coplot or pairs.

Usage

panel.smooth(x, y, col = par("col"), bg = NA, pch = par("pch"),
    cex = 1, col.smooth = "red", span = 2/3, iter = 3,
    ...)
Arguments

- **x, y** numeric vectors of the same length
- **col, bg, pch, cex** numeric or character codes for the color(s), point type and size of points; see also par.
- **col.smooth** color to be used by lines for drawing the smooths.
- **span** smoothing parameter \( f \) for lowess, see there.
- **iter** number of robustness iterations for lowess.
- **...** further arguments to lines.

See Also

coplot and pairs where panel.smooth is typically used; lowess.

Examples

```r
pairs(swiss, panel = panel.smooth, pch = ".") # emphasize the smooths
pairs(swiss, panel = panel.smooth, lwd = 2, cex= 1.5, col="blue") # hmm...
```

Description

par can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to par in tag = value form, or by passing them as a list of tagged values.

Usage

```r
par(..., no.readonly = FALSE)
```

```r
<highlevel plot> (..., <tag> = <value>)
```

Arguments

- **...** arguments in tag = value form, or a list of tagged values. The tags must come from the graphical parameters described below.
- **no.readonly** logical; if TRUE and there are no other arguments, only parameters are returned which can be set by a subsequent par() call.

Details

Parameters are queried by giving one or more character vectors to par.

par() (no arguments) or par(no.readonly=TRUE) is used to get all the graphical parameters (as a named list). Their names are currently taken from the variable .Pars. .Pars.readonly contains the names of the par arguments which are readonly.

R.O. indicates read-only arguments: These may only be used in queries, i.e., they do not set anything.

All but these R.O. and the following low-level parameters can be set as well in high-level and mid-level plot functions, such as plot, points, lines, axis, title, text, mtext:
par

- "ask"
- "family", "fig", "fin"
- "lend", "lheight", "ljoin", "lmitre"
- "mai", "mar", "mex"
- "mfrow", "mfcol", "mfg"
- "new"
- "oma", "ome", "omi"
- "pin", "plt", "ps", "pty"
- "usr"
- "xlog", "ylog"

However, see the comments on bg and cex, which may be taken as arguments to certain plot functions rather than as graphical parameters.

Value

When parameters are set, their former values are returned in an invisible named list. Such a list can be passed as an argument to par to restore the parameter values. Use `par(no.readonly = TRUE)` for the full list of parameters that can be restored.

When just one parameter is queried, the value is a character string. When two or more parameters are queried, the result is a list of character strings, with the list names giving the parameters.

Note the inconsistency: setting one parameter returns a list, but querying one parameter returns a vector.

Graphical Parameters

adj The value of adj determines the way in which text strings are justified. A value of 0 produces left-justified text, 0.5 centered text and 1 right-justified text. (Any value in [0, 1] is allowed, and on most devices values outside that interval will also work.) Note that the adj argument of text also allows adj = c(x, y) for different adjustment in x- and y-direction.

ann If set to FALSE, high-level plotting functions do not annotate the plots they produce with axis and overall titles. The default is to do annotation.

ask logical. If TRUE, the user is asked for input, before a new figure is drawn.

bg The color to be used for the background of plots. A description of how colors are specified is given below. Note that some graphics functions such as plot.default and points have an argument of this name with a different meaning.

bty A character string which determined the type of box which is drawn about plots. If bty is one of "o", "l", "7", "c", "u", or "n" the resulting box resembles the corresponding upper case letter. A value of "n" suppresses the box.

cex A numerical value giving the amount by which plotting text and symbols should be scaled relative to the default. Note that some graphics functions such as plot.default have an argument of this name which multiplies this graphical parameter.

cex.axis The magnification to be used for axis annotation relative to the current setting of cex.

cex.lab The magnification to be used for x and y labels relative to the current setting of cex.

cex.main The magnification to be used for main titles relative to the current setting of cex.

cex.sub The magnification to be used for sub-titles relative to the current setting of cex.

cin R.O.; character size (width, height) in inches.
col  A specification for the default plotting color. A description of how colors are specified is given below.

col.axis The color to be used for axis annotation.

col.lab The color to be used for x and y labels.

col.main The color to be used for plot main titles.

col.sub The color to be used for plot sub-titles.

cra R.O.; size of default character (width, height) in “rasters” (pixels).

crt A numerical value specifying (in degrees) how single characters should be rotated. It is unwise to expect values other than multiples of 90 to work. Compare with srt which does string rotation.

csi R.O.; height of (default sized) characters in inches.

cxy R.O.; size of default character (width, height) in user coordinate units. \text{par("cxy")} is \text{par("cin")}/par("pin") scaled to user coordinates. Note that \text{c(strwidth(ch), strwidth(ch))} for a given string ch is usually much more precise.

din R.O.; the device dimensions, (width, height), in inches.

err (Unimplemented; R is silent when points outside the plot region are not plotted.) The degree of error reporting desired.

family The name of a font family for drawing text. This name is device-independent and gets mapped by each graphics device to a device-specific font description. The default value is "" which means that the default device font will be used. Standard values are "serif", "sans", "mono", and "symbol" and the Hershey font families are also available. Different devices may define others. Some devices will ignore this setting completely.

fg The color to be used for the foreground of plots. This is the default color used for things like axes and boxes around plots. A description of how colors are specified is given below.

fig A numerical vector of the form c(x1, x2, y1, y2) which gives the (NDC) coordinates of the figure region in the display region of the device. If you set this, unlike S, you start a new plot, so to add to an existing plot use new=TRUE as well.

fin The figure region dimensions, (width, height), in inches. If you set this, unlike S, you start a new plot.

font An integer which specifies which font to use for text. If possible, device drivers arrange so that 1 corresponds to plain text, 2 to bold face, 3 to italic and 4 to bold italic. Also, font 5 is expected to be the symbol font, in Adobe symbol encoding.

font.axis The font to be used for axis annotation.

font.lab The font to be used for x and y labels.

font.main The font to be used for plot main titles.

font.sub The font to be used for plot sub-titles.

gamma the gamma correction, see argument gamma to hsv.

lab A numerical vector of the form c(x, y, len) which modifies the way that axes are annotated. The values of x and y give the (approximate) number of tickmarks on the x and y axes and len specifies the label size. The default is c(5, 5, 7). Currently, len is unimplemented.

las numeric in \{0,1,2,3\}; the style of axis labels.

0: always parallel to the axis [default],
1: always horizontal,
2: always perpendicular to the axis,
par

3: always vertical.

Note that other string/character rotation (via argument srt to par) does not affect the axis labels.

lend The line end style. This can be specified as an integer or string: 0 and "round" mean rounded line caps; 1 and "butt" mean butt line caps; 2 and "square" mean square line caps.

lheight The line height multiplier. The height of a line of text (used to vertically space multi-line text) is found by multiplying the current font size both by the current character expansion and by the line height multiplier. Default value is 1.

ljoin The line join style. This can be specified as an integer or string: 0 and "round" mean rounded line joins; 1 and "mitre" mean mitred line joins; 2 and "bevel" mean bevelled line joins.

lmitre The line mitre limit. This controls when mitred line joins are automatically converted into bevelled line joins. The value must be larger than 1 and the default is 10. Not all devices will honour this setting.

lty The line type. Line types can either be specified as an integer (0=blank, 1=solid, 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash", where "blank" uses 'invisible lines' (i.e., doesn't draw them).

Alternatively, a string of up to 8 characters (from c(1:9, "A":"F")) may be given, giving the length of line segments which are alternatively drawn and skipped. See section 'Line Type Specification' below.

lwd The line width, a positive number, defaulting to 1. The interpretation is device-specific, and some devices do not implement line widths less than one.

mai A numerical vector of the form c(bottom, left, top, right) which gives the margin size specified in inches.

mar A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot. The default is c(5, 4, 2) + 0.1.

mex mrx is a character size expansion factor which is used to describe coordinates in the margins of plots. Note that this does not change the font size, rather specifies the size of font used to convert between mar and mai, and between oma and omi.

mfcol, mfrow A vector of the form c(nr, nc). Subsequent figures will be drawn in an nr-by-nc array on the device by columns (mfcol), or rows (mfrow), respectively.

In a layout with exactly two rows and columns the base value of "cex" is reduced by a factor of 0.83: if there are three or more of either rows or columns, the reduction factor is 0.66.

Consider the alternatives, layout and split.screen.

mfg A numerical vector of the form c(i, j) where i and j indicate which figure in an array of figures is to be drawn next (if setting) or is being drawn (if enquiring). The array must already have been set by mfcol or mfrow.

For compatibility with S, the form c(i, j, nr, nc) is also accepted, when nr and nc should be the current number of rows and number of columns. Mismatches will be ignored, with a warning.

mgp The margin line (in mex units) for the axis title, axis labels and axis line. The default is c(3, 1, 0).

mkh The height in inches of symbols to be drawn when the value of pch is an integer. Completely ignored currently.
par

new logical, defaulting to FALSE. If set to TRUE, the next high-level plotting command (actually plot.new) should not clean the frame before drawing "as if it was on a new device".

oma A vector of the form c(bottom, left, top, right) giving the size of the outer margins in lines of text.

omd A vector of the form c(x1, x2, y1, y2) giving the outer margin region in NDC (= normalized device coordinates), i.e., as fraction (in [0,1]) of the device region.

omi A vector of the form c(bottom, left, top, right) giving the size of the outer margins in inches.

pch Either an integer specifying a symbol or a single character to be used as the default in plotting points. See points for possible values and their interpretation.

pin The current plot dimensions, (width,height), in inches.

plt A vector of the form c(x1, x2, y1, y2) giving the coordinates of the plot region as fractions of the current figure region.

ps integer; the pointsize of text and symbols.

pty A character specifying the type of plot region to be used; "s" generates a square plotting region and "m" generates the maximal plotting region.

smo (Unimplemented) a value which indicates how smooth circles and circular arcs should be.

srt The string rotation in degrees. See the comment about crt.

tck The length of tick marks as a fraction of the smaller of the width or height of the plotting region. If tck >= 0.5 it is interpreted as a fraction of the relevant side, so if tck=1 grid lines are drawn. The default setting (tck = NA) is to use tcl = -0.5 (see below).

tcl The length of tick marks as a fraction of the height of a line of text. The default value is -0.5; setting tcl = NA sets tck = -0.01 which is S’ default.

tmag A number specifying the enlargement of text of the main title relative to the other annotating text of the plot.

type character; the default plot type desired, see plot.default (type=...), defaulting to "p".

usr A vector of the form c(x1, x2, y1, y2) giving the extremes of the user coordinates of the plotting region. When a logarithmic scale is in use (i.e., par("xlog") is true, see below), then the x-limits will be 10 ^ par("usr")[1:2]. Similarly for the y-axis.

xaxs A vector of the form c(x1, x2, n) giving the coordinates of the extreme tick marks and the number of intervals between tick-marks when par("xlog") is false. Otherwise, when log coordinates are active, the three values have a different meaning: For a small range, n is negative, and the ticks are as in the linear case, otherwise, n is in 1:3, specifying a case number, and x1 and x2 are the lowest and highest power of 10 inside the user coordinates, 10 ^ par("usr")[1:2]. (The "usr" coordinates are log10-transformed here!)

n=1 will produce tick marks at 10^j for integer j,

n=2 gives marks k10^j with k ∈ {1,5},

n=3 gives marks k10^j with k ∈ {1,2,5}.

See axTicks() for a pure R implementation of this.

xaxs The style of axis interval calculation to be used for the x-axis. Possible values are "r", "i", "e", "s", "d". The styles are generally controlled by the range of data or xlim, if given. Style "r" (regular) first extends the data range by 4 percent and then finds an axis with pretty labels that fits within the range. Style "i" (internal) just finds an axis with pretty labels that fits within the original data range. Style "s" (standard) finds an axis with pretty labels within which the original data range fits. Style "e" (extended) is like style "s", except that it is also
ensured that there is room for plotting symbols within the bounding box. Style "d" (direct) specifies that the current axis should be used on subsequent plots. (Only "e" and "i" styles are currently implemented)

**xaxt** A character which specifies the axis type. Specifying "n" causes an axis to be set up, but not plotted. The standard value is "s": for compatibility with S values "l" and "e" are accepted but are equivalent to "s".

**xlog** logical value (see log in `plot.default`). If TRUE, a logarithmic scale is in use (e.g., after `plot(*, log = "x")`). For a new device, it defaults to FALSE, i.e., linear scale.

**xpd** A logical value or NA. If FALSE, all plotting is clipped to the plot region, if TRUE, all plotting is clipped to the figure region, and if NA, all plotting is clipped to the device region.

**yaxp** A vector of the form c(y1, y2, n) giving the coordinates of the extreme tick marks and the number of intervals between tick-marks unless for log coordinates, see xaxp above.

**yaxs** The style of axis interval calculation to be used for the y-axis. See xaxs above.

**yaxt** A character which specifies the axis type. Specifying "n" causes an axis to be set up, but not plotted.

**ylog** a logical value; see xlog above.

**Color Specification**

Colors can be specified in several different ways. The simplest way is with a character string giving the color name (e.g., "red"). A list of the possible colors can be obtained with the function `colors`. Alternatively, colors can be specified directly in terms of their RGB components with a string of the form "#RRGGBB" where each of the pairs RR, GG, BB consist of two hexadecimal digits giving a value in the range 00 to FF. Colors can also be specified by giving an index into a small table of colors, the `palette`. This provides compatibility with S. Index 0 corresponds to the background color.

Additionally, "transparent" or (integer) NA is transparent, useful for filled areas (such as the background!), and just invisible for things like lines or text.

The functions `rgb`, `hsv`, `gray` and `rainbow` provide additional ways of generating colors.

**Line Type Specification**

Line types can either be specified by giving an index into a small built in table of line types (1 = solid, 2 = dashed, etc, see `lty` above) or directly as the lengths of on/off stretches of line. This is done with a string of an even number (up to eight) of characters, namely non-zero (hexadecimal) digits which give the lengths in consecutive positions in the string. For example, the string "33" specifies three units on followed by three off and "3313" specifies three units on followed by three off followed by one on and finally three off. The 'units' here are (on most devices) proportional to lwd, and with lwd = 1 are in pixels or points.

The five standard dash-dot line types (`lty = 2:6`) correspond to c("44", "13", "1343", "73", "2262").

Note that NA is not a valid value for `lty`.

**Note**

The effect of restoring all the (settable) graphics parameters as in the examples is hard to predict if the device has been resized. Several of them are attempting to set the same things in different ways, and those last in the alphabet will win. In particular, the settings of `mai`, `mar`, `pin`, `plt` and `pty` interact, as do the outer margin settings, the figure layout and figure region size.
References


See Also

`plot.default` for some high-level plotting parameters; `colors`, `gray`, `rainbow`, `rgb`; `options` for other setup parameters; graphic devices `x11`, `postscript` and setting up device regions by `layout` and `split.screen`.

Examples

```r
op <- par(mfrow = c(2, 2), # 2 x 2 pictures on one plot
        pty = "s") # square plotting region,
        # independent of device size

## At end of plotting, reset to previous settings:
par(op)

## Alternatively,
op <- par(no.readonly = TRUE) # the whole list of settable par's.
## do lots of plotting and par(.) calls, then reset:
par(op)
par("ylog") # FALSE
plot(1 : 12, log = "y")
par("ylog") # TRUE

plot(1:2, xaxs = "i") # 'inner axis' w/o extra space
stopifnot(par("xaxp")[1:2] == 1:2 &&
          par("usr")[1:2] == 1:2)

(nr.prof <-
c(prof.pilots=16,lawyers=11,farmers=10,salesmen=9,physicians=9,
    mechanics=6,policemen=6,managers=6,engineers=5,teachers=4,
    housewives=3,students=3,armed.forces=1))
par(las = 3)
barplot(rbind(nr.prof)) # R 0.63.2: shows alignment problem
par(las = 0)# reset to default

## 'fg' use:
plot(1:12, type = "b", main = "fg' : axes, ticks and box in gray",
     fg = gray(0.7), bty = "7", sub = R.version.string)

ex <- function() {
    old.par <- par(no.readonly = TRUE) # all par settings which
    # could be changed.
    on.exit(par(old.par))
    ## ...
    ## ... do lots of par() settings and plots
    ## ...
    invisible() #-- now, par(old.par) will be executed
}
ex()
```
persp

Perspective Plots

Description

This function draws perspective plots of surfaces over the x–y plane. persp is a generic function.

Usage

persp(x, ...)

## Default S3 method:
persp(x = seq(0, 1, len = nrow(z)), y = seq(0, 1, len = ncol(z)), z,
xlim = range(x), ylim = range(y), zlim = range(z, na.rm = TRUE),
lab = NULL, ylab = NULL, zlab = NULL, main = NULL, sub = NULL,
theta = 0, phi = 15, r = sqrt(3), d = 1, scale = TRUE,
expand = 1, col = "white", border = NULL, ltheta = -135, lphi = 0,
shade = NA, box = TRUE, axes = TRUE, nticks = 5,
ticktype = "simple", ...)

Arguments

x, y locations of grid lines at which the values in z are measured. These must be in ascending order. By default, equally spaced values from 0 to 1 are used. If x is a list, its components x$x and x$y are used for x and y, respectively.

z a matrix containing the values to be plotted (NA s are allowed). Note that x can be used instead of z for convenience.

xlim, ylim, zlim x-, y- and z-limits. The plot is produced so that the rectangular volume defined by these limits is visible.

xlab, ylab, zlab titles for the axes. N.B. These must be character strings; expressions are not accepted. Numbers will be coerced to character strings.

main, sub main and sub title, as for title.

theta, phi angles defining the viewing direction. theta gives the azimuthal direction and phi the colatitude.

r the distance of the eyepoint from the centre of the plotting box.

d a value which can be used to vary the strength of the perspective transformation. Values of d greater than 1 will lessen the perspective effect and values less and 1 will exaggerate it.

scale before viewing the x, y and z coordinates of the points defining the surface are transformed to the interval [0,1]. If scale is TRUE the x, y and z coordinates are transformed separately. If scale is FALSE the coordinates are scaled so that aspect ratios are retained. This is useful for rendering things like DEM information.

expand a expansion factor applied to the z coordinates. Often used with 0 < expand < 1 to shrink the plotting box in the z direction.

col the color(s) of the surface facets. Transparent colours are ignored. This is recycled to the (nx – 1)(ny – 1) facets.
persp

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border</td>
<td>the color of the line drawn around the surface facets. A value of NA will disable the drawing of borders. This is sometimes useful when the surface is shaded.</td>
</tr>
<tr>
<td>ltheta, lphi</td>
<td>if finite values are specified for ltheta and lphi, the surface is shaded as though it was being illuminated from the direction specified by azimuth ltheta and colatitude lphi.</td>
</tr>
<tr>
<td>shade</td>
<td>the shade at a surface facet is computed as (((1+d)/2)^\text{shade}), where (d) is the dot product of a unit vector normal to the facet and a unit vector in the direction of a light source. Values of shade close to one yield shading similar to a point light source model and values close to zero produce no shading. Values in the range 0.5 to 0.75 provide an approximation to daylight illumination.</td>
</tr>
<tr>
<td>box</td>
<td>should the bounding box for the surface be displayed. The default is TRUE.</td>
</tr>
<tr>
<td>axes</td>
<td>should ticks and labels be added to the box. The default is TRUE. If box is FALSE then no ticks or labels are drawn.</td>
</tr>
<tr>
<td>ticktype</td>
<td>character: &quot;simple&quot; draws just an arrow parallel to the axis to indicate direction of increase; &quot;detailed&quot; draws normal ticks as per 2D plots.</td>
</tr>
<tr>
<td>nticks</td>
<td>the (approximate) number of tick marks to draw on the axes. Has no effect if ticktype is &quot;simple&quot;.</td>
</tr>
<tr>
<td>...</td>
<td>additional graphical parameters (see par).</td>
</tr>
</tbody>
</table>

Details

The plots are produced by first transforming the coordinates to the interval [0,1]. The surface is then viewed by looking at the origin from a direction defined by theta and phi. If theta and phi are both zero the viewing direction is directly down the negative y axis. Changing theta will vary the azimuth and changing phi the colatitude.

There is a hook called "persp" (see setHook) called after the plot is completed, which is used in the testing code to annotate the plot page. The hook function(s) are called with no argument.

Notice that persp interprets the z matrix as a table of \(f(x[i], y[j])\) values, so that the x axis corresponds to row number and the y axis to column number, with column 1 at the bottom, so that with the standard rotation angles, the top left corner of the matrix is displayed at the left hand side, closest to the user.

Value

persp() returns the viewing transformation matrix, say VT, a 4 x 4 matrix suitable for projecting 3D coordinates \((x, y, z)\) into the 2D plane using homogenous 4D coordinates \((x, y, z, t)\). It can be used to superimpose additional graphical elements on the 3D plot, by lines() or points(), using the simple function trans3d().

References


See Also

contour and image; trans3d.
Examples

## More examples in demo(persp) !!
-------------

# (1) The Obligatory Mathematical surface.
# Rotated sinc function.
x <- seq(-10, 10, length=30)
y <- x
f <- function(x,y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
z <- outer(x, y, f)
z[is.na(z)] <- 1
op <- par(bg = "white")
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue",
     ltheta = 120, shade = 0.75, ticktype = "detailed",
     xlab = "X", ylab = "Y", zlab = "Sinc( r )"
) -> res
round(res, 3)

# (2) Add to existing persp plot - using trans3d() :
xE <- c(-10,10); xy <- expand.grid(xE, xE)
points(trans3d(xy[,1], xy[,2], 6, pm = res), col = 2, pch =16)
lines (trans3d(x, y=10, z= 6 + sin(x), pm = res), col = 3)
phi <- seq(0, 2*pi, len = 201)
rl <- 7.725 # radius of 2nd maximum
xr <- rl * cos(phi)
yr <- rl * sin(phi)
lines(trans3d(xr,yr, f(xr,yr), res), col = "pink", lwd=2)## (no hidden lines)

# (3) Visualizing a simple DEM model
z <- 2 * volcano # Exaggerate the relief
x <- 10 * (1:nrow(z)) # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z)) # 10 meter spacing (E to W)
## Don't draw the grid lines : border = NA
par(bg = "slategray")
persp(x, y, z, theta = 135, phi = 30, col = "green3", scale = FALSE,
     ltheta = -120, shade = 0.75, border = NA, box = FALSE)
par(op)

pie

Pie Charts

Description

Draw a pie chart.

Usage

pie(x, labels = names(x), edges = 200, radius = 0.8,
clockwise = FALSE, init.angle = if(clockwise) 90 else 0,
Arguments

- **x**: A vector of positive quantities. The values in x are displayed as the areas of pie slices.
- **labels**: A vector of character strings giving names for the slices. For empty or NA labels, no pointing line is drawn either.
- **edges**: The circular outline of the pie is approximated by a polygon with this many edges.
- **radius**: The pie is drawn centered in a square box whose sides range from −1 to 1. If the character strings labeling the slices are long it may be necessary to use a smaller radius.
- **clockwise**: Logical indicating if slices are drawn clockwise or counter-clockwise (i.e., mathematically positive direction), the latter is default.
- **init.angle**: Number specifying the starting angle (in degrees) for the slices. Defaults to 0 (i.e., '3 o'clock') unless clockwise is true where init.angle defaults to 90 (degrees), (i.e., '12 o'clock').
- **density**: The density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.
- **angle**: The slope of shading lines, given as an angle in degrees (counter-clockwise).
- **col**: A vector of colors to be used in filling or shading the slices. If missing a set of 6 pastel colours is used, unless density is specified when par("fg") is used.
- **border, lty** (possibly vectors) arguments passed to polygon which draws each slice.
- **main**: An overall title for the plot.
- **...**: Graphical parameters can be given as arguments to pie. They will affect the main title and labels only.

Note

Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.

Cleveland (1985), page 264: “Data that can be shown by pie charts always can be shown by a dot chart. This means that judgements of position along a common scale can be made instead of the less accurate angle judgements.” This statement is based on the empirical investigations of Cleveland and McGill as well as investigations by perceptual psychologists.

Prior to R 1.5.0 this was known as piechart, which is the name of a Trellis function, so the name was changed to be compatible with S.

References


See Also
dotchart.

Examples

```r
pie(rep(1, 24), col = rainbow(24), radius = 0.9)

pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
names(pie.sales) <- c("Blueberry", "Cherry", "Apple", "Boston Cream", "Other", "Vanilla Cream")
pie(pie.sales) # default colours
pie(pie.sales, col = c("purple", "violetred1", "green3", "cornsilk", "cyan", "white"))
pie(pie.sales, col = gray(seq(0.4, 1.0, length=6)))
pie(pie.sales, clockwise=TRUE, main="pie(*, clockwise=TRUE)")
segments(0,0, 0,1, col= "red", lwd = 2)
text(0,1, "init.angle = 90", col= "red")

n <- 200
pie(rep(1,n), labels="", col=rainbow(n), border=NA,
    main = "pie(*, labels="", col=rainbow(n), border=NA,"")
```
• "S" for other steps, see Details below.
• "n" for no plotting.

All other types give a warning or an error; using, e.g., type = "punkte" being equivalent to type = "p" for S compatibility.

main
an overall title for the plot: see title.
sub
a sub title for the plot: see title.
xlab
a title for the x axis: see title.
ylab
a title for the y axis: see title.

Details

For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.

The two step types differ in their x-y preference: Going from \((x_1, y_1)\) to \((x_2, y_2)\) with \(x_1 < x_2\), type = "s" moves first horizontal, then vertical, whereas type = "S" moves the other way around.

See Also

plot.default, plot.formula and other methods: points, lines, par.

Examples

plot(cars)
lines(lowess(cars))

plot(sin, -pi, 2*pi)

## Discrete Distribution Plot:
plot(table(rpois(100,5)), type = "h", col = "red", lwd=10, main="rpois(100,lambda=5)"

## Simple quantiles/ECDF, see ecdf() {library(stats)} for a better one:
plot(x <- sort(rnorm(47)), type = "s", main = "plot(x, type = "s")")
points(x, cex = .5, col = "dark red")

plot.data.frame  Plot Method for Data Frames

Description

plot.data.frame, a method for the plot generic. It is designed for a quick look at numeric data frames.

Usage

## S3 method for class 'data.frame':
plot(x, ...
Arguments

- **x**: object of class `data.frame`.
- **...**: further arguments to `stripchart`, `plot.default` or `pairs`.

Details

This is intended for data frames with numeric columns. For more than two columns it first calls `data.matrix` to convert the data frame to a numeric matrix and then calls `pairs` to produce a scatterplot matrix. This can fail and may well be inappropriate: for example numerical conversion of dates will lose their special meaning and a warning will be given.

For a two-column data frame it plots the second column against the first by the most appropriate method for the first column.

For a single numeric column it uses `stripchart`, and for other single-column data frames tries to find a plot method for the single column.

See Also

- `data.frame`

Examples

```r
plot(OrchardSprays[1], method="jitter")
plot(OrchardSprays[,c(4,1)])
plot(OrchardSprays)
plot(iris)
plot(iris[,5:4])
plot(women)
```
Arguments

\( x, y \)

the \( x \) and \( y \) arguments provide the \( x \) and \( y \) coordinates for the plot. Any reasonable way of defining the coordinates is acceptable. See the function \texttt{xy.coords} for details.

\texttt{type}

1-character string giving the type of plot desired. The following values are possible, for details, see \texttt{plot}: "p" for points, "l" for lines, "o" for overplotted points and lines, "b", "c") for (empty if "c") points joined by lines, "s" and "S" for stair steps and "h" for histogram-like vertical lines. Finally, "n" does not produce any points or lines.

\texttt{xlim}

the \( x \) limits (min,max) of the plot.

\texttt{ylim}

the \( y \) limits of the plot.

\texttt{log}

a character string which contains "x" if the \( x \) axis is to be logarithmic, "y" if the \( y \) axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.

\texttt{main}

a main title for the plot.

\texttt{sub}

a sub title for the plot.

\texttt{xlab}

a label for the \( x \) axis.

\texttt{ylab}

a label for the \( y \) axis.

\texttt{ann}

a logical value indicating whether the default annotation (title and \( x \) and \( y \) axis labels) should appear on the plot.

\texttt{axes}

a logical value indicating whether axes should be drawn on the plot.

\texttt{frame.plot}

a logical indicating whether a box should be drawn around the plot.

\texttt{panel.first}

an expression to be evaluated after the plot axes are set up but before any plotting takes place. This can be useful for drawing background grids or scatterplot smooths.

\texttt{panel.last}

an expression to be evaluated after plotting has taken place.

\texttt{col}

The colors for lines and points. Multiple colors can be specified so that each point can be given its own color. If there are fewer colors than points they are recycled in the standard fashion. Lines will all be plotted in the first colour specified.

\texttt{bg}

a vector of background colors for open plot symbols, see \texttt{points}. Note: this is \texttt{not} the same setting as \texttt{par ("bg").}

\texttt{pch}

a vector of plotting characters or symbols: see \texttt{points}.

\texttt{cex}

a numerical vector giving the amount by which plotting text and symbols should be scaled relative to the default. This works as a multiple of \texttt{par ("cex").} NULL and \texttt{NA} are equivalent to 1.0.

\texttt{lty}

the line type, see \texttt{par}.

\texttt{lab}

the specification for the (approximate) numbers of tick marks on the \( x \) and \( y \) axes, see \texttt{par}.

\texttt{lwd}

the line width, see \texttt{par}.

\texttt{asp}

the \( y/x \) aspect ratio, see \texttt{plot.window}.

\texttt{...}

other graphical parameters as in \texttt{par} may also be passed as arguments.
References


See Also

`plot.plot.window, xy.coords`

Examples

```r
Speed <- cars$speed
Distance <- cars$dist
plot(Speed, Distance, panel.first = grid(8,8),
     pch = 0, cex = 1.2, col = "blue")
plot(Speed, Distance,
     panel.first = lines(lowess(Speed, Distance), lty = "dashed"),
     pch = 0, cex = 1.2, col = "blue")

## Show the different plot types
x <- 0:12
y <- sin(pi/5 * x)
op <- par(mfrow = c(3,3), mar = .1+ c(2,2,3,1))
for (tp in c("p","l","b", "c","o","h", "s","S","n")) {
  plot(y ~ x, type = tp,
       main = paste("plot( * , type = ",tp,"\")",sep=""))
  if(tp == "S") {
    lines(x,y, type = "s", col = "red", lty = 2)
    mtext("lines( * , type = "s\", ...)", col = "red", cex=.8)
  }
}
par(op)

##--- Log-Log Plot with custom axes
lx <- seq(1,5, length=41)
yl <- expression(e^{-frac(1,2) * {log[10](x)}^2})
y <- exp(-.5*lx^2)
op <- par(mfrow=c(2,1), mar=par("mar") + c(0,1,0,0))
plot(10^lx, y, log="xy", type="l", col="purple",
     main="Log-Log plot", ylab=yl, xlab="x")
plot(10^lx, y, log="xy", type="o", pch='.', col="forestgreen",
     main="Log-Log plot with custom axes", ylab=yl, xlab="x",
     axes = FALSE, frame.plot = TRUE)
axis(1, at = my.at <- 10^(1:5), labels = formatC(my.at, format="fg"))
at.y <- 10^(-5:-1)
axis(2, at = at.y, labels = formatC(at.y, format="fg"), col.axis="red")
par(op)
```

plot.design

*Plot Univariate Effects of a ‘Design’ or Model*
plot.design

Description
Plot univariate effects of one or more factors, typically for a designed experiment as analyzed by aov(). Further, in S this a method of the plot generic function for design objects.

Usage
plot.design(x, y = NULL, fun = mean, data = NULL, ..., ylim = NULL, xlab = "Factors", ylab = NULL, main = NULL, ask = NULL, xaxt = par("xaxt"), axes = TRUE, xtick = FALSE)

Arguments
x
either a data frame containing the design factors and optionally the response, or a formula or terms object.

y
the response, if not given in x.

fun
a function (or name of one) to be applied to each subset. It must return one number for a numeric (vector) input.

data
data frame containing the variables referenced by x when that is formula like.

... graphical arguments such as col, see par.

ylim range of y values, as in plot.default.

xlab x axis label, see title.

ylab y axis label with a “smart” default.

main main title, see title.

ask logical indicating if the user should be asked before a new page is started – in the case of multiple y’s.

xaxt character giving the type of x axis.

axes logical indicating if axes should be drawn.

xtick logical indicating if “ticks” (one per factor) should be drawn on the x axis.

Details
The supplied function will be called once for each level of each factor in the design and the plot will show these summary values. The levels of a particular factor are shown along a vertical line, and the overall value of fun() for the response is drawn as a horizontal line.

This is a new R implementation which will not be completely compatible to the earlier S implementations. This is not a bug but might still change.

Note
A big effort was taken to make this closely compatible to the S version. However, col (and fg) specification has different effects.

Author(s)
Roberto Frisullo and Martin Maechler
plot.factor

Plotting Factor Variables

Description

This function implements a "scatterplot" method for factor arguments of the generic plot function. Actually, boxplot is used when y is numeric and a spineplot when y is a factor. For a single factor x (i.e., with y missing) a simple barplot is produced.

Usage

## S3 method for class 'factor':
plot(x, y, legend.text = NULL, ...)

Arguments

x, y numeric or factor. y may be missing.
legend.text character vector for annotation of y axis, defaults to levels(y). Can be used instead of yaxlabels (for backward compatibility).
... Further arguments to plot, see also par.

Examples

plot.design(warpbreaks)# automatic for data frame with one numeric var.
Form <- breaks ~ wool + tension
summary(fm1 <- aov(Form, data = warpbreaks))
plot.design(
  Form, data = warpbreaks, col = 2)# same as above

## More than one y :
utils::str(esoph)
plot.design(esoph) ## two plots; if interactive you are "ask"ed

## or rather, compare mean and median:
op <- par(mfcol = 1:2)
plot.design(ncases/ncontrols ~ ., data = esoph, ylim = c(0, 0.8))
plot.design(ncases/ncontrols ~ ., data = esoph, ylim = c(0, 0.8),
          fun = median)
par(op)

References


See Also

interaction.plot for a “standard graphic” of designed experiments.
plot.formula

See Also

plot.default, plot.formula, barplot, boxplot, spineplot.

Examples

```
plot(PlantGrowth)  # -> plot.data.frame
plot(weight ~ group, data = PlantGrowth)  # numeric vector ~ factor
plot(cut(weight, 2) ~ group, data = PlantGrowth)  # factor ~ factor
## passing "..." to spineplot() eventually:
plot(cut(weight, 2) ~ group, data = PlantGrowth, col = hcl(c(0, 120, 240), 50, 70))
plot(PlantGrowth$group, axes=FALSE, main="no axes")  # extremly silly
```

Description

Specify a scatterplot or add points or lines via a formula.

Usage

```
## S3 method for class 'formula':
plot(formula, data = parent.frame(), ..., subset, ylab = varnames[response], ask = TRUE)

## S3 method for class 'formula':
points(formula, data = parent.frame(), ..., subset)

## S3 method for class 'formula':
lines(formula, data = parent.frame(), ..., subset)
```

Arguments

- `formula`: a formula, such as `y ~ x`.
- `data`: a data.frame (or list) from which the variables in `formula` should be taken.
- `...`: Further graphical parameters may also be passed as arguments, see `par`. `horizontal = TRUE` is also accepted.
- `subset`: an optional vector specifying a subset of observations to be used in the fitting process.
- `ylab`: the y label of the plot(s).
- `ask`: logical, see `par`.

Details

Both the terms in the formula and the `...` arguments are evaluated in `data` enclosed in `parent.frame()` if `data` is a list or a data frame. The terms of the formula and those arguments in `...` that are of the same length as `data` are subjected to the subsetting specified in `subset`. If the formula in `plot.formula` contains more than one non-response term, a series of
plots of y against each term is given. A plot against the running index can be specified as `plot(y ~ 1)`. Missing values are not considered in these methods, and in particular cases with missing values are not removed.

If y is an object (i.e. has a `class` attribute) then `plot.formula` looks for a plot method for that class first. Otherwise, the class of x will determine the type of the plot. For factors this will be a parallel boxplot, and argument `horizontal = TRUE` can be used (see `boxplot`).

Value

These functions are invoked for their side effect of drawing in the active graphics device.

See Also

`plot.default`, `plot.factor`.

Examples

```r
op <- par(mfrow=c(2,1))
plot(Ozone ~ Wind, data = airquality, pch=as.character(Month))
plot(Ozone ~ Wind, data = airquality, pch=as.character(Month),
     subset = Month != 7)
par(op)
```

Description

These are methods for objects of class "histogram", typically produced by `hist`.

Usage

```r
## S3 method for class 'histogram':
plot(x, freq = equidist, density = NULL, angle = 45,
     col = NULL, border = par("fg"), lty = NULL,
     main = paste("Histogram of", paste(x$xname, collapse="\n")),
     sub = NULL, xlab = x$xname, ylab,
     xlim = range(x$breaks), ylim = NULL,
     axes = TRUE, labels = FALSE, add = FALSE, ...)

## S3 method for class 'histogram':
lines(x, ...)
```

Arguments

- `x`: a histogram object, or a list with components `density`, `mid`, etc, see `hist` for information about the components of `x`.
- `freq`: logical; if TRUE, the histogram graphic is to present a representation of frequencies, i.e. `x$counts`; if FALSE, relative frequencies ("probabilities"), i.e., `x$density`, are plotted. The default is true for equidistant breaks and false otherwise.
col  a colour to be used to fill the bars. The default of NULL yields unfilled bars.
border the color of the border around the bars.
angle, density select shading of bars by lines: see rect.
lty the line type used for the bars, see also lines.
main, sub, xlab, ylab these arguments to title have useful defaults here.
xlim, ylim the range of x and y values with sensible defaults.
axes logical, indicating if axes should be drawn.
labels logical or character. Additionally draw labels on top of bars, if not FALSE; if TRUE, draw the counts or rounded densities; if labels is a character, draw itself.
add logical. If TRUE, only the bars are added to the current plot. This is what lines.histogram(*) does.
... further graphical parameters to title and axis.

Details
lines.histogram(*) is the same as plot.histogram(*, add = TRUE).

See Also
hist, stem, density.

Examples

(wwt <- hist(women$weight, nc= 7, plot = FALSE))
plot(wwt, labels = TRUE) # default main & xlab using wwt$xname
plot(wwt, border = "dark blue", col = "light blue",
     main = "Histogram of 15 women's weights", xlab = "weight [pounds]")

## Fake "lines" example, using non-default labels:
w2 <- wwt; w2$counts <- w2$counts - 1
lines(w2, col = "Midnight Blue", labels = ifelse(w2$counts, "> 1", "1"))

Description

This is a method of the generic plot function for (contingency) table objects. Whereas for two- and more dimensional tables, a mosaicplot is drawn, one-dimensional ones are plotted “bar like”.

Usage

## S3 method for class 'table':
plot(x, type = "h", ylim = c(0, max(x)), lwd = 2,
     xlab = NULL, ylab = NULL, frame.plot = is.num, ...)

plot.table  Plot Methods for 'table' Objects
Arguments

- **x**: a table (like) object.
- **type**: plotting type.
- **ylim**: range of y-axis.
- **lwd**: line width for bars when type = "h" is used in the 1D case.
- **xlab**, **ylab**: x- and y-axis labels.
- **frame.plot**: logical indicating if a frame (box) should be drawn in the 1D case. Defaults to true when x has dimnames coercable to numbers.
- **...**: further graphical arguments, see `plot.default`.

Details

The current implementation (R 1.2) is somewhat experimental and will be improved and extended.

See Also

`plot.factor`, the `plot` method for factors.

Examples

```r
## 1-d tables
(Pois.tab <- table(N = rpois(200, lam= 5)))
plot(Pois.tab, main = "plot(table(rpois(200, lam=5)))")

plot(table(state.division))

## 4-D :
plot(Titanic, main ="plot(Titanic, main= *)")
```

---

**plot.window**  
*Set up World Coordinates for Graphics Window*

Description

This function sets up the world coordinate system for a graphics window. It is called by higher level functions such as `plot.default` (after `plot.new`).

Usage

```r
plot.window(xlim, ylim, log = "", asp = NA, ...)
```

Arguments

- **xlim**, **ylim**: numeric of length 2, giving the x and y coordinates ranges.
- **log**: character; indicating which axes should be in log scale.
- **asp**: numeric, giving the aspect ratio y/x.
- **...**: further graphical parameters as in `par`. 
Details

Note that if \( asp \) is a finite positive value then the window is set up so that one data unit in the \( x \) direction is equal in length to \( asp \times \) one data unit in the \( y \) direction.

The special case \( asp == 1 \) produces plots where distances between points are represented accurately on screen. Values with \( asp > 1 \) can be used to produce more accurate maps when using latitude and longitude.

The function attempts to produce a plausible set of scales if one or both of \( \text{xlim} \) and \( \text{ylim} \) is of length one or the two values given are identical, but it is better to avoid that case.

Usually, one should rather use the higher level functions such as \text{plot}, \text{hist}, \text{image}, \ldots, instead and refer to their help pages for explanation of the arguments.

See Also

\text{xy.coords}, \text{plot.xy}, \text{plot.default}.

Examples

```r
##--- An example for the use of 'asp' :
require(stats) # normally loaded
loc <- cmdscale(eurodist)
rx <- range(x <- loc[,1])
ry <- range(y <- -loc[,2])
plot(x, y, type="n", asp=1, xlab="", ylab="")
abline(h = pretty(rx, 10), v = pretty(ry, 10), col = "lightgray")
text(x, y, names(eurodist), cex=0.8)
```

---

**plot.xy**

Basic Internal Plot Function

Description

This is the internal function that does the basic plotting of points and lines. Usually, one should rather use the higher level functions instead and refer to their help pages for explanation of the arguments.

Usage

```r
plot.xy(xy, type, pch = 1, lty = "solid", col = par("fg"), bg = NA,
cex = 1, lwd = par("lwd"), ...)
```

Arguments

- \( xy \) A four-element list as results from \text{xy.coords}.
- \( type \) 1 character code: see \text{plot.default}.
- \( pch \) character or integer code for kind of points/lines, see \text{points.default}.
- \( lty \) line type code, see \text{lines}.
- \( col \) color code or name, see \text{colors,palette}.
- \( bg \) background (“fill”) color for the open plot symbols 21:25: see \text{points.default}.
points

points is a generic function to draw a sequence of points at the specified coordinates. The specified character(s) are plotted, centered at the coordinates.

Usage

points(x, ...)  

## Default S3 method:
points(x, y = NULL, type = "p", pch = par("pch"), col = par("col"), bg = NA, cex = 1, ...)

Arguments

x, y coordinate vectors of points to plot.
type character indicating the type of plotting; actually any of the types as in plot.  

plotting "character", i.e., symbol to use. pch can either be a single character or an integer code for one of a set of graphics symbols. The full set of S symbols is available with pch=0:18, see the last picture from example(points), i.e., the examples below.

In addition, there is a special set of R plotting symbols which can be obtained with pch=19:25 and 21:25 can be colored and filled with different colors:
  * pch=19: solid circle,
  * pch=20: bullet (smaller circle),
  * pch=21: circle,
  * pch=22: square,
  * pch=23: diamond,
  * pch=24: triangle point-up,
Points

- `pch=25`: triangle point down.

Values `pch=26:32` are currently unused, and `pch=32:255` give the text symbol in a single-byte locale. In a multi-byte locale such as UTF-8, numeric values of `pch` greater than or equal to 32 specify a Unicode code point.

If `pch` is an integer or character `NA` or an empty character string, the point is omitted from the plot.

Value `pch="."` is handled specially. It is a rectangle of side 0.01 inch (scaled by `cex`). In addition, if `cex = 1` (the default), each side is at least one pixel (1/72 inch on the `pdf`, `postscript` and `xfig` devices). The details here have been changed for 2.1.0 and are subject to change.

- `col`: color code or name, see `par`.
- `cex`: character (or symbol) expansion: a numerical vector. This works as a multiple of `par("cex")`.

... Further graphical parameters (see `plot.xy` and `par`) may also be supplied as arguments.

Details

The coordinates can be passed in a plotting structure (a list with `x` and `y` components), a two-column matrix, a time series, ... See `xy.coords`.

Arguments `pch`, `col`, `bg`, `cex` and `lwd` can be vectors (which will be recycled as needed) giving a value for each point plotted. Points whose `x`, `y`, `pch`, `col` or `cex` value is `NA` are omitted from the plot.

Graphical parameters are permitted as arguments to this function.

Note

What is meant by 'a single character' is locale-dependent.

The encoding may not have symbols for some or all of the characters in `pch=128:255`

References


See Also

`plot`, `lines`, and the underlying "primitive" `plot.xy`.

Examples

```r
plot(-4:4, -4:4, type = "n")# setting up coord. system
points(rnorm(200), rnorm(200), col = "red")
points(rnorm(100)/2, rnorm(100)/2, col = "blue", cex = 1.5)

op <- par(bg = "light blue")
x <- seq(0,2*pi, len=51)
## something "between type='b' and type='o'":
plot(x, sin(x), type="o", pch=21, bg=par("bg"), col = "blue", cex=1.5,
     main="plot(..., type="o", pch=21, bg=par("bg"))")
par(op)
```
## Showing all the extra & some char graphics symbols ------------
Pex <- 3  ## good for both .Device=="postscript" and "x11"
ipch <- 1:(np <- 25+11); k <- floor(sqrt(np)); dd <- c(-1,1)/2
rx <- dd + range(ix <- (ipch-1) %/% k)
ry <- dd + range(iy <- 3 + (k-1)-(ipch-1) %% k)
pch <- as.list(ipch)
pch[25+ 1:11] <- as.list(c(" * ", ".", "o", "O", "0", "+", "-", ":", "|", "%;"))
plot(rx, ry, type="n", axes = FALSE, xlab = "", ylab = "",
     main = paste("plot symbols : points (... pch = * , cex =", Pex,"))")
abline(v = ix, h = iy, col = "lightgray", lty = "dotted")
for(i in 1:np) {
  pc <- pch[[i]]
  points(ix[i], iy[i], pch = pc, col = "red", bg = "yellow", cex = Pex)
  ## red symbols with a yellow interior (where available)
  text(ix[i] - .3, iy[i], pc, col = "brown", cex = 1.2)
}

---

**polygon**  
**Polygon Drawing**

**Description**

polygon draws the polygons whose vertices are given in x and y.

**Usage**

```r
polygon(x, y = NULL, density = NULL, angle = 45, 
border = NULL, col = NA, lty = NULL, xpd = NULL, ...)
```

**Arguments**

- **x, y** vectors containing the coordinates of the vertices of the polygon.
- **density** the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. A zero value of density means no shading lines whereas negative values (and NA) suppress shading (and so allow color filling).
- **angle** the slope of shading lines, given as an angle in degrees (counter-clockwise).
- **col** the color for filling the polygon. The default, NA, is to leave polygons unfilled.
- **border** the color to draw the border. The default, NULL, uses `par("fg")`. Use `border = NA` to omit borders.
  
  For compatibility with S, `border` can also be logical, in which case `FALSE` is equivalent to `NA` (borders omitted) and `TRUE` is equivalent to `NULL` (use the foreground colour).
- **lty** the line type to be used, as in `par`.
- **xpd** (where) should clipping take place? Defaults to `par("xpd")`.
- **...** graphical parameters can be given as arguments to `polygon`. 

```r
## -------- Showing all the extra & some char graphics symbols ---------
Pex <- 3  ## good for both .Device=="postscript" and "x11"
ipch <- 1:(np <- 25+11); k <- floor(sqrt(np)); dd <- c(-1,1)/2
rx <- dd + range(ix <- (ipch-1) %/% k)
ry <- dd + range(iy <- 3 + (k-1)-(ipch-1) %% k)
pch <- as.list(ipch)
pch[25+ 1:11] <- as.list(c(" * ", ".", "o", "O", "0", "+", "-", ":", "|", "%;"))
plot(rx, ry, type="n", axes = FALSE, xlab = "", ylab = "",
     main = paste("plot symbols : points (... pch = * , cex =", Pex,"))")
abline(v = ix, h = iy, col = "lightgray", lty = "dotted")
for(i in 1:np) {
  pc <- pch[[i]]
  points(ix[i], iy[i], pch = pc, col = "red", bg = "yellow", cex = Pex)
  ## red symbols with a yellow interior (where available)
  text(ix[i] - .3, iy[i], pc, col = "brown", cex = 1.2)
}
```
Details

The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, ..... See `xy.coords`.

It is assumed that the polygon is to be closed by joining the last point to the first point.

The coordinates can contain missing values. The behaviour is similar to that of `lines`, except that instead of breaking a line into several lines, NA values break the polygon into several complete polygons (including closing the last point to the first point). See the examples below.

When multiple polygons are produced, the values of `density`, `angle`, `col`, `border`, and `lty` are recycled in the usual manner.

Bugs

The present shading algorithm can produce incorrect results for self-intersecting polygons.

Author(s)

The code implementing polygon shading was donated by Kevin Buhr (<buhr@stat.wisc.edu>).

References


See Also

`segments` for even more flexibility, `lines`, `rect`, `box`, `abline`. `par` for how to specify colors.

Examples

```r
x <- c(1:9,8:1)
y <- c(1,2*(5:3),2,-1,17,9,8,2:9)
opt <- par(mfcol=c(3,1))
for(xpd in c(FALSE,TRUE,NA)) {
  plot(1:10, main=paste("xpd =", xpd)) ; box("figure", col = "pink", lwd=3)
  polygon(x,y, xpd=xpd, col = "orange", lty=2, lwd=2, border = "red")
}
par(opt)

n <- 100
xx <- c(0:n, n:0)
yy <- c(c(0,cumsum(rnorm(n))), rev(c(0,cumsum(rnorm(n)))))
plot (xx, yy, type="n", xlab="Time", ylab="Distance")
polygon(xx, yy, col="gray", border = "red")
title("Distance Between Brownian Motions")

# Multiple polygons from NA values
# and recycling of col, border, and lty
opt <- par(mfrow=c(2,1))
plot(c(1:9), 1:2, type="n")
polygon(1:9, c(2,1,2,1,2,1,2,1,2),
  col=c("red", "blue"),
  border=c("green", "yellow"),
  lwd=3, lty=c("dashed", "solid"))
```
rect

Description

rect draws a rectangle (or sequence of rectangles) with the given coordinates, fill and border colors.

Usage

rect(xleft, ybottom, xright, ytop, density = NULL, angle = 45,
     col = NULL, border = NULL, lty = NULL, lwd = par("lwd"),
     xpd = NULL, ...)  

Arguments

xleft  a vector (or scalar) of left x positions.
ybottom  a vector (or scalar) of bottom y positions.
xright  a vector (or scalar) of right x positions.
ytop  a vector (or scalar) of top y positions.
density  the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. A zero value of density means no shading lines whereas negative values (and NA) suppress shading (and so allow color filling).
angle  angle (in degrees) of the shading lines.
col  color(s) to fill or shade the rectangle(s) with. The default NULL, or also NA do not fill, i.e., draw transparent rectangles, unless density is specified.
border  color for rectangle border(s). Can also be FALSE to suppress the border, or TRUE in which case col is used.
lty  line type for borders and shading; defaults to "solid".
lwd  line width for borders and shading.
xpd  logical ("expand"); defaults to par("xpd"). See par(xpd= ).
...  other graphical parameters can be given as arguments.
Details

The positions supplied, i.e., \texttt{xleft}, \ldots, are relative to the current plotting region. If the x-axis goes from 100 to 200 then \texttt{xleft} must be larger than 100 and \texttt{xright} must be less than 200.

It is a primitive function used in \texttt{hist}, \texttt{barplot}, \texttt{legend}, etc.

See Also

\texttt{box} for the “standard” box around the plot; \texttt{polygon} and \texttt{segments} for flexible line drawing. \texttt{par} for how to specify colors.

Examples

\begin{verbatim}
## set up the plot region:
op <- par(bg = "thistle")
plot(c(100, 250), c(300, 450), type = "n", xlab="", ylab="",
     main = "2 x 11 rectangles; 'rect(100+i,300+i, 150+i,380+i)'")
i <- 4*z(0:10)
## draw rectangles with bottom left (100, 300)+i and top right (150, 380)+i
rect(100+i, 300+i, 150+i, 380+i, col=rainbow(11, start=.7,end=.1))
rect(240-i, 320+i, 250-i, 410+i, col=heat.colors(11), lwd=i/5)
## Background alternating ( transparent / "bg" ) : 
j <- 10*(0.5:
rect(125+j, 360+j, 141+j, 405+j/2, col = c(NA,0), border = "gold", lwd = 2)
rect(125+j, 296+j/2, 141+j, 331+j/5, col = c(NA,"m..blue"))
mtext("+ 2 x 6 rect(*, col = c(NA,0)) and col = c(NA,\"m..blue\")")
## an example showing colouring and shading
plot(c(100, 200), c(300, 450), type= "n", xlab="", ylab="")
rect(100, 300, 125, 350) # transparent
rect(100, 400, 125, 450, col="green", border="blue") # coloured
rect(115, 375, 150, 425, col=par("bg"), border="transparent")
rect(150, 300, 175, 350, density=10, border="red")
rect(150, 400, 175, 450, density=30, col="blue",
    angle=-30, border="transparent")
legend(180, 450, legend=1:4, fill=c(NA, "green", par("fg"), "blue"),
      density=c(NA, NA, 10, 30), angle=c(NA, NA, 30, -30))
par(op)
\end{verbatim}

rug

Add a Rug to a Plot

Description

Adds a rug representation (1-d plot) of the data to the plot.

Usage

\begin{verbatim}
rug(x, ticksize=0.03, side=1, lwd=0.5, col, quiet = getOption("warn") < 0, ...)\end{verbatim}
Arguments

- `x`: A numeric vector
- `ticksize`: The length of the ticks making up the ‘rug’. Positive lengths give inwards ticks.
- `side`: On which side of the plot box the rug will be plotted. Normally 1 (bottom) or 3 (top).
- `lwd`: The line width of the ticks.
- `col`: The colour the ticks are plotted in, default is black.
- `quiet`: logical indicating if there should be a warning about clipped values.
- `...`: further arguments, passed to `axis(...)` such as `line` or `pos` for specifying the location of the rug.

Details

Because of the way `rug` is implemented, only values of `x` that fall within the plot region are included. There will be a warning if any finite values are omitted, but non-finite values are omitted silently.

Because of the way colours are done the axis itself is coloured the same as the ticks. You can always replot the box in black if you don’t like this feature.

References


See Also

- `jitter` which you may want for ties in `x`.

Examples

```r
require(stats)# both 'density' and its default method
with(faithful, {
  plot(density(eruptions, bw=0.15))
  rug(eruptions)
  rug(jitter(eruptions, amount = .01), side = 3, col = "light blue")
})
```

---

Description

`split.screen` defines a number of regions within the current device which can, to some extent, be treated as separate graphics devices. It is useful for generating multiple plots on a single device. Screens can themselves be split, allowing for quite complex arrangements of plots.

`screen` is used to select which screen to draw in.

`erase.screen` is used to clear a single screen, which it does by filling with the background colour.

`close.screen` removes the specified screen definition(s).
Usage

split.screen(figs, screen, erase = TRUE)
screen(n = , new = TRUE)
erase.screen(n = )
close.screen(n, all.screens = FALSE)

Arguments

figs A two-element vector describing the number of rows and the number of columns in a screen matrix or a matrix with 4 columns. If a matrix, then each row describes a screen with values for the left, right, bottom, and top of the screen (in that order) in NDC units, that is 0 at the lower left corner of the device surface, and 1 at the upper right corner.

screen A number giving the screen to be split. It defaults to the current screen if there is one, otherwise the whole device region.

erase logical: should be selected screen be cleared?

n A number indicating which screen to prepare for drawing (screen), erase (erase.screen), or close (close.screen). (close.screen will accept a vector of screen numbers.)

new A logical value indicating whether the screen should be erased as part of the preparation for drawing in the screen.

all.screens A logical value indicating whether all of the screens should be closed.

Details

The first call to split.screen places R into split-screen mode. The other split-screen functions only work within this mode. While in this mode, certain other commands should be avoided (see the Warnings section below). Split-screen mode is exited by the command close.screen(all = TRUE).

If the current screen is closed, close.screen sets the current screen to be the next larger screen number if there is one, otherwise to the first available screen.

Value

split.screen returns a vector of screen numbers for the newly-created screens. With no arguments, split.screen returns a vector of valid screen numbers.

screen invisibly returns the number of the selected screen. With no arguments, screen returns the number of the current screen.

close.screen returns a vector of valid screen numbers.

screen, erase.screen, and close.screen all return FALSE if R is not in split-screen mode.

Warnings

The recommended way to use these functions is to completely draw a plot and all additions (i.e. points and lines) to the base plot, prior to selecting and plotting on another screen. The behavior associated with returning to a screen to add to an existing plot is unpredictable and may result in problems that are not readily visible.

These functions are totally incompatible with the other mechanisms for arranging plots on a device: par(mfrow), par(mfcol) and layout().
segments

The functions are also incompatible with some plotting functions, such as `coplot`, which make use of these other mechanisms.

`erase.screen` will appear not to work if the background colour is transparent (as it is by default on most devices).

References


See Also

`par`, `layout`, `Devices`, `dev.*`

Examples

```r
if (interactive()) {
  par(bg = "white") # default is likely to be transparent
  split.screen(c(2,1)) # split display into two screens
  split.screen(c(1,3), screen = 2) # now split the bottom half into 3
  screen(1) # prepare screen 1 for output
  plot(10:1)
  screen(4) # prepare screen 4 for output
  plot(10:1)
  close.screen(all = TRUE) # exit split-screen mode
  split.screen(c(2,1)) # split display into two screens
  split.screen(c(1,2),2) # split bottom half in two
  plot(1:10) # screen 3 is active, draw plot
  erase.screen() # forgot label, erase and redraw
  plot(1:10, ylab = "ylab 3")
  screen(1) # prepare screen 1 for output
  plot(1:10)
  screen(4) # prepare screen 4 for output
  plot(1:10, ylab="ylab 4") # return to screen 1, but do not clear
  screen(1, FALSE)
  plot(10:1, axes=FALSE, lty=2, ylab="") # overlay second plot
  axis(4) # add tic marks to right-hand axis
  title("Plot 1")
  close.screen(all = TRUE) # exit split-screen mode
}
```

segments

Add Line Segments to a Plot

Description

Draw line segments between pairs of points.

Usage

```r
segments(x0, y0, x1, y1,
  col = par("fg"), lty = par("lty"), lwd = par("lwd"), ...)
```
spineplot

Arguments
- `x0,y0` coordinates of points from which to draw.
- `x1,y1` coordinates of points to which to draw.
- `col, lty, lwd` usual graphical parameters as in `par`.
- `...` further graphical parameters (from `par`).

Details
For each i, a line segment is drawn between the point \((x0[i], y0[i])\) and the point \((x1[i], y1[i])\).

The graphical parameters `col` and `lty` can be used to specify a color and line texture for the line segments (`col` may be a vector).

References

See Also
- `arrows`, `polygon` for slightly easier and less flexible line drawing, and `lines` for the usual polygons.

Examples
```r
x <- runif(12); y <- rnorm(12)
i <- order(x,y); x <- x[i]; y <- y[i]
plot(x,y, main="arrows(.) and segments(.)")
## draw arrows from point to point :
s <- seq(length(x)-1)# one shorter than data
arrows(x[s], y[s], x[s+1], y[s+1], col= 1:3)
s <- s[-length(s)]
segments(x[s], y[s], x[s+2], y[s+2], col= 'pink')
```

spineplot

**Spine Plots and Spinograms**

Description
Spine plots are a special cases of mosaic plots, and can be seen as a generalization of stacked (or highlighted) bar plots. Analogously, spinograms are an extension of histograms.

Usage
```r
spineplot(x, ...)
## Default S3 method:
spineplot(x, y = NULL,
  breaks = NULL, tol.ylab = 0.05, off = NULL,
  col = NULL, main = "", xlab = NULL, ylab = NULL,
  xaxlabels = NULL, yaxlabels = NULL,
```

spineplot

xlim = NULL, ylim = c(0, 1), ...
```### S3 method for class 'formula':
spineplot(formula, data = list(),
  breaks = NULL, tol.ylab = 0.05, off = NULL,
  col = NULL, main = "", xlab = NULL, ylab = NULL,
  xaxlabels = NULL, yaxlabels = NULL,
  xlim = NULL, ylim = c(0, 1), ...
  subset = NULL)

Arguments

x
  an object, the default method expects either a single variable (interpreted to be
  the explanatory variable) or a 2-way table. See details.

y
  a "factor" interpreted to be the dependent variable

formula
  a "formula" of type \( y \sim x \) with a single dependent "factor" and a single
  explanatory variable.

data
  an optional data frame.

breaks
  if the explantory variable is numeric, this controls how it is discretized. breaks
  is passed to hist and can be a list of arguments.

tol.ylab
  convenience tolerance parameter for y-axis annotation. If the distance between
  two labels drops under this threshold, they are plotted equidistantly.

off
  vertical offset between the bars (in per cent). It is fixed to 0 for spinograms and
  defaults to 2 for spine plots.

col
  a vector of fill colors of the same length as levels(y). The default is to call
  gray.colors.

main, xlab, ylab
  character strings for annotation

xaxlabels, yaxlabels
  character vectors for annotation of x and y axis. Default to levels(y) and
  levels(x), respectively for the spine plot. For xaxlabels in the spinogram, the breaks are used.

xlim, ylim
  the range of x and y values with sensible defaults.

...  
  additional arguments passed to rect.

subset
  an optional vector specifying a subset of observations to be used for plotting.

Details

spineplot creates either a spinogram or a spine plot. It can be called via spineplot(x,
  y) or spineplot(y \sim x) where y is interpreted to be the dependent variable (and has to be
categorical) and x the explanatory variable. x can be either categorical (then a spine plot is created)
or numerical (then a spinogram is plotted). Additionally, spineplot can also be called with only
  a single argument which then has to be a 2-way table, interpreted to correspond to table(x, y).

Both, spine plots and spinograms, are essentially mosaic plots with special formatting of spacing
and shading. Conceptually, they plot \( P(y|x) \) against \( P(x) \). For the spine plot (where both x and y
are categorical), both quantities are approximated by the corresponding empirical relative frequencies.
For the spinogram (where x is numerical), x is first discretized (by calling hist with breaks
argument) and then empirical relative frequencies are taken.

Thus, spine plots can also be seen as a generalization of stacked bar plots where not the heights
but the widths of the bars corresponds to the relative frequencies of x. The heights of the bars then
correspond to the conditional relative frequencies of y in every x group. Analogously, spinograms
extend stacked histograms.
spineplot

Value

The table visualized is returned invisibly.

Author(s)

Achim Zeileis (Achim.Zeileis@R-project.org)

References


See Also

mosaicplot.hist, cdplot

Examples

## treatment and improvement of patients with rheumatoid arthritis

treatment <- factor(rep(c(1, 2), c(43, 41)), levels = c(1, 2),
                      labels = c("placebo", "treated"))
improved <- factor(rep(c(1, 2, 3, 1, 2, 3), c(29, 7, 7, 13, 7, 21)),
                   levels = c(1, 2, 3), labels = c("none", "some", "marked"))

## (dependence on a categorical variable)
(spineplot(improved ~ treatment))

## applications and admissions by department at UC Berkeley
## (two-way tables)
(spineplot(margin.table(UCBAdmissions, c(3, 2)), main = "Applications at UCB"))
(spineplot(margin.table(UCBAdmissions, c(3, 1)), main = "Admissions at UCB"))

## NASA space shuttle o-ring failures

fail <- factor(c(2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1),
               levels = c(1, 2), labels = c("no", "yes"))
temperature <- c(53, 57, 58, 63, 66, 67, 67, 68, 69, 70, 70, 70, 70, 72, 73, 75, 75, 76, 76, 76, 78, 79, 81)

## (dependence on a numerical variable)
(spineplot(fail ~ temperature))
(spineplot(fail ~ temperature, breaks = 3))
(spineplot(fail ~ temperature, breaks = quantile(temperature)))
stars

Star (Spider/Radar) Plots and Segment Diagrams

Description

Draw star plots or segment diagrams of a multivariate data set. With one single location, also draws “spider” (or “radar”) plots.

Usage

stars(x, full = TRUE, scale = TRUE, radius = TRUE,
labels = dimnames(x)[[1]], locations = NULL,
nrow = NULL, ncol = NULL, len = 1,
key.loc = NULL, key.labels = dimnames(x)[[2]], key.xpd = TRUE,
xlim = NULL, ylim = NULL, flip.labels = NULL,
draw.segments = FALSE, col.segments = 1:n.seg, col.stars = NA,
axes = FALSE, frame.plot = axes,
main = NULL, sub = NULL, xlab = "", ylab = ",
cex = 0.8, lwd = 0.25, lty = par("lty"), xpd = FALSE,
mar = pmin(par("mar"),
1.1 + c(2*axes+ (xlab != ""),
2*axes+ (ylab != ""), 1,0)),
add = FALSE, plot = TRUE, ...

Arguments

x matrix or data frame of data. One star or segment plot will be produced for each row of x. Missing values (NA) are allowed, but they are treated as if they were 0 (after scaling, if relevant).

full logical flag: if TRUE, the segment plots will occupy a full circle. Otherwise, they occupy the (upper) semicircle only.

scale logical flag: if TRUE, the columns of the data matrix are scaled independently so that the maximum value in each column is 1 and the minimum is 0. If FALSE, the presumption is that the data have been scaled by some other algorithm to the range [0, 1].

radius logical flag: in TRUE, the radii corresponding to each variable in the data will be drawn.

labels vector of character strings for labeling the plots. Unlike the S function stars, no attempt is made to construct labels if labels = NULL.

locations Either two column matrix with the x and y coordinates used to place each of the segment plots; or numeric of length 2 when all plots should be superimposed (for a "spider plot"). By default, locations = NULL, the segment plots will be placed in a rectangular grid.

nrow, ncol integers giving the number of rows and columns to use when locations is NULL. By default, nrow == ncol, a square layout will be used.

len scale factor for the length of radii or segments.

key.loc vector with x and y coordinates of the unit key.

key.labels vector of character strings for labeling the segments of the unit key. If omitted, the second component of dimnames (x) is used, if available.
key.xpd clipping switch for the unit key (drawing and labeling), see par("xpd").
xlim vector with the range of x coordinates to plot.
ylim vector with the range of y coordinates to plot.
flip.labels logical indicating if the label locations should flip up and down from diagram to
   diagram. Defaults to a somewhat smart heuristic.
draw.segments logical. If TRUE draw a segment diagram.
col.segments color vector (integer or character, see par), each specifying a color for one of
   the segments (variables). Ignored if draw.segments = FALSE.
col.stars color vector (integer or character, see par), each specifying a color for one of
   the stars (cases). Ignored if draw.segments = TRUE.
axes logical flag: if TRUE axes are added to the plot.
frame.plot logical flag: if TRUE, the plot region is framed.
main a main title for the plot.
sub a sub title for the plot.
xlab a label for the x axis.
ylab a label for the y axis.
cex character expansion factor for the labels.
lwd line width used for drawing.
lty line type used for drawing.
xpd logical or NA indicating if clipping should be done, see par(xpd = .).
mar argument to par(mar = *), typically choosing smaller margings than by de-
   fault.
... further arguments, passed to the first call of plot(), see plot.default and
   to box() if frame.plot is true.
add logical, if TRUE add stars to current plot.
plot logical, if FALSE, nothing is plotted.

Details

Missing values are treated as 0.

Each star plot or segment diagram represents one row of the input x. Variables (columns) start on
the right and wind counterclockwise around the circle. The size of the (scaled) column is shown by
the distance from the center to the point on the star or the radius of the segment representing the
variable.

Only one page of output is produced.

Note

This code started life as spatial star plots by David A. Andrews. See http://www.udallas.
edu:8080/~andrews/software/software.html.

Prior to 1.4.1, scaling only shifted the maximum to 1, although documented as here.

Author(s)

Thomas S. Dye
Stem-and-Leaf Plots

References


Examples

```r
stars(mtcars[, 1:7], key.loc = c(14, 2),
main = "Motor Trend Cars : stars(*, full = F)", full = FALSE)
stars(mtcars[, 1:7], key.loc = c(14, 1.5),
main = "Motor Trend Cars : full stars()", flip.labels=FALSE)

## 'Spider' or 'Radar' plot:
stars(mtcars[, 1:7], locations = c(0,0), radius = FALSE,
key.loc=c(0,0), main="Motor Trend Cars", lty = 2)

## Segment Diagrams:
palette(rainbow(12, s = 0.6, v = 0.75))
stars(mtcars[, 1:7], len = 0.8, key.loc = c(12, 1.5),
main = "Motor Trend Cars", draw.segments = TRUE)
stars(mtcars[, 1:7], len = 0.6, key.loc = c(1.5, 0),
main = "Motor Trend Cars", draw.segments = TRUE,
frame.plot=TRUE, nrow = 4, cex = .7)

## scale linearly (not affinely) to [0, 1]
USJudge <- apply(USJudgeRatings, 2, function(x) x/max(x))
Jnam <- row.names(USJudgeRatings)
Snam <- abbreviate(substring(Jnam,1,regexpr("[\[,\]]",Jnam) - 1), 7)
stars(USJudge, labels = Jnam, scale = FALSE,
key.loc = c(13, 1.5), main = "Judge not ...", len = 0.8)
stars(USJudge, labels = Snam, scale = FALSE,
key.loc = c(13, 1.5), radius = FALSE)

loc <- stars(USJudge, labels = NULL, scale = FALSE,
radius = FALSE, frame.plot = TRUE,
key.loc = c(13, 1.5), main = "Judge not ...", len = 1.2)
text(loc, Snam, col = "blue", cex = 0.8, xpd = TRUE)

## 'Segments':
stars(USJudge, draw.segments = TRUE, scale = FALSE, key.loc = c(13,1.5))

## 'Spider':
stars(USJudgeRatings, locations=c(0,0), scale=FALSE,radius = FALSE,
col.stars=1:10, key.loc = c(0,0), main="US Judges rated")

## 'Radar-Segments'
stars(USJudgeRatings[,1:10], locations = 0:1, scale=FALSE,
draw.segments = TRUE, col.segments=0, col.stars=1:10,key.loc= 0:1,
main="US Judges 1-10 ")
palette("default")
stars(cbind(1:16,10*(16:1)),draw.segments=TRUE,
main = "A Joke -- do *not* use symbols on 2D data!")
```

Stem-and-Leaf Plots
**Description**

Stem produces a stem-and-leaf plot of the values in `x`. The parameter `scale` can be used to expand the scale of the plot. A value of `scale=2` will cause the plot to be roughly twice as long as the default.

**Usage**

```r
stem(x, scale = 1, width = 80, atom = 1e-08)
```

**Arguments**

- `x`  
a numeric vector.
- `scale`  
This controls the plot length.
- `width`  
The desired width of plot.
- `atom`  
a tolerance.

**References**


**Examples**

```r
stem(islands)
stem(log10(islands))
```

---

**Description**

Stripchart produces one dimensional scatter plots (or dot plots) of the given data. These plots are a good alternative to `boxplots` when sample sizes are small.

**Usage**

```r
stripchart(x, method = "overplot", jitter = 0.1, offset = 1/3,
        vertical = FALSE, group.names, add = FALSE,
        at = NULL, xlim = NULL, ylim = NULL,
        main = "", ylab = "", xlab = "",
        log = "", pch = 0, col = par("fg"), cex = par("cex"))
```

**Arguments**

- `x`  
the data from which the plots are to be produced. The data can be specified as a single vector, or as list of vectors, each corresponding to a component plot. Alternatively a symbolic specification of the form `x ~ g` can be given, indicating the the observations in the vector `x` are to be grouped according to the levels of the factor `g`. NAs are allowed in the data.
method 

the method to be used to separate coincident points. The default method "overplot" causes such points to be overplotted, but it is also possible to specify "jitter" to jitter the points, or "stack" have coincident points stacked. The last method only makes sense for very granular data.

jitter

when method="jitter" is used, jitter gives the amount of jittering applied.

offset

when stacking is used, points are stacked this many line-heights (symbol widths) apart.

vertical

when vertical is TRUE the plots are drawn vertically rather than the default horizontal.

group.names

group labels which will be printed alongside (or underneath) each plot.

add

logical, if true add the chart to the current plot.

at

numeric vector giving the locations where the charts should be drawn, particularly when add = TRUE; defaults to 1:n where n is the number of boxes.

xlim, ylim, main, ylab, xlab, log, pch, col, cex

Graphical parameters.

Details

Extensive examples of the use of this kind of plot can be found in Box, Hunter and Hunter or Seber and Wild.

Examples

x <- rnorm(50)
xr <- round(x, 1)
stripchart(x) ; m <- mean(par("usr")[1:2])
text(m, 1.04, "stripchart(x, "overplot")")
stripchart(xr, method = "stack", add = TRUE, at = 1.2)
text(m, 1.35, "stripchart(round(x,1), "stack")")
stripchart(xr, method = "jitter", add = TRUE, at = 0.7)
text(m, 0.85, "stripchart(round(x,1), "jitter")")

with(OrchardSprays,
    stripchart(decrease ~ treatment,
               main = "stripchart(Orchardsprays)", ylab = "decrease",
               vertical = TRUE, log = "y")
)

with(OrchardSprays,
    stripchart(decrease ~ treatment, at = c(1:8)^2,
               main = "stripchart(Orchardsprays)", ylab = "decrease",
               vertical = TRUE, log = "y")
)

strwidth

Plotting Dimensions of Character Strings and Math Expressions

Description

These functions compute the width or height, respectively, of the given strings or mathematical expressions s[i] on the current plotting device in user coordinates, inches or as fraction of the figure width par("fin").
Usage

```r
strwidth(s, units = "user", cex = NULL)
strheight(s, units = "user", cex = NULL)
```

Arguments

- `s` character vector or expressions whose string widths in plotting units are to be determined. An attempt is made to coerce other vectors to character, and other language objects to expressions.

- `units` character indicating in which units `s` is measured; should be one of "user", "inches", "figure"; partial matching is performed.

- `cex` numeric character expansion factor; multiplied by `par("cex")` yields the final character size; the default `NULL` is equivalent to 1.

Value

Numeric vector with the same length as `s`, giving the width or height for each `s[i]`. NA strings are given width and height 0 (as they are not plotted).

See Also

text, nchar

Examples

```r
str.ex <- c("W","w","I",".","WwI.")
op <- par(pty='s'); plot(1:100,1:100, type="n")
sw <- strwidth(str.ex); sw
all.equal(sum(sw[1:4]), sw[5])#- since the last string contains the others
sw.i <- strwidth(str.ex, "inches"); 25.4 * sw.i # width in [mm]
unique(sw / sw.i)
# constant factor: 1 value
mean(sw.i / strwidth(str.ex, "fig")) / par("fin")[1] # = 1: are the same

## See how letters fall in classes -- depending on graphics device and font!
all.lett <- c(letters, LETTERS)
shL <- strheight(all.lett, units = "inches") * 72 # 'big points'
table(shL) # all have same heights ... 
mean(shL)/par("cin")[2] # around 0.6

(swl <- strwidth(all.lett, units="inches") * 72) # 'big points'
split(all.lett, factor(round(swl, 2)))
sumex <- expression(sum(x[i], i=1,n), e^(i * pi) == -1)
strwidth(sumex)
strheight(sumex)
par(op)##- reset to previous setting
```
**sunflowerplot**

**Produce a Sunflower Scatter Plot**

**Description**

Multiple points are plotted as “sunflowers” with multiple leaves (“petals”) such that overplotting is visualized instead of accidental and invisible.

**Usage**

```r
sunflowerplot(x, y = NULL, number, log = "", digits = 6, xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL, add = FALSE, rotate = FALSE, pch = 16, cex = 0.8, cex.fact = 1.5, col = par("col"), bg = NA, size = 1/8, seg.col = 2, seg.lwd = 1.5, ...)```

**Arguments**

- **x**: numeric vector of x-coordinates of length n, say, or another valid plotting structure, as for `plot.default`, see also `xy.coords`.
- **y**: numeric vector of y-coordinates of length n.
- **number**: integer vector of length n. `number[i]` = number of replicates for `(x[i],y[i])`. may be 0. Default: compute the exact multiplicity of the points `x[i],y[i]`.
- **log**: character indicating log coordinate scale, see `plot.default`.
- **digits**: when `number` is computed (i.e., not specified), x and y are rounded to `digits` significant digits before multiplicities are computed.
- **xlab, ylab**: character label for x-, or y-axis, respectively.
- **xlim, ylim**: numeric(2) limiting the extents of the x-, or y-axis.
- **add**: logical; should the plot be added on a previous one? Default is `FALSE`.
- **rotate**: logical; if TRUE, randomly rotate the sunflowers (preventing artefacts).
- **pch**: plotting character to be used for points `(number[i]==1)` and center of sunflowers.
- **cex**: numeric; character size expansion of center points (s. pch).
- **cex.fact**: numeric shrinking factor to be used for the center points when there are flower leaves, i.e., `cex / cex.fact` is used for these.
- **col, bg**: colors for the plot symbols, passed to `plot.default`.
- **size**: of sunflower leaves in inches, 1[in] := 2.54[cm]. Default: 1/8, approximately 3.2mm.
- **seg.col**: color to be used for the segments which make the sunflowers leaves, see `par(col=col):col = "gold"` reminds of real sunflowers.
- **seg.lwd**: numeric; the line width for the leaves’ segments.
- **...**: further arguments to `plot` [if add=FALSE].
Details

For number[i]==1, a (slightly enlarged) usual plotting symbol (pch) is drawn. For number[i] > 1, a small plotting symbol is drawn and number[i] equi-angular "rays" emanate from it.

If rotate=TRUE and number[i] >= 2, a random direction is chosen (instead of the y-axis) for the first ray. The goal is to jitter the orientations of the sunflowers in order to prevent artefactual visual impressions.

Value

A list with three components of same length,

- x: x coordinates
- y: y coordinates
- number: number

Side Effects

A scatter plot is drawn with “sunflowers” as symbols.

Author(s)


References


See Also
density

Examples

```r
## 'number' is computed automatically:
sunflowerplot(iris[, 3:4])
## Imitating Chambers et al., p.109, closely:
sunflowerplot(iris[, 3:4], cex=.2, cex.f=1, size=.035, seg.lwd=.8)

sunflowerplot(x=sort(2*round(rnorm(100))), y= round(rnorm(100),0),
main = "Sunflower Plot of Rounded N(0,1)")
## A 'point process' (explicit 'number' argument):
sunflowerplot(rnorm(100),rnorm(100), number=rpois(n=100,lambda=2),
rotate=TRUE, main="Sunflower plot", col = "blue4")
```
text draws the strings given in the vector labels at the coordinates given by x and y. y may be missing since xy.coords(x, y) is used for construction of the coordinates.

Usage

```r
text(x, ...)  # Default S3 method:
text(x, y = NULL, labels = seq(along = x), adj = NULL,
pos = NULL, offset = 0.5, vfont = NULL,
cex = 1, col = NULL, font = NULL, xpd = NULL, ...)  # Argument details
```

Arguments

- `x, y` numeric vectors of coordinates where the text labels should be written. If the length of x and y differs, the shorter one is recycled.
- `labels` one or more character strings or expressions specifying the text to be written. An attempt is made to coerce other vectors (and factors) to character, and other language objects to expressions. If labels is longer than x and y, the coordinates are recycled to the length of labels.
- `adj` one or two values in [0, 1] which specify the x (and optionally y) adjustment of the labels. On most devices values outside that interval will also work.
- `pos` a position specifier for the text. If specified this overrides any adj value given. Values of 1, 2, 3 and 4, respectively indicate positions below, to the left of, above and to the right of the specified coordinates.
- `offset` when pos is specified, this value gives the offset of the label from the specified coordinate in fractions of a character width.
- `vfont` if a character vector of length 2 is specified, then Hershey vector fonts are used. The first element of the vector selects a typeface and the second element selects a style.
- `cex` numeric character expansion factor; multiplied by `par("cex")` yields the final character size.
- `col, font` the color and font to be used; these default to the values of the global graphical parameters in `par()`. NULL and NA are equivalent to 1.0.
- `xpd` (where) should clipping take place? Defaults to `par("xpd")`.
- `...` further graphical parameters (from `par`).

Details

labels must be of type character or expression (or be coercible to such a type). In the latter case, quite a bit of mathematical notation is available such as sub- and superscripts, greek letters, fractions, etc.

adj allows adjustment of the text with respect to (x, y). Values of 0, 0.5, and 1 specify left/bottom, middle and right/top, respectively. The default is for centered text, i.e., adj =
Accurate vertical centering needs character metric information on individual characters, which is only available on some devices.

The `pos` and `offset` arguments can be used in conjunction with values returned by `identify` to recreate an interactively labelled plot.

Text can be rotated by using graphical parameters `srt` (see `par`); this rotates about the centre set by `adj`.

Graphical parameters `col`, `cex` and `font` can be vectors and will then be applied cyclically to the labels (and extra values will be ignored).

Labels whose `x`, `y`, `labels`, `cex` or `col` value is NA are omitted from the plot.

References


See Also

`mtext`, `title`, `Hershey` for details on Hershey vector fonts, `plotmath` for details and more examples on mathematical annotation.

Examples

```r
plot(-1:1,-1:1, type = "n", xlab = "Re", ylab = "Im")
K <- 16; text(exp(1i * 2 * pi * (1:K) / K), col = 2)

## The following two examples use latin1 characters: these may not
## appear correctly (or be omitted entirely).
plot(1:10, 1:10, main = "text(...) examples\n~~~~~~~~~~~~~~",
     sub = "R is GNU ©, but not ® ...")
mtext("^Latin-1 accented chars": éè øØ å<Å æ<Æ", side=3)
points(c(6,2), c(2,1), pch = 3, cex = 4, col = "red")
text(6, 2, "the text is CENTERED around (x,y) = (6,2) by default",
     cex = .8)
text(2, 1, "or Left/Bottom - JUSTIFIED at (2,1) by 'adj = c(0,0)'",
     adj = c(0,0))
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y)", cex = .75)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))

## Two more latin1 examples

```r
text(5,10.2,
    "Le français, c'est facile: Règles, Liberté, Egalité, Fraternité...")
text(5,9.8, "Jetz no chli züritüütsch: (noch ein bißchen Zürcher deutsch)"

```

Description

This function can be used to add labels to a plot. Its first four principal arguments can also be used as arguments in most high-level plotting functions. They must be of type `character` or `expression`. In the latter case, quite a bit of mathematical notation is available such as sub- and superscripts, greek letters, fractions, etc.
Usage

```r
title(main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
       line = NA, outer = FALSE, ...)
```

Arguments

- **main**
  The main title (on top) using font and size (character expansion) `par("font.main")` and color `par("col.main")`.

- **sub**
  Sub-title (at bottom) using font and size `par("font.sub")` and color `par("col.sub")`.

- **xlab**
  X axis label using font and character expansion `par("font.axis")` and color `par("col.axis")`.

- **ylab**
  Y axis label, same font attributes as `xlab`.

- **line**
  specifying a value for `line` overrides the default placement of labels, and places them this many lines from the plot.

- **outer**
  a logical value. If `TRUE`, the titles are placed in the outer margins of the plot.

- **...**
  further graphical parameters from `par`. Use e.g., `col.main` or `cex.sub` instead of just `col` or `cex`.

Details

The labels passed to `title` can be simple strings or expressions, or they can be a list containing the string to be plotted, and a selection of the optional modifying graphical parameters `cex=`, `col=`, `font=`.

References


See Also

`mtext`, `text`; `plotmath` for details on mathematical annotation.

Examples

```r
plot(cars, main = "") # here, could use main directly
title(main = "Stopping Distance versus Speed")

plot(cars, main = "")
title(main = list("Stopping Distance versus Speed", cex=1.5,
                 col="red", font=3))

### Specifying "...":
plot(1, col.axis = "sky blue", col.lab = "thistle")
title("Main Title", sub = "sub title",
      cex.main = 2, font.main= 4, col.main= "blue",
      cex.sub = 0.75, font.sub = 3, col.sub = "red")

x <- seq(-4, 4, len = 101)
y <- cbind(sin(x), cos(x))
matplot(x, y, type = "l", xaxt = "n",
       main = expression(paste(plain(sin) * phi, " and ",
                           cos(phi))))
```
plain(cos) * phi)),
  ylab = expression("sin" * phi, "cos" * phi), # only 1st is taken
  xlab = expression(paste("Phase Angle \(\), phi)),
  col.main = "blue")
axis(1, at = c(-pi, -pi/2, 0, pi/2, pi),
     lab = expression(-pi, -pi/2, 0, pi/2, pi))
abline(h = 0, v = pi/2 * c(-1,1), lty = 2, lwd = .1, col = "gray70")

## Graphical Units

### Description

xinch and yinch convert the specified number of inches given as their arguments into the correct units for plotting with graphics functions. Usually, this only makes sense when normal coordinates are used, i.e., no log scale (see the `log` argument to `par`).

xyinch does the same for a pair of numbers xy, simultaneously.

### Usage

```r
xinch(x = 1, warn.log = TRUE)
yinch(y = 1, warn.log = TRUE)
xyinch(xy = 1, warn.log = TRUE)
```

### Arguments

- **x, y** numeric vector
- **xy** numeric of length 1 or 2.
- **warn.log** logical; if TRUE, a warning is printed in case of active log scale.

### Examples

```r
all(c(xinch(), yinch()) == xyinch()) # TRUE
xyinch()
xyinch # to see that is really delta("usr") / "pin"

## plot labels offset 0.12 inches to the right
## of plotted symbols in a plot
with(mtcars, {
  plot(mpg, disp, pch=19, main= "Motor Trend Cars")
  text(mpg + xinch(0.12), disp, row.names(mtcars),
       adj = 0, cex = .7, col = 'blue')
})
```
Chapter 5

The grid package

---

**absolute.size**  
*Absolute Size of a Grob*

**Description**

This function converts a unit object into absolute units. Absolute units are unaffected, but non-absolute units are converted into "null" units.

**Usage**

```r
absolute.size(unit)
```

**Arguments**

- `unit`  
  An object of class "unit".

**Details**

Absolute units are things like "inches", "cm", and "lines". Non-absolute units are "npc" and "native".

This function is designed to be used in widthDetails and heightDetails methods.

**Value**

An object of class "unit".

**Author(s)**

Paul Murrell

**See Also**

`widthDetails` and `heightDetails` methods.
convertNative  Convert a Unit Object to Native units

Description

This function is deprecated in grid version 0.8 and will be made defunct in grid version 1.9.

You should use the convertUnit() function or one of its close allies instead.

This function returns a numeric vector containing the specified x or y locations or dimensions, converted to "user" or "data" units, relative to the current viewport.

Usage

convertNative(unit, dimension="x", type="location")

Arguments

- unit: A unit object.
- dimension: Either "x" or "y".
- type: Either "location" or "dimension".

Value

A numeric vector.

WARNING

If you draw objects based on output from these conversion functions, then resize your device, the objects will be drawn incorrectly – the base R display list will not recalculate these conversions. This means that you can only rely on the results of these calculations if the size of your device is fixed.

Author(s)

Paul Murrell

See Also

grid.convert.unit

Examples

grid.newpage()
pushViewport(viewport(width=unit(.5, "npc"),
                     height=unit(.5, "npc")))
grid.rect()
w <- convertNative(unit(1, "inches"))
h <- convertNative(unit(1, "inches"), "y")
# This rectangle starts off life as 1in square, but if you
# resize the device it will no longer be 1in square
grid.rect(width=unit(w, "native"), height=unit(h, "native"),
          gp=gpar(col="red"))
popViewport(1)
# How to use grid.convert(), etc instead
convertNative(unit(1, "inches")) ==
  convertX(unit(1, "inches"), "native", valueOnly=TRUE)
convertNative(unit(1, "inches"), "y", "dimension") ==
  convertHeight(unit(1, "inches"), "native", valueOnly=TRUE)

### dataViewport

**Create a Viewport with Scales based on Data**

#### Description

This is a convenience function for producing a viewport with x- and/or y-scales based on numeric values passed to the function.

#### Usage

```r
dataViewport(xData = NULL, yData = NULL, xscale = NULL,
  yscale = NULL, extension = 0.05, ...)
```

#### Arguments

- **xData**: A numeric vector of data.
- **yData**: A numeric vector of data.
- **xscale**: A numeric vector (length 2).
- **yscale**: A numeric vector (length 2).
- **extension**: A numeric. If length greater than 1, then first value is used to extend the xscale and second value is used to extend the yscale.
- **...**: All other arguments will be passed to a call to the `viewport()` function.

#### Details

If `xscale` is not specified then the values in `x` are used to generate an x-scale based on the range of `x`, extended by the proportion specified in `extension`. Similarly for the y-scale.

#### Value

A grid viewport object.

#### Author(s)

Paul Murrell

#### See Also

`viewport` and `plotViewport`.
drawDetails

Customising grid Drawing

Description

These generic hook functions are called whenever a grid grob is drawn. They provide an opportunity for customising the drawing of a new class derived from grob (or gTree).

Usage

```r
drawDetails(x, recording)
draw.details(x, recording)
predrawDetails(x)
postDrawDetails(x)
```

Arguments

- `x`  
  A grid grob.

- `recording`  
  A logical value indicating whether a grob is being added to the display list or redrawn from the display list.

Details

These functions are called by the `grid.draw` methods for grobs and gTrees.

`predrawDetails` is called first during the drawing of a grob. This is where any additional viewports should be pushed (see, for example, `grid:::predrawDetails.frame`). Note that the default behaviour for grobs is to push any viewports in the `vp` slot, and for gTrees is to also push and up any viewports in the `childrenvp` slot so there is typically nothing to do here.

`drawDetails` is called next and is where any additional calculations and graphical output should occur (see, for example, `grid:::drawDetails.xaxis`). Note that the default behaviour for gTrees is to draw all grobs in the `children` slot so there is typically nothing to do here.

`postDrawDetails` is called last and should reverse anything done in `predrawDetails` (i.e., pop or up any viewports that were pushed; again, see, for example, `grid:::postDrawDetails.frame`). Note that the default behaviour for grobs is to pop any viewports that were pushed so there is typically nothing to do here.

Note that `predrawDetails` and `postDrawDetails` are also called in the calculation of "grobwidth" and "grobheight" units.

Value

None of these functions are expected to return a value.

Author(s)

Paul Murrell

See Also

`grid.draw`
**editDetails**

*Customising grid Editing*

**Description**

This generic hook function is called whenever a grid grob is edited via `grid.edit` or `editGrob`. This provides an opportunity for customising the editing of a new class derived from grob (or gTree).

**Usage**

```r
editDetails(x, specs)
```

**Arguments**

- `x` A grid grob.
- `specs` A list of named elements. The names indicate the grob slots to modify and the values are the new values for the slots.

**Details**

This function is called by `grid.edit` and `editGrob`. A method should be written for classes derived from grob or gTree if a change in a slot has an effect on other slots in the grob or children of a gTree (e.g., see `grid:::editDetails.xaxis`). Note that the slot already has the new value.

**Value**

The function MUST return the modified grob.

**Author(s)**

Paul Murrell

**See Also**

- `grid.edit`
- `gEdit`

---

**gEdit**

*Create and Apply Edit Objects*

**Description**

The functions `gEdit` and `gEditList` create objects representing an edit operation (essentially a list of arguments to `editGrob`).

The functions `applyEdit` and `applyEdits` apply one or more edit operations to a graphical object.

These functions are most useful for developers creating new graphical functions and objects.
Usage

```r
gEdit(...)  
gEditList(...)  
applyEdit(x, edit)  
applyEdits(x, edits)
```

Arguments

- `...` one or more arguments to the `editGrob` function (for `gEdit`) or one or more "gEdit" objects (for `gEditList`).
- `x` a grob (grid graphical object).
- `edit` a "gEdit" object.
- `edits` either a "gEdit" object or a "gEditList" object.

Value

- `gEdit` returns an object of class "gEdit".
- `gEditList` returns an object of class "gEditList".
- `applyEdit` and `applyEditList` return the modified grob.

Author(s)

Paul Murrell

See Also

grob editGrob

Examples

```r
grid.rect(gp=gpar(col="red"))  
# same thing, but more verbose  
grid.draw(applyEdit(rectGrob(), gEdit(gp=gpar(col="red"))))
```

---

**getNames**

List the names of grobs on the display list

Description

Returns a character vector containing the names of all top-level grobs on the display list.

Usage

```r
getNames()
```

Value

A character vector.
Description

**gpar()** should be used to create a set of graphical parameter settings. It returns an object of class "gpar". This is basically a list of name-value pairs.

**get.gpar()** can be used to query the current graphical parameter settings.

Usage

```r
gr1 = gpar(col = "red", lty = 2)
```

Arguments

- ... Any number of named arguments.
- `names` A character vector of valid graphical parameter names.

Details

All grid viewports and (predefined) graphical objects have a slot called `gp`, which contains a "gpar" object. When a viewport is pushed onto the viewport stack and when a graphical object is drawn, the settings in the "gpar" object are enforced. In this way, the graphical output is modified by the `gp` settings until the graphical object has finished drawing, or until the viewport is popped off the viewport stack, or until some other viewport or graphical object is pushed or begins drawing.

Valid parameter names are:

- **col** Colour for lines and borders.
- **fill** Colour for filling rectangles, polygons, ...
- **alpha** Alpha channel for transparency
- **lty** Line type
- **lwd** Line width
- **lex** Multiplier applied to line width
- **lineend** Line end style (round, butt, square)
- **linejoin** Line join style (round, mitre, bevel)
- **linemitre** Line mitre limit (number greater than 1)
- **fontsize** The size of text (in points)
- **cex** Multiplier applied to fontsize
- **fontfamily** The font family
- **fontface** The font face (bold, italic, ...)
- **lineheight** The height of a line as a multiple of the size of text
- **font** Font face (alias for fontface; for backward compatibility)
The alpha setting is combined with the alpha channel for individual colours by multiplying (with both alpha settings normalised to the range 0 to 1).

The size of text is \texttt{fontsize*cex}. The size of a line is \texttt{fontsize*cex*lineheight}.

The \texttt{cex} setting is cumulative; if a viewport is pushed with a \texttt{cex} of 0.5 then another viewport is pushed with a \texttt{cex} of 0.5, the effective \texttt{cex} is 0.25.

The alpha and \texttt{lex} settings are also cumulative.

Changes to the font family may be ignored by some devices, but is supported by PostScript, PDF, X11, Windows, and Quartz. The font family may be used to specify one of the Hershey Font families (e.g., HersheySerif) and this specification will be honoured on all devices.

The specification of font face can be an integer or a string. If an integer, then it follows the R base graphics standard: 1 = plain, 2 = bold, 3 = italic, 4 = bold italic. If a string, then valid values are: "plain", "bold", "italic", "oblique", and "bold.italic". For the special case of the HersheySerif font family, "cyrillic", "cyrillic.oblique", and "EUC" are also available.

Specifying the value NULL for a parameter is the same as not specifying any value for that parameter, except for \texttt{col} and \texttt{fill}, where NULL indicates not to draw a border or not to fill an area (respectively).

All parameter values can be vectors of multiple values. (This will not always make sense – for example, viewports will only take notice of the first parameter value.)

The gamma parameter is deprecated.

\texttt{get.gpar()} returns all current graphical parameter settings.

\textbf{Value}

An object of class "gpar".

\textbf{Author(s)}

Paul Murrell

\textbf{See Also}

Hershey.

\textbf{Examples}

\begin{verbatim}
gp <- get.gpar()
utils::str(gp)
## These *do* nothing but produce a "gpar" object:
gpar(col = "red")
gpar(col = "blue", lty = "solid", lwd = 3, fontsize = 16)
get.gpar(c("col", "lty"))
grid.newpage()
vp <- viewport(w = .8, h = .8, gp = gpar(col="blue"))
grid.draw(gTree(children=gList(rectGrob(gp = gpar(col="red")),
    textGrob(paste("The rect is its own colour (red)",
        "but this text is the colour",
        "set by the gTree (green)",
        sep = "\\n")))),
    gp = gpar(col="green"), vp = vp))
grid.text("This text is the colour set by the viewport (blue)",
y = 1, just = c("center", "bottom"),
\end{verbatim}
### gPath

This function can be used to generate a grob path for use in `grid.edit` and friends.

A grob path is a list of nested grob names.

#### Usage

```r
gPath(...)```

#### Arguments

...  
Character values which are grob names.

#### Details

Grob names must only be unique amongst grobs which share the same parent in a gTree.

This function can be used to generate a specification for a grob that includes the grob’s parent’s name (and the name of its parent and so on).

For interactive use, it is possible to directly specify a path, but it is strongly recommended that this function is used otherwise in case the path separator is changed in future versions of grid.

#### Value

A `gPath` object.

#### See Also

`grob, editGrob, addGrob, removeGrob, getGrob, setGrob`

#### Examples

```r
gPath("g1", "g2")```
Description

General information about the grid graphics package.

Details

Grid graphics provides an alternative to the standard R graphics. The user is able to define arbitrary rectangular regions (called viewports) on the graphics device and define a number of coordinate systems for each region. Drawing can be specified to occur in any viewport using any of the available coordinate systems.

Grid graphics and standard R graphics do not mix!

Type `library(help = grid)` to see a list of (public) Grid graphics functions.

Author(s)

Paul Murrell

See Also

viewport, grid.layout, and unit.

Examples

```r
## Diagram of a simple layout
grid.show.layout(grid.layout(4, 2,
                      heights=unit(rep(1, 4),
                          c("lines", "lines", "lines", "null")),
                      widths=unit(c(1, 1), "inches")))
## Diagram of a sample viewport
grid.show.viewport(viewport(x=0.6, y=0.6,
                     w=unit(1, "inches"), h=unit(1, "inches")))
## A flash plotting example
grid.multipanel(vp=viewport(0.5, 0.5, 0.8, 0.8))
```

Create a Grid Viewport

Description

These functions create viewports, which describe a rectangular regions on a graphics device and define a number of coordinate systems within those regions.
### Grid Viewports

**Usage**

```r
grid viewport(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
width = unit(1, "npc"), height = unit(1, "npc"),
default.units = "npc", just = "centre",
gp = gpar(), clip = "inherit",
xscale = c(0, 1), yscale = c(0, 1),
angle = 0,
layout = NULL,
layout.pos.row = NULL, layout.pos.col = NULL,
name = NULL)
```  

```r
vpList(...)  
vpStack(...)  
vpTree(parent, children)
```

**Arguments**

- **x**
  - A numeric vector or unit object specifying x-location.

- **y**
  - A numeric vector or unit object specifying y-location.

- **width**
  - A numeric vector or unit object specifying width.

- **height**
  - A numeric vector or unit object specifying height.

- **default.units**
  - A string indicating the default units to use if x, y, width, or height are only given as numeric vectors.

- **just**
  - A string or numeric vector specifying the justification of the viewport relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left alignment and 1 means right alignment.

- **gp**
  - An object of class gpar, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.

- **clip**
  - One of "on", "inherit", or "off", indicating whether to clip to the extent of this viewport, inherit the clipping region from the parent viewport, or turn clipping off altogether. For back-compatibility, a logical value of TRUE corresponds to "on" and FALSE corresponds to "inherit".

- **xscale**
  - A numeric vector of length two indicating the minimum and maximum on the x-scale.

- **yscale**
  - A numeric vector of length two indicating the minimum and maximum on the y-scale.

- **angle**
  - A numeric value indicating the angle of rotation of the viewport. Positive values indicate the amount of rotation, in degrees, anticlockwise from the positive x-axis.

- **layout**
  - A Grid layout object which splits the viewport into subregions.

- **layout.pos.row**
  - A numeric vector giving the rows occupied by this viewport in its parent’s layout.

- **layout.pos.col**
  - A numeric vector giving the columns occupied by this viewport in its parent’s layout.
name A character value to uniquely identify the viewport once it has been pushed onto the viewport tree.

... Any number of grid viewport objects.

parent A grid viewport object.

children A vpList object.

Details

The location and size of a viewport are relative to the coordinate systems defined by the viewport's parent (either a graphical device or another viewport). The location and size can be specified in a very flexible way by specifying them with unit objects. When specifying the location of a viewport, specifying both layout.pos.row and layout.pos.col as NULL indicates that the viewport ignores its parent's layout and specifies its own location and size (via its locn). If only one of layout.pos.row and layout.pos.col is NULL, this means occupy ALL of the appropriate row(s)/column(s). For example, layout.pos.row = 1 and layout.pos.col = NULL means occupy all of row 1. Specifying non-NULL values for both layout.pos.row and layout.pos.col means occupy the intersection of the appropriate rows and columns. If a vector of length two is specified for layout.pos.row or layout.pos.col, this indicates a range of rows or columns to occupy. For example, layout.pos.row = c(1, 3) and layout.pos.col = c(2, 4) means occupy cells in the intersection of rows 1, 2, and 3, and columns, 2, 3, and 4.

Clipping obeys only the most recent viewport clip setting. For example, if you clip to viewport1, then clip to viewport2, the clipping region is determined wholly by viewport2, the size and shape of viewport1 is irrelevant (until viewport2 is popped of course).

If a viewport is rotated (because of its own angle setting or because it is within another viewport which is rotated) then the clip flag is ignored.

Viewport names need not be unique. When pushed, viewports sharing the same parent must have unique names, which means that if you push a viewport with the same name as an existing viewport, the existing viewport will be replaced in the viewport tree. A viewport name can be any string, but grid uses the reserved name "ROOT" for the top-level viewport. Also, when specifying a viewport name in downViewport and seekViewport, it is possible to provide a viewport path, which consists of several names concatenated using the separator (currently ::). Consequently, it is not advisable to use this separator in viewport names.

The viewports in a vpList are pushed in parallel. The viewports in a vpStack are pushed in series. When a vpTree is pushed, the parent is pushed first, then the children are pushed in parallel.

Value

An R object of class viewport.

Author(s)

Paul Murrell

See Also

Grid, pushViewport, popViewport, downViewport, seekViewport, upViewport, unit, grid.layout, grid.show.layout.
# Diagram of a sample viewport
grid.show.viewport(viewport(x=0.6, y=0.6,
    w=unit(1, "inches"), h=unit(1, "inches")))

# Demonstrate viewport clipping
clip.demo <- function(i, j, clip1, clip2) {
    pushViewport(viewport(layout.pos.col=i,
        layout.pos.row=j))
    pushViewport(viewport(width=0.6, height=0.6, clip=clip1))
    grid.rect(gp=gpar(fill="white"))
    grid.circle(r=0.55, gp=gpar(col="red", fill="pink"))
    popViewport()
    pushViewport(viewport(width=0.6, height=0.6, clip=clip2))
    grid.polygon(x=c(0.5, 1.1, 0.6, 1.1, 0.5, -0.1, 0.4, -0.1),
        y=c(0.6, 1.1, 0.5, -0.1, 0.4, -0.1, 0.5, 1.1),
        gp=gpar(col="blue", fill="light blue"))
    popViewport(2)
}

grid.newpage()
grid.rect(gp=gpar(fill="grey"))
pushViewport(viewport(layout=grid.layout(2, 2)))
clip.demo(1, 1, FALSE, FALSE)
clip.demo(1, 2, TRUE, FALSE)
clip.demo(2, 1, FALSE, TRUE)
clip.demo(2, 2, TRUE, TRUE)
popViewport()

# Demonstrate turning clipping off
grid.newpage()
pushViewport(viewport(w=.5, h=.5, clip="on"))
grid.rect()
grid.circle(r=.6, gp=gpar(lwd=10))
pushViewport(viewport(clip="inherit"))
grid.circle(r=.6, gp=gpar(lwd=5, col="grey"))
pushViewport(viewport(clip="off"))
grid.circle(r=.6)
popViewport(3)

# Demonstrate vpList, vpStack, and vpTree
grid.newpage()
tree <- vpTree(viewport(w=0.8, h=0.8, name="A"),
    vpList(vpStack(viewport(x=0.1, y=0.1, w=0.5, h=0.5,
        just=c("left", "bottom"), name="B"),
        viewport(x=0.1, y=0.1, w=0.5, h=0.5,
        just=c("left", "bottom"), name="C"),
        viewport(x=0.1, y=0.1, w=0.5, h=0.5,
        just=c("left", "bottom"), name="D")),
        viewport(x=0.5, w=0.4, h=0.9,
        just="left", name="E")))
pushViewport(tree)
for (i in LETTERS[1:5]) {
    seekViewport(i)
    grid.rect()
    grid.text(current.vpTree(FALSE),
        x=unit(1, "mm"), y=unit(1, "npc") - unit(1, "mm"),
        just=c("left", "top"),
        gp=gpar(fontsize=8))
}
The Grid Graphics Package

Description

A rewrite of the graphics layout capabilities, plus some support for interaction.

Details

This package contains a new graphics system, which replaces (or extends) old-style S graphics.

Further information is available in the following vignettes:

- displaylist: Display Lists in grid (source, pdf)
- frame: Frames and packing grobs (source, pdf)
- grid: Introduction to grid (source, pdf)
- grobs: Working with grid grobs (source, pdf)
- interactive: Editing grid Graphics (source, pdf)
- locndimm: Locations versus Dimensions (source, pdf)
- moveline: Demonstrating move-to and line-to (source, pdf)
- nonfinite: Non-finite values (source, pdf)
- plotexample: Writing grid Code (source, pdf)
- rotated: Rotated Viewports (source, pdf)
- saveload: Persistent representations (source, pdf)
- sharing: Modifying multiple grobs simultaneously (source, pdf)
- viewports: Working with viewports (source, pdf)

For a complete list of functions, use `library(help="grid")`.

Author(s)

Paul Murrell (paul@stat.auckland.ac.nz)
Maintainer: R Core Team (R-core@r-project.org)

grid.add  

Add a Grid Graphical Object

Description

Add a grob to a gTree or a descendant of a gTree.

Usage

```r
grid.add(gPath, child, strict = FALSE, grep = FALSE, global = FALSE, allDevices = FALSE, redraw = TRUE)
addGrob(gTree, child, gPath = NULL, strict = FALSE, grep = FALSE, global = FALSE)
setChildren(x, children)
```
Arguments

- **gTree, x**: A gTree object.
- **gPath**: A gPath object. For `grid.add` this specifies a gTree on the display list. For `addGrob` this specifies a descendant of the specified gTree.
- **child**: A grob object.
- **children**: A gList object.
- **strict**: A boolean indicating whether the gPath must be matched exactly.
- **grep**: A boolean indicating whether the gPath should be treated as a regular expression. Values are recycled across elements of the gPath (e.g., c(TRUE, FALSE) means that every odd element of the gPath will be treated as a regular expression).
- **global**: A boolean indicating whether the function should affect just the first match of the gPath, or whether all matches should be affected.
- **allDevices**: A boolean indicating whether all open devices should be searched for matches, or just the current device. NOT YET IMPLEMENTED.
- **redraw**: A logical value to indicate whether to redraw the grob.

Details

- `addGrob` copies the specified grob and returns a modified grob.
- `grid.add` destructively modifies a grob on the display list. If `redraw` is TRUE it then redraws everything to reflect the change.
- `setChildren` is a basic function for setting all children of a gTree at once (instead of repeated calls to `addGrob`).

Value

- `addGrob` returns a grob object; `grid.add` returns NULL.

Author(s)

Paul Murrell

See Also

- `grob`, `getGrob`, `addGrob`, `removeGrob`.

Description

Functions to create and draw arrows at either end of a line, or at either end of a line.to, lines, or segments grob.
Usage

grid.arrows(x = c(0.25, 0.75), y = 0.5, default.units = "npc",
grob = NULL,
    angle = 30, length = unit(0.25, "inches"),
    ends = "last", type = "open", name = NULL,
    gp = gpar(), draw = TRUE, vp = NULL)

arrowsGrob(x = c(0.25, 0.75), y = 0.5, default.units = "npc",
grob = NULL,
    angle = 30, length = unit(0.25, "inches"),
    ends = "last", type = "open", name = NULL,
    gp = gpar(), vp = NULL)

Arguments

x A numeric vector or unit object specifying x-values.
y A numeric vector or unit object specifying y-values.
default.units A string indicating the default units to use if x or y are only given as numeric vectors.
grob A grob to add arrows to; currently can only be a line.to, lines, or segments grob.
angle A numeric specifying (half) the width of the arrow head (in degrees).
length A unit object specifying the length of the arrow head.
ends One of "first", "last", or "both", indicating which end of the line to add arrow heads.
type Either "open" or "closed" to indicate the type of arrow head.
name A character identifier.
gp An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw A logical value indicating whether graphics output should be produced.
vp A Grid viewport object (or NULL).

Details

Both functions create an arrows grob (a graphical object describing arrows), but only grid.arrows() draws the arrows (and then only if draw is TRUE).

If the grob argument is specified, this overrides any x and/or y arguments.

Value

An arrows grob. grid.arrows() returns the value invisibly.

Author(s)

Paul Murrell

See Also

Grid, viewport, grid.line.to, grid.lines, grid.segments
Examples

```r
pushViewport(viewport(layout=grid.layout(2, 4)))
pushViewport(viewport(layout.pos.col=1, layout.pos.row=1))
grid.rect(gp=gpar(col="grey"))
grid.arrows()
popViewport()
pushViewport(viewport(layout.pos.col=2, layout.pos.row=1))
grid.rect(gp=gpar(col="grey"))
grid.arrows(angle=15, type="closed")
popViewport()
pushViewport(viewport(layout.pos.col=3, layout.pos.row=1))
grid.rect(gp=gpar(col="grey"))
grid.arrows(angle=5, length=unit(0.1, "npc"),
  type="closed", gp=gpar(fill="white"))
popViewport()
pushViewport(viewport(layout.pos.col=4, layout.pos.row=1))
grid.rect(gp=gpar(col="grey"))
grid.arrows(x=unit(0:80/100, "npc"),
  y=unit(1 - (0:80/100)^2, "npc"))
popViewport()
pushViewport(viewport(layout.pos.col=1, layout.pos.row=2))
grid.rect(gp=gpar(col="grey"))
grid.arrows(ends="both")
popViewport()
pushViewport(viewport(layout.pos.col=2, layout.pos.row=2))
grid.rect(gp=gpar(col="grey"))
# Recycling arguments
grid.arrows(x=unit(1:10/11, "npc"), y=unit(1:3/4, "npc"))
popViewport()
pushViewport(viewport(layout.pos.col=3, layout.pos.row=2))
grid.rect(gp=gpar(col="grey"))
# Drawing arrows on a segments grob
gs <- segmentsGrob(x0=unit(1:4/5, "npc"),
  x1=unit(1:4/5, "npc"))
grid.arrows(grob=gs, length=unit(0.1, "npc"),
  type="closed", gp=gpar(fill="white"))
popViewport()
pushViewport(viewport(layout.pos.col=4, layout.pos.row=2))
grid.rect(gp=gpar(col="grey"))
# Arrows on a lines grob
# Name these because going to grid.edit them later
gl <- linesGrob(name="curve", x=unit(0:80/100, "npc"),
  y=unit((0:80/100)^2, "npc"))
grid.arrows(name="arrowOnLine", grob=gl, angle=15, type="closed",
  gp=gpar(fill="black"))
popViewport()
pushViewport(viewport(layout.pos.col=1, layout.pos.row=2))
```
grid.move.to(x=0.5, y=0.8)
popViewport()
pushViewport(viewport(layout.pos.col=4,
        layout.pos.row=1))

# Arrows on a line.to grob
glt <- lineToGrob(x=0.5, y=0.2, gp=gpar(lwd=3))
grid.arrows(grob=glt, ends="first", gp=gpar(lwd=3))
popViewport(2)
grid.edit(gPath("arrowOnLine", "curve"), y=unit((0:80/100)^3, "npc"))

grid.circle

Description
Functions to create and draw a circle.

Usage

grid.circle(x=0.5, y=0.5, r=0.5, default.units="npc", name=NULL,
        gp=gpar(), draw=TRUE, vp=NULL)
circleGrob(x=0.5, y=0.5, r=0.5, default.units="npc", name=NULL,
        gp=gpar(), vp=NULL)

Arguments

x        A numeric vector or unit object specifying x-locations.
y        A numeric vector or unit object specifying y-locations.
r        A numeric vector or unit object specifying radii.
default.units
        A string indicating the default units to use if x, y, width, or height are only
given as numeric vectors.
name
        A character identifier.
gp
        An object of class gpar, typically the output from a call to the function gpar.
        This is basically a list of graphical parameter settings.
draw
        A logical value indicating whether graphics output should be produced.
vp
        A Grid viewport object (or NULL).

Details
Both functions create a circle grob (a graphical object describing a circle), but only
grid.circle() draws the circle (and then only if draw is TRUE).
The radius may be given in any units; if the units are relative (e.g., "npc" or "native") then
the radius will be different depending on whether it is interpreted as a width or as a height. In such
cases, the smaller of these two values will be the result. To see the effect, type grid.circle()
and adjust the size of the window.

Value
A circle grob. grid.circle() returns the value invisibly.
grid.collection

Create a Coherent Group of Grid Graphical Objects

Description

This function is deprecated; please use gTree.

This function creates a graphical object which contains several other graphical objects. When it is
drawn, it draws all of its children.

It may be convenient to name the elements of the collection.

Usage

grid.collection(..., gp=gpar(), draw=TRUE, vp=NULL)

Arguments

... Zero or more objects of class "grob".

gp An object of class gpar, typically the output from a call to the function gpar.
This is basically a list of graphical parameter settings.

draw A logical value to indicate whether to produce graphical output.

vp A Grid viewport object (or NULL).

Value

A collection grob.

Author(s)

Paul Murrell

See Also

grid.grob.
grid.convert  
Convert Between Different grid Coordinate Systems

Description
These functions take a unit object and convert it to an equivalent unit object in a different coordinate system.

Usage
convertX(x, unitTo, valueOnly = FALSE)
convertY(x, unitTo, valueOnly = FALSE)
convertWidth(x, unitTo, valueOnly = FALSE)
convertHeight(x, unitTo, valueOnly = FALSE)
convertUnit(x, unitTo,
  axisFrom = "x", typeFrom = "location",
  axisTo = axisFrom, typeTo = typeFrom,
  valueOnly = FALSE)
grid.convertX(x, unitTo, valueOnly = FALSE)
grid.convertY(x, unitTo, valueOnly = FALSE)
grid.convertWidth(x, unitTo, valueOnly = FALSE)
grid.convertHeight(x, unitTo, valueOnly = FALSE)
grid.convert(x, unitTo,
  axisFrom = "x", typeFrom = "location",
  axisTo = axisFrom, typeTo = typeFrom,
  valueOnly = FALSE)

Arguments
x  A unit object.
unitTo The coordinate system to convert the unit to. See the unit function for valid coordinate systems.
axisFrom Either "x" or "y" to indicate whether the unit object represents a value in the x- or y-direction.
typeFrom Either "location" or "dimension" to indicate whether the unit object represents a location or a length.
axisTo Same as axisFrom, but applies to the unit object that is to be created.
typeTo Same as typeFrom, but applies to the unit object that is to be created.
valueOnly A logical indicating. If TRUE then the function does not return a unit object, but rather only the converted numeric values.

Details
The convertUnit function allows for general-purpose conversions. The other four functions are just more convenient front-ends to it for the most common conversions.

The conversions occur within the current viewport.

It is not currently possible to convert to all valid coordinate systems (e.g., "strwidth" or "grob-width"). I’m not sure if all of these are impossible, they just seem implausible at this stage.
In normal usage of grid, these functions should not be necessary. If you want to express a location or dimension in inches rather than user coordinates then you should simply do something like `unit(1, "inches")` rather than something like `unit(0.134, "native")`.

In some cases, however, it is necessary for the user to perform calculations on a unit value and this function becomes necessary. In such cases, please take note of the warning below.

The grid.* versions are just previous incarnations which have been deprecated.

**Value**

A unit object in the specified coordinate system (unless `valueOnly` is `TRUE` in which case the returned value is a numeric).

**Warning**

The conversion is only valid for the current device size. If the device is resized then at least some conversions will become invalid. For example, suppose that I create a unit object as follows: `oneinch <- convertUnit(unit(1, "inches"), "native")`. Now if I resize the device, the unit object in `oneinch` no longer corresponds to a physical length of 1 inch.

**Author(s)**

Paul Murrell

**See Also**

`unit`

**Examples**

```r
## A tautology
convertX(unit(1, "inches"), "inches")
## The physical units
convertX(unit(2.54, "cm"), "inches")
convertX(unit(25.4, "mm"), "inches")
convertX(unit(72.27, "points"), "inches")
convertX(unit(1/12*72.27, "picas"), "inches")
convertX(unit(72, "bigpts"), "inches")
convertX(unit(1157/1238*72.27, "dida"), "inches")
convertX(unit(1/12*1157/1238*72.27, "cicero"), "inches")
convertX(unit(65536*72.27, "scaledpts"), "inches")
convertX(unit(1/2.54, "inches"), "cm")
convertX(unit(1/25.4, "inches"), "mm")
convertX(unit(1/72.27, "inches"), "points")
convertX(unit(1/(1/12*72.27), "inches"), "picas")
convertX(unit(1/72, "inches"), "bigpts")
convertX(unit(1/(1157/1238*72.27), "inches"), "dida")
convertX(unit(1/(1/12*1157/1238*72.27), "inches"), "cicero")
convertX(unit(1/(65536*72.27), "inches"), "scaledpts")

pushViewport(viewport(width=unit(1, "inches"),
                     height=unit(2, "inches"),
                     xscale=c(0, 1),
                     yscale=c(1, 3)))

## Location versus dimension
convertY(unit(2, "native"), "inches")
```
grid.copy  Make a Copy of a Grid Graphical Object

Description

This function is redundant and will disappear in future versions.

Usage

grid.copy(grob)

Arguments

grob  A grob object.

Value

A copy of the grob object.

Author(s)

Paul Murrell

See Also

grid.grob.

grid.display.list  Control the Grid Display List

Description

Turn the Grid display list on or off.

Usage

grid.display.list(on=TRUE)
engine.display.list(on=TRUE)
grid.draw

Arguments

  on          A logical value to indicate whether the display list should be on or off.

Details

  All drawing and viewport-setting operations are (by default) recorded in the Grid display list. This allows redrawing to occur following an editing operation.
  This display list could get very large so it may be useful to turn it off in some cases; this will of course disable redrawing.
  All graphics output is also recorded on the main display list of the R graphics engine (by default). This supports redrawing following a device resize and allows copying between devices.
  Turning off this display list means that grid will redraw from its own display list for device resizes and copies. This will be slower than using the graphics engine display list.

Value

  None.

WARNING

  Turning the display list on causes the display list to be erased!
  Turning off both the grid display list and the graphics engine display list will result in no redrawing whatsoever.

Author(s)

  Paul Murrell

grid.draw  Draw a grid grob

description

  Produces graphical output from a graphical object.

Usage

  grid.draw(x, recording=TRUE)

Arguments

  x          An object of class "grob" or NULL.
  recording  A logical value to indicate whether the drawing operation should be recorded on the Grid display list.
Details

This is a generic function with methods for grob and gTree objects.

The grob and gTree methods automatically push any viewports in a vp slot and automatically apply any gpar settings in a gp slot. In addition, the gTree method pushes and ups any viewports in a childrenvp slot and automatically calls grid.draw for any grobs in a children slot.

The methods for grob and gTree call the generic hook functions preDrawDetails, drawDetails, and postDrawDetails to allow classes derived from grob or gTree to perform additional viewport pushing/popping and produce additional output beyond the default behaviour for grobs and gTrees.

Value

None.

Author(s)

Paul Murrell

See Also

grob.

Examples

grid.newpage()
## Create a graphical object, but don't draw it
l <- linesGrob()
## Draw it
grid.draw(l)

grid.edit Edit the Description of a Grid Graphical Object

Description

Changes the value of one of the slots of a grob and redraws the grob.

Usage

grid.edit(gPath, ..., strict = FALSE, grep = FALSE, global = FALSE, allDevices = FALSE, redraw = TRUE)
editGrob(grob, gPath = NULL, ..., strict = FALSE, grep = FALSE, global = FALSE)

Arguments

grob A grob object.
...
Zero or more named arguments specifying new slot values.
gPath A gPath object. For grid.edit this specifies a grob on the display list. For editGrob this specifies a descendant of the specified grob.
grid.frame 689

strict A boolean indicating whether the gPath must be matched exactly.
grep A boolean indicating whether the gPath should be treated as a regular expression. Values are recycled across elements of the gPath (e.g., c(TRUE, FALSE) means that every odd element of the gPath will be treated as a regular expression).
global A boolean indicating whether the function should affect just the first match of the gPath, or whether all matches should be affected.
allDevices A boolean indicating whether all open devices should be searched for matches, or just the current device. NOT YET IMPLEMENTED.
redraw A logical value to indicate whether to redraw the grob.

Details
editGrob copies the specified grob and returns a modified grob.
grid.edit destructively modifies a grob on the display list. If redraw is TRUE it then redraws everything to reflect the change.
Both functions call editDetails to allow a grob to perform custom actions and validDetails to check that the modified grob is still coherent.

Value
editGrob returns a grob object; grid.edit returns NULL.

Author(s)
Paul Murrell

See Also
grob, getGrob, addGrob, removeGrob.

Examples
grid.newpage()
grid.xaxis(name = "xa", vp = viewport(width=.5, height=.5))
grid.edit("xa", gp = gpar(col="red"))
  # won't work because no ticks (at is NULL)
try(grid.edit(gPath("xa", "ticks"), gp = gpar(col="green")))
grid.edit("xa", at = 1:4/5)
  # Now it should work
try(grid.edit(gPath("xa", "ticks"), gp = gpar(col="green")))

grid.frame Create a Frame for Packing Objects

Description
These functions, together with grid.pack, grid.place, packGrob, and placeGrob are part of a GUI-builder-like interface to constructing graphical images. The idea is that you create a frame with this function then use grid.pack or whatever to pack/place objects into the frame.
grid.frame

Usage

grid.frame(layout=NULL, name=NULL, gp=gpar(), vp=NULL, draw=TRUE)
frameGrob(layout=NULL, name=NULL, gp=gpar(), vp=NULL)

Arguments

layout A Grid layout, or NULL. This can be used to initialise the frame with a number
of rows and columns, with initial widths and heights, etc.
name A character identifier.
vp An object of class viewport, or NULL.
gp An object of class gpar; typically the output from a call to the function gpar.
draw Should the frame be drawn.

Details

Both functions create a frame grob (a graphical object describing a frame), but only
grid.frame() draws the frame (and then only if draw is TRUE). Nothing will actually be
drawn, but it will put the frame on the display list, which means that the output will be dynamically
updated as objects are packed into the frame. Possibly useful for debugging.

Value

A frame grob. grid.frame() returns the value invisibly.

Author(s)

Paul Murrell

See Also

grid.pack

Examples

grid.newpage()
grid.frame(name="gf", draw=TRUE)
grid.pack("gf", rectGrob(gp=gpar(fill="grey")), width=unit(1, "null"))
grid.pack("gf", textGrob("hi there"), side="right")

grid.get

Get a Grid Graphical Object

Description

Retrieve a grob or a descendant of a grob.

Usage

grid.get(gPath, strict = FALSE, grep = FALSE, global = FALSE,
allDevices = FALSE)

grob(gTree, gPath, strict = FALSE, grep = FALSE, global = FALSE)
Arguments

- **gTree**: A gTree object.
- **gPath**: A gPath object. For `grid.get` this specifies a grob on the display list. For `getGrob` this specifies a descendant of the specified gTree.
- **strict**: A boolean indicating whether the gPath must be matched exactly.
- **grep**: A boolean indicating whether the gPath should be treated as a regular expression. Values are recycled across elements of the gPath (e.g., `c(TRUE, FALSE)` means that every odd element of the gPath will be treated as a regular expression).
- **global**: A boolean indicating whether the function should affect just the first match of the gPath, or whether all matches should be affected.
- **allDevices**: A boolean indicating whether all open devices should be searched for matches, or just the current device. NOT YET IMPLEMENTED.

Value

A grob object.

Author(s)

Paul Murrell

See Also

`grob`, `getGrob`, `addGrob`, `removeGrob`.

Examples

```r
grid.xaxis(name="xa")
grid.get("xa")
grid.get(gPath("xa", "ticks"))

grid.draw(gTree(name="gt", children=gList(xaxisGrob(name="axis"))))
grid.get(gPath("gt", "axis", "ticks"))
```

```
grid.grab

Grab the current grid output
```

Description

Creates a gTree object from the current grid display list or from a scene generated by user-specified code.

Usage

```r
grid.grab(warn = 2, wrap = FALSE, ...)
grid.grabExpr(expr, warn = 2, wrap = FALSE, ...)
```
Arguments

expr  An expression to be evaluated. Typically, some calls to grid drawing functions.

warn  An integer specifying the amount of warnings to emit. 0 means no warnings, 1 means warn when it is certain that the grab will not faithfully represent the original scene. 2 means warn if there’s any possibility that the grab will not faithfully represent the original scene.

wrap  A logical indicating how the output should be captured. If TRUE, each non-grob element on the display list is captured by wrapping it in a grob.

... arguments passed to gTree, for example, a name and/or class for the gTree that is created.

Details

There are four ways to capture grid output as a gTree.

There are two functions for capturing output: use grid.grab to capture an existing drawing and grid.grabExpr to capture the output from an expression (without drawing anything).

For each of these functions, the output can be captured in two ways. One way tries to be clever and make a gTree with a childrenvp slot containing all viewports on the display list (including those that are popped) and every grob on the display list as a child of the new gTree; each child has a vpPath in the vp slot so that it is drawn in the appropriate viewport. In other words, the gTree contains all elements on the display list, but in a slightly altered form.

The other way, wrap=TRUE, is to create a grob for every element on the display list (and make all of those grobs children of the gTree).

The first approach creates a more compact and elegant gTree, which is more flexible to work with, but is not guaranteed to faithfully replicate all possible grid output. The second approach is more brute force, and harder to work with, but should always faithfully replicate the original output.

Value

A gTree object.

See Also

gTree

Examples

pushViewport(viewport(w=.5, h=.5))
grid.rect()
grid.points(runif(10), runif(10))
popViewport()
grab <- grid.grab()
grid.newpage()
grid.draw(grab)
**grid.grill**

*Draw a Grill*

**Description**

This function draws a grill within a Grid viewport.

**Usage**

```r
grid.grill(h = unit(seq(0.25, 0.75, 0.25), "npc"),
          v = unit(seq(0.25, 0.75, 0.25), "npc"),
          default.units = "npc", gp=gpar(col = "grey"), vp = NULL)
```

**Arguments**

- **h**: A numeric vector or unit object indicating the horizontal location of the vertical grill lines.
- **v**: A numeric vector or unit object indicating the vertical location of the horizontal grill lines.
- **default.units**: A string indicating the default units to use if `h` or `v` are only given as numeric vectors.
- **gp**: An object of class `gpar`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
- **vp**: A Grid viewport object.

**Value**

None.

**Author(s)**

Paul Murrell

**See Also**

- `Grid`, `viewport`.

---

**grid.grob**

*Create a Grid Graphical Object*

**Description**

These functions create grid graphical objects.
grid.grob

Usage

grid.grob(list.struct, cl = NULL, draw = TRUE)
grob(..., name = NULL, gp = NULL, vp = NULL, cl = NULL)
gTree(..., name = NULL, gp = NULL, vp = NULL, children = NULL,
  childrenvp = NULL, cl = NULL)
childNames(gTree)
gList(...)

Arguments

... For grob and gTree, the named slots describing important features of the
  graphical object. For gList, a series of grob objects.
list.struct A list (preferably with each element named).
name A character identifier for the grob. Used to find the grob on the display list
  and/or as a child of another grob.
children A gList object.
childrenvp A viewport object (or NULL).
gp A gpar object, typically the output from a call to the function gpar. This is
  basically a list of graphical parameter settings.
vp A viewport object (or NULL).
cl A string giving the class attribute for the list.struct
draw A logical value to indicate whether to produce graphical output.
gTree A gTree object.

Details

These functions can be used to create a basic grob, gTree, or gList object, or a new class derived
from one of these.

A grid graphical object (grob) is a description of a graphical item. These basic classes provide
default behaviour for validating, drawing, and modifying graphical objects. Both call the function
validDetails to check that the object returned is coherent.

A gTree can have other grobs as children; when a gTree is drawn, it draws all of its children.
Before drawing its children, a gTree pushes its childrenvp slot and then navigates back up (calls
upViewport) so that the children can specify their location within the childrenvp via a vpPath.

Grob names need not be unique in general, but all children of a gTree must have different names.
A grob name can be any string, though it is not advisable to use the gPath separator (currently ::)
in grob names.

The function childNames returns the names of the grobs which are children of a gTree.

All grid primitives (grid.lines, grid.rect,...) and some higher-level grid components (e.g.,
grid.xaxis and grid.yaxis) are derived from these classes.

grid.grob is deprecated.

Value

A grob object.

Author(s)

Paul Murrell
grid.layout

See Also
grid.draw, grid.edit, grid.get.

Examples

grid.layout(nrow = 1, ncol = 1,
            widths = unit(rep(1, ncol), "null"),
            heights = unit(rep(1, nrow), "null"),
            default.units = "null", respect = FALSE,
            just="centre")

Description

This function returns a Grid layout, which describes a subdivision of a rectangular region.

Usage

grid.layout(nrow = 1, ncol = 1,
            widths = unit(rep(1, ncol), "null"),
            heights = unit(rep(1, nrow), "null"),
            default.units = "null", respect = FALSE,
            just="centre")

Arguments

nrow An integer describing the number of rows in the layout.
ncol An integer describing the number of columns in the layout.
widths A numeric vector or unit object describing the widths of the columns in the layout.
heights A numeric vector or unit object describing the heights of the rows in the layout.
default.units A string indicating the default units to use if widths or heights are only given as numeric vectors.
respect A logical value or a numeric matrix. If a logical, this indicates whether row heights and column widths should respect each other. If a matrix, non-zero values indicate that the corresponding row and column should be respected (see examples below).
just A string vector indicating how the layout should be justified if it is not the same size as its parent viewport. If there are two values, the first value specifes horizontal justification and the second value specifies vertical justification. Possible values are: "left", "right", "centre", "center", "bottom", and "top". NOTE that in this context, "left", for example, means align the left edge of the left-most layout column with the left edge of the parent viewport.

Details

The unit objects given for the widths and heights of a layout may use a special units that only has meaning for layouts. This is the "null" unit, which indicates what relative fraction of the available width/height the column/row occupies. See the reference for a better description of relative widths and heights in layouts.
A Grid layout object.

**WARNING**

This function must NOT be confused with the base R graphics function `layout`. In particular, do not use `layout` in combination with Grid graphics. The documentation for `layout` may provide some useful information and this function should behave identically in comparable situations. The `grid.layout` function has added the ability to specify a broader range of units for row heights and column widths, and allows for nested layouts (see `viewport`).

**Author(s)**

Paul Murrell

**References**


**See Also**

`Grid`, `grid.show.layout`, `viewport`, `layout`

**Examples**

```r
## A variety of layouts (some a bit mid-bending ...)
layout.torture()
## Demonstration of layout justification
grid.newpage()

testlay <- function(just="centre") {
  pushViewport(viewport(layout=grid.layout(1, 1, widths=unit(1, "inches"),
                          height=unit(0.25, "npc"),
                          just=just)))
  pushViewport(viewport(layout.pos.col=1, layout.pos.row=1))
  grid.rect()
  grid.text(paste(just, collapse="-"))
  popViewport(2)
}

  testlay()
  testlay(c("left", "top"))
  testlay(c("right", "top"))
  testlay(c("right", "bottom"))
  testlay(c("left", "bottom"))
  testlay(c("left"))
  testlay(c("right"))
  testlay(c("bottom"))
  testlay(c("top"));
```
grid.lines

Draw Lines in a Grid Viewport

Description

These functions create and draw a series of lines.

Usage

grid.lines(x = unit(c(0, 1), "npc", units.per.obs),
           y = unit(c(0, 1), "npc", units.per.obs),
           default.units = "npc", units.per.obs = FALSE, name = NULL,
           gp=gpar(), draw = TRUE, vp = NULL)

linesGrob(x = unit(c(0, 1), "npc", units.per.obs),
          y = unit(c(0, 1), "npc", units.per.obs),
          default.units = "npc", units.per.obs = FALSE, name = NULL,
          gp=gpar(), vp = NULL)

Arguments

x
A numeric vector or unit object specifying x-values.

y
A numeric vector or unit object specifying y-values.

default.units
A string indicating the default units to use if x or y are only given as numeric vectors.

units.per.obs
A logical value to indicate whether each individual (x, y) location has its own unit(s) specified.

name
A character identifier.

gp
An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.

draw
A logical value indicating whether graphics output should be produced.

vp
A Grid viewport object (or NULL).

Details

Both functions create a lines grob (a graphical object describing lines), but only grid.lines draws the lines (and then only if draw is TRUE).

Value

A lines grob. grid.lines returns the value invisibly.

Author(s)

Paul Murrell

See Also

Grid, viewport
Capture a Mouse Click

Description

Allows the user to click the mouse once within the current graphics device and returns the location of the mouse click within the current viewport, in the specified coordinate system.

Usage

grid.locator(unit = "native")

Arguments

unit

The coordinate system in which to return the location of the mouse click. See the unit function for valid coordinate systems.

Details

This function is modal (like the base function locator) so the command line and graphics drawing is blocked until the use has clicked the mouse in the current device.

Value

A unit object representing the location of the mouse click within the current viewport, in the specified coordinate system.

If the user did not click mouse button 1, the function (invisibly) returns NULL.

Author(s)

Paul Murrell

See Also

viewport, unit, locator

Examples

if (interactive()) {
  ## Need to write a more sophisticated unit as.character method
  unittrim <- function(unit) {
    sub("^([0-9]+|[0-9]+[.][0-9]+)[0-9]*", "\1", as.character(unit))
  }
  do.click <- function(unit) {
    click.locn <- grid.locator(unit)
    grid.segments(unit.c(click.locn$x, unit(0, "npc")),
                   unit.c(unit(0, "npc"), click.locn$y),
                   click.locn$x, click.locn$y,
                   gp=gpar(lty="dashed", col="grey"))
    grid.points(click.locn$x, click.locn$y, pch=16, size=unit(1, "mm"))
    clickx <- unittrim(click.locn$x)
    clicky <- unittrim(click.locn$y)
    grid.text(paste("(*", clickx, ",", clicky, ")", sep=""),
```
grid.move.to

```
click.locn$x + unit(2, "mm"), click.locn$y,
just="left")
)
dclick("inches")
pushViewport(viewport(width=0.5, height=0.5,
    xscale=c(0, 100), yscale=c(0, 10)))
grid.rect()
grid.xaxis()
grid.yaxis()
do.click("native")
popViewport()
```

---

**grid.move.to**  
*Move or Draw to a Specified Position*

**Description**

Grid has the notion of a current location. These functions sets that location.

**Usage**

```
grid.move.to(x = 0, y = 0, default.units = "npc", name = NULL,
draw = TRUE, vp = NULL)
moveToGrob(x = 0, y = 0, default.units = "npc", name = NULL, vp = NULL)
grid.line.to(x = 1, y = 1, default.units = "npc", name = NULL,
gp = gpar(), draw = TRUE, vp = NULL)
lineToGrob(x = 1, y = 1, default.units = "npc", name = NULL,
gp = gpar(), vp = NULL)
```

**Arguments**

- `x` A numeric value or a unit object specifying an x-value.
- `y` A numeric value or a unit object specifying a y-value.
- `default.units` A string indicating the default units to use if `x` or `y` are only given as numeric values.
- `name` A character identifier.
- `draw` A logical value indicating whether graphics output should be produced.
- `gp` An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
- `vp` A Grid viewport object (or NULL).

**Details**

Both functions create a move.to/line.to grob (a graphical object describing a move-to/line-to), but only **grid.move.to/line.to()** draws the move.to/line.to (and then only if `draw` is TRUE).
Value

A move.to/line.to grob. `grid.move.to/line.to()` returns the value invisibly.

Author(s)

Paul Murrell

See Also

`Grid`, `viewport`

Examples

```r
grid.newpage()
grid.move.to(0.5, 0.5)
grid.line.to(1, 1)
grid.line.to(0.5, 0)
pushViewport(viewport(x=0, y=0, w=0.25, h=0.25, just=c("left", "bottom")))
grid.rect()
grid.grill()
grid.line.to(0.5, 0.5)
popViewport()
```

---

**grid.newpage**

Move to a New Page on a Grid Device

Description

This function erases the current device or moves to a new page.

Usage

```r
grid.newpage(recording = TRUE)
```

Arguments

- **recording**: A logical value to indicate whether the new-page operation should be saved onto the Grid display list.

Details

There is a hook called "grid.newpage" (see `setHook`) which is used in the testing code to annotate the new page. The hook function(s) are called with no argument. (If the value is a character string, `get` is called on it from within the `grid` namespace.)

Value

None.

Author(s)

Paul Murrell
grid.pack

See Also
Grid

grid.pack  Pack an Object within a Frame

Description
This function, together with grid.frame and frameGrob, are part of a GUI-builder-like interface to constructing graphical images. The idea is that you create a frame with grid.frame or frameGrob then use this function to pack objects into the frame.

Usage
grid.pack(gPath, grob, redraw = TRUE, side = NULL,
   row = NULL, row.before = NULL, row.after = NULL,
   col = NULL, col.before = NULL, col.after = NULL,
   width = NULL, height = NULL,
   force.width = FALSE, force.height = FALSE, border = NULL,
   dynamic = FALSE)

packGrob(frame, grob, side = NULL,
   row = NULL, row.before = NULL, row.after = NULL,
   col = NULL, col.before = NULL, col.after = NULL,
   width = NULL, height = NULL,
   force.width = FALSE, force.height = FALSE, border = NULL,
   dynamic = FALSE)

Arguments
gPath  A gPath object, which specifies a frame on the display list.
frame  An object of class frame, typically the output from a call to grid.frame.
grob  An object of class grob. The object to be packed.
redraw  A boolean indicating whether the output should be updated.
side  One of "left", "top", "right", "bottom" to indicate which side to pack the object on.
row  Which row to add the object to. Must be between 1 and the-number-of-rows-currently-in-the-frame + 1, or NULL in which case the object occupies all rows.
row.before  Add the object to a new row just before this row.
row.after  Add the object to a new row just after this row.
col  Which col to add the object to. Must be between 1 and the-number-of-cols-currently-in-the-frame + 1, or NULL in which case the object occupies all cols.
col.before  Add the object to a new col just before this col.
col.after  Add the object to a new col just after this col.
width  Specifies the width of the column that the object is added to (rather than allowing the width to be taken from the object).
height  Specifies the height of the row that the object is added to (rather than allowing the height to be taken from the object).

force.width  A logical value indicating whether the width of the column that the grob is being packed into should be EITHER the width specified in the call to grid.pack OR the maximum of that width and the pre-existing width.

force.height  A logical value indicating whether the height of the column that the grob is being packed into should be EITHER the height specified in the call to grid.pack OR the maximum of that height and the pre-existing height.

border  A unit object of length 4 indicating the borders around the object.

dynamic  If the width/height is taken from the grob being packed, this boolean flag indicates whether the grobwidth/height unit refers directly to the grob, or uses a gPath to the grob. In the latter case, changes to the grob will trigger a recalculation of the width/height.

Details

packGrob modifies the given frame grob and returns the modified frame grob.

grid.pack destructively modifies a frame grob on the display list (and redraws the display list if redraw is TRUE).

These are (meant to be) very flexible functions. There are many different ways to specify where the new object is to be added relative to the objects already in the frame. The function checks that the specification is not self-contradictory.

NOTE that the width/height of the row/col that the object is added to is taken from the object itself unless the width/height is specified.

Value

packGrob returns a frame grob, but grid.pack returns NULL.

Author(s)

Paul Murrell

See Also

grid.frame, grid.place, grid.edit, and gPath.

grid.place  Place an Object within a Frame

Description

These functions provide a simpler (and faster) alternative to the grid.pack() and packGrob functions. They can be used to place objects within the existing rows and columns of a frame layout. They do not provide the ability to add new rows and columns nor do they affect the heights and widths of the rows and columns.

Usage

grid.place(gPath, grob, row = 1, col = 1, redraw = TRUE)
placeGrob(frame, grob, row = NULL, col = NULL)
grid.plot.and.legend

Arguments

- **gPath**: A gPath object, which specifies a frame on the display list.
- **frame**: An object of class `frame`, typically the output from a call to `grid.frame`.
- **grob**: An object of class `grob`. The object to be placed.
- **row**: Which row to add the object to. Must be between 1 and the-number-of-rows-currently-in-the-frame.
- **col**: Which col to add the object to. Must be between 1 and the-number-of-cols-currently-in-the-frame.
- **redraw**: A boolean indicating whether the output should be updated.

Details

- `placeGrob` modifies the given frame grob and returns the modified frame grob.
- `grid.place` destructively modifies a frame grob on the display list (and redraws the display list if `redraw` is `TRUE`).

Value

- `placeGrob` returns a frame grob, but `grid.place` returns `NULL`.

Author(s)

- Paul Murrell

See Also

- `grid.frame`, `grid.pack`, `grid.edit`, and `gPath`.

---

grid.plot.and.legend

*A Simple Plot and Legend Demo*

Description

This function is just a wrapper for a simple demonstration of how a basic plot and legend can be drawn from scratch using grid.

Usage

- `grid.plot.and.legend()`

Author(s)

- Paul Murrell

Examples

- `grid.plot.and.legend()`
grid.points  

**Description**

These functions create and draw data symbols.

**Usage**

```r
grid.points(x = runif(10),
            y = runif(10),
            pch = 1, size = unit(1, "char"),
            default.units = "native", name = NULL,
            gp=gpar(), draw = TRUE, vp = NULL)
pointsGrob(x = runif(10),
           y = runif(10),
           pch = 1, size = unit(1, "char"),
           default.units = "native", name = NULL,
           gp=gpar(), vp = NULL)
```

**Arguments**

- `x`: A numeric vector or unit object specifying x-values.
- `y`: A numeric vector or unit object specifying y-values.
- `pch`: A numeric or character vector indicating what sort of plotting symbol to use.
- `size`: A unit object specifying the size of the plotting symbols.
- `default.units`: A string indicating the default units to use if `x` or `y` are only given as numeric vectors.
- `name`: A character identifier.
- `gp`: An object of class `gpar`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
- `draw`: A logical value indicating whether graphics output should be produced.
- `vp`: A Grid viewport object (or NULL).

**Details**

Both functions create a points grob (a graphical object describing points), but only `grid.points` draws the points (and then only if `draw` isTRUE).

**Value**

A points grob. `grid.points` returns the value invisibly.

**Author(s)**

Paul Murrell

**See Also**

`Grid, viewport`
grid.polygon

Draw a Polygon

Description

These functions create and draw a polygon. The final point will automatically be connected to the initial point.

Usage

grid.polygon(x=c(0, 0.5, 1, 0.5), y=c(0.5, 1, 0.5, 0),
     id=NULL, id.lengths=NULL,
     default.units="npc", name=NULL,
     gp=gpar(), draw=TRUE, vp=NULL)

drawgrob(x=c(0, 0.5, 1, 0.5), y=c(0.5, 1, 0.5, 0),
     id=NULL, id.lengths=NULL,
     default.units="npc", name=NULL,
     gp=gpar(), vp=NULL)

Arguments

x A numeric vector or unit object specifying x-locations.
y A numeric vector or unit object specifying y-locations.
id A numeric vector used to separate locations in x and y into multiple polygons. All locations with the same id belong to the same polygon.
id.lengths A numeric vector used to separate locations in x and y into multiple polygons. Specifies consecutive blocks of locations which make up separate polygons.
default.units A string indicating the default units to use if x, y, width, or height are only given as numeric vectors.
name A character identifier.
gp An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
draw A logical value indicating whether graphics output should be produced.
vp A Grid viewport object (or NULL).

Details

Both functions create a polygon grob (a graphical object describing a polygon), but only grid.polygon draws the polygon (and then only if draw is TRUE).

Value

A grob object.

Author(s)

Paul Murrell
See Also

Grid.viewport

Examples

grid.polygon()
# Using id (NOTE: locations are not in consecutive blocks)
grid.newpage()
grid.polygon(x=c((0:4)/10, rep(.5, 5), (10:6)/10, rep(.5, 5)),
    y=c(rep(.5, 5), (10:6/10), rep(.5, 5), (0:4)/10),
    id=rep(1:5, 4),
    gp=gpar(fill=1:5))
# Using id.lengths
grid.newpage()
grid.polygon(x=outer(c(0, .5, 1, .5), 5:1/5),
    y=outer(c(.5, 1, .5, 0), 5:1/5),
    id.lengths=rep(4, 5),
    gp=gpar(fill=1:5))
**grid.prompt**  
*Prompt before new page*

**Description**

This function can be used to control whether the user is prompted before starting a new page of output.

**Usage**

```
grid.prompt(ask)
```

**Arguments**

- **ask** a logical value. If `TRUE`, the user is prompted before a new page of output is started.

**Value**

The current prompt setting *before* any new setting is applied.

**Author(s)**

Paul Murrell

**See Also**

- `grid.newpage`

---

**grid.record**  
*Encapsulate calculations and drawing*

**Description**

Evaluates an expression that includes both calculations and drawing that depends on the calculations so that both the calculations and the drawing will be rerun when the scene is redrawn (e.g., device resize or editing).

Intended only *for expert use*.

**Usage**

```
recordGrob(expr, list, name=NULL, gp=NULL, vp=NULL)
grid.record(expr, list, name=NULL, gp=NULL, vp=NULL)
```
Arguments

- `expr`: object of mode `expression` or call or an “unevaluated expression”.
- `list`: a list defining the environment in which `expr` is to be evaluated.
- `name`: A character identifier.
- `gp`: An object of class `gpar`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
- `vp`: A Grid viewport object (or NULL).

Details

A grob is created of special class "recordedGrob" (and drawn, in the case of `grid.record`). The `drawDetails` method for this class evaluates the expression with the list as the evaluation environment (and the grid Namespace as the parent of that environment).

Note

This function **must** be used instead of the function `recordGraphics`; all of the dire warnings about using `recordGraphics` responsibly also apply here.

Author(s)

Paul Murrell

See Also

- `recordGraphics`

Examples

```r
grid.record({
  w <- convertWidth(unit(1, "inches"), "npc")
  grid.rect(width=w)
},
  list())
```

Description

These functions create and draw rectangles.

Usage

```r
grid.rect(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
  width = unit(1, "npc"), height = unit(1, "npc"),
  just = "centre", hjust = NULL, vjust = NULL,
  default.units = "npc", name = NULL,
  gp=gpar(), draw = TRUE, vp = NULL)
rectGrob(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
  width = unit(1, "npc"), height = unit(1, "npc"),
  gp=gpar(), draw = TRUE, vp = NULL)
```
Arguments

x
A numeric vector or unit object specifying x-location.

y
A numeric vector or unit object specifying y-location.

width
A numeric vector or unit object specifying width.

height
A numeric vector or unit object specifying height.

just
The justification of the rectangle relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left alignment and 1 means right alignment.

hjust
A numeric vector specifying horizontal justification. If specified, overrides the just setting.

vjust
A numeric vector specifying vertical justification. If specified, overrides the just setting.

default.units
A string indicating the default units to use if x, y, width, or height are only given as numeric vectors.

name
A character identifier.

gp
An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.

draw
A logical value indicating whether graphics output should be produced.

vp
A Grid viewport object (or NULL).

Details

Both functions create a rect grob (a graphical object describing rectangles), but only grid.rect draws the rectangles (and then only if draw is TRUE).

Value

A rect grob. grid.rect returns the value invisibly.

Author(s)

Paul Murrell

See Also

Grid, viewport
grid.refresh  
*Refresh the current grid scene*

**Description**
Replays the current grid display list.

**Usage**

```r
grid.refresh()
```

**Author(s)**
Paul Murrell

---

grid.remove  
*Remove a Grid Graphical Object*

**Description**
Remove a grob from a gTree or a descendant of a gTree.

**Usage**

```r
grid.remove(gPath, warn = TRUE, strict = FALSE, grep = FALSE,
            global = FALSE, allDevices = FALSE, redraw = TRUE)
```

```r
removeGrob(gTree, gPath, strict = FALSE, grep = FALSE, global = FALSE,
           warn = TRUE)
```

**Arguments**
- **gTree**: A gTree object.
- **gPath**: A gPath object. For grid.remove this specifies a gTree on the display list. For removeGrob this specifies a descendant of the specified gTree.
- **strict**: A boolean indicating whether the gPath must be matched exactly.
- **grep**: A boolean indicating whether the gPath should be treated as a regular expression. Values are recycled across elements of the gPath (e.g., c(TRUE, FALSE) means that every odd element of the gPath will be treated as a regular expression).
- **global**: A boolean indicating whether the function should affect just the first match of the gPath, or whether all matches should be affected.
- **allDevices**: A boolean indicating whether all open devices should be searched for matches, or just the current device. NOT YET IMPLEMENTED.
- **warn**: A logical to indicate whether failing to find the specified grob should trigger an error.
- **redraw**: A logical value to indicate whether to redraw the grob.
grid.segments

Details

removeGrob copies the specified grob and returns a modified grob.

grid.remove destructively modifies a grob on the display list. If redraw is TRUE it then redraws everything to reflect the change.

Value

removeGrob returns a grob object; grid.remove returns NULL.

Author(s)

Paul Murrell

See Also

grob, getGrob, removeGrob, removeGrob.

grid.segments Draw Line Segments

Description

These functions create and draw line segments.

Usage

grid.segments(x0 = unit(0, "npc"), y0 = unit(0, "npc"),
             x1 = unit(1, "npc"), y1 = unit(1, "npc"),
             default.units = "npc", units.per.obs = FALSE,
             name = NULL, gp = gpar(), draw = TRUE, vp = NULL)

segmentsGrob(x0 = unit(0, "npc"), y0 = unit(0, "npc"),
             x1 = unit(1, "npc"), y1 = unit(1, "npc"),
             default.units = "npc", units.per.obs = FALSE,
             name = NULL, gp = gpar(), draw = TRUE, vp = NULL)

Arguments

- **x0** Numeric indicating the starting x-values of the line segments.
- **y0** Numeric indicating the starting y-values of the line segments.
- **x1** Numeric indicating the stopping x-values of the line segments.
- **y1** Numeric indicating the stopping y-values of the line segments.
- **default.units** A string.
- **units.per.obs** A boolean indicating whether distinct units are given for each x/y-value.
- **name** A character identifier.
- **gp** An object of class gpar.
- **draw** A logical value indicating whether graphics output should be produced.
- **vp** A Grid viewport object (or NULL)
Both functions create a segments grob (a graphical object describing segments), but only \texttt{grid.segments} draws the segments (and then only if \texttt{draw} is \texttt{TRUE}).

\textbf{Value}

A segments grob. \texttt{grid.segments} returns the value invisibly.

\textbf{Author(s)}

Paul Murrell

\textbf{See Also}

\texttt{Grid}, \texttt{viewport}
grid.show.layout

Draw a Diagram of a Grid Layout

Description
This function uses Grid graphics to draw a diagram of a Grid layout.

Usage
grid.show.layout(l, newpage=TRUE, bg = "light grey",
cell.border = "blue", cell.fill = "light blue",
cell.label = TRUE, label.col = "blue",
unit.col = "red", vp = NULL)

Arguments
l A Grid layout object.
newpage A logical value indicating whether to move on to a new page before drawing the
diagram.
bg The colour used for the background.
cell.border The colour used to draw the borders of the cells in the layout.
cell.fill The colour used to fill the cells in the layout.
cell.label A logical indicating whether the layout cells should be labelled.
label.col The colour used for layout cell labels.
unit.col The colour used for labelling the widths/heights of columns/rows.
vp A Grid viewport object (or NULL).

Details
A viewport is created within vp to provide a margin for annotation, and the layout is drawn within
that new viewport. The margin is filled with light grey, the new viewport is filled with white and
framed with a black border, and the layout regions are filled with light blue and framed with a blue
border. The diagram is annotated with the widths and heights (including units) of the columns and
rows of the layout using red text. (All colours are defaults and may be customised via function
arguments.)

Value
None.
grid.show.viewport

Author(s)

Paul Murrell

See Also

Grid, viewport, grid.layout

Examples

```r
## Diagram of a simple layout
grid.show.layout(grid.layout(4, 2,
  heights=unit(rep(1, 4),
    c("lines", "lines", "lines", "null")),
  widths=unit(c(1, 1), "inches"))
```

draw a Diagram of a Grid Viewport

Description

This function uses Grid graphics to draw a diagram of a Grid viewport.

Usage

```r
grid.show.viewport(v, parent.layout = NULL, newpage = TRUE,
  border.fill="light grey",
  vp.col="blue", vp.fill="light blue",
  scale.col="red",
  vp = NULL)
```

Arguments

- **v**: A Grid viewport object.
- **parent.layout**: A grid layout object. If this is not NULL and the viewport given in `v` has its location specified relative to the layout, then the diagram shows the layout and which cells `v` occupies within the layout.
- **newpage**: A logical value to indicate whether to move to a new page before drawing the diagram.
- **border.fill**: Colour to fill the border margin.
- **vp.col**: Colour for the border of the viewport region.
- **vp.fill**: Colour to fill the viewport region.
- **scale.col**: Colour to draw the viewport axes.
- **vp**: A Grid viewport object (or NULL).
Details

A viewport is created within \( \text{vp} \) to provide a margin for annotation, and the diagram is drawn within that new viewport. By default, the margin is filled with light grey, the new viewport is filled with white and framed with a black border, and the viewport region is filled with light blue and framed with a blue border. The diagram is annotated with the width and height (including units) of the viewport, the \((x, y)\) location of the viewport, and the \(x\)- and \(y\)-scales of the viewport, using red lines and text.

Value

None.

Author(s)

Paul Murrell

See Also

\texttt{Grid.viewport}

Examples

```r
## Diagram of a sample viewport
grid.show.viewport(viewport(x=0.6, y=0.6,
                           w=unit(1, "inches"), h=unit(1, "inches")))
grid.show.viewport(viewport(layout.pos.row=2, layout.pos.col=2:3),
                   grid.layout(3, 4))
```

---

\texttt{grid.text} \hspace{1cm} \textit{Draw Text}

Description

These functions create and draw text.

Usage

```r
\texttt{grid.text(label, x = unit(0.5, "npc"), y = unit(0.5, "npc"),
                just = "centre", hjust = NULL, vjust = NULL, rot = 0,
                check.overlap = FALSE, default.units = "npc",
                name = NULL, gp = gpar(), draw = TRUE, vp = NULL)}
```

```r
\texttt{textGrob(label, x = unit(0.5, "npc"), y = unit(0.5, "npc"),
                just = "centre", hjust = NULL, vjust = NULL, rot = 0,
                check.overlap = FALSE, default.units = "npc",
                name = NULL, gp = gpar(), vp = NULL)}
```
Arguments

- **label**: A vector of strings or expressions to draw.
- **x**: A numeric vector or unit object specifying x-values.
- **y**: A numeric vector or unit object specifying y-values.
- **just**: The justification of the text relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left alignment and 1 means right alignment.
- **hjust**: A numeric vector specifying horizontal justification. If specified, overrides the just setting.
- **vjust**: A numeric vector specifying vertical justification. If specified, overrides the just setting.
- **rot**: The angle to rotate the text.
- **check.overlap**: A logical value to indicate whether to check for and omit overlapping text.
- **default.units**: A string indicating the default units to use if x or y are only given as numeric vectors.
- **name**: A character identifier.
- **gp**: An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
- **draw**: A logical value indicating whether graphics output should be produced.
- **vp**: A Grid viewport object (or NULL).

Details

Both functions create a text grob (a graphical object describing text), but only `grid.text` draws the text (and then only if draw is TRUE).

If the label argument is an expression, the output is formatted as a mathematical annotation, as for base graphics text.

Value

A text grob. `grid.text` returns the value invisibly.

Author(s)

Paul Murrell

See Also

- `Grid`, `viewport`
grid.xaxis

Examples

```r
grid.newpage()
x <- stats::runif(20)
y <- stats::runif(20)
rot <- stats::runif(20, 0, 360)
grid.text("SOMETHING NICE AND BIG", x=x, y=y, rot=rot,
gp=gpar(fontsize=20, col="grey"))
grid.text("SOMETHING NICE AND BIG", x=x, y=y, rot=rot,
gp=gpar(fontsize=20), check=TRUE)
grid.newpage()
draw.text <- function(just, i, j) {
  grid.text("ABCD", x=x[j], y=y[i], just=just)
grid.text(deparse(substitute(just)), x=x[j], y=y[i] + unit(2, "lines"),
gp=gpar(col="grey", fontsize=8))
}
x <- unit(1:4/5, "npc")
y <- unit(1:4/5, "npc")
grid.grill(h=y, v=x, gp=gpar(col="grey"))
draw.text(c("bottom"), 1, 1)
draw.text(c("left", "bottom"), 2, 1)
draw.text(c("right", "bottom"), 3, 1)
draw.text(c("centre", "bottom"), 4, 1)
draw.text(c("centre"), 1, 2)
draw.text(c("left", "centre"), 2, 2)
draw.text(c("right", "centre"), 3, 2)
draw.text(c("centre", "centre"), 4, 2)
draw.text(c("top"), 1, 3)
draw.text(c("left", "top"), 2, 3)
draw.text(c("right", "top"), 3, 3)
draw.text(c("centre", "top"), 4, 3)
draw.text(c(), 1, 4)
draw.text(c("right"), 3, 4)
draw.text(c("centre"), 4, 4)
```

---

grid.xaxis

**Draw an X-Axis**

Description

These functions create and draw an x-axis.

Usage

```r
grid.xaxis(at = NULL, label = TRUE, main = TRUE,
edits = NULL, name = NULL,
gp = gpar(), draw = TRUE, vp = NULL)
xaxisGrob(at = NULL, label = TRUE, main = TRUE,
edits = NULL, name = NULL,
gp = gpar(), vp = NULL)
```
grid.xaxis

Arguments

- **at**: A numeric vector of x-value locations for the tick marks.
- **label**: A logical value indicating whether to draw the labels on the tick marks, or an expression or string vector which specify the labels to use. If not logical, must be the same length as the at argument.
- **main**: A logical value indicating whether to draw the axis at the bottom (TRUE) or at the top (FALSE) of the viewport.
- **edits**: A gEdit or gEditList containing edit operations to apply (to the children of the axis) when the axis is first created and during redrawing whenever at is NULL.
- **name**: A character identifier.
- **gp**: An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
- **draw**: A logical value indicating whether graphics output should be produced.
- **vp**: A Grid viewport object (or NULL).

Details

Both functions create an xaxis grob (a graphical object describing an axis), but only grid.xaxis draws the axis (and then only if draw is TRUE).

Value

An xaxis grob. grid.xaxis returns the value invisibly.

Children

If the at slot of an xaxis grob is not NULL then the xaxis will have the following children:

- **major**: representing the line at the base of the tick marks.
- **ticks**: representing the tick marks.
- **labels**: representing the tick labels.

If the at slot is NULL then there are no children and ticks are drawn based on the current viewport scale.

Author(s)

Paul Murrell

See Also

Grid, viewport, grid.yaxis
grid.yaxis  

Draw a Y-Axis

Description

These functions create and draw a y-axis.

Usage

grid.yaxis(at = NULL, label = TRUE, main = TRUE, 
edits = NULL, name = NULL, 
gp = gpar(), draw = TRUE, vp = NULL)  
yaxisGrob(at = NULL, label = TRUE, main = TRUE, 
edits = NULL, name = NULL, 
gp = gpar(), vp = NULL)

Arguments

at  
A numeric vector of y-value locations for the tick marks.

label  
A logical value indicating whether to draw the labels on the tick marks, or an expression or string vector which specify the labels to use. If not logical, must be the same length as the at argument.

main  
A logical value indicating whether to draw the axis at the left (TRUE) or at the right (FALSE) of the viewport.

edits  
A gEdit or gEditList containing edit operations to apply (to the children of the axis) when the axis is first created and during redrawing whenever at is NULL.

name  
A character identifier.

gp  
An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.

draw  
A logical value indicating whether graphics output should be produced.

vp  
A Grid viewport obect (or NULL).

Details

Both functions create a yaxis grob (a graphical object describing a yaxis), but only grid.yaxis draws the yaxis (and then only if draw is TRUE).

Value

A yaxis grob. grid.yaxis returns the value invisibly.

Children

If the at slot of an xaxis grob is not NULL then the xaxis will have the following children:

major representing the line at the base of the tick marks.
ticks representing the tick marks.
labels representing the tick labels.

If the at slot is NULL then there are no children and ticks are drawn based on the current viewport scale.
grobWidth                  Create a Unit Describing the Width of a Grob

Description
These functions create a unit object describing the width or height of a grob. They are generic.

Usage
  grobWidth(x)
grobHeight(x)

Arguments
  x       A grob object.

Value
  A unit object.

Author(s)
  Paul Murrell

See Also
  unit and stringWidth

plotViewport              Create a Viewport with a Standard Plot Layout

Description
  This is a convenience function for producing a viewport with the common S-style plot layout – i.e.,
  a central plot region surrounded by margins given in terms of a number of lines of text.

Usage
  plotViewport(margins=c(5.1, 4.1, 4.1, 2.1), ...)

Arguments
  margins       A numeric vector interpreted in the same way as par(mar) in base graphics.
  ...           All other arguments will be passed to a call to the viewport() function.
Value
A grid viewport object.

Author(s)
Paul Murrell

See Also
viewport and dataViewport.

Description
Grid maintains a viewport stack — a list of nested drawing contexts. This function makes the parent of the specified viewport the new default viewport.

Usage
pop.viewport(n=1, recording=TRUE)

Arguments
n An integer giving the number of viewports to pop. Defaults to 1.
recording A logical value to indicate whether the set-viewport operation should be recorded on the Grid display list.

Value
None.

Warning
This function has been deprecated. Please use popViewport instead.

Author(s)
Paul Murrell

See Also
push.viewport.
push.viewport  

*Push a Viewport onto the Grid Viewport Stack*

**Description**

Grid maintains a viewport stack — a list of nested drawing contexts. This function makes the specified viewport the default viewport and makes its parent the previous default viewport (i.e., nests the specified context within the previous default context).

**Usage**

```r
push.viewport(..., recording=TRUE)
```

**Arguments**

- `...`: One or more objects of class "viewport", or NULL.
- `recording`: A logical value to indicate whether the set-viewport operation should be recorded on the Grid display list.

**Value**

None.

**Warning**

This function has been deprecated. Please use `pushViewport` instead.

**Author(s)**

Paul Murrell

**See Also**

- `pop.viewport`

---

**Querying the Viewport Tree**  

*Get the Current Grid Viewport (Tree)*

**Description**

- `current.viewport()` returns the viewport that Grid is going to draw into.
- `current.vpTree` returns the entire Grid viewport tree.
- `current.vpPath` returns the viewport path to the current viewport.
- `current.transform` returns the transformation matrix for the current viewport.
Querying the Viewport Tree

Usage

current.viewport(vp=NULL)
current.vpTree(all=TRUE)
current.vpPath()
current.transform()

Arguments

vp A Grid viewport object. Use of this argument has been deprecated.
all A logical value indicating whether the entire viewport tree should be returned.

Details

If all is FALSE then current.vpTree only returns the subtree below the current viewport.

Value

A Grid viewport object from current.viewport or current.vpTree.
current.transform returns a 4x4 transformation matrix.
The viewport path returned by current.vpPath is NULL if the current viewport is the ROOT viewport.

Author(s)

Paul Murrell

See Also

viewport

Examples

grid.newpage()
pushViewport(viewport(width=0.8, height=0.8, name="A"))
pushViewport(viewport(x=0.1, width=0.3, height=0.6,
     just="left", name="B"))
upViewport(1)
pushViewport(viewport(x=0.5, width=0.4, height=0.8,
     just="left", name="C"))
pushViewport(viewport(width=0.8, height=0.8, name="D"))
current.vpPath()
upViewport(1)
current.vpPath()
current.vpTree()
current.viewport()
current.vpTree(all=FALSE)
popViewport(0)
stringWidth

Create a Unit Describing the Width of a String

Description

These functions create a unit object describing the width or height of a string.

Usage

`stringWidth(string)`
`stringHeight(string)`

Arguments

`string`  
A character vector.

Value

A unit object.

Author(s)

Paul Murrell

See Also

`unit` and `grobWidth`

unit

Function to Create a Unit Object

Description

This function creates a unit object — a vector of unit values. A unit value is typically just a single numeric value with an associated unit.

Usage

`unit(x, units, data=NULL)`

Arguments

`x`  
A numeric vector.

`units`  
A character vector specifying the units for the corresponding numeric values.

`data`  
This argument is used to supply extra information for special `unit` types.
Details

Unit objects allow the user to specify locations and dimensions in a large number of different coordinate systems. All drawing occurs relative to a viewport and the units specifies what coordinate system to use within that viewport.

Possible units (coordinate systems) are:

"npc" Normalised Parent Coordinates (the default). The origin of the viewport is (0, 0) and the viewport has a width and height of 1 unit. For example, (0.5, 0.5) is the centre of the viewport.

"cm" Centimetres.

"inches" Inches. 1 in = 2.54 cm.

"mm" Millimetres. 10 mm = 1 cm.

"points" Points. 72.27 pt = 1 in.

"picas" Picas. 1 pc = 12 pt.

"bigpts" Big Points. 72 bp = 1 in.

"dida" Dida. 1157 dd = 1238 pt.

"cicero" Cicero. 1 cc = 12 dd.

"scaledpts" Scaled Points. 65536 sp = 1 pt.

"lines" Lines of text. Locations and dimensions are in terms of multiples of the default text size of the viewport (as specified by the viewport's fontsize and lineheight).

"char" Multiples of nominal font height of the viewport (as specified by the viewport's fontsize).

"native" Locations and dimensions are relative to the viewport's xscale and yscale.

"snpc" Square Normalised Parent Coordinates. Same as Normalised Parent Coordinates, except gives the same answer for horizontal and vertical locations/dimensions. It uses the lesser of npc-width and npc-height. This is useful for making things which are a proportion of the viewport, but have to be square (or have a fixed aspect ratio).

"strwidth" Multiples of the width of the string specified in the data argument. The font size is determined by the pointsize of the viewport.

"strheight" Multiples of the height of the string specified in the data argument. The font size is determined by the pointsize of the viewport.

"grobwidth" Multiples of the width of the grob specified in the data argument.

"grobheight" Multiples of the height of the grob specified in the data argument.

A special units value of "null" is also allowed, but only makes sense when used in specifying widths of columns or heights of rows in grid layouts (see grid.layout).

The data argument must be a list when the unit.length() is greater than 1. For example, unit(rep(1, 3), c("npc", "strwidth", "inches"), data=list(NULL, "my string", NULL)).

It is possible to subset unit objects in the normal way (e.g., unit(1:5, "npc") [2:4]), but a special function unit.c is provided for combining unit objects.

Certain arithmetic and summary operations are defined for unit objects. In particular, it is possible to add and subtract unit objects (e.g., unit(1, "npc") - unit(1, "inches")), and to specify the minimum or maximum of a list of unit objects (e.g., min(unit(0.5, "npc"), unit(1, "inches"))).
Value

An object of class "unit".

WARNING

A special function unit.length is provided for determining the number of unit values in a unit object.
The length function will work in some cases, but in general will not give the right answer.
There is also a special function unit.c for concatenating several unit objects.
The c function will not give the right answer.
There used to be "mylines", "mychar", "mystrwidth", "mystrheight" units. These will still be accepted, but work exactly the same as "lines", "char", "strwidth", "strheight".

Author(s)

Paul Murrell

See Also

unit.c and unit.length

Examples

unit(1, "npc")
unit(1:3/4, "npc")
unit(1:3/4, "npc") + unit(1, "inches")
min(unit(0.5, "npc"), unit(1, "inches"))
unit.c(unit(0.5, "npc"), unit(2, "inches") + unit(1:3/4, "npc"),
      unit(1, "strwidth", "hi there"))

unit.c Combine Unit Objects

Description

This function produces a new unit object by combining the unit objects specified as arguments.

Usage

unit.c(...)
**unit.length**

**Author(s)**
Paul Murrell

**See Also**
unit.

---

**Description**

The length of a unit object is defined as the number of unit values in the unit object.

**Usage**

unit.length(unit)

**Arguments**

unit A unit object.

**Value**

An object of class unit.

**Author(s)**
Paul Murrell

**See Also**

unit

---

**unit.pmin** *Parallel Unit Minima and Maxima*

**Description**

Returns a unit object whose i’th value is the minimum (or maximum) of the i’th values of the arguments.

**Usage**

unit.pmin(…)
unit.pmax(…)

**Arguments**

… One or more unit objects.
Details

The length of the result is the maximum of the lengths of the arguments; shorter arguments are recycled in the usual manner.

Value

A unit object.

Author(s)

Paul Murrell

Examples

```r
max(unit(1:3, "cm"), unit(0.5, "npc"))
unit.pmax(unit(1:3, "cm"), unit(0.5, "npc"))
```

unit.rep  Replicate Elements of Unit Objects

Description

Replicates the units according to the values given in `times` and `length.out`.

Usage

```r
unit.rep(x, times, length.out)
```

Arguments

- `x` An object of class "unit".
- `times` integer. A vector giving the number of times to repeat each element. Either of `length 1` or `length(x)`.
- `length.out` integer. (Optional.) The desired length of the output vector.

Value

An object of class "unit".

Author(s)

Paul Murrell

See Also

`rep`
validDetails

Examples

unit.rep(unit(1:3, "npc"), 3)
unit.rep(unit(1:3, "npc"), 1:3)
unit.rep(unit(1:3, "npc") + unit(1, "inches"), 3)
unit.rep(max(unit(1:3, "npc") + unit(1, "inches")), 3)
unit.rep(max(unit(1:3, "npc") + unit(1, "strwidth", "a")*4, 3)
unit.rep(unit(1:3, "npc") + unit(1, "strwidth", "a")*4, 3)

validDetails Customising grid grob Validation

Description

This generic hook function is called whenever a grid grob is created or edited via grob, gTree, grid.edit or editGrob. This provides an opportunity for customising the validation of a new class derived from grob (or gTree).

Usage

validDetails(x)

Arguments

x A grid grob.

Details

This function is called by grob, gTree, grid.edit and editGrob. A method should be written for classes derived from grob or gTree to validate the values of slots specific to the new class. (e.g., see grid:::validDetails.axis).

Note that the standard slots for grobs and gTrees are automatically validated (e.g., vp, gp slots for grobs and, in addition, children, and childrenvp slots for gTrees) so only slots specific to a new class need to be addressed.

Value

The function MUST return the validated grob.

Author(s)

Paul Murrell

See Also

grid.edit
vpPath

Concatenate Viewport Names

Description

This function can be used to generate a viewport path for use in downViewport or seekViewport.

A viewport path is a list of nested viewport names.

Usage

vpPath(...)

Arguments

... Character values which are viewport names.

Details

Viewport names must only be unique amongst viewports which share the same parent in the viewport tree.

This function can be used to generate a specification for a viewport that includes the viewport’s parent’s name (and the name of its parent and so on).

For interactive use, it is possible to directly specify a path, but it is strongly recommended that this function is used otherwise in case the path separator is changed in future versions of grid.

Value

A vpPath object.

See Also

viewport, pushViewport, popViewport, downViewport, seekViewport, upViewport

Examples

vpPath("vp1", "vp2")
widthDetails

Description

These generic functions are used to determine the size of grid grobs.

Usage

widthDetails(x)
heightDetails(x)

Arguments

x
A grid grob.

Details

These functions are called in the calculation of "grobwidth" and "grobheight" units. Methods should be written for classes derived from grob or gTree where the size of the grob can be determined (see, for example grid::widthDetails.frame).

Value

A unit object.

Author(s)

Paul Murrell

See Also

absolute.size.

Working with Viewports

Maintaining and Navigating the Grid Viewport Tree

Description

Grid maintains a tree of viewports — nested drawing contexts. These functions provide ways to add or remove viewports and to navigate amongst viewports in the tree.

Usage

pushViewport(..., recording=TRUE)
popViewport(n, recording=TRUE)
downViewport(name, strict=FALSE, recording=TRUE)
seekViewport(name, recording=TRUE)
upViewport(n, recording=TRUE)
Arguments

... One or more objects of class "viewport".

n An integer value indicating how many viewports to pop or navigate up. The special value 0 indicates to pop or navigate viewports right up to the root viewport.

name A character value to identify a viewport in the tree.

strict A boolean indicating whether the vpPath must be matched exactly.

recording A logical value to indicate whether the viewport operation should be recorded on the Grid display list.

Details

Objects created by the viewport() function are only descriptions of a drawing context. A viewport object must be pushed onto the viewport tree before it has any effect on drawing.

The viewport tree always has a single root viewport (created by the system) which corresponds to the entire device (and default graphical parameter settings). Viewports may be added to the tree using pushViewport() and removed from the tree using popViewport().

There is only ever one current viewport, which is the current position within the viewport tree. All drawing and viewport operations are relative to the current viewport. When a viewport is pushed it becomes the current viewport. When a viewport is popped, the parent viewport becomes the current viewport. Use upViewport to navigate to the parent of the current viewport, without removing the current viewport from the viewport tree. Use downViewport to navigate to a viewport further down the viewport tree and seekViewport to navigate to a viewport anywhere else in the tree.

If a viewport is pushed and it has the same name as a viewport at the same level in the tree, then it replaces the existing viewport in the tree.

Value
downViewport returns the number of viewports it went down.

This can be useful for returning to your starting point by doing something like depth <- downViewport() then upViewport(depth).

Author(s)

Paul Murrell

See Also

viewport and vpPath.

Examples

# push the same viewport several times
grid.newpage()
vp <- viewport(width=0.5, height=0.5)
pushViewport(vp)
grid.rect(gp=gpar(col="blue"))
grid.text("Quarter of the device",
    y=unit(1, "npc") - unit(1, "lines"), gp=gpar(col="blue"))
pushViewport(vp)
grid.rect(gp=gpar(col="red"))
grid.text("Quarter of the parent viewport",

Working with Viewports

```r
y = unit(1, "npc") - unit(1, "lines"), gp=gpar(col="red"))
popViewport(2)

# push several viewports then navigate amongst them
grid.newpage()
grid.rect(gp=gpar(col="grey"))
grid.text("Top-level viewport",
    y = unit(1, "npc") - unit(1, "lines"), gp=gpar(col="grey"))
if (interactive()) Sys.sleep(1.0)
pushViewport(viewport(width=0.8, height=0.7, name="A"))
grid.rect(gp=gpar(col="blue"))
grid.text("1. Push Viewport A",
    y = unit(1, "npc") - unit(1, "lines"), gp=gpar(col="blue"))
if (interactive()) Sys.sleep(1.0)
upViewport(1)
grid.text("3. Up from B to A",
    y = unit(1, "npc") - unit(2, "lines"), gp=gpar(col="blue"))
if (interactive()) Sys.sleep(1.0)
pushViewport(viewport(x=0.5, width=0.4, height=0.8, just="left", name="C"))
grid.rect(gp=gpar(col="green"))
grid.text("4. Push Viewport C (in A)",
    y = unit(1, "npc") - unit(1, "lines"), gp=gpar(col="green"))
if (interactive()) Sys.sleep(1.0)
pushViewport(viewport(width=0.8, height=0.6, name="D"))
grid.rect()
grid.text("5. Push Viewport D (in C)",
    y = unit(1, "npc") - unit(1, "lines"))
if (interactive()) Sys.sleep(1.0)
upViewport(0)
grid.text("6. Up from D to top-level",
    y = unit(1, "npc") - unit(2, "lines"), gp=gpar(col="grey"))
if (interactive()) Sys.sleep(1.0)
downViewport("D")
grid.text("7. Down from top-level to D",
    y = unit(1, "npc") - unit(2, "lines"))
if (interactive()) Sys.sleep(1.0)
seekViewport("D")
grid.text("8. Seek from D to B",
    y = unit(1, "npc") - unit(2, "lines"), gp=gpar(col="red"))
pushViewport(viewport(width=0.9, height=0.5, name="A"))
grid.rect()
grid.text("9. Push Viewport A (in B)",
    y = unit(1, "npc") - unit(1, "lines"))
if (interactive()) Sys.sleep(1.0)
seekViewport("A")
grid.text("10. Seek from B to A (in ROOT)",
    y = unit(1, "npc") - unit(3, "lines"), gp=gpar(col="blue"))
if (interactive()) Sys.sleep(1.0)
seekViewport(vpPath("B", "A"))
grid.text("11. Seek from (in ROOT) into A (in B)")

popViewport(0)
```
Chapter 6

The methods package

---

**.BasicFunsList**  
*List of Builtin and Special Functions*

**Description**

A named list providing instructions for turning builtin and special functions into generic functions. Functions in R that are defined as `Primitive(<name>)` are not suitable for formal methods, because they lack the basic reflectance property. You can’t find the argument list for these functions by examining the function object itself.

Future versions of R may fix this by attaching a formal argument list to the corresponding function. While generally the names of arguments are not checked by the internal code implementing the function, the number of arguments frequently is.

In any case, some definition of a formal argument list is needed if users are to define methods for these functions. In particular, if methods are to be merged from multiple packages, the different sets of methods need to agree on the formal arguments.

In the absence of reflectance, this list provides the relevant information via a dummy function associated with each of the known specials for which methods are allowed.

At the same, the list flags those specials for which methods are meaningless (e.g., `for`) or just a very bad idea (e.g., `Primitive`).

A generic function created via `setMethod`, for example, for one of these special functions will have the argument list from `.BasicFunsList`. If no entry exists, the argument list `(x, ...)` is assumed.

---

**as**  
*Force an Object to Belong to a Class*

**Description**

These functions manage the relations that allow coercing an object to a given class.
Usage

as(object, Class, strict=TRUE, ext)

as(object, Class) <- value

setAs(from, to, def, replace, where = topenv(parent.frame()))

Arguments

object any R object.

Class the name of the class to which object should be coerced.

strict logical flag. If TRUE, the returned object must be strictly from the target class (unless that class is a virtual class, in which case the object will be from the closest actual class (often the original object, if that class extends the virtual class directly).

If strict = FALSE, any simple extension of the target class will be returned, without further change. A simple extension is, roughly, one that just adds slots to an existing class.

value The value to use to modify object (see the discussion below). You should supply an object with class Class; some coercion is done, but you’re unwise to rely on it.

from, to The classes between which def performs coercion. (In the case of the coerce function these are objects from the classes, not the names of the classes, but you’re not expected to call coerce directly.)

def function of one argument. It will get an object from class from and had better return an object of class to. (If you want to save setAs a little work, make the name of the argument from, but don’t worry about it, setAs will do the conversion.)

replace if supplied, the function to use as a replacement method.

where the position or environment in which to store the resulting method for coerce.

ext the optional object defining how Class is extended by the class of the object (as returned by possibleExtends). This argument is used internally (to provide essential information for non-public classes), but you are unlikely to want to use it directly.

Summary of Functions

as: Returns the version of this object coerced to be the given Class. If the corresponding is(object, Class) relation is true, it will be used. In particular, if the relation has a coerce method, the method will be invoked on object. However, if the object’s class extends Class in a simple way (e.g., by including the superclass in the definition, then the actual coercion will be done only if strict is TRUE (non-strict coercion, is used in passing objects to methods).

Coerce methods are pre-defined for basic classes (including all the types of vectors, functions and a few others). See showMethods(coerce) for a list of these.

Beyond these two sources of methods, further methods are defined by calls to the setAs function.
**as**

**coerce**: Coerce from to be of the same class as to.

Not a function you should usually call explicitly. The function `setAs` creates methods for coerce for the as function to use.

**setAs**: The function supplied as the third argument is to be called to implement as(x, to) when x has class from. Need we add that the function should return a suitable object with class to.

### How Functions ‘as’ and ‘setAs’ Work

The function as contrives to turn object into an object with class Class. In doing so, it uses information about classes and methods, but in a somewhat special way. Keep in mind that objects from one class can turn into objects from another class either automatically or by an explicit call to the as function. Automatic conversion is special, and comes from the designer of one class of objects asserting that this class extends another class (see `setClass` and `setIs`).

Because inheritance is a powerful assertion, it should be used sparingly (otherwise your computations may produce unexpected, and perhaps incorrect, results). But objects can also be converted explicitly, by calling as, and that conversion is designed to use any inheritance information, as well as explicit methods.

As a first step in conversion, the as function determines whether is(object, Class) is TRUE. This can be the case either because the class definition of object includes Class as a “super class” (directly or indirectly), or because a call to setIs established the relationship.

Either way, the inheritance relation defines a method to coerce object to Class. In the most common case, the method is just to extract from object the slots needed for Class, but it’s also possible to specify a method explicitly in a setIs call.

So, if inheritance applies, the as function calls the appropriate method. If inheritance does not apply, and coerceFlag is FALSE, NULL is returned.

By default, coerceFlag is TRUE. In this case the as function goes on to look for a method for the function coerce for the signature c(from = class(object), to = Class).

Method selection is used in the as function in two special ways. First, inheritance is applied for the argument from but not for the argument to (if you think about it, you’ll probably agree that you wouldn’t want the result to be from some class other than the Class specified). Second, the function tries to use inheritance information to convert the object indirectly, by first converting it to an inherited class. It does this by examining the classes that the from class extends, to see if any of them has an explicit conversion method. Suppose class "by" does: Then the as function implicitly computes as(as(object, "by"), Class).

With this explanation as background, the function setAs does a fairly obvious computation: It constructs and sets a method for the function coerce with signature c(from = class(object), to = Class).

The function coerce exists almost entirely as a repository for such methods, to be selected as described above by the as function. In fact, it would usually be a bad idea to call coerce directly, since then you would get inheritance on the to argument; as mentioned, this is not likely to be what you want.

### The Function ‘as’ Used in Replacements

When as appears on the left of an assignment, the intuitive meaning is “Replace the part of object that was inherited from Class by the value on the right of the assignment.”
This usually has a straightforward interpretation, but you can control explicitly what happens, and sometimes you should to avoid possible corruption of objects.

When `object` inherits from `Class` in the usual way, by including the slots of `Class`, the default `as` method is to set the corresponding slots in `object` to those in value.

The default computation may be reasonable, but usually only if all other slots in `object` are unrelated to the slots being changed. Often, however, this is not the case. The class of `object` may have extended `Class` with a new slot whose value depends on the inherited slots. In this case, you may want to define a method for replacing the inherited information that recomputes all the dependent information. Or, you may just want to prohibit replacing the inherited information directly.

The way to control such replacements is through the `replace` argument to function `setIs`. This argument is a method that function `as` calls when used for replacement. It can do whatever you like, including calling `stop` if you want to prohibit replacements. It should return a modified object with the same class as the `object` argument to `as`.

In R, you can also explicitly supply a replacement method, even in the case that inheritance does not apply, through the `replace` argument to `setAs`. It works essentially the same way, but in this case by constructing a method for `"coerce<-"`. (Replace methods for coercion without inheritance are not in the original description and so may not be compatible with S-Plus, at least not yet.)

When inheritance does apply, coerce and replace methods can be specified through either `setIs` or `setAs`; the effect is essentially the same.

**Basic Coercion Methods**

Methods are pre-defined for coercing any object to one of the basic datatypes. For example, `as(x, "numeric")` uses the existing `as.numeric` function. These built-in methods can be listed by `showMethods("coerce")`.

**References**

The R package `methods` implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the `methods` package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.

**Examples**

```r
## using the definition of class "track" from Classes

setAs("track", "numeric", function(from) from@y)

t1 <- new("track", x=1:20, y=(1:20)^2)

as(t1, "numeric")
```
## The next example shows:
## 1. A virtual class to define setAs for several classes at once.
## 2. as() using inherited information

```r
setClass("ca", representation(a = "character", id = "numeric"))
setClass("cb", representation(b = "character", id = "numeric"))
setClass("id")
setIs("ca", "id")
setIs("cb", "id")
setAs("id", "numeric", function(from) from@id)
CA <- new("ca", a = "A", id = 1)
CB <- new("cb", b = "B", id = 2)
setAs("cb", "ca", function(from, to) new(to, a=from$b, id = from$id))
as(CB, "numeric")
```

---

### Description

Formal classes exist corresponding to the basic R data types, allowing these types to be used in method signatures, as slots in class definitions, and to be extended by new classes.

### Usage

```r
### The following are all basic vector classes.
### They can appear as class names in method signatures, in calls to as(), is(), and new().
"character"
"complex"
"double"
"expression"
"integer"
"list"
"logical"
"numeric"
"single"
"raw"

### the class
"vector"
### is a virtual class, extended by all the above

### The following are additional basic classes
"NULL"  #  NULL objects
```
Objects from the Classes

Objects can be created by calls of the form `new(Class, ...)`, where `Class` is the quoted class name, and the remaining arguments if any are objects to be interpreted as vectors of this class.

Multiple arguments will be concatenated.

The class "expression" is slightly odd, in that the ... arguments will not be evaluated; therefore, don’t enclose them in a call to `quote()`.

Extends

Class "vector", directly.

Methods

**coerce**  Methods are defined to coerce arbitrary objects to these classes, by calling the corresponding basic function, for example, `as(x, "numeric")` calls `as.numeric(x)`.

---

**callNextMethod**  *Call an Inherited Method*

**Description**

A call to `callNextMethod` can only appear inside a method definition. It then results in a call to the first inherited method after the current method, with the arguments to the current method passed down to the next method. The value of that method call is the value of `callNextMethod`.

**Usage**

`callNextMethod(...)`

**Arguments**

...  Optionally, the arguments to the function in its next call (but note that the dispatch is as in the detailed description below; the arguments have no effect on selecting the next method.)

If no arguments are included in the call to `callNextMethod`, the effect is to call the method with the current arguments. See the detailed description for what this really means.

Calling with no arguments is often the natural way to use `callNextMethod`; see the examples.
callNextMethod

Details

The “next” method (i.e., the first inherited method) is defined to be that method which would have been called if the current method did not exist. This is more-or-less literally what happens: The current method is deleted from a copy of the methods for the current generic, and `selectMethod` is called to find the next method (the result is cached in a special object, so the search only typically happens once per session per combination of argument classes).

It is also legal, and often useful, for the method called by `callNextMethod` to itself have a call to `callNextMethod`. This generally works as you would expect, but for completeness be aware that it is possible to have ambiguous inheritance in the S structure, in the sense that the same two classes can appear as superclasses in the opposite order in two other class definitions. In this case the effect of a nested instance of `callNextMethod` is not well defined. Such inconsistent class hierarchies are both rare and nearly always the result of bad design, but they are possible, and currently undetected.

The statement that the method is called with the current arguments is more precisely as follows. Arguments that were missing in the current call are still missing (remember that "missing" is a valid class in a method signature). For a formal argument, say x, that appears in the original call, there is a corresponding argument in the next method call equivalent to “x = x”. In effect, this means that the next method sees the same actual arguments, but arguments are evaluated only once.

Value

The value returned by the selected method.

References

The R package `methods` implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the `methods` package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.

See Also

`Methods` for the general behavior of method dispatch

Examples

```r
## some class definitions with simple inheritance
setClass("B0", representation(b0 = "numeric"))
setClass("B1", representation(b1 = "character"), contains = "B0")
setClass("B2", representation(b2 = "logical"), contains = "B1")
## and a rather silly function to illustrate callNextMethod
f <- function(x) class(x)
```
```r
setMethod("f", "B0", function(x) c(x@b0^2, callNextMethod()))
setMethod("f", "B1", function(x) c(paste(x@b1,":"), callNextMethod()))
setMethod("f", "B2", function(x) c(x@b2, callNextMethod()))

b1 <- new("B1", b0 = 2, b1 = "Testing")
b2 <- new("B2", b2 = FALSE, b1 = "More testing", b0 = 10)
f(b2)
f(b1)
```

---

**cbind2**

Combine two Objects by Columns or Rows

**Description**

Combine two “matrix-like” R objects by columns (cbind2) or rows (rbind2). These are (S4) generic functions with default methods.

**Usage**

```r
cbind2(x, y)
rbind2(x, y)
```

**Arguments**

- `x` any R object, typically “matrix-like”.
- `y` any R object, typically similar to `x`, or missing completely.

**Details**

The main use of cbind2 (rbind2) is to be called by cbind (rbind) if these are activated. This allows cbind (rbind) to “work” for formally classed (aka ‘S4’) objects by providing S4 methods for these objects. Currently, a call

`methods:::bind_activation(TRUE)`

is needed to install a "cbind2-calling" version of cbind (into the base namespace) and the same for rbind. 

`methods:::bind_activation(FALSE)` reverts to the previous internal version of cbind which does not build on cbind2, see the examples.

**Value**

A matrix (or matrix like object) combining the columns (or rows) of `x` and `y`.

**Methods**

- `x = "ANY", y = "ANY"` the default method using R’s internal code.
- `x = "ANY", y = "missing"` the default method for one argument using R’s internal code.
Classes

Description

Class definitions are objects that contain the formal definition of a class of R objects.

Details

When a class is defined, an object is stored that contains the information about that class, including:

- **slots**: Each slot is a component object. Like elements of a list these may be extracted (by name) and set. However, they differ from list components in important ways. All the objects from a particular class have the same set of slot names; specifically, the slot names that are contained in the class definition. Each slot in each object always has the same class; again, this is defined by the overall class definition. Classes don’t need to have any slots, and many useful classes do not. These objects usually extend other, simple objects, such as numeric or character vectors. Finally, classes can have no data at all—these are known as virtual classes and are in fact very important programming tools. They are used to group together ordinary classes that want to share some programming behavior, without necessarily restricting how the behavior is implemented.

- **extends**: The names of the classes that this class extends. A class Fancy, say, extends a class Simple if an object from the Fancy class has all the capabilities of the Simple class (and probably some more as well). In particular, and very usefully, any method defined to work for a Simple object can be applied to a Fancy object as well.

Examples

```
cbind2(1:3, 4)
m <- matrix(3:8, 2,3, dimnames=list(c("a","b"), LETTERS[1:3]))
cbind2(1:2, m) # keeps dimnames from m
```

### Note: Use the following activation if you want `cbind()` to work
### ---- on S4 objects -- be careful otherwise!

```
methods:::bind_activation(on = TRUE)
trace("cbind2")
cbind(a=1:3)# no call to cbind2()
cbind(a=1:3, four=4, 7:9)# calling cbind2() twice
untrace("cbind2")
```

```
## The following fails currently,
## since cbind() works recursively from the tail:
try( cbind(m, a=1, b=3) )
```

```
## turn off the 'special cbind()':
methods:::bind_activation(FALSE)
```
In other programming languages, this relationship is sometimes expressed by saying that Simple is a superclass of Fancy, or that Fancy is a subclass of Simple.

The actual class definition object contains the names of all the classes this class extends. But those classes can themselves extend other classes also, so the complete extension can only be known by obtaining all those class definitions.

Class extension is usually defined when the class itself is defined, by including the names of superclasses as unnamed elements in the representation argument to `setClass`.

An object from a given class will then have all the slots defined for its own class and all the slots defined for its superclasses as well.

Note that extends relations can be defined in other ways as well, by using the `setIs` function.

**prototype** Each class definition contains a prototype object from the class. This must have all the slots, if any, defined by the class definition.

The prototype most commonly just consists of the prototypes of all its slots. But that need not be the case: the definition of the class can specify any valid object for any of the slots.

There are a number of “basic” classes, corresponding to the ordinary kinds of data occurring in R. For example, "numeric" is a class corresponding to numeric vectors. These classes are predefined and can then be used as slots or as superclasses for any other class definitions.

The prototypes for the vector classes are vectors of length 0 of the corresponding type.

There are also a few basic virtual classes, the most important being "vector", grouping together all the vector classes; and "language", grouping together all the types of objects making up the R language.

**Author(s)**

John Chambers

**References**

The web page [http://www.omegahat.org/RSMethods/index.html](http://www.omegahat.org/RSMethods/index.html) is the primary documentation.


**See Also**

`Methods`, `setClass`, `is`, `as`, `new`, `slot`
Details

Class definitions are stored as metadata in various packages. Additional metadata supplies information on inheritance (the result of calls to `setIs`). Inheritance information implied by the class definition itself (because the class contains one or more other classes) is also constructed automatically.

When a class is to be used in an R session, this information is assembled to complete the class definition. The completion is a second object of class "classRepresentation", cached for the session or until something happens to change the information. A call to `getClass` returns the completed definition of a class; a call to `getClassDef` returns the stored definition (uncompleted).

In particular, completion fills in the upward- and downward-pointing inheritance information for the class, in slots `contains` and `subclasses` respectively. It’s in principle important to note that this information can depend on which packages are installed, since these may define additional subclasses or superclasses.

Slots

- **slots**: A named list of the slots in this class; the elements of the list are the classes to which the slots must belong (or extend), and the names of the list gives the corresponding slot names.
- **contains**: A named list of the classes this class “contains”; the elements of the list are objects of `SClassExtension-class`. The list may be only the direct extensions or all the currently known extensions (see the details).
- **virtual**: Logical flag, set to `TRUE` if this is a virtual class.
- **prototype**: The object that represents the standard prototype for this class; i.e., the data and slots returned by a call to `new` for this class with no special arguments. Don’t mess with the prototype object directly.
- **validity**: Optionally, a function to be used to test the validity of objects from this class. See `validObject`.
- **access**: Access control information. Not currently used.
- **className**: The character string name of the class.
- **package**: The character string name of the package to which the class belongs. Nearly always the package on which the metadata for the class is stored, but in operations such as constructing inheritance information, the internal package name rules.
- **subclasses**: A named list of the classes known to extend this class'; the elements of the list are objects of `SClassExtension-class`. The list is currently only filled in when completing the class definition (see the details).
- **versionKey**: Object of class "externalptr"; eventually will perhaps hold some versioning information, but not currently used.
- **sealed**: Object of class "logical"; is this class sealed? If so, no modifications are allowed.

See Also

See function `setClass` to supply the information in the class definition. See `Classes` for a more basic discussion of class information.
Using and Creating On-line Documentation for Classes and Methods

**Description**

Special documentation can be supplied to describe the classes and methods that are created by the software in the methods package. Techniques to access this documentation and to create it in R help files are described here.

**Getting documentation on classes and methods**

You can ask for on-line help for class definitions, for specific methods for a generic function, and for general discussion of methods for a generic function. These requests use the `?` operator (see `help` for a general description of the operator). Of course, you are at the mercy of the implementer as to whether there is any documentation on the corresponding topics.

Documentation on a class uses the argument `class` on the left of the `?`, and the name of the class on the right; for example,

```
class ? genericFunction
```

to ask for documentation on the class "genericFunction".

When you want documentation for the methods defined for a particular function, you can ask either for a general discussion of the methods or for documentation of a particular method (that is, the method that would be selected for a particular set of actual arguments).

Overall methods documentation is requested by calling the `?` operator with `methods` as the left-side argument and the name of the function as the right-side argument. For example,

```
methods ? initialize
```

Asking for documentation on a particular method is done by giving a function call expression as the right-hand argument to the "?" operator. There are two forms, depending on whether you prefer to give the class names for the arguments or expressions that you intend to use in the actual call.

If you planned to evaluate a function call, say `myFun(x, sqrt(wt))` and wanted to find out something about the method that would be used for this call, put the call on the right of the "?" operator:

```
?myFun(x, sqrt(wt))
```

A method will be selected, as it would be for the call itself, and documentation for that method will be requested. If `myFun` is not a generic function, ordinary documentation for the function will be requested.

If you know the actual classes for which you would like method documentation, you can supply these explicitly in place of the argument expressions. In the example above, if you want method documentation for the first argument having class "maybeNumber" and the second "logical", call the "?" operator, this time with a left-side argument `method`, and with a function call on the right using the class names as arguments:

```
method ? myFun("maybeNumber", "logical")
```

Once again, a method will be selected, this time corresponding to the specified classes, and method documentation will be requested. This version only works with generic functions.

The two forms each have advantages. The version with actual arguments doesn’t require you to figure out (or guess at) the classes of the arguments. On the other hand, evaluating the arguments
may take some time, depending on the example. The version with class names does require you to pick classes, but it’s otherwise unambiguous. It has a subtler advantage, in that the classes supplied may be virtual classes, in which case no actual argument will have specifically this class. The class "maybeNumber", for example, might be a class union (see the example for setClassUnion).

In either form, methods will be selected as they would be in actual computation, including use of inheritance and group generic functions. See selectMethod for the details, since it is the function used to find the appropriate method.

Writing Documentation for Methods

The on-line documentation for methods and classes uses some extensions to the R documentation format to implement the requests for class and method documentation described above. See the document Writing R Extensions for the available markup commands (you should have consulted this document already if you are at the stage of documenting your software).

In addition to the specific markup commands to be described, you can create an initial, overall file with a skeleton of documentation for the methods defined for a particular generic function:

promptMethods("myFun")

will create a file, ‘myFun-methods.Rd’ with a skeleton of documentation for the methods defined for function myFun. The output from promptMethods is suitable if you want to describe all or most of the methods for the function in one file, separate from the documentation of the generic function itself. Once the file has been filled in and moved to the ‘man’ subdirectory of your source package, requests for methods documentation will use that file, both for specific methods documentation as described above, and for overall documentation requested by

methods ? myFun

You are not required to use promptMethods, and if you do, you may not want to use the entire file created:

- If you want to document the methods in the file containing the documentation for the generic function itself, you can cut-and-paste to move the \alias lines and the Methods section from the file created by promptMethods to the existing file.
- On the other hand, if these are auxiliary methods, and you only want to document the added or modified software, you should strip out all but the relevant \alias lines for the methods of interest, and remove all but the corresponding \item entries in the Methods section. Note that in this case you will usually remove the first \alias line as well, since that is the marker for general methods documentation on this function (in the example, \alias{myfun-methods}).

If you simply want to direct documentation for one or more methods to a particular R documentation file, insert the appropriate alias.

---

environment-class  Class "environment"

Description

A formal class for R environments.

Objects from the Class

Objects can be created by calls of the form new("environment", ...). The arguments in ..., if any, should be named and will be assigned to the newly created environment.
Methods

**coerce** signature(from = "ANY", to = "environment"): calls `as.environment`.

**initialize** signature(object = "environment"): Implements the assignments in the new environment. Note that the object argument is ignored; a new environment is always created, since environments are not protected by copying.

See Also

`new.env`

---

**fixPre1.8** *Fix Objects Saved from R Versions Previous to 1.8*

Description

Beginning with R version 1.8.0, the class of an object contains the identification of the package in which the class is defined. The function `fixPre1.8` fixes and re-assigns objects missing that information (typically because they were loaded from a file saved with a previous version of R.)

Usage

`fixPre1.8(names, where)`

Arguments

- **names**: Character vector of the names of all the objects to be fixed and re-assigned.
- **where**: The environment from which to look for the objects, and for class definitions. Defaults to the top environment of the call to `fixPre1.8`, the global environment if the function is used interactively.

Details

The named object will be saved where it was found. Its class attribute will be changed to the full form required by R 1.8; otherwise, the contents of the object should be unchanged.

Objects will be fixed and re-assigned only if all the following conditions hold:

1. The named object exists.
2. It is from a defined class (not a basic datatype which has no actual class attribute).
3. The object appears to be from an earlier version of R.
4. The class is currently defined.
5. The object is consistent with the current class definition.

If any condition except the second fails, a warning message is generated.

Note that `fixPre1.8` currently fixes only the change in class attributes. In particular, it will not fix binary versions of packages installed with earlier versions of R if these use incompatible features. Such packages must be re-installed from source, which is the wise approach always when major version changes occur in R.

Value

The names of all the objects that were in fact re-assigned.
Generics are extended function objects, containing additional information used in creating and dispatching methods for this function. They also identify the package associated with the function and its methods.

Objects from the Class

Generics are created and assigned by `setGeneric` or `setGroupGeneric` and, indirectly, by `setMethod`.

As you might expect, `setGeneric` and `setGroupGeneric` create objects of class "genericFunction" and "groupGenericFunction" respectively.

Slots

- **.Data**: Object of class "function", the function definition of the generic, usually created automatically as a call to `standardGeneric`.
- **generic**: Object of class "character", the name of the generic function.
- **package**: Object of class "character", the name of the package to which the function definition belongs (and not necessarily where the generic function is stored). If the package is not specified explicitly in the call to `setGeneric`, it is usually the package on which the corresponding non-generic function exists.
- **group**: Object of class "list", the group or groups to which this generic function belongs. Empty by default.
- **valueClass**: Object of class "character"; if not an empty character vector, identifies one or more classes. It is asserted that all methods for this function return objects from these class (or from classes that extend them).
- **signature**: Object of class "character", the vector of formal argument names that can appear in the signature of methods for this generic function. By default, it is all the formal arguments, except for .... Order matters for efficiency: the most commonly used arguments in specifying methods should come first.
- **default**: Object of class "OptionalMethods", the default method for this function. Generated automatically and used to initialize method dispatch.
- **skeleton**: Object of class "call", a slot used internally in method dispatch. Don’t expect to use it directly.

Extends

Class "function", from data part.
Class "OptionalMethods", by class "function".
Class "PossibleMethod", by class "function".
Methods

Generic function objects are used in the creation and dispatch of formal methods; information from the object is used to create methods list objects and to merge or update the existing methods for this generic.

Description

The functions documented here manage collections of methods associated with a generic function, as well as providing information about the generic functions themselves.

Usage

```r
isGeneric(f, where, fdef, getName = FALSE)
isGroup(f, where, fdef)
removeGeneric(f, where)
dumpMethod(f, signature, file, where, def)
findFunction(f, generic = TRUE, where = topenv(parent.frame()))
dumpMethods(f, file, signature, methods, where)
signature(...)
removeMethods(f, where = topenv(parent.frame()), all = TRUE)
setReplaceMethod(f, ..., where = topenv(parent.frame()))

generics(where, searchForm = FALSE)
allGenerics(where, searchForm = FALSE)
callGeneric(...)```

Arguments

- **f**: The character string naming the function.
- **where**: The environment, namespace, or search-list position from which to search for objects. By default, start at the top-level environment of the calling function, typically the global environment (i.e., use the search list), or the namespace of a package from which the call came. It is important to supply this argument when calling any of these functions indirectly. With package namespaces, the default is likely to be wrong in such calls.
- **signature**: The class signature of the relevant method. A signature is a named or unnamed vector of character strings. If named, the names must be formal argument names for the generic function. If `signature` is unnamed, the default is to use the first `length(signature)` formal arguments of the function.
- **file**: The file on which to dump method definitions.
- **def**: The function object defining the method; if omitted, the current method definition corresponding to the signature.
- **...**: Named or unnamed arguments to form a signature.
In testing or finding functions, should generic functions be included. Supply as FALSE to get only non-generic functions.

Optional, the generic function definition. Usually omitted in calls to isGeneric.

If TRUE, isGeneric returns the name of the generic. By default, it returns TRUE.

The methods object containing the methods to be dumped. By default, the methods defined for this generic (optionally on the specified where location).

logical indicating if all (default) or only the first method found should be removed.

In getGenerics, if TRUE, the package slot of the returned result is in the form used by search(), otherwise as the simple package name (e.g, "package:base" vs "base").

**Summary of Functions**

**isGeneric:** Is there a function named f, and if so, is it a generic?

The getName argument allows a function to find the name from a function definition. If it is TRUE then the name of the generic is returned, or FALSE if this is not a generic function definition.

The behavior of isGeneric and getGeneric for primitive functions is slightly different. These functions don't exist as formal function objects (for efficiency and historical reasons), regardless of whether methods have been defined for them. A call to isGeneric tells you whether methods have been defined for this primitive function, anywhere in the current search list, or in the specified position where. In contrast, a call to getGeneric will return what the generic for that function would be, even if no methods have been currently defined for it.

**removeGeneric, removeMethods:** Remove all the methods for the generic function of this name. In addition, removeGeneric removes the function itself; removeMethods restores the non-generic function which was the default method. If there was no default method, removeMethods leaves a generic function with no methods.

**standardGeneric:** Dispatches a method from the current function call for the generic function f. It is an error to call standardGeneric anywhere except in the body of the corresponding generic function.

Note that standardGeneric is a primitive function in the base package for efficiency reasons, but rather documented here where it belongs naturally.

**dumpMethod:** Dump the method for this generic function and signature.

**findFunction:** return a list of either the positions on the search list, or the current top-level environment, on which a function object for name exists. The returned value is always a list, use the first element to access the first visible version of the function. See the example.

**NOTE:** Use this rather than find with mode="function", which is not as meaningful, and has a few subtle bugs from its use of regular expressions. Also, findFunction works correctly in the code for a package when attaching the package via a call to library.

**dumpMethods:** Dump all the methods for this generic.

**signature:** Returns a named list of classes to be matched to arguments of a generic function.

**getGenerics:** Returns the names of the generic functions that have methods defined on where; this argument can be an environment or an index into the search list. By default, the whole search list is used.

The methods definitions are stored with package qualifiers; for example, methods for function "initialize" might refer to two different functions of that name, on different packages.
The package names corresponding to the method list object are contained in the slot package of the returned object. The form of the returned name can be plain (e.g., "base"), or in the form used in the search list ("package:base") according to the value of searchForm.

callGeneric: In the body of a method, this function will make a call to the current generic function. If no arguments are passed to callGeneric, the arguments to the current call are passed down; otherwise, the arguments are interpreted as in a call to the generic function.

Details

setGeneric: If there is already a non-generic function of this name, it will be used to define the generic unless def is supplied, and the current function will become the default method for the generic.

If def is supplied, this defines the generic function, and no default method will exist (often a good feature, if the function should only be available for a meaningful subset of all objects). Arguments group and valueClass are retained for consistency with S-Plus, but are currently not used.

isGeneric: If the fdef argument is supplied, take this as the definition of the generic, and test whether it is really a generic, with f as the name of the generic. (This argument is not available in S-Plus.)

removeGeneric: If where supplied, just remove the version on this element of the search list; otherwise, removes the first version encountered.

standardGeneric: Generic functions should usually have a call to standardGeneric as their entire body. They can, however, do any other computations as well. The usual setGeneric (directly or through calling setMethod) creates a function with a call to standardGeneric.

dumpMethod: The resulting source file will recreate the method.

findFunction: If generic is FALSE, ignore generic functions.

dumpMethods: If signature is supplied only the methods matching this initial signature are dumped. (This feature is not found in S-Plus: don’t use it if you want compatibility.)

signature: The advantage of using signature is to provide a check on which arguments you meant, as well as clearer documentation in your method specification. In addition, signature checks that each of the elements is a single character string.

removeMethods: Returns TRUE if f was a generic function, FALSE (silently) otherwise.

If there is a default method, the function will be re-assigned as a simple function with this definition. Otherwise, the generic function remains but with no methods (so any call to it will generate an error). In either case, a following call to setMethod will consistently re-establish the same generic function as before.

References

The R package methods implements, with a few exceptions, the programming interface for classes and methods in the book Programming with Data (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the methods package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.
See Also

getMethod (also for selectMethod), setGeneric, setClass, showMethods

Examples

```r
## get the function "myFun" -- throw an error if 0 or > 1 versions visible:
findFuncStrict <- function(fName) {
  allF <- findFunction(fName)
  if(length(allF) == 0)
    stop("No versions of ", fName," visible")
  else if(length(allF) > 1)
    stop(fName," is ambiguous: ", length(allF), ", " versions")
  else
    get(fName, allF[[1]])
}

try(findFuncStrict("myFun"))# Error: no version
lm <- function(x) x+1
try(findFuncStrict("lm"))# Error: 2 versions
findFuncStrict("findFuncStrict")# just 1 version
rm(lm)
```

---

goingClass

### Description

Get the definition of a class.

### Usage

```r
gtclass(Class, .Force = FALSE, where)
gtclassDef(Class, where, package)
```

### Arguments

- **Class**
  - the character-string name of the class.

- **.Force**
  - if TRUE, return NULL if the class is undefined; otherwise, an undefined class results in an error.

- **where**
  - environment from which to begin the search for the definition; by default, start at the top-level (global) environment and proceed through the search list.

- **package**
  - the name of the package asserted to hold the definition. Supplied instead of where, with the distinction that the package need not be currently attached.

### Details

A call to getClass returns the complete definition of the class supplied as a string, including all slots, etc. in classes that this class extends. A call to getClassDef returns the definition of the class from the environment where, unadorned. It's usually getClass you want.

If you really want to know whether a class is formally defined, call isClass.
Value

The object defining the class. This is an object of class "classRepEnvironment". However, *do not* deal with the contents of the object directly unless you are very sure you know what you’re doing. Even then, it is nearly always better practice to use functions such as `setClass` and `setIs`. Messing up a class object will cause great confusion.

References

The R package *methods* implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the methods package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.

See Also

`Classes, setClass, setIs`.

Examples

```r
getClass("numeric") # a built in class

cld <- getClass("thisIsAnUndefinedClass", .Force = TRUE)
cld # a NULL prototype
## If you are really curious:
str(cld)
## Whereas these generate errors:
try(getClass("thisIsAnUndefinedClass"))
try(getClassDef("thisIsAnUndefinedClass"))
```

getMethod

*Get or Test for the Definition of a Method*

Description

The functions `getMethod` and `selectMethod` get the definition of a particular method; the functions `existsMethod` and `hasMethod` test for the existence of a method. In both cases the first function only gets direct definitions and the second uses inheritance. The function `findMethod` returns the package(s) in the search list (or in the packages specified by the `where` argument) that contain a method for this function and signature.

The other functions are support functions: see the details below.
getMethod

Usage

getMethod(f, signature=character(), where, optional=FALSE, mlist)

findMethod(f, signature, where)

getMethods(f, where)

existsMethod(f, signature = character(), where)

hasMethod(f, signature=character(), where)

selectMethod(f, signature, optional = FALSE, useInherited = TRUE,
        mlist = (if (is.null(fdef)) NULL else
            getMethodsForDispatch(f, fdef)),
        fdef = getGeneric(f, !optional))

MethodsListSelect(f, env, mlist, fEnv, finalDefault, evalArgs,
        useInherited, fdef, reset Allowed)

Arguments

f The character-string name of the generic function.
signature the signature of classes to match to the arguments of f. See the details below.
For selectMethod, the signature can optionally be an environment with classes assigned to the names of the corresponding arguments. Note: the names correspond to the names of the classes, not to the objects supplied in a call to the generic function. (You are not likely to find this approach convenient, but it is used internally and is marginally more efficient.)
where The position or environment in which to look for the method(s): by default, anywhere in the current search list.
optional If the selection does not produce a unique result, an error is generated, unless this argument is TRUE. In that case, the value returned is either a MethodsList object, if more than one method matches this signature, or NULL if no method matches.
mlist Optionally, the list of methods in which to search. By default, the function finds the methods for the corresponding generic function. To restrict the search to a particular package or environment, e.g., supply this argument as getMethodsMetaData(f, where). For selectMethod, see the discussion of argument fdef.
fdef In selectMethod, the MethodsList object and/or the generic function object can be explicitly supplied. (Unlikely to be used, except in the recursive call that finds matches to more than one argument.)
env The environment in which argument evaluations are done in MethodsListSelect. Currently must be supplied, but should usually be sys.frame(sys.parent()) when calling the function explicitly for debugging purposes.
fEnv, finalDefault, evalArgs, useInherited, reset Allowed

Internal-use arguments for the function’s environment, the method to use as the overall default, whether to evaluate arguments, which arguments should use inheritance, and whether the cached methods are allowed to be reset.
**Details**

The `signature` argument specifies classes, in an extended sense, corresponding to formal arguments of the generic function. As supplied, the argument may be a vector of strings identifying classes, and may be named or not. Names, if supplied, match the names of those formal arguments included in the signature of the generic. That signature is normally all the arguments except …. However, generic functions can be specified with only a subset of the arguments permitted, or with the signature taking the arguments in a different order.

It’s a good idea to name the arguments in the signature to avoid confusion, if you’re dealing with a generic that does something special with its signature. In any case, the elements of the signature are matched to the formal signature by the same rules used in matching arguments in function calls (see `match.call`).

The strings in the signature may be class names, "missing" or "ANY". See Methods for the meaning of these in method selection. Arguments not supplied in the signature implicitly correspond to class "ANY"; in particular, giving an empty signature means to look for the default method.

A call to `getMethod` returns the method for a particular function and signature. As with other `get` functions, argument `where` controls where the function looks (by default anywhere in the search list) and argument `optional` controls whether the function returns `NULL` or generates an error if the method is not found. The search for the method makes no use of inheritance.

The function `selectMethod` also looks for a method given the function and signature, but makes full use of the method dispatch mechanism; i.e., inherited methods and group generics are taken into account just as they would be in dispatching a method for the corresponding signature, with the one exception that conditional inheritance is not used. Like `getMethod`, `selectMethod` returns `NULL` or generates an error if the method is not found, depending on the argument `optional`.

The functions `existsMethod` and `hasMethod` return TRUE or FALSE according to whether a method is found, the first corresponding to `getMethod` (no inheritance) and the second to `selectMethod`.

The function `getMethods` returns all the methods for a particular generic (in the form of a generic function with the methods information in its environment). The function is called from the evaluator to merge method information, and is not intended to be called directly. Note that it gets all the visible methods for the specified functions. If you want only the methods defined explicitly in a particular environment, use the function `getMethodsMetaData` instead.

The function `MethodsListSelect` performs a full search (including all inheritance and group generic information: see the `Methods` documentation page for details on how this works). The call returns a possibly revised methods list object, incorporating any method found as part of the `allMethods` slot.

Normally you won’t call `MethodsListSelect` directly, but it is possible to use it for debugging purposes (only for distinctly advanced users!).

Note that the statement that `MethodsListSelect` corresponds to the selection done by the evaluator is a fact, not an assertion, in the sense that the evaluator code constructs and executes a call to `MethodsListSelect` when it does not already have a cached method for this generic function and signature. (The value returned is stored by the evaluator so that the search is not required next time.)

**Value**

The call to `selectMethod` or `getMethod` returns a `MethodDefinition-class` object, the selected method, if a unique selection exists. (This class extends `function-class`, so you can use the result directly as a function if that is what you want.) Otherwise an error is thrown if
optional is FALSE. If optional is TRUE, the value returned is NULL if no method matched, or a MethodsList object if multiple methods matched.

The call to getMethods returns the MethodsList object containing all the methods requested. If there are none, NULL is returned: getMethods does not generate an error in this case.

References

The R package methods implements, with a few exceptions, the programming interface for classes and methods in the book Programming with Data (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the methods package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.

See Also

GenericFunctions

Examples

setGeneric("testFun", function(x) standardGeneric("testFun"))
setMethod("testFun", "numeric", function(x)x+1)
hasMethod("testFun", "numeric")
## Not run: [1] TRUE
hasMethod("testFun", "integer") # inherited
## Not run: [1] TRUE
existsMethod("testFun", "integer")
## Not run: [1] TRUE
hasMethod("testFun") # default method
## Not run: [1] FALSE
hasMethod("testFun", "ANY")
## Not run: [1] FALSE

getPackageName

The Name associated with a Given Package

Description

The functions below produce the package associated with a particular environment or position on the search list, or of the package containing a particular function. They are primarily used to support computations that need to differentiate objects on multiple packages.

Usage

packageName(where)
packageSlot(object)
packageSlot(object) <- value
Arguments

where the environment or position on the search list associated with the desired package.

object object providing a character string name, plus the package in which this object is to be found.

value the name of the package.

Details

Package names are normally installed during loading of the package, by the INSTALL script or by the library function. (Currently, the name is stored as the object .packageName but don’t trust this for the future.)

Value

packageName return the character-string name of the package (without the extraneous "package:" found in the search list).

packageSlot returns or sets the package name slot (currently an attribute, not a formal slot, but this will likely change).

See Also

search

Examples

## both the following usually return "base"
getPackageName(length(search()))

hasArg

Look for an Argument in the Call

Description

Returns TRUE if name corresponds to an argument in the call, either a formal argument to the function, or a component of ..., and FALSE otherwise.

Usage

hasArg(name)

Arguments

name The unquoted name of a potential argument.

Details

The expression hasArg(x), for example, is similar to !missing(x), with two exceptions. First, hasArg will look for an argument named x in the call if x is not a formal argument to the calling function, but ... is. Second, hasArg never generates an error if given a name as an argument, whereas missing(x) generates an error if x is not a formal argument.
Value

Always TRUE or FALSE as described above.

See Also

missing

Examples

```r
ftest <- function(x1, ...) c(hasArg(x1), hasArg(y2))

ftest(1)  ## c(TRUE, FALSE)
ftest(1, 2)  ## c(TRUE, FALSE)
ftest(y2 = 2)  ## c(FALSE, TRUE) (no partial matching)
ftest(y2 = 2, x = 1)  ## c(TRUE, TRUE) partial match x1
```

Methods

Methods to Initialize New Objects from a Class

The arguments to function `new` to create an object from a particular class can be interpreted specially for that class, by the definition of a method for function `initialize` for the class. This documentation describes some existing methods, and also outlines how to write new ones.

### Methods

- **Object = "ANY"**
The default method for `initialize` takes either named or unnamed arguments. Argument names must be the names of slots in this class definition, and the corresponding arguments must be valid objects for the slot (that is, have the same class as specified for the slot, or some superclass of that class). If the object comes from a superclass, it is not coerced strictly, so normally it will retain its current class (specifically, `as(object, Class, strict = FALSE)`).

  Unnamed arguments must be objects of this class, of one of its superclasses, or one of its subclasses (from the class, from a class this class extends, or from a class that extends this class). If the object is from a superclass, this normally defines some of the slots in the object. If the object is from a subclass, the new object is that argument, coerced to the current class. Unnamed arguments are processed first, in the order they appear. Then named arguments are processed. Therefore, explicit values for slots always override any values inferred from superclass or subclass arguments.

- **Object = "traceable"**

  Objects of a class that extends `traceable` are used to implement debug tracing (see `traceable-class` and `trace`).

  The `initialize` method for these classes takes special arguments `def`, `tracer`, `exit`, `at`, `print`. The first of these is the object to use as the original definition (e.g., a function). The others correspond to the arguments to `trace`.

- **Object = "environment"**

  The `initialize` method for environments takes a named list of objects to be used to initialize the environment.

- **Object = "signature"**

  This is a method for internal use only. It takes an optional `functionDef` argument to provide a generic function with a `signature` slot to define the argument names. See `Methods` for details.
Writing Initialization Methods

Initialization methods provide a general mechanism corresponding to generator functions in other languages.

The arguments to \texttt{initialize} are \texttt{.Object} and \ldots. Nearly always, \texttt{initialize} is called from \texttt{new}, not directly. The \texttt{.Object} argument is then the prototype object from the class.

Two techniques are often appropriate for \texttt{initialize} methods: special argument names and \texttt{callNextMethod}.

You may want argument names that are more natural to your users than the (default) slot names. These will be the formal arguments to your method definition, in addition to \texttt{.Object} (always) and \ldots (optionally). For example, the method for class \texttt{"traceable"} documented above would be created by a call to \texttt{setMethod} of the form:

\begin{verbatim}
setMethod("initialize", "traceable",
    function(.Object, def, tracer, exit, at, print) ... )
\end{verbatim}

In this example, no other arguments are meaningful, and the resulting method will throw an error if other names are supplied.

When your new class extends another class, you may want to call the initialize method for this superclass (either a special method or the default). For example, suppose you want to define a method for your class, with special argument \texttt{x}, but you also want users to be able to set slots specifically. If you want \texttt{x} to override the slot information, the beginning of your method definition might look something like this:

\begin{verbatim}
function(.Object, x, ...) {
    Object <- callNextMethod(.Object, ...)
    if(!missing(x)) { # do something with x
\end{verbatim}

You could also choose to have the inherited method override, by first interpreting \texttt{x}, and then calling the next method.

\[\text{is} \quad \text{Is an Object from a Class}\]

\textbf{Description}

- \texttt{is}: With two arguments, tests whether \texttt{object} can be treated as from \texttt{class2}.
- With one argument, returns all the super-classes of this object’s class.
- \texttt{extends}: Does the first class extend the second class? Returns \texttt{maybe} if the extension includes a test.
- \texttt{setIs}: Defines \texttt{class1} to be an extension of \texttt{class2}.

\textbf{Usage}

\begin{verbatim}
is(object, class2)
extends(class1, class2, maybe=TRUE, fullInfo = FALSE)
setIs(class1, class2, test=NULL, coerce=NULL, replace=NULL,
    by = character(), where = topenv(parent.frame()), classDef =,
    extensionObject = NULL, doComplete = TRUE)
\end{verbatim}
Arguments

object: any R object.
class1, class2: the names of the classes between which `is` relations are to be defined.
maybe, fullInfo: In a call to `extends`, maybe is a flag to include/exclude conditional relations, and fullInfo is a flag, which if TRUE causes object(s) of class `classExtension` to be returned, rather than just the names of the classes or a logical value. See the details below.
extensionObject: alternative to the test, coerce, replace, by arguments; an object from class `SClassExtension` describing the relation. (Used in internal calls.)
doComplete: when TRUE, the class definitions will be augmented with indirect relations as well. (Used in internal calls.)
test, coerce, replace: In a call to `setIs`, functions optionally supplied to test whether the relation is defined, to coerce the object to `class2`, and to alter the object so that `is(object, class2)` is identical to value.
by: In a call to `setIs`, the name of an intermediary class. Coercion will proceed by first coercing to this class and from there to the target class. (The intermediate coercions have to be valid.)
where: In a call to `setIs`, where to store the metadata defining the relationship. Default is the global environment.
classDef: Optional class definition for `class`, required internally when `setIs` is called during the initial definition of the class by a call to `setClass`. Don’t use this argument, unless you really know why you’re doing so.

details

details: Given two class names, `extends` by default says whether the first class extends the second; that is, it does for class names what `is` does for an object and a class. Given one class name, it returns all the classes that class extends (the “superclasses” of that class), including the class itself. If the flag `fullInfo` is TRUE, the result is a list, each element of which is an object describing the relationship; otherwise, and by default, the value returned is only the names of the classes.

setIs: This function establishes an inheritance relation between two classes, by some means other than having one class contain the other. It should not be used for ordinary relationships: either include the second class in the `contains=` argument to `setClass` if the class is contained in the usual way, or consider `setClassUnion` to define a virtual class that is extended by several ordinary classes. A call to `setIs` makes sense, for example, if one class ought to be automatically convertible into a second class, but they have different representations, so that the conversion must be done by an explicit computation, not just be inheriting slots, for example. In this case, you will typically need to provide both a `coerce=` and `replace=` argument to `setIs`. The `coerce`, `replace`, and `by` arguments behave as described for the `setAs` function. It’s unlikely you would use the `by` argument directly, but it is used in defining cached information about classes. The value returned (invisibly) by `setIs` is the extension information, as a list. The `coerce` argument is a function that turns a `class1` object into a `class2` object. The `replace` argument is a function of two arguments that modifies a `class1` object (the first
argument) to replace the part of it that corresponds to class2 (supplied as value, the second argument). It then returns the modified object as the value of the call. In other words, it acts as a replacement method to implement the expression as(object, class2) <- value. The easiest way to think of the coerce and replace functions is by thinking of the case that class1 contains class2 in the usual sense, by including the slots of the second class. (To repeat, in this situation you would not call setIs, but the analogy shows what happens when you do.)

The coerce function in this case would just make a class2 object by extracting the corresponding slots from the class1 object. The replace function would replace in the class1 object the slots corresponding to class2, and return the modified object as its value.

The relationship can also be conditional, if a function is supplied as the test argument. This should be a function of one argument that returns TRUE or FALSE according to whether the object supplied satisfies the relation is(object, class2). If you worry about such things, conditional relations between classes are slightly deprecated because they cannot be implemented as efficiently as ordinary relations and because they sometimes can lead to confusion (in thinking about what methods are dispatched for a particular function, for example). But they can correspond to useful distinctions, such as when two classes have the same representation, but only one of them obeys certain additional constraints.

Because only global environment information is saved, it rarely makes sense to give a value other than the default for argument where. One exception is where=0, which modifies the cached (i.e., session-scope) information about the class. Class completion computations use this version, but don’t use it yourself unless you are quite sure you know what you’re doing.

References

The R package methods implements, with a few exceptions, the programming interface for classes and methods in the book Programming with Data (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the methods package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.

Examples

```r
## a class definition (see setClass for the example)
setClass("trackCurve",
    representation("track", smooth = "numeric"))
## A class similar to "trackCurve", but with different structure
## allowing matrices for the "y" and "smooth" slots
setClass("trackMultiCurve",
    representation(x="numeric", y="matrix", smooth="matrix"),
    prototype = structure(list(), x=numeric(), y=matrix(0,0,0),
                           smooth= matrix(0,0,0)))
## Automatically convert an object from class "trackCurve" into
## "trackMultiCurve", by making the y, smooth slots into 1-column matrices
setIs("trackCurve", "trackMultiCurve",
```
isSealedMethod

Check for a Sealed Method or Class

Description

These functions check for either a method or a class that has been "sealed" when it was defined, and which therefore cannot be re-defined.

Usage

isSealedMethod(f, signature, fdef, where)

isSealedClass(Class, where)

Arguments

f
signatures
fdef
Class
where

The quoted name of the generic function.

The class names in the method’s signature, as they would be supplied to setMethod.

Optional, and usually omitted: the generic function definition for f.

The quoted name of the class.

where to search for the method or class definition. By default, searches from the top environment of the call to isSealedMethod or isSealedClass, typically the global environment or the namespace of a package containing a call to one of the functions.
Details

In the R implementation of classes and methods, it is possible to seal the definition of either a class or a method. The basic classes (numeric and other types of vectors, matrix and array data) are sealed. So also are the methods for the primitive functions on those data types. The effect is that programmers cannot re-define the meaning of these basic data types and computations. More precisely, for primitive functions that depend on only one data argument, methods cannot be specified for basic classes. For functions (such as the arithmetic operators) that depend on two arguments, methods can be specified if one of those arguments is a basic class, but not if both are.

Programmers can seal other class and method definitions by using the sealed argument to setClass or setMethod.

Value

The functions return FALSE if the method or class is not sealed (including the case that it is not defined); TRUE if it is.

References

The R package methods implements, with a few exceptions, the programming interface for classes and methods in the book Programming with Data (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the methods package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.

Examples

```r
## these are both TRUE
isSealedMethod("+", c("numeric", "character"))
isSealedClass("matrix")

setClass("track",
  representation(x="numeric", y="numeric"))
## but this is FALSE
isSealedClass("track")
## and so is this
isSealedClass("A Name for an undefined Class")
## and so are these, because only one of the two arguments is basic
isSealedMethod("+", c("track", "numeric"))
isSealedMethod("+", c("numeric", "track"))
```
Description

The virtual class "language" and the specific classes that extend it represent unevaluated objects, as produced for example by the parser or by functions such as quote.

Usage

### each of these classes corresponds to an unevaluated object
### in the S language. The class name can appear in method signatures,
### and in a few other contexts (such as some calls to as()).

"("  "<-"  "call"
"for"  "if"
"repeat"  "while"
"name"
"{ "

### Each of the classes above extends the virtual class

"language"

Objects from the Class

"language" is a virtual class; no objects may be created from it.

Objects from the other classes can be generated by a call to new(Class, ...), where Class is the quoted class name, and the ...arguments are either empty or a single object that is from this class (or an extension).

Methods

coerce signature(from = "ANY", to = "call"). A method exists for as(object, "call"), calling as.call().

LinearMethodsList-class

_class "LinearMethodsList"

Description

A version of methods lists that has been “linearized” for producing summary information. The actual objects from class "MethodsList" used for method dispatch are defined recursively over the arguments involved.

Objects from the Class

The function linearizeMlist converts an ordinary methods list object into the linearized form.
Slots

- **methods**: Object of class "list", the method definitions.
- **arguments**: Object of class "list", the corresponding formal arguments.
- **classes**: Object of class "list", the corresponding classes in the signatures.
- **fromClasses**: Object of class "list"

Future Note

The current version of `linearizeMlist` does not take advantage of the `MethodDefinition` class, and therefore does more work for less effect than it could. In particular, we may move to redefine both the function and the class to take advantage of the stored signatures. Don’t write code depending precisely on the present form, although all the current information will be obtainable in the future.

See Also

Function `linearizeMlist` for the computation, and `MethodsList-class` for the original, recursive form.

---

**makeClassRepresentation**

*Create a Class Definition*

Description

Constructs a `classRepresentation-class` object to describe a particular class. Mostly a utility function, but you can call it to create a class definition without assigning it, as `setClass` would do.

Usage

```r
makeClassRepresentation(name, slots=list(), superClasses=character(),
                       prototype=NULL, package, validity, access,
                       version, sealed, virtual=NA, where)
```

Arguments

- **name**: character string name for the class
- **slots**: named list of slot classes as would be supplied to `setClass`, but without the unnamed arguments for `superClasses` if any.
- **superClasses**: what classes does this class extend
- **prototype**: an object providing the default data for the class, e.g., the result of a call to `prototype`.
- **package**: The character string name for the package in which the class will be stored; see `getPackageName`.
- **validity**: Optional validity method. See `validObject`, and the discussion of validity methods in the reference.
- **access**: Access information. Not currently used.
version Optional version key for version control. Currently generated, but not used.
sealed Is the class sealed? See setClass.
virtual Is this known to be a virtual class?
where The environment from which to look for class definitions needed (e.g., for slots or superclasses). See the discussion of this argument under GenericFunctions.

References

The R package methods implements, with a few exceptions, the programming interface for classes and methods in the book Programming with Data (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the methods package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.

See Also

setClass

MethodDefinition-class

Classes to Represent Method Definitions

Description

These classes extend the basic class "function" when functions are to be stored and used as method definitions.

Details

Method definition objects are functions with additional information defining how the function is being used as a method. The target slot is the class signature for which the method will be dispatched, and the defined slot the signature for which the method was originally specified (that is, the one that appeared in some call to setMethod).

Objects from the Class

The action of setting a method by a call to setMethod creates an object of this class. It’s unwise to create them directly.

The class "SealedMethodDefinition" is created by a call to setMethod with argument sealed = TRUE. It has the same representation as "MethodDefinition".

Slots

>Data: Object of class "function"; the data part of the definition.
>target: Object of class "signature"; the signature for which the method was wanted.
>defined: Object of class "signature"; the signature for which a method was found. If the method was inherited, this will not be identical to target.
Methods

Description

This documentation section covers some general topics on how methods work and how the methods package interacts with the rest of R. The information is usually not needed to get started with methods and classes, but may be helpful for moderately ambitious projects, or when something doesn’t work as expected.

The section How Methods Work describes the underlying mechanism; Class Inheritance and Method Selection provides more details on how class definitions determine which methods are used.

The section Changes with the Methods Package outlines possible effects on other computations when running with package methods.

How Methods Work

A generic function is a function that has associated with it a collection of other functions (the methods), all of which agree in formal arguments with the generic. In R, the “collection” is an object of class "MethodsList", which contains a named list of methods (the methods slot), and the name of one of the formal arguments to the function (the argument slot). The names of the methods are the names of classes, and the corresponding element defines the method or methods to be used if the corresponding argument has that class. For example, suppose a function f has formal arguments x and y. The methods list object for that function has the object as.name("x") as its argument slot. An element of the methods named "track" is selected if the actual argument corresponding to x is an object of class "track". If there is such an element, it can generally be either a function or another methods list object.

In the first case, the function defines the method to use for any call in which x is of class "track".

In the second case, the new methods list object defines the selection of methods depending on the remaining formal arguments, in this example, y. The same selection process takes place, recursively, using the new methods list. Eventually, the selection returns either a function or NULL, meaning that no method matched the actual arguments.

Each method selected corresponds conceptually to a signature; that is a named list of classes, with names corresponding to some or all of the formal arguments. In the previous example, if selecting class "track" for x, finding that the selection was another methods list and then selecting class "numeric" for y would produce a method associated with the signature x = "track", y = "numeric".
The actual selection is done recursively, but you can see the methods arranged by signature by calling the function `showMethods`, and objects with the methods arranged this way (in two different forms) are returned by the functions `listFromMlist` and `linearizeMlist`.

In an R session, each generic function has a single methods list object defining all the currently available methods. The session methods list object is created the first time the function is called by merging all the relevant method definitions currently visible. Whenever something happens that might change the definitions (such as attaching or detaching a package with methods for this function, or explicitly defining or removing methods), the merged methods list object is removed. The next call to the function will recompute the merged definitions.

When methods list are merged, they can come from two sources:

1. Methods list objects for the same function anywhere on the current search list. These are merged so that methods in an environment earlier in the search list override methods for the same function later in the search list. A method overrides only another method for the same signature. See the comments on class "ANY" in the section on Inheritance.

2. Methods list objects corresponding the group generic functions, if any, for this function. Any generic function can be defined to belong to a group generic. The methods for the group generic are available as methods for this function. The group generic can itself be defined as belong to a group; as a result there is a list of group generic functions. A method defined for a function and a particular signature overrides a method for the same signature for that function’s group generic.

Merging is done first on all methods for a particular function, and then over the generic and its group generics.

The result is a single methods list object that contains all the methods directly defined for this function. As calls to the function occur, this information may be supplemented by inherited methods, which we consider next.

### Class Inheritance and Method Selection

If no method is found directly for the actual arguments in a call to a generic function, an attempt is made to match the available methods to the arguments by using inheritance.

Each class definition potentially includes the names of one or more classes that the new class contains. (These are sometimes called the superclasses of the new class.) These classes themselves may extend other classes. Putting all this information together produces the full list of superclasses for this class. (You can see this list for any class "A" from the expression `extends("A")`.) In addition, any class implicitly extends class "ANY". When all the superclasses are needed, as they are for dispatching methods, they are ordered by how direct they are: first, the direct classes contained directly in the definition of this class, then the superclasses of these classes, etc.

The S language has an additional, explicit mechanism for defining superclasses, the `setIs` mechanism. This mechanism allows a class to extend another even though they do not have the same representation. The extension is made possible by defining explicit methods to coerce an object to its superclass and to replace the data in the object corresponding to the superclass. The `setIs` mechanism will be used less often and only when directly including the superclass does not make sense, but once defined, the superclass acts just as directly contained classes as far as method selection is concerned.

A method will be selected by inheritance if we can find a method in the methods list for a signature corresponding to any combination of superclasses for each of the relevant arguments. The search for such a method is performed by the function `MethodsListSelect`, working as follows.

The generic, `f`, say, has a signature, which by default is all its formal arguments, except ...(see `setGeneric`). For each of the formal arguments in that signature, in order, the class of the
actual argument is matched against available methods. A missing argument corresponds to class "missing". If no method corresponds to the class of the argument, the evaluator looks for a method corresponding to the the superclasses (the other classes that the actual class extends, always including "ANY"). If no match is found, the dispatch fails, with an error. (But if there is a default method, that will always match.)

If the match succeeds, it can find either a single method, or a methods list. In the first case, the search is over, and returns the method. In the second case, the search proceeds, with the next argument in the signature of the generic. That search may succeed or fail. If it fails, the dispatch will try again with the next best match for the current argument, if there is one. The last match always corresponds to class "ANY".

The effect of this definition of the selection process is to order all possible inherited methods, first by the superclasses for the first argument, then within this by the superclasses for the second argument, and so on.

Changes with the Methods Package

The methods package is designed to leave other computations in R unchanged. There are, however, a few areas where the default functions and behavior are overridden when running with the methods package attached. This section outlines those known to have some possible effect.

class: The methods package enforces the notion that every object has a class; in particular, `class(x)` is never NULL, as it would be for basic vectors, for example, when not using methods.

In addition, when assigning a class, the value is required to be a single string. (However, objects can have multiple class names if these were generated by old-style class computations. The methods package does not hide the “extra” class names.)

Computations using the notion of NULL class attributes or of class attributes with multiple class names are not really compatible with the ideas in the methods package. Formal classes and class inheritance are designed to give more flexible and reliable implementations of similar ideas.

If you do have to mix the two approaches, any operations that use class attributes in the old sense should be written in terms of `attr(x, "class")`, not `class(x)`. In particular, test for no class having been assigned with `is.null(attr(x, "class")).`

Printing: To provide appropriate printing automatically for objects with formal class definitions, the methods package overrides `print.default`, to look for methods for the generic function `show`, and to use a default method for objects with formal class definitions.

The revised version of `print.default` is intended to produce identical printing to the original version for any object that does not have a formally defined class, including honoring old-style print methods. So far, no exceptions are known.

References

The R package methods implements, with a few exceptions, the programming interface for classes and methods in the book Programming with Data (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the methods package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.
**See Also**

`setGeneric, setClass`

---

**methods-package**  
*Formal Methods and Classes*

---

**Description**

Formally defined methods and classes for R objects, plus other programming tools, as described in the reference.

---

**Details**

This package provides the so-called “S version 4” object oriented system.

For a complete list of functions and classes, use `library(help = "methods")`.

---

**Author(s)**

R Development Core Team

Maintainer: R Core Team (R-core@r-project.org)

---

**References**

[http://www.omegahat.org/RSMethods/Intro.ps](http://www.omegahat.org/RSMethods/Intro.ps) and  

---

**MethodsList-class**  
*Class MethodsList, Representation of Methods for a Generic Function*

---

**Description**

Objects from this class are generated and revised by the definition of methods for a generic function.

---

**Slots**

- **argument**: Object of class "name". The name of the argument being used for dispatch at this level.

- **methods**: A named list of the methods (and method lists) defined explicitly for this argument, with the names being the classes for which the methods have been defined.

- **allMethods**: A named list, which may be empty if this object has not been used in dispatch yet. Otherwise, it contains all the directly defined methods from the `methods` slot, plus any inherited methods.

---

**Extends**

Class "OptionalMethods", directly.
MethodWithNext-class

Class MethodWithNext

Description

Class of method definitions set up for callNextMethod

Objects from the Class

Objects from this class are generated as a side-effect of calls to callNextMethod.

Slots

.Data: Object of class "function"; the actual function definition.

nextMethod: Object of class "PossibleMethod" the method to use in response to a callNextMethod() call.

excluded: Object of class "list"; one or more signatures excluded in finding the next method.

target: Object of class "signature", from class "MethodDefinition"

defined: Object of class "signature", from class "MethodDefinition"

Extends

Class "MethodDefinition", directly.
Class "function", from data part.
Class "PossibleMethod", by class "MethodDefinition".
Class "OptionalMethods", by class "MethodDefinition".

Methods

findNextMethod signature(method = "MethodWithNext"): used internally by method dispatch.

loadMethod signature(method = "MethodWithNext"): used internally by method dispatch.

show signature(object = "MethodWithNext")

See Also

callNextMethod, and MethodDefinition-class.
Generate an Object from a Class

Description

Given the name or the definition of a class, plus optionally data to be included in the object, new returns an object from that class.

Usage

new(Class, ...)

initialize(.Object, ...)

Arguments

Class Either the name of a class (the usual case) or the object describing the class (e.g., the value returned by getClass).

... Data to include in the new object. Named arguments correspond to slots in the class definition. Unnamed arguments must be objects from classes that this class extends.

.Object An object: see the Details section.

Details

The function new begins by copying the prototype object from the class definition. Then information is inserted according to the ... arguments, if any.

The interpretation of the ... arguments can be specialized to particular classes, if an appropriate method has been defined for the generic function "initialize". The new function calls initialize with the object generated from the prototype as the .Object argument to initialize.

By default, unnamed arguments in the ... are interpreted as objects from a superclass, and named arguments are interpreted as objects to be assigned into the correspondingly named slots. Thus, explicit slots override inherited information for the same slot, regardless of the order in which the arguments appear.

The initialize methods do not have to have ... as their second argument (see the examples), and generally it is better design not to have ... as a formal argument, if only a fixed set of arguments make sense.

For examples of initialize methods, see initialize-methods for existing methods for classes "traceable" and "environment", among others.

Note that the basic vector classes, "numeric", etc. are implicitly defined, so one can use new for these classes.

References

The web page http://www.omegahat.org/RSMethods/index.html is the primary documentation.

## using the definition of class "track" from Classes

### a new object with two slots specified
\[
t1 <- \text{new("track", } x = \text{seq(}\text{along=}}y\text{data), } y = y\text{data)\]

### a new object including an object from a superclass, plus a slot
\[
t2 <- \text{new("trackCurve", } t1, \text{smooth = }}ysmooth)\]

### define a method for initialize, to ensure that new objects have
### equal-length x and y slots.
\[
\text{setMethod("initialize", }
\text{"track",}
\text{function(.Object, } x = \text{numeric(}}0)\text{), } y = \text{numeric(}}0))
\{ 
  \text{if(nargs()} > 1) \{ 
    \text{if(length(x) != length(y))}
    \text{stop("specified x and y of different lengths")}
    \text{.Object@x <- x}
    \text{.Object@y <- y}
  \}
  \text{.Object}
\}
\]

### the next example will cause an error (x will be numeric(0)),
### because we didn't build in defaults for x,
### although we could with a more elaborate method for initialize
\[
\text{try(new("track", } y = \text{sort(rnorm(}}10)))\]

### a better way to implement the previous initialize method.
### Why? By using \text{callNextMethod} to call the default initialize method
### we don't inhibit classes that extend "track" from using the general
### form of the \text{new()} function. In the previous version, they could only
### use x and y as arguments to \text{new, unless they wrote their own
### initialize method.}
\[
\text{setMethod("initialize", } "track", \text{function(.Object, } \ldots) \{ 
  \text{.Object <- callNextMethod()}
  \text{if(length(.Object@x) != length(.Object@y))}
  \text{stop("specified x and y of different lengths")}
  \text{.Object}
\})\]
**promptClass**

**Description**

This class of objects is used to represent ordinary character string object names, extended with a package slot naming the package associated with each object.

**Objects from the Class**

The function `getGenerics` returns an object of this class.

**Slots**

- **Data**: Object of class "character": the object names.
- **package**: Object of class "character": the package names.

**Extends**

Class "character", from data part.
Class "vector", by class "character".

**See Also**

Methods for general background.

---

**promptClass**

*Generate a Shell for Documentation of a Formal Class*

**Description**

Assembles all relevant slot and method information for a class, with minimal markup for Rd processing; no QC facilities at present.

**Usage**

```
promptClass(clName, filename = NULL, type = "class", keywords = "classes", where = toopenv(parent.frame()))
```

**Arguments**

- **clName**: a character string naming the class to be documented.
- **filename**: usually, a connection or a character string giving the name of the file to which the documentation shell should be written. The default corresponds to a file whose name is the topic name for the class documentation, followed by ".Rd". Can also be NA (see below).
- **type**: the documentation type to be declared in the output file.
- **keywords**: the keywords to include in the shell of the documentation. The keyword "classes" should be one of them.
- **where**: where to look for the definition of the class and of methods that use it.
Details

The class definition is found on the search list. Using that definition, information about classes extended and slots is determined.

In addition, the currently available generics with methods for this class are found (using getGenerics). Note that these methods need not be in the same environment as the class definition; in particular, this part of the output may depend on which packages are currently in the search list.

As with other prompt-style functions, unless filename is NA, the documentation shell is written to a file, and a message about this is given. The file will need editing to give information about the meaning of the class. The output of promptClass can only contain information from the metadata about the formal definition and how it is used.

If filename is NA, a list-style representation of the documentation shell is created and returned. Writing the shell to a file amounts to cat(unlist(x), file = filename, sep = "\n"), where x is the list-style representation.

Value

If filename is NA, a list-style representation of the documentation shell. Otherwise, the name of the file written to is returned invisibly.

Author(s)

VJ Carey (stvjc@channing.harvard.edu) and John Chambers

References

The web page http://www.omegahat.org/RSMethods/index.html is the primary documentation.


See Also

prompt for documentation of functions, promptMethods for documentation of method definitions.

For processing of the edited documentation, either use R CMD Rdconv, or include the edited file in the ‘man’ subdirectory of a package.

Examples

```r
## Not run:
> promptClass("track")
A shell of class documentation has been written to the file "track-class.Rd".
## End(Not run)
```
promptMethods  

*Generate a Shell for Documentation of Formal Methods*

**Description**

Generates a shell of documentation for the methods of a generic function.

**Usage**

```r
promptMethods(f, filename = NULL, methods)
```

**Arguments**

- `f`  
  a character string naming the generic function whose methods are to be documented.

- `filename`  
  usually, a connection or a character string giving the name of the file to which the documentation shell should be written. The default corresponds to the coded topic name for these methods (currently, `f` followed by "-methods.Rd"). Can also be `FALSE` or `NA` (see below).

- `methods`  
  Optional methods list object giving the methods to be documented. By default, the first methods object for this generic is used (for example, if the current global environment has some methods for `f`, these would be documented). If this argument is supplied, it is likely to be `getMethods(f, where)` with `where` some package containing methods for `f`.

**Details**

If `filename` is `FALSE`, the text created is returned, presumably to be inserted some other documentation file, such as the documentation of the generic function itself (see `prompt`).

If `filename` is `NA`, a list-style representation of the documentation shell is created and returned. Writing the shell to a file amounts to

```r
cat(unlist(x), file = filename, sep = "\n"),
```

where `x` is the list-style representation.

Otherwise, the documentation shell is written to the file specified by `filename`.

**Value**

If `filename` is `FALSE`, the text generated; if `filename` is `NA`, a list-style representation of the documentation shell. Otherwise, the name of the file written to is returned invisibly.

**References**

The R package `methods` implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the `methods` package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.
representation

Construct a Representation or a Prototype for a Class Definition

Description

In calls to `setClass`, these two functions construct, respectively, the representation and prototype arguments. They do various checks and handle special cases. You're encouraged to use them when defining classes that, for example, extend other classes as a data part or have multiple superclasses, or that combine extending a class and slots.

Usage

representation(...)
prototype(...)

Arguments

... The call to representation takes arguments that are single character strings. Unnamed arguments are classes that a newly defined class extends; named arguments name the explicit slots in the new class, and specify what class each slot should have.

In the call to prototype, if an unnamed argument is supplied, it unconditionally forms the basis for the prototype object. Remaining arguments are taken to correspond to slots of this object. It is an error to supply more than one unnamed argument.

Details

The `representation` function applies tests for the validity of the arguments. Each must specify the name of a class.

The classes named don't have to exist when `representation` is called, but if they do, then the function will check for any duplicate slot names introduced by each of the inherited classes.

The arguments to `prototype` are usually named initial values for slots, plus an optional first argument that gives the object itself. The unnamed argument is typically useful if there is a data part to the definition (see the examples below).

Value

The value of `representation` is just the list of arguments, after these have been checked for validity.

The value of `prototype` is the object to be used as the prototype. Slots will have been set consistently with the arguments, but the construction does not use the class definition to test validity of the contents (it hardly can, since the prototype object is usually supplied to create the definition).
References

The R package **methods** implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the **methods** package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.

See Also

`setClass`

Examples

```r
## representation for a new class with a directly define slot "smooth"
## which should be a "numeric" object, and extending class "track"
representation("track", smooth = "numeric")

setClass("Character", representation("character"))
setClass("TypedCharacter", representation("Character", type = "character"),
  prototype(character(0), type = "plain"))

ttt <- new("TypedCharacter", "foo", type = "character")

setClass("num1", representation(comment = "character"),
  contains = "numeric",
  prototype = prototype(pi, comment = "Start with pi"))
```

---

**SClassExtension-class**

*Class to Represent Inheritance (Extension) Relations*

Description

An object from this class represents a single “is” relationship; lists of these objects are used to represent all the extensions (superclasses) and subclasses for a given class. The object contains information about how the relation is defined and methods to coerce, test, and replace correspondingly.

Objects from the Class

Objects from this class are generated by `setIs`, both from direct calls.
Slots

subClass, superClass: The classes being extended: corresponding to the from, and to arguments to setIs.

package: The package to which that class belongs.

coerce: A function to carry out the as() computation implied by the relation. Note that these functions should not be used directly. They only deal with the strict=TRUE calls to the as function, with the full method constructed from this mechanically.

test: The function that would test whether the relation holds. Except for explicitly specified test arguments to setIs, this function is trivial.

replace: The method used to implement as(x, Class) <- value.

simple: A "logical" flag, TRUE if this is a simple relation, either because one class is contained in the definition of another, or because a class has been explicitly stated to extend a virtual class. For simple extensions, the three methods are generated automatically.

by: If this relation has been constructed transitively, the first intermediate class from the subclass.

dataPart: A "logical" flag, TRUE if the extended class is in fact the data part of the subclass.

In this case the extended class is a basic class (i.e., a type).

Methods

No methods defined with class "SClassExtension" in the signature.

See Also

is, as, and classRepresentation-class.

setClass Create a Class Definition

Description

Functions to create (setClass) and manipulate class definitions.

Usage

setClass(Class, representation, prototype, contains=character(),
  validity, access, where, version, sealed, package)

removeClass(Class, where)

isclass(Class, formal=TRUE, where)

getClasses(where, inherits = missing(where))

findClass(Class, where, unique = "")

resetClass(Class, classDef, where)

sealClass(Class, where)
**Arguments**

**Class** character string name for the class. Other than `setClass`, the functions will usually take a class definition instead of the string (allowing the caller to identify the class uniquely).

**representation**

the slots that the new class should have and/or other classes that this class extends. Usually a call to the `representation` function.

**prototype** an object (usually a list) providing the default data for the slots specified in the representation.

**contains** what classes does this class extend? (These are called superclasses in some languages.) When these classes have slots, all their slots will be contained in the new class as well.

**where** For `setClass` and `removeClass`, the environment in which to store or remove the definition. Defaults to the top-level environment of the calling function (the global environment for ordinary computations, but the environment or namespace of a package when loading that package).

For other functions, `where` defines where to do the search for the class definition, and the default is to search from the top-level environment or namespace of the caller to this function.

**unique** if `findClass` expects a unique location for the class, `unique` is a character string explaining the purpose of the search (and is used in warning and error messages). By default, multiple locations are possible and the function always returns a list.

**inherits** in a call to `getClasses`, should the value returned include all parent environments of `where`, or that environment only? Defaults to TRUE if `where` is omitted, and to FALSE otherwise.

**validity** if supplied, should be a validity-checking method for objects from this class (a function that returns TRUE if its argument is a valid object of this class and one or more strings describing the failures otherwise). See `validObject` for details.

**access** Access list for the class. Saved in the definition, but not currently used.

**version** A version indicator for this definition. Saved in the definition, but not currently used.

**sealed** If TRUE, the class definition will be sealed, so that another call to `setClass` will fail on this class name.

**package** An optional package name for the class. By default (and usually) the package where the class definition is assigned will be used.

**formal** Should a formal definition be required?

**classDef** For `removeClass`, the optional class definition (but usually it’s better for `Class` to be the class definition, and to omit `classDef`).

**Details**

These are the functions that create and manipulate formal class definitions. Brief documentation is provided below. See the references for an introduction and for more details.

**setClass**: Define `Class` to be an S-style class. The effect is to create an object, of class "classRepEnvironment", and store this (hidden) in the specified environment or database. Objects can be created from the class (e.g., by calling `new`), manipulated (e.g., by
accessing the object’s slots), and methods may be defined including the class name in the signature (see `setMethod`).

**removeClass**: Remove the definition of this class, from the environment where if this argument is supplied; if not, `removeClass` will search for a definition, starting in the top-level environment of the call to `removeClass`, and remove the (first) definition found.

**isClass**: Is this the name of a formally defined class? (Argument `formal` is for compatibility and is ignored.)

**getClasses**: The names of all the classes formally defined on where. If called with no argument, all the classes visible from the calling function (if called from the top-level, all the classes in any of the environments on the search list). The `inherits` argument can be used to search a particular environment and all its parents, but usually the default setting is what you want.

**findClass**: The list of environments or positions on the search list in which a class definition of `Class` is found. If `where` is supplied, this is an environment (or namespace) from which the search takes place; otherwise the top-level environment of the caller is used. If `unique` is supplied as a character string, `findClass` returns a single environment or position. By default, it always returns a list. The calling function should select, say, the first element as a position or environment for functions such as `get`. If `unique` is supplied as a character string, `findClass` will warn if there is more than one definition visible (using the string to identify the purpose of the call), and will generate an error if no definition can be found.

**resetClass**: Reset the internal definition of a class. Causes the complete definition of the class to be re-computed, from the representation and superclasses specified in the original call to `setClass`.

This function is called when aspects of the class definition are changed. You would need to call it explicitly if you changed the definition of a class that this class extends (but doing that in the middle of a session is living dangerously, since it may invalidate existing objects).

**sealClass**: Seal the current definition of the specified class, to prevent further changes. It is possible to seal a class in the call to `setClass`, but sometimes further changes have to be made (e.g., by calls to `setIs`). If so, call `sealClass` after all the relevant changes have been made.

### Inheritance and Prototypes

Defining new classes that inherit from ("extend") other classes is a powerful technique, but has to be used carefully and not over-used. Otherwise, you will often get unintended results when you start to compute with objects from the new class.

As shown in the examples below, the simplest and safest form of inheritance is to start with an explicit class, with some slots, that does not extend anything else. It only does what we say it does. Then extensions will add some new slots and new behavior.

Another variety of extension starts with one of the built-in data types, perhaps with the intention of modifying R’s standard behavior for that class. In this case, the new class inherits the built-in data type as its “data” part. See the “numWithId” example below.

When such a class definition is printed, the data part shows up as a pseudo-slot named “.Data”.

### S3 Classes

Earlier, informal classes of objects (usually referred to as “S3” classes) are used by many R functions. It’s natural to consider including them as the class for a slot in a formal class, or even as a class to be extended by the new class. This isn’t prohibited but there are some disadvantages,
and if you do want to include S3 classes, they should be declared by including them in a call to `setOldClass`. Here are some considerations:

- Using S3 classes somewhat defeats the purpose of defining a formal class: An important advantage to your users is that a formal class provides guarantees of what the object contains (minimally, the classes of the slots and therefore what data they contain; optionally, any other requirements imposed by a validity method).

  But there is no guarantee whatever about the data in an object from an S3 class. It’s entirely up to the functions that create or modify such objects. If you want to provide guarantees to your users, you will need a validity method that explicitly checks the contents of S3-class objects.

- To get the minimal guarantee (that the object in a slot has, or extends, the class for the slot) you should ensure that the S3 classes are known to be S3 classes, with the possible inheritance. To do this, include a call to `setOldClass` for the S3 classes used.

  Otherwise, the S3 class is undefined (and the code used by `setClass` will issue a warning). Slot assignments, for example, will not then check for possible errors.

- These caveats apply to S3 classes; that is, objects with a class assigned by some R function but without a formal class definition. In contrast, the built-in data types (numeric, list, etc.) are generally fine as slots or for `contains= classes (see the previous section). These data types don’t have formal slots, but the base code in the system essentially forces them to contain the type of data they claim to have.

  The data types `matrix` and `array` are somewhat in between. They do not have an explicit S3 class, but do have one or two attributes. There is no general problem in having these as slots, but because there is no guarantee of a dimnames slot, they don’t work as formal classes.

  The `ts` class is treated as a formal class, extending class `vector`.

References

The R package `methods` implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the `methods` package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.

See Also

`setClassUnion`, `Methods`, `makeClassRepresentation`

Examples

```r
## A simple class with two slots
setClass("track",
   representation(x="numeric", y="numeric"))

## A class extending the previous, adding one more slot
setClass("trackCurve",
   representation("track", smooth = "numeric"))

## A class similar to "trackCurve", but with different structure
## allowing matrices for the "y" and "smooth" slots
setClass("trackMultiCurve",
   representation(y="matrix", smooth="numeric"))
```

```r
```
representation(x="numeric", y="matrix", smooth="matrix"),
prototype = list(x=numeric(), y=matrix(0,0,0),
                      smooth= matrix(0,0,0)))
##
## Suppose we want trackMultiCurve to be like trackCurve when there's
## only one column.
## First, the wrong way.
try(setIs("trackMultiCurve", "trackCurve",
    test = function(obj) {ncol(slot(obj, "y")) == 1})
## Why didn't that work? You can only override the slots "x", "y",
## and "smooth" if you provide an explicit coerce function to correct
## any inconsistencies:
setIs("trackMultiCurve", "trackCurve",
    test = function(obj) {ncol(slot(obj, "y")) == 1},
    coerce = function(obj) {
        new("trackCurve",
            x = slot(obj, "x"),
            y = as.numeric(slot(obj,"y")),
            smooth = as.numeric(slot(obj, "smooth")))
    })
## A class that extends the built-in data type "numeric"
setClass("numWithId", representation(id = "character"),
         contains = "numeric")
new("numWithId", 1:3, id = "An Example")

setClassUnion

Classes Defined as the Union of Other Classes

Description

A class may be defined as the union of other classes; that is, as a virtual class defined as a superclass
of several other classes. Class unions are useful in method signatures or as slots in other classes,
when we want to allow one of several classes to be supplied.

Usage

setClassUnion(name, members, where)
isClassUnion(Class)

Arguments

name the name for the new union class.
members the classes that should be members of this union.
where where to save the new class definition; by default, the environment of the package
in which the setClassUnion call appears, or the global environment if
called outside of the source of a package.
Class the name or definition of a class.
Details

The classes in members must be defined before creating the union. However, members can be added later on to an existing union, as shown in the example below. Class unions can be members of other class unions.

Class unions are the only way to create a class that is extended by a class whose definition is sealed (for example, the basic datatypes or other classes defined in the base or methods package in R are sealed). You cannot say setIs("function", "other") unless "other" is a class union. In general, a setIs call of this form changes the definition of the first class mentioned (adding "other" to the list of superclasses contained in the definition of "function").

Class unions get around this by not modifying the first class definition, relying instead on storing information in the subclasses slot of the class union. In order for this technique to work, the internal computations for expressions such as extends(class1, class2) work differently for class unions than for regular classes; specifically, they test whether any class is in common between the superclasses of class1 and the subclasses of class2.

The different behavior for class unions is made possible because the class definition object for class unions has itself a special class, "ClassUnionRepresentation", an extension of "classRepresentation" (see classRepresentation-class).

References

The R package methods implements, with a few exceptions, the programming interface for classes and methods in the book Programming with Data (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the methods package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.

Examples

## a class for either numeric or logical data
setClassUnion("maybeNumber", c("numeric", "logical"))

## use the union as the data part of another class
setClass("withId", representation("maybeNumber", id = "character"))

w1 <- new("withId", 1:10, id = "test 1")
w2 <- new("withId", sqrt(w1)%%1 < .01, id = "Perfect squares")

## add class "complex" to the union "maybeNumber"
setIs("complex", "maybeNumber")
w3 <- new("withId", complex(real = 1:10, imaginary = sqrt(1:10)))

## a class union containing the existing class union "OptionalFunction"
setClassUnion("maybeCode",
    c("expression", "language", "OptionalFunction"))

is(quote(sqrt(1:10)), "maybeCode")  ## TRUE
The character string name of the generic function. In the simplest and most
common case, a function of this name is already defined. The existing function
may be non-generic or already a generic (see the details).

An optional function object, defining the generic. This argument is usually only
needed (and is then required) if there is no current function of this name. In
that case, the formal arguments and default values for the generic are taken from
def. You can also supply this argument if you want the generic function to do
something other than just dispatch methods (an advanced topic best left alone
unless you are sure you want it).

Note that def is not the default method; use argument useAsDefault if you
want to specify the default separately.

Optionally, a character string giving the group of generic functions to which
this function belongs. Methods can be defined for the corresponding group
generic, and these will then define methods for this specific generic function,
if no method has been explicitly defined for the corresponding signature. See
the references for more discussion.

An optional character vector or unevaluated expression. The value returned by
the generic function must have (or extend) this class, or one of the classes; other-
wise, an error is generated. See the details section for supplying an expression.

The name of the package with which this function is associated. Usually deter-
mined automatically (as the package containing the non-generic version if there
is one, or else the package where this generic is to be saved).

Where to store the resulting initial methods definition, and possibly the generic
function; by default, stored into the top-level environment.

Options, the signature of arguments in the function that can be used in meth-
ods for this generic. By default, all arguments other than ... can be used.
The signature argument can prohibit methods from using some arguments. The
argument, if provided, is a vector of formal argument names.
genericFunction
The object to be used as a (nonstandard) generic function definition. Supply this explicitly only if you know what you are doing!

useAsDefault
Override the usual choice of default argument (an existing non-generic function or no default if there is no such function). Argument useAsDefault can be supplied, either as a function to use for the default, or as a logical value. FALSE says not to have a default method at all, so that an error occurs if there is not an explicit or inherited method for a call. TRUE says to use the existing function as default, unconditionally (hardly ever needed as an explicit argument). See the section on details.

knownMembers
(For setGroupGeneric only) The names of functions that are known to be members of this group. This information is used to reset cached definitions of the member generics when information about the group generic is changed.

Details

The setGeneric function is called to initialize a generic function in an environment (usually the global environment), as preparation for defining some methods for that function.

The simplest and most common situation is that name is already an ordinary non-generic function, and you now want to turn this function into a generic. In this case you will most often supply only name. The existing function becomes the default method, and the special group and valueClass properties remain unspecified.

A second situation is that you want to create a new, generic function, unrelated to any existing function. In this case, you need to supply a skeleton of the function definition, to define the arguments for the function. The body of a generic function is usually a standard form, standardGeneric(name) where name is the quoted name of the generic function.

When calling setGeneric in this form, you would normally supply the def argument as a function of this form. If not told otherwise, setGeneric will try to find a non-generic version of the function to use as a default. If you don’t want this to happen, supply the argument useAsDefault. That argument can be the function you want to be the default method. You can supply the argument as FALSE to force no default (i.e., to cause an error if there is not direct or inherited method on call to the function).

The same no-default situation occurs if there is no non-generic form of the function, and useAsDefault=FALSE. Remember, though, you can also just assign the default you want (even one that generates an error) rather than relying on the prior situation.

You cannot (and never need to) create an explicit generic for the primitive functions in the base library. These are dispatched from C code for efficiency and are not to be redefined in any case.

As mentioned, the body of a generic function usually does nothing except for dispatching methods by a call to standardGeneric. Under some circumstances you might just want to do some additional computation in the generic function itself. As long as your function eventually calls standardGeneric that is permissible (though perhaps not a good idea, in that it makes the behavior of your function different from the usual S model). If your explicit definition of the generic function does not call standardGeneric you are in trouble, because none of the methods for the function will ever be dispatched.

By default, the generic function can return any object. If valueClass is supplied, it should be a vector of class names; the value returned by a method is then required to satisfy is(object, Class) for one of the specified classes. An empty (i.e., zero length) vector of classes means anything is allowed. Note that more complicated requirements on the result can be specified explicitly, by defining a non-standard generic function.
The `setGroupGeneric` function behaves like `setGeneric` except that it constructs a group generic function, differing in two ways from an ordinary generic function. First, this function cannot be called directly, and the body of the function created will contain a stop call with this information. Second, the group generic function contains information about the known members of the group, used to keep the members up to date when the group definition changes, through changes in the search list or direct specification of methods, etc.

**Value**

The `setGeneric` function exists for its side effect: saving the generic function to allow methods to be specified later. It returns `name`.

**Generic Functions and Primitive Functions**

A number of the basic R functions are specially implemented as primitive functions, to be evaluated directly in the underlying C code rather than by evaluating an S language definition. Primitive functions are eligible to have methods, but are handled differently by `setGeneric` and `setGroupGeneric`. A call to `setGeneric` for a primitive function does not create a new definition of the function, and the call is allowed only to “turn on” methods for that function.

A call to `setGeneric` for a primitive causes the evaluator to look for methods for that generic; a call to `setGroupGeneric` for any of the groups that include primitives ("Arith", "Logic", "Compare", "Ops", "Math", "Math2", "Summary", and "Complex") does the same for each of the functions in that group.

You usually only need to use either function if the methods are being defined only for the group generic. Defining a method for a primitive function, say "+", by a call to `setMethod` turns on method dispatch for that function. But in R defining a method for the corresponding group generic, "Arith", does not currently turn on method dispatch (for efficiency reasons). If there are no non-group methods for the functions, you have two choices.

You can turn on method dispatch for all the functions in the group by calling `setGroupGeneric("Arith")`, or you can turn on method dispatch for only some of the functions by calling `setGeneric("+", etc. Note that in either case you should give the name of the generic function as the only argument.

**References**

The R package `methods` implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the `methods` package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.

**See Also**

`Methods` for a discussion of other functions to specify and manipulate the methods of generic functions.
Examples

### A non-standard generic function. It insists that the methods
### return a non-empty character vector (a stronger requirement than
### valueClass = "character" in the call to setGeneric)

```
setGeneric("authorNames",
  function(text) {
    value <- standardGeneric("authorNames")
    if(!(is(value, "character") & any(nchar(value)>0)))
      stop("authorNames methods must return non-empty strings")
    value
  })
```

## An example of group generic methods, using the class
## "track"; see the documentation of setClass for its definition

```
# define a method for the Arith group

setMethod("Arith", c("track", "numeric"),
  function(e1, e2){
    e1@y <- callGeneric(e1@y , e2)
    e1
  })

setMethod("Arith", c("numeric", "track"),
  function(e1, e2){
    e2@y <- callGeneric(e1, e2@y)
    e2
  })

# now arithmetic operators will dispatch methods:

t1 <- new("track", x=1:10, y=sort(rnorm(10)))

t1 - 100

1/t1
```

setMethod

Create and Save a Method

Description

Create and save a formal method for a given function and list of classes.

Usage

```
setMethod(f, signature=character(), definition,
```
removeMethod(f, signature, where)

Arguments

- **f**
  - The character-string name of the generic function.

- **signature**
  - A match of formal argument names for `f` with the character-string names of corresponding classes. This argument can also just be the vector of class names, in which case the first name corresponds to the first formal argument, the next to the second formal argument, etc.

- **definition**
  - A function definition, which will become the method called when the arguments in a call to `f` match the classes in `signature`, directly or through inheritance.

- **where**
  - the database in which to store the definition of the method; For `removeMethod`, the default is the location of the (first) instance of the method for this signature.

- **valueClass**
  - If supplied, this argument asserts that the method will return a value of this class. (At present this argument is stored but not explicitly used.)

- **sealed**
  - If `TRUE`, the method so defined cannot be redefined by another call to `setMethod` (although it can be removed and then re-assigned). Note that this argument is an extension to the definition of `setMethod` in the reference.

Details

R methods for a particular generic function are stored in an object of class `MethodsList`. The effect of calling `setMethod` is to store `definition` in a `MethodsList` object on database `where`. If `f` doesn’t exist as a generic function, but there is an ordinary function of the same name and the same formal arguments, a new generic function is created, and the previous non-generic version of `f` becomes the default method. This is equivalent to the programmer calling `setGeneric` for the same function; it’s better practice to do the call explicitly, since it shows that you intend to turn `f` into a generic function.

Methods are stored in a hierarchical structure: see `Methods` for how the objects are used to select a method, and `MethodsList` for functions that manipulate the objects.

The class names in the signature can be any formal class, plus predefined basic classes such as "numeric", "character", and "matrix". Two additional special class names can appear: "ANY", meaning that this argument can have any class at all; and "missing", meaning that this argument must not appear in the call in order to match this signature. Don’t confuse these two: if an argument isn’t mentioned in a signature, it corresponds implicitly to class "ANY", not to "missing". See the example below. Old-style (“S3”) classes can also be used, if you need compatibility with these, but you should definitely declare these classes by calling `setOldClass` if you want S3-style inheritance to work.

While `f` can correspond to methods defined on several packages or environments, the underlying model is that these together make up the definition for a single generic function. When R proceeds to select and evaluate methods for `f`, the methods on the current search list are merged to form a single generic function and associated methods list. When `f` is called and a method is “dispatched”, the evaluator matches the classes of the actual arguments to the signatures of the available methods. When a match is found, the body of the corresponding method is evaluated, but without rematching the arguments to `f`. Aside from not rematching the arguments, the computation proceeds as if the call had been to the method. In particular, the lexical scope of the method is used.
Method definitions can have default expressions for arguments. If those arguments are then missing in the call to the generic function, the default expression in the method is used. If the method definition has no default for the argument, then the expression (if any) supplied in the definition of the generic function itself is used. But note that this expression will be evaluated in the environment defined by the method.

It is possible to have some differences between the formal arguments to a method supplied to `setMethod` and those of the generic. Roughly, if the generic has ...as one of its arguments, then the method may have extra formal arguments, which will be matched from the arguments matching ...in the call to f. (What actually happens is that a local function is created inside the method, with its formal arguments, and the method is re-defined to call that local function.)

Method dispatch tries to match the class of the actual arguments in a call to the available methods collected for f. Roughly, for each formal argument in turn, we look for the best match (the exact same class or the nearest element in the value of `extends` for that class) for which there is any possible method matching the remaining arguments. See Methods for more details.

Value

These functions exist for their side-effect, in setting or removing a method in the object defining methods for the specified generic.

The value returned by `removeMethod` is `TRUE` if a method was found to be removed.

References

The R package `methods` implements, with a few exceptions, the programming interface for classes and methods in the book Programming with Data (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the `methods` package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page http://developer.r-project.org/methodsPackage.html and the pointers from that page.

See Also

Methods, MethodsList for details of the implementation

Examples

```r
## methods for plotting track objects (see the example for setClass)

## First, with only one object as argument:
setMethod("plot", signature(x="track", y="missing"),
  function(x, y, ...) plot(slot(x, "x"), slot(x, "y"), ...)
)

## Second, plot the data from the track on the y-axis against anything
## as the x data.
setMethod("plot", signature(y = "track"),
  function(x, y, ...) plot(x, slot(y, "y"), ...)
)

## and similarly with the track on the x-axis (using the short form of
```
### Specification for Signatures)

```r
setMethod("plot", "track",
  function(x, y, ...) plot(slot(x, "y"), y, ...)
)
t1 <- new("track", x=1:20, y=(1:20)^2)
tcl <- new("trackCurve", t1)
slot(tcl, "smooth") <- smooth.spline(slot(tcl, "x"), slot(tcl, "y"))$y
plot(t1)
plot(qnorm(ppoints(20)), t1)
```

### An example of Inherited Methods, and of Conforming Method Arguments

```r
setMethod("plot", c("trackCurve", "missing"),
  function(x, y, dotCurve = FALSE, ...) {
    plot(as(x, "track"))
    if(length(slot(x, "smooth") > 0))
      lines(slot(x, "x"), slot(x, "smooth"),
        lty = if(dotCurve) 2 else 1)
  })
```

### the plot of tcl alone has an added curve; other uses of tcl
### are treated as if it were a "track" object.

```r
plot(tcl, dotCurve = TRUE)
plot(qnorm(ppoints(20)), tcl)
```

### Defining Methods for a Special Function.

```r
setMethod("[", "track",
  function(x, i, j, ..., drop) {
    x@x <- x@x[i]; x@y <- x@y[i]
    x
  })
plot(t1[1:15])
```

### Methods can be defined for Missing Arguments as Well

```r
setMethod("length", "track", function(x)length(x@y))
length(t1)
```

### Methods Can Be Defined for Missing Arguments as Well

```r
setGeneric("summary") # make the function into a generic
```

### A Method for summary()

```r
setMethod("summary", "missing", function() "<No Object>")
```

---

### setOldClass

#### Specify Names for Old-Style Classes

**Description**

Register an old-style (a.k.a. 'S3') class as a formally defined class. The Classes argument is the character vector used as the class attribute; in particular, if there is more than one string, old-style
class inheritance is mimiced. Registering via setOldClass allows S3 classes to appear as slots
or in method signatures.

Usage

setOldClass(Classes, where, test = FALSE)

Arguments

Classes  A character vector, giving the names for old-style classes, as they would appear
         on the right side of an assignment of the class attribute.

where    Where to store the class definitions, the global or top-level environment by de-
          fault. (When either function is called in the source for a package, the class
          definitions will be included in the package's environment by default.)

test     flag, if TRUE, inheritance must be tested explicitly for each object, needed if the
          S3 class can have a different set of class strings, with the same first string. See
          the details below.

Details

Each of the names will be defined as a virtual class, extending the remaining classes in Classes,
and the class oldClass, which is the "root" of all old-style classes. See Methods for the details
of method dispatch and inheritance. See the section Register or Convert? for comments on the
alternative of defining "real" S4 classes rather than using setOldClass.

S3 classes have no formal definition, and some of them cannot be represented as an ordinary combi-
nation of S4 classes and superclasses. It is still possible to register the classes as S4 classes, but now
the inheritance has to be verified for each object, and you must call setOldClass with argument
test=TRUE.

For example, ordered factors always have the S3 class c("ordered", "factor"). This is
proper behavior, and maps simply into two S4 classes, with "ordered" extending "factor".

But objects whose class attribute has "POSIXt" as the first string may have either (or neither) of
"POSIXct" or "POSIXlt" as the second string. This behavior can be mapped into S4 classes
but now to evaluate is(x, "POSIXlt"). For example, requires checking the S3 class attribute
on each object. Supplying the test=TRUE argument to setOldClass causes an explicit test to
be included in the class definitions. It's never wrong to have this test, but since it adds significant
overhead to methods defined for the inherited classes, you should only supply this argument if it's
known that object-specific tests are needed.

The list .OldClassesList contains the old-style classes that are defined by the methods pack-

Register or Convert?

A call to setOldClass creates formal classes corresponding to S3 classes, allows these to be
used as slots in other classes or in a signature in setMethod, and mimics the S3 inheritance.

However, all such classes are created as virtual classes, meaning that you cannot generally create
new objects from the class by calling new, and that objects cannot be coerced automatically from
or to these classes. All these restrictions just reflect the fact that nothing is inherently known about
the “structure” of S3 classes, or whether in fact they define a consistent set of attributes that can be mapped into slots in a formal class definition.

If your class does in fact have a consistent structure, so that every object from the class has the same structure, you may prefer to take some extra time to write down a specific definition in a call to setClass to convert the class to a fully functional formal class. On the other hand, if the actual contents of the class vary from one object to another, you may have to redesign most of the software using the class, in which case converting it may not be worth the effort. You should still register the class via setOldClass, unless its class attribute is hopelessly unpredictable.

An S3 class has consistent structure if each object has the same set of attributes, both the names and the classes of the attributes being the same for every object in the class. In practice, you can convert classes that are slightly less well behaved. If a few attributes appear in some but not all objects, you can include these optional attributes as slots that always appear in the objects, if you can supply a default value that is equivalent to the attribute being missing. Sometimes NULL can be that value: A slot (but not an attribute) can have the value NULL. If version, for example, was an optional attribute, the old test is.null(attr(x,"version")) for a missing version attribute could turn into is.null(x@version) for the formal class.

The requirement that slots have a fixed class can be satisfied indirectly as well. Slots can be specified with class "ANY", allowing an arbitrary object. However, this eliminates an important benefit of formal class definitions; namely, automatic validation of objects assigned to a slot. If just a few different classes are possible, consider using setClassUnion to define valid objects for a slot.

See Also

setClass, setMethod

Examples

setOldClass(c("mlm", "lm"))
setGeneric("dfResidual", function(model) standardGeneric("dfResidual"))
setMethod("dfResidual", "lm", function(model) model$df.residual)

## dfResidual will work on mlm objects as well as lm objects
myData <- data.frame(time = 1:10, y = (1:10)^.5)
myLm <- lm(cbind(y, y^3) ~ time, myData)

rm(myData, myLm)
removeGeneric("dfResidual")

show

Show an Object

Description

Display the object, by printing, plotting or whatever suits its class. This function exists to be specialized by methods. The default method calls showDefault.

Formal methods for show will usually be invoked for automatic printing (see the details).

Usage

show(object)
show

Arguments

object Any R object

Details

The methods package overrides the base definition of print.default to arrange for automatic printing to honor methods for the function show. This does not quite manage to override old-style printing methods, since the automatic printing in the evaluator will look first for the old-style method.

If you have a class myClass and want to define a method for show, all will be well unless there is already a function named print.myClass. In that case, to get your method dispatched for automatic printing, it will have to be a method for print. A slight cheat is to override the function print.myClass yourself, and then call that function also in the method for show with signature "myClass".

Value

show returns an invisible NULL.

See Also

showMethods prints all the methods for one or more functions; showMlist prints individual methods lists; showClass prints class definitions. Neither of the latter two normally needs to be called directly.

Examples

## following the example shown in the setMethod documentation ...
setClass("track",
        representation(x="numeric", y="numeric"))
setClass("trackCurve",
        representation("track", smooth = "numeric"))
t1 <- new("track", x=1:20, y=(1:20)^2)
tc1 <- new("trackCurve", t1)
setMethod("show", "track",
          function(object)print(rbind(x = object@x, y=object@y))
        )
## The method will now be used for automatic printing of t1

t1

## Not run:
  x  1  2  3  4  5  6  7  8  9 10 11 12
  y  1  4  9 16 25 36 49 64 81 100 121 144
  x 13 14 15 16 17 18 19 20
  y 169 196 225 256 289 324 361 400
## End(Not run)

## and also for tcl, an object of a class that extends "track"
tc1
showMethods  

Show all the methods for the specified function(s)

Description

Show a summary of the methods for one or more generic functions, possibly restricted to those involving specified classes.

Usage

showMethods(f = character(), where = toopenv(parent.frame()),
            classes = NULL, includeDefs = FALSE, inherited = TRUE,
            showEmpty = TRUE, printTo = stdout())

Arguments

f
one or more function names. If omitted, all functions will be examined.

where
If where is supplied, the methods definition from that position will be used; otherwise, the current definition is used (which will include inherited methods that have arisen so far in the session). If f is omitted, where controls where to look for generic functions.

classes
If argument classes is supplied, it is a vector of class names that restricts the displayed results to those methods whose signatures include one or more of those classes.

includeDefs
If includeDefs is TRUE, include the definitions of the individual methods in the printout.

inherited
If inherited is TRUE, then methods that have been found by inheritance, so far in the session, will be included and marked as inherited. Note that an inherited method will not usually appear until it has been used in this session. See selectMethod if you want to know what method is dispatched for particular classes of arguments.

showEmpty
logical indicating if methods with empty method lists should be shown at all. Note that FALSE is not yet implemented.

printTo
The connection on which the printed information will be written. If printTo is FALSE, the output will be collected as a character vector and returned as the value of the call to showMethod. See show.

Details

The output style is different from S-Plus in that it does not show the database from which the definition comes, but can optionally include the method definitions.
Value

If `printTo` is `FALSE`, the character vector that would have been printed is returned; otherwise the value is the connection or filename.

References

The R package `methods` implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the `methods` package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.

See Also

`setMethod`, and `GenericFunctions` for other tools involving methods; `selectMethod` will show you the method dispatched for a particular function and signature of classes for the arguments.

Examples

```r
## Assuming the methods for plot
## are set up as in the example of help(setMethod),
## print (without definitions) the methods that involve class "track":
showMethods("plot", classes = "track")
## Not run:
Function "plot":
x = ANY, y = track
x = track, y = missing
x = track, y = ANY
## End(Not run)

## Show all methods from the same place that a class is defined:
not.there <- !any("package:stats4" == search())
if(not.there) library(stats4)
showMethods(class = "mle") # not really helpful
showMethods(class = "mle", where = "package:stats4") # much better
if(not.there) detach("package:stats4")
```

signature-class

Class “signature” For Method Definitions

Description

This class represents the mapping of some of the formal arguments of a function onto the names of some classes. It is used as one of two slots in the `MethodDefinition-class`. 
Objects from the Class

Objects can be created by calls of the form `new("signature", functionDef, ...)`. The `functionDef` argument, if it is supplied as a function object, defines the formal names. The other arguments define the classes.

Slots

- **Data**: Object of class "character" the classes.
- **names**: Object of class "character" the corresponding argument names.

Extends

Class "character", from data part. Class "vector", by class "character".

Methods

- **initialize** signature(object = "signature"): see the discussion of objects from the class, above.

See Also

- `MethodDefinition-class` for the use of this class

---

**slot**

*The Slots in an Object from a Formal Class*

---

Description

These functions return or set information about the individual slots in an object.

Usage

- `object@name`
- `object@name <- value`
- `slot(object, name)`
- `slot(object, name, check = TRUE) <- value`
- `slotNames(x)`

Arguments

- **object** An object from a formally defined class.
- **name** The character-string name of the slot. The name must be a valid slot name: see Details below.
- **value** A new value for the named slot. The value must be valid for this slot in this object’s class.
- **x** Either the name of a class or an object from that class. Print `getClass(class)` to see the full description of the slots.
- **check** If TRUE, check the assigned value for validity as the value of this slot. You should never set this to FALSE in normal use, since the result can create invalid objects.
Details

The "@" operator and the `slot` function extract or replace the formally defined slots for the object. The operator takes a fixed name, which can be unquoted if it is syntactically a name in the language. A slot name can be any non-empty string, but if the name is not made up of letters, numbers, and ".", it needs to be quoted.

In the case of the `slot` function, the slot name can be any expression that evaluates to a valid slot in the class definition. Generally, the only reason to use the functional form rather than the simpler operator is because the slot name has to be computed.

The definition of the class contains the names of all slots directly and indirectly defined. Each slot has a name and an associated class. Extracting a slot returns an object from that class. Setting a slot first coerces the value to the specified slot and then stores it.

Unlike attributes, slots are not partially matched, and asking for (or trying to set) a slot with an invalid name for that class generates an error.

Note that currently, `slotNames()` behaves particularly for class representation objects – this is considered bogus and likely to be changed.

References

The R package `methods` implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the `methods` package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.

See Also

`@`, `Classes`, `Methods`, `getClass`

Examples

```r
setClass("track", representation(x="numeric", y="numeric"))
myTrack <- new("track", x = -4:4, y = exp(-4:4))
slot(myTrack, "x")
slot(myTrack, "y") <- log(slot(myTrack, "y"))
str(myTrack)

slotNames("track") # is the same as
slotNames(myTrack)
```
StructureClasses  Classes Corresponding to Basic Structures

Description

The virtual class `structure` and classes that extend it are formal classes analogous to S language structures such as arrays and time-series.

Usage

```
## The following class names can appear in method signatures, 
## as the class in as() and is() expressions, and, except for 
## the classes commented as VIRTUAL, in calls to new()

"matrix"
"array"
"ts"

"structure"  ## VIRTUAL
```

Objects from the Classes

Objects can be created by calls of the form `new(Class, ...)`, where `Class` is the quoted name of the specific class (e.g., "matrix"), and the other arguments, if any, are interpreted as arguments to the corresponding function, e.g., to function `matrix()`. There is no particular advantage over calling those functions directly, unless you are writing software designed to work for multiple classes, perhaps with the class name and the arguments passed in.

Extends

The specific classes all extend class "structure", directly, and class "vector", by class "structure".

Methods

coerce  Methods are defined to coerce arbitrary objects to these classes, by calling the corresponding basic function, for example, `as(x, "matrix")` calls `as.matrix(x)`.

TraceClasses  Classes Used Internally to Control Tracing

Description

The classes described here are used by the R function `trace` to create versions of functions and methods including browser calls, etc., and also to `untrace` the same objects.
Usage

### Objects from the following classes are generated
### by calling trace() on an object from the corresponding
### class without the "WithTrace" in the name.

"functionWithTrace"
"MethodDefinitionWithTrace"
"MethodWithNextWithTrace"
"genericFunctionWithTrace"
"groupGenericFunctionWithTrace"

### the following is a virtual class extended by each of the
### classes above

"traceable"

Objects from the Class

Objects will be created from these classes by calls to trace. (There is an initialize method
for class "traceable", but you are unlikely to need it directly.)

Slots

.Data: The data part, which will be "function" for class "functionWithTrace", and
similarly for the other classes.

original: Object of the original class; e.g., "function" for class
"functionWithTrace".

Extends

Each of the classes extends the corresponding untraced class, from the data part; e.g.,
"functionWithTrace" extends "function". Each of the specific classes extends
"traceable", directly, and class "VIRTUAL", by class "traceable".

Methods

The point of the specific classes is that objects generated from them, by function trace(). remain
callable or dispatchable, in addition to their new trace information.

See Also

function trace

---

validObject Test the Validity of an Object
Description

The validity of object related to its class definition is tested. If the object is valid, TRUE is returned; otherwise, either a vector of strings describing validity failures is returned, or an error is generated (according to whether test is TRUE).

The function setValidity sets the validity method of a class (but more normally, this method will be supplied as the validity argument to setClass). The method should be a function of one object that returns TRUE or a description of the non-validity.

Usage

validObject(object, test = FALSE)

setValidity(Class, method, where = topenv(parent.frame()) )

Arguments

object Any object, but not much will happen unless the object’s class has a formal definition.

test If test is TRUE, and validity fails the function returns a vector of strings describing the problems. If test is FALSE (the default) validity failure generates an error.

Class the name or class definition of the class whose validity method is to be set.

method a validity method; that is, either NULL or a function of one argument (the object). Like validObject, the function should return TRUE if the object is valid, and one or more descriptive strings if any problems are found. Unlike validObject, it should never generate an error.

where the modified class definition will be stored in this environment.

Note that validity methods do not have to check validity of any slots or superclasses: the logic of validObject ensures these tests are done once only. As a consequence, if one validity method wants to use another, it should extract and call the method from the other definition of the other class by calling getValidity: it should not call validObject.

Details

Validity testing takes place “bottom up”: first the validity of the object’s slots, if any, is tested. Then for each of the classes that this class extends (the “superclasses”), the explicit validity method of that class is called, if one exists. Finally, the validity method of object’s class is called, if there is one.

Testing generally stops at the first stage of finding an error, except that all the slots will be examined even if a slot has failed its validity test.

Value

validObject returns TRUE if the object is valid. Otherwise a vector of strings describing problems found, except that if test is FALSE, validity failure generates an error, with the corresponding strings in the error message.
References

The R package **methods** implements, with a few exceptions, the programming interface for classes and methods in the book *Programming with Data* (John M. Chambers, Springer, 1998), in particular sections 1.6, 2.7, 2.8, and chapters 7 and 8.

While the programming interface for the **methods** package follows the reference, the R software is an original implementation, so details in the reference that reflect the S4 implementation may appear differently in R. Also, there are extensions to the programming interface developed more recently than the reference. For a discussion of details and ongoing development, see the web page [http://developer.r-project.org/methodsPackage.html](http://developer.r-project.org/methodsPackage.html) and the pointers from that page.

See Also

`setClass`.

Examples

```r
setClass("track", representation(x="numeric", y = "numeric"));
t1 <- new("track", x=1:10, y=sort(rnorm(10)));
## A valid "track" object has the same number of x, y values
validTrackObject <- function(x){
  if(length(x@x) == length(x@y)) TRUE
  else paste("Unequal x,y lengths: ", length(x@x), ", ", length(x@y), sep="")
}
## assign the function as the validity method for the class
setValidity("track", validTrackObject);
## t1 should be a valid "track" object
validObject(t1);
## Now we do something bad
validObject(t1)
## This should generate an error
## Not run: try(validObject(t1))
```
validObject
Chapter 7

The stats package

.checkMFClasses Functions to Check the Type of Variables passed to Model Frames

Description

.checkMFClasses checks if the variables used in a predict method agree in type with those used for fitting.

.MFclass categorizes variables for this purpose.

Usage

.checkMFClasses(cl, m, ordNotOK = FALSE)
.MFclass(x)
.getXlevels(Terms, m)

Arguments

cl a character vector of class descriptions to match.
m a model frame.
x any R object.
ordNotOK logical: are ordered factors different?
Terms a terms object.

Details

For applications involving model.matrix such as linear models we do not need to differentiate between ordered factors and factors as although these affect the coding, the coding used in the fit is already recorded and imposed during prediction. However, other applications may treat ordered factors differently: rpart does, for example.

Value

.MFclass returns a character string, one of "logical", "ordered", "factor", "numeric", "nmatrix.*" (a numeric matrix with a number of columns appended) or "other".
.getXlevels returns a named character vector, or NULL.
Description

The function `acf` computes (and by default plots) estimates of the autocovariance or autocorrelation function. Function `pacf` is the function used for the partial autocorrelations. Function `ccf` computes the cross-correlation or cross-covariance of two univariate series.

Usage

```r
acf(x, lag.max = NULL,
    type = c("correlation", "covariance", "partial"),
    plot = TRUE, na.action = na.fail, demean = TRUE, ...)

pacf(x, lag.max, plot, na.action, ...)
```

```
## Default S3 method:
pacf(x, lag.max = NULL, plot = TRUE, na.action = na.fail,
...)
```

```r
ccf(x, y, lag.max = NULL, type = c("correlation", "covariance"),
    plot = TRUE, na.action = na.fail, ...)
```

`acf.obj[i, j]`

Arguments

- `x, y` a univariate or multivariate (not `ccf`) numeric time series object or a numeric vector or matrix.
- `lag.max` maximum number of lags at which to calculate the acf. Default is $10 \log_{10}(N/m)$ where $N$ is the number of observations and $m$ the number of series.
- `type` character string giving the type of acf to be computed. Allowed values are "correlation" (the default), "covariance" or "partial".
- `plot` logical. If TRUE (the default) the acf is plotted.
- `na.action` function to be called to handle missing values. `na.pass` can be used.
- `demean` logical. Should the covariances be about the sample means?
- `...` further arguments to be passed to `plot.acf`.
- `acf.obj` an object of class "acf" resulting from a call to `acf`.
- `i` a set of lags to retain.
- `j` a set of series to retain.

Details

For `type = "correlation"` and "covariance", the estimates are based on the sample covariance.

By default, no missing values are allowed. If the `na.action` function passes through missing values (as `na.pass` does), the covariances are computed from the complete cases. This means
that the estimate computed may well not be a valid autocorrelation sequence, and may contain
missing values. Missing values are not allowed when computing the PACF of a multivariate time
series.

The partial correlation coefficient is estimated by fitting autoregressive models of successively
higher orders up to lag.max.

The generic function plot has a method for objects of class "acf".

The lag is returned and plotted in units of time, and not numbers of observations.

There are print and subsetting methods for objects of class "acf".

Value

An object of class "acf", which is a list with the following elements:

- **lag**: A three dimensional array containing the lags at which the acf is estimated.
- **acf**: An array with the same dimensions as lag containing the estimated acf.
- **type**: The type of correlation (same as the type argument).
- **n.used**: The number of observations in the time series.
- **series**: The name of the series x.
- **snames**: The series names for a multivariate time series.

The result is returned invisibly if plot is TRUE.

Author(s)

Original: Paul Gilbert, Martyn Plummer. Extensive modifications and univariate case of pacf by
B.D. Ripley.

See Also

plot.acf, ARMAacf for the exact autocorrelations of a given ARMA process.

Examples

```r
## Examples from Venables & Ripley
acf(lh)
acf(lh, type = "covariance")
pacf(lh)

acf(ldeaths)
acf(ldeaths, ci.type = "ma")
acf(ts.union(mdeaths, fdeaths))
ccf(mdeaths, fdeaths) # just the cross-correlations.

presidents # contains missing values
acf(presidents, na.action = na.pass)
pacf(presidents, na.action = na.pass)
```
acf2AR  Compute an AR Process Exactly Fitting an ACF

Description

Compute an AR process exactly fitting an autocorrelation function.

Usage

acf2AR(acf)

Arguments

acf  An autocorrelation or autocovariance sequence.

Value

A matrix, with one row for the computed AR(p) coefficients for 1 <= p <= length(acf).

See Also

ARMAacf, ar.yw which does this from an empirical ACF.

Examples

(Acf <- ARMAacf(c(0.6, 0.3, -0.2)))
acf2AR(Acf)

add1  Add or Drop All Possible Single Terms to a Model

Description

Compute all the single terms in the scope argument that can be added to or dropped from the model, fit those models and compute a table of the changes in fit.

Usage

add1(object, scope, ...)

## Default S3 method:
add1(object, scope, scale = 0, test = c("none", "Chisq"),
    k = 2, trace = FALSE, ...)

## S3 method for class 'lm':
add1(object, scope, scale = 0, test = c("none", "Chisq", "F"),
    x = NULL, k = 2, ...)

## S3 method for class 'glm':
add1(object, scope, scale = 0, test = c("none", "Chisq", "F"),
    ...
\texttt{x} = \texttt{NULL}, \ k = 2, \ldots)

\texttt{drop1(object, scope, \ldots)}

## Default S3 method:
drop1(object, scope, scale = 0, test = c("none", "Chisq"),
  k = 2, trace = FALSE, \ldots)

## S3 method for class 'lm':
drop1(object, scope, scale = 0, all.cols = TRUE,
  test = c("none", "Chisq", "F"), k = 2, \ldots)

## S3 method for class 'glm':
drop1(object, scope, scale = 0, test = c("none", "Chisq", "F"),
  k = 2, \ldots)

### Arguments

- **object**: a fitted model object.
- **scope**: a formula giving the terms to be considered for adding or dropping.
- **scale**: an estimate of the residual mean square to be used in computing \( C_p \). Ignored if 0 or \texttt{NULL}.
- **test**: should the results include a test statistic relative to the original model? The F test is only appropriate for \texttt{lm} and \texttt{aov} models or perhaps for \texttt{glm} fits with estimated dispersion. The \( \chi^2 \) test can be an exact test (\texttt{lm} models with known scale) or a likelihood-ratio test or a test of the reduction in scaled deviance depending on the method.
- **k**: the penalty constant in AIC / \( C_p \).
- **trace**: if TRUE, print out progress reports.
- **x**: a model matrix containing columns for the fitted model and all terms in the upper scope. Useful if \texttt{add1} is to be called repeatedly. **Warning**: no checks are done on its validity.
- **all.cols**: (Provided for compatibility with S.) Logical to specify whether all columns of the design matrix should be used. If \texttt{FALSE} then non-estimable columns are dropped, but the result is not usually statistically meaningful.
- ... further arguments passed to or from other methods.

### Details

For \texttt{drop1} methods, a missing \texttt{scope} is taken to be all terms in the model. The hierarchy is respected when considering terms to be added or dropped: all main effects contained in a second-order interaction must remain, and so on.

In a \texttt{scope} formula, \texttt{.} means ‘what is already there’.

The methods for \texttt{lm} and \texttt{glm} are more efficient in that they do not recompute the model matrix and call the \texttt{fit} methods directly.

The default output table gives AIC, defined as minus twice log likelihood plus \( 2p \) where \( p \) is the rank of the model (the number of effective parameters). This is only defined up to an additive constant (like log-likelihoods). For linear Gaussian models with fixed scale, the constant is chosen to give Mallows’ \( C_p \), \( RSS/scale + 2p - n \). Where \( C_p \) is used, the column is labelled as \( Cp \) rather than AIC.
addmargins

Value

An object of class "anova" summarizing the differences in fit between the models.

Warning

The model fitting must apply the models to the same dataset. Most methods will attempt to use a subset of the data with no missing values for any of the variables if `na.action=na.omit`, but this may give biased results. Only use these functions with data containing missing values with great care.

Note

These are not fully equivalent to the functions in S. There is no `keep` argument, and the methods used are not quite so computationally efficient.

Their authors’ definitions of Mallows’ $C_p$ and Akaike’s AIC are used, not those of the authors of the models chapter of S.

Author(s)

The design was inspired by the S functions of the same names described in Chambers (1992).

References


See Also

`step.aov`, `lm`, `extractAIC`, `anova`

Examples

```r
example(step)#-> swiss
add1(lm1, ~ I(Education^2) + .^2)
drop1(lm1, test="F")  # So called 'type II' anova
```

```r
example(glm)
drop1(glm.D93, test="Chisq")
drop1(glm.D93, test="F")
```

---

**addmargins**

*Puts arbitrary margins on multidimensional tables or arrays.*

Description

For a given table one can specify which of the classifying factors to expand by one or more levels to hold margins to be calculated. One may for example form sums and means over the first dimension and medians over the second. The resulting table will then have two extra levels for the first dimension and one extra level for the second. The default is to sum over all margins in the table. Other possibilities may give results that depend on the order in which the margins are computed. This is flagged in the printed output from the function.
Usage

addmargins(A, margin = 1:length(dim(A)), FUN = sum, quiet = FALSE)

Arguments

A A table or array. The function uses the presence of the "dim" and "dimnames" attributes of A
margin Vector of dimensions over which to form margins. Margins are formed in the order in which dimensions are specified in margin.
FUN List of the same length as margin, each element of the list being either a function or a list of functions. Names of the list elements will appear as levels in dimnames of the result. Unnamed list elements will have names constructed: the name of a function or a constructed name based on the position in the table.
quiet Logical which suppresses the message telling the order in which the margins were computed.

Details

If the functions used to form margins are not commutative the result depends on the order in which margins are computed. Annotation of margins is done via naming the FUN list.

Value

A table with the same number of dimensions as A, but with extra levels of the dimensions mentioned in margin. The number of levels added to each dimension is the length of the entries in FUN. A message with the order of computation of margins is printed.

Author(s)


See Also

table, ftable, margin.table.

Examples

Aye <- sample( c("Yes","Si","Oui"), 177, replace=TRUE )
Bee <- sample( c("Hum","Buzz"), 177, replace=TRUE )
Sea <- sample( c("White","Black","Red","Dead"), 177, replace=TRUE )
A <- table( Aye, Bee, Sea )
A
addmargins( A )
ftable( A )
ftable( addmargins( A ) )

# Non commutative functions - note differences between resulting tables:
ftable(addmargins(A, c(1,3),
    FUN = list(Sum=sum, list(Min=min, Max=max)))))
ftable(addmargins(A, c(3,1),
    FUN = list(list(Min=min, Max=max), Sum=sum)))
# Weird function needed to return the N when computing percentages
sqsm <- function(x) sum(x)^2/100
B <- table(Sea, Bee)
round(sweep(addmargins(B, 1, list(list(All=sum, N=sqsm)))), 2,
apply(B, 2, sum)/100, "/" , 1)
round(sweep(addmargins(B, 2, list(list(All=sum, N=sqsm)))), 1,
apply(B, 1, sum)/100, "/" , 1)

# A total over Bee requires formation of the Bee-margin first:
mB <- addmargins(B, 2, FUN=list(list(Total=sum)) )
round(ftable(sweep(addmargins(mB, 1, list(list(All=sum, N=sqsm))), 2,
apply(mB,2,sum)/100, "/" ), 1)

---

**aggregate**

*Compute Summary Statistics of Data Subsets*

**Description**

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

**Usage**

aggregate(x, ...)

## Default S3 method:
aggregate(x, ...)

## S3 method for class 'data.frame':
aggregate(x, by, FUN, ...)

## S3 method for class 'ts':
aggregate(x, nfrequency = 1, FUN = sum, ndeltat = 1,
        ts.eps = getOption("ts.eps"), ...)

**Arguments**

- **x**:
  an R object.

- **by**:
  a list of grouping elements, each as long as the variables in x. Names for the grouping variables are provided if they are not given. The elements of the list will be coerced to factors (if they are not already factors).

- **FUN**:
  a scalar function to compute the summary statistics which can be applied to all data subsets.

- **nfrequency**:
  new number of observations per unit of time; must be a divisor of the frequency of x.

- **ndeltat**:
  new fraction of the sampling period between successive observations; must be a divisor of the sampling interval of x.

- **ts.eps**:
  tolerance used to decide if nfrequency is a sub-multiple of the original frequency.

- **...**:
  further arguments passed to or used by methods.
**aggregate**

**Details**

`aggregate` is a generic function with methods for data frames and time series.

The default method `aggregate.default` uses the time series method if `x` is a time series, and otherwise coerces `x` to a data frame and calls the data frame method.

`aggregate.data.frame` is the data frame method. If `x` is not a data frame, it is coerced to one. Then, each of the variables (columns) in `x` is split into subsets of cases (rows) of identical combinations of the components of `by`, and `FUN` is applied to each such subset with further arguments in ... passed to it. (i.e., `tapply(VAR, by, FUN, ..., simplify = FALSE)` is done for each variable `VAR` in `x`, conveniently wrapped into one call to `lapply()`.) Empty subsets are removed, and the result is reformatted into a data frame containing the variables in `by` and `x`. The ones arising from `by` contain the unique combinations of grouping values used for determining the subsets, and the ones arising from `x` the corresponding summary statistics for the subset of the respective variables in `x`.

`aggregate.ts` is the time series method. If `x` is not a time series, it is coerced to one. Then, the variables in `x` are split into appropriate blocks of length `frequency(x) / nfrequency`, and `FUN` is applied to each such block, with further (named) arguments in ... passed to it. The result returned is a time series with frequency `nfrequency` holding the aggregated values.

**Author(s)**

Kurt Hornik

**References**


**See Also**

`apply`, `lapply`, `tapply`.

**Examples**

```r
## Compute the averages for the variables in 'state.x77', grouped
## according to the region (Northeast, South, North Central, West) that
## each state belongs to.
aggregate(state.x77, list(Region = state.region), mean)

## Compute the averages according to region and the occurrence of more
## than 130 days of frost.
aggregate(state.x77,
  list(Region = state.region,
       Cold = state.x77[,"Frost"] > 130),
  mean)
## (Note that no state in 'South' is THAT cold.)

## Compute the average annual approval ratings for American presidents.
aggregate(presidents, nf = 1, FUN = mean)
## Give the summer less weight.
aggregate(presidents, nf = 1, FUN = weighted.mean, w = c(1, 1, 0.5, 1))
```
**AIC**  
Akaike’s An Information Criterion

**Description**

Generic function calculating the Akaike information criterion for one or several fitted model objects for which a log-likelihood value can be obtained, according to the formula $-2\text{log-likelihood} + k n_{\text{par}}$, where $n_{\text{par}}$ represents the number of parameters in the fitted model, and $k = 2$ for the usual AIC, or $k = \log(n)$ ($n$ the number of observations) for the so-called BIC or SBC (Schwarz’s Bayesian criterion).

**Usage**

AIC(object, ..., k = 2)

**Arguments**

- **object**  
a fitted model object, for which there exists a `logLik` method to extract the corresponding log-likelihood, or an object inheriting from class `logLik`.

- **...**  
optionally more fitted model objects.

- **k**  
umeric, the “penalty” per parameter to be used; the default $k = 2$ is the classical AIC.

**Details**

The default method for `AIC`, `AIC.default()` entirely relies on the existence of a `logLik` method computing the log-likelihood for the given class.

When comparing fitted objects, the smaller the AIC, the better the fit.

**Value**

If just one object is provided, returns a numeric value with the corresponding AIC (or BIC, or ..., depending on $k$); if more than one object are provided, returns a `data.frame` with rows corresponding to the objects and columns representing the number of parameters in the model (`df`) and the AIC.

**Author(s)**

Jose Pinheiro and Douglas Bates

**References**


**See Also**

`extractAIC`, `logLik`. 

---

### AIC

**AIC**  
Akaike’s An Information Criterion

**Description**

Generic function calculating the Akaike information criterion for one or several fitted model objects for which a log-likelihood value can be obtained, according to the formula $-2\text{log-likelihood} + k n_{\text{par}}$, where $n_{\text{par}}$ represents the number of parameters in the fitted model, and $k = 2$ for the usual AIC, or $k = \log(n)$ ($n$ the number of observations) for the so-called BIC or SBC (Schwarz’s Bayesian criterion).

**Usage**

AIC(object, ..., k = 2)

**Arguments**

- **object**  
a fitted model object, for which there exists a `logLik` method to extract the corresponding log-likelihood, or an object inheriting from class `logLik`.

- **...**  
optionally more fitted model objects.

- **k**  
umeric, the “penalty” per parameter to be used; the default $k = 2$ is the classical AIC.

**Details**

The default method for `AIC`, `AIC.default()` entirely relies on the existence of a `logLik` method computing the log-likelihood for the given class.

When comparing fitted objects, the smaller the AIC, the better the fit.

**Value**

If just one object is provided, returns a numeric value with the corresponding AIC (or BIC, or ..., depending on $k$); if more than one object are provided, returns a `data.frame` with rows corresponding to the objects and columns representing the number of parameters in the model (`df`) and the AIC.

**Author(s)**

Jose Pinheiro and Douglas Bates

**References**


**See Also**

`extractAIC`, `logLik`. 

---
Examples

```r
lm1 <- lm(Fertility ~ ., data = swiss)
AIC(lm1)
stopifnot(all.equal(AIC(lm1),
                    AIC(logLik(lm1))))
## a version of BIC or Schwarz' BC :
AIC(lm1, k = log(nrow(swiss)))
```

### alias

Find Aliases (Dependencies) in a Model

**Description**

Find aliases (linearly dependent terms) in a linear model specified by a formula.

**Usage**

```r
alias(object, ...

## S3 method for class 'formula':
alias(object, data, ...)

## S3 method for class 'lm':
alias(object, complete = TRUE, partial = FALSE,
       partial.pattern = FALSE, ...)
```

**Arguments**

- **object**
  A fitted model object, for example from `lm` or `aov`, or a formula for `alias.formula`.
- **data**
  Optionally, a data frame to search for the objects in the formula.
- **complete**
  Should information on complete aliasing be included?
- **partial**
  Should information on partial aliasing be included?
- **partial.pattern**
  Should partial aliasing be presented in a schematic way? If this is done, the results are presented in a more compact way, usually giving the deciles of the coefficients.
- **...**
  Further arguments passed to or from other methods.

**Details**

Although the main method is for class "$\text{lm}"`, `alias` is most useful for experimental designs and so is used with fits from `aov`. Complete aliasing refers to effects in linear models that cannot be estimated independently of the terms which occur earlier in the model and so have their coefficients omitted from the fit. Partial aliasing refers to effects that can be estimated less precisely because of correlations induced by the design.
Value

A list (of class "listof") containing components

- **Model**: Description of the model; usually the formula.
- **Complete**: A matrix with columns corresponding to effects that are linearly dependent on the rows; may be of class "mtable" which has its own print method.
- **Partial**: The correlations of the estimable effects, with a zero diagonal.

Note

The aliasing pattern may depend on the contrasts in use: Helmert contrasts are probably most useful. The defaults are different from those in S.

Author(s)

The design was inspired by the S function of the same name described in Chambers *et al.* (1992).

References


Examples

```r
had.VR <- "package:MASS" %in% search()
## The next line is for fractions() which gives neater results
if(!had.VR) res <- require(MASS)
N <- c(0,1,0,1,1,0,0,0,1,1,0,1,0,1,0,1,0,1,0,1,1,0,0,1)
P <- c(1,1,0,0,0,1,0,1,1,0,0,0,1,0,1,1,0,0,1,1,0,0,1,0)
K <- c(1,0,1,0,1,1,0,0,1,1,0,0,1,0,1,0,1,0,0,1,1,0,0,1)
yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,55.0,
          62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)
npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P),
                  K=factor(K), yield=yield)

op <- options(contrasts=c("contr.helmert", "contr.poly"))
npk.aov <- aov(yield ~ block + N*P*K, npk)
alias(npk.aov)
if(!had.VR && res) detach(package:MASS)
options(op)# reset
```

**anova**

**Anova Tables**

Description

Compute analysis of variance (or deviance) tables for one or more fitted model objects.

Usage

`anova(object, ...)`
anova.glm

Arguments

object an object containing the results returned by a model fitting function (e.g., `lm` or `glm`).
...
additional objects of the same type.

Value

This (generic) function returns an object of class `anova`. These objects represent analysis-of-
variance and analysis-of-deviance tables. When given a single argument it produces a table which
tests whether the model terms are significant.

When given a sequence of objects, `anova` tests the models against one another in the order speci-
fied.

The print method for `anova` objects prints tables in a “pretty” form.

Warning

The comparison between two or more models will only be valid if they are fitted to the same dataset.
This may be a problem if there are missing values and R’s default of `na.action = na.omit` is used.

References


See Also

coefficients, effects, fitted.values, residuals, summary, drop1, add1.

anova.glm Analysis of Deviance for Generalized Linear Model Fits

Description

Compute an analysis of deviance table for one or more generalized linear model fits.

Usage

```r
## S3 method for class 'glm':
anova(object, ..., dispersion = NULL, test = NULL)
```

Arguments

object, ... objects of class `glm`, typically the result of a call to `glm`, or a list of objects
for the "glmlist" method.
dispersion the dispersion parameter for the fitting family. By default it is obtained from
`glm.obj`.
test a character string, (partially) matching one of "Chisq", "F" or "Cp". See
`stat.anova`.
Details

Specifying a single object gives a sequential analysis of deviance table for that fit. That is, the reductions in the residual deviance as each term of the formula is added in turn are given in as the rows of a table, plus the residual deviances themselves.

If more than one object is specified, the table has a row for the residual degrees of freedom and deviance for each model. For all but the first model, the change in degrees of freedom and deviance is also given. (This only makes statistical sense if the models are nested.) It is conventional to list the models from smallest to largest, but this is up to the user.

The table will optionally contain test statistics (and P values) comparing the reduction in deviance for the row to the residuals. For models with known dispersion (e.g., binomial and Poisson fits) the chi-squared test is most appropriate, and for those with dispersion estimated by moments (e.g., gaussian, quasibinomial and quasipoisson fits) the F test is most appropriate. Mallovs’ $C_p$ statistic is the residual deviance plus twice the estimate of $\sigma^2$ times the residual degrees of freedom, which is closely related to AIC (and a multiple of it if the dispersion is known).

Value

An object of class "anova" inheriting from class "data.frame".

Warning

The comparison between two or more models by anova or anova.glmlist will only be valid if they are fitted to the same dataset. This may be a problem if there are missing values and R’s default of na.action = na.omit is used, and anova.glmlist will detect this with an error.

References


See Also

glm, anova.
drop1 for so-called 'type II' anova where each term is dropped one at a time respecting their hierarchy.

Examples

```r
## --- Continuing the Example from '?glm':

anova(glm.D93)
anova(glm.D93, test = "Cp")
anova(glm.D93, test = "Chisq")
```
ANOVA for Linear Model Fits

Description

Compute an analysis of variance table for one or more linear model fits.

Usage

## S3 method for class 'lm':
anova(object, ...)
anova.lmlist(object, ..., scale = 0, test = "F")

Arguments

object, ... objects of class lm, usually, a result of a call to lm.
test a character string specifying the test statistic to be used. Can be one of "F", "Chisq" or "Cp", with partial matching allowed, or NULL for no test.
scale numeric. An estimate of the noise variance \( \sigma^2 \). If zero this will be estimated from the largest model considered.

Details

Specifying a single object gives a sequential analysis of variance table for that fit. That is, the reductions in the residual sum of squares as each term of the formula is added in turn are given in as the rows of a table, plus the residual sum of squares.

The table will contain F statistics (and P values) comparing the mean square for the row to the residual mean square.

If more than one object is specified, the table has a row for the residual degrees of freedom and sum of squares for each model. For all but the first model, the change in degrees of freedom and sum of squares is also given. (This only make statistical sense if the models are nested.) It is conventional to list the models from smallest to largest, but this is up to the user.

Optionally the table can include test statistics. Normally the F statistic is most appropriate, which compares the mean square for a row to the residual sum of squares for the largest model considered. If scale is specified chi-squared tests can be used. Mallows’ \( C_p \) statistic is the residual sum of squares plus twice the estimate of \( \sigma^2 \) times the residual degrees of freedom.

Value

An object of class "anova" inheriting from class "data.frame".

Warning

The comparison between two or more models will only be valid if they are fitted to the same dataset. This may be a problem if there are missing values and R’s default of na.action = na.omit is used, and anova.lmlist will detect this with an error.
Note

Versions of R prior to 1.2.0 based F tests on pairwise comparisons, and this behaviour can still be obtained by a direct call to `anovalist.lm`.

References


See Also

The model fitting function `lm`, `anova`, `drop1` for so-called ‘type II’ anova where each term is dropped one at a time respecting their hierarchy.

Examples

```r
## sequential table
fit <- lm(sr ~ ., data = LifeCycleSavings)
anova(fit)

## same effect via separate models
fit0 <- lm(sr ~ 1, data = LifeCycleSavings)
fit1 <- update(fit0, . ~ . + pop15)
fit2 <- update(fit1, . ~ . + pop75)
fit3 <- update(fit2, . ~ . + dpi)
fit4 <- update(fit3, . ~ . + ddpi)
anova(fit0, fit1, fit2, fit3, fit4, test="F")
anova(fit4, fit2, fit0, test="F") # unconventional order
```

---

### `anova.mlm` Comparisons between Multivariate Linear Models

Compute generalized analysis of variance table for one or more multivariate linear models.

#### Usage

```r
## S3 method for class 'mlm':
anova(object, ...,

   test = c("Pillai", "Wilks", "Hotelling-Lawley", "Roy", "Spherical"),
   Sigma = diag(nrow = p),
   T = Thin.row(proj(M) - proj(X)), M = diag(nrow = p), X = ~0,
   idata = data.frame(index = seq(length=p)))
```
**Arguments**

- `object`: An object of class "mlm".
- `...`: Further objects of class "mlm".
- `test`: Choice of test statistic (see below).
- `Sigma`: (Only relevant if `test`=="Spherical"). Covariance matrix assumed proportional to `Sigma`.
- `M`: Formula or matrix describing the outer projection (see below).
- `X`: Formula or matrix describing the inner projection (see below).
- `idata`: Data frame describing intra-block design.

**Details**

The `anova.mlm` method uses either a multivariate test statistic for the summary table, or a test based on sphericity assumptions (i.e. that the covariance is proportional to a given matrix).

For the multivariate test, Wilks’ statistic is most popular in the literature, but the default Pillai–Bartlett statistic is recommended by Hand and Taylor (1987).

For the "Spherical" test, proportionality is usually with the identity matrix but a different matrix can be specified using `Sigma`). Corrections for asphericity known as the Greenhouse–Geisser, respectively Huynh–Feldt, epsilons are given and adjusted $F$ tests are performed.

It is common to transform the observations prior to testing. This typically involves transformation to intra-block differences, but more complicated within-block designs can be encountered, making more elaborate transformations necessary. A transformation matrix $T$ can be given directly or specified as the difference between two projections onto the spaces spanned by $M$ and $X$, which in turn can be given as matrices or as model formulas with respect to `idata` (the tests will be invariant to parametrization of the quotient space $M/X$).

As with `anova.lm`, all test statistics use the SSD matrix from the largest model considered as the (generalized) denominator.

Contrary to other `anova` methods, the intercept is not excluded from the display in the single-model case. When contrast transformations are involved, it often makes good sense to test for a zero intercept.

**Value**

An object of class "anova" inheriting from class "data.frame"

**Note**

The Huynh–Feldt epsilon differs from that calculated by SAS (as of v. 8.2) except when the DF is equal to the number of observations minus one. This is believed to be a bug in SAS, not in R.

**References**


**See Also**

`summary.manova`
Examples

```r
eexample(SSD) # Brings in the mlmfit and reacttime objects

mlmfit0 <- update(mlmfit, ~0)

### Traditional tests of intrasubj. contrasts
## Using MANOVA techniques on contrasts:
anova(mlmfit, mlmfit0, X=~1)

### Assuming sphericity
anova(mlmfit, mlmfit0, X=~1, test="Spherical")

### tests using intra-subject 3x2 design
idata <- data.frame(deg=gl(3,1,6,labels=c(0,4,8)),
                  noise=gl(2,3,6,labels=c("A","P")))
anova(mlmfit, mlmfit0, X = ~ deg + noise, idata = idata, test = "Spherical")
anova(mlmfit, mlmfit0, M = ~ deg + noise, X = ~ noise, idata = idata, test="Spherical")
anova(mlmfit, mlmfit0, M = ~ deg + noise, X = ~ deg, idata = idata, test="Spherical")

f <- factor(rep(1:2,5)) # bogus, just for illustration
mlmfit2 <- update(mlmfit, ~f)
anova(mlmfit2, mlmfit, mlmfit0, X=~1, test="Spherical")
anova(mlmfit2, X=~1, test="Spherical") # one-model form, equiv. to previous

### There seems to be a strong interaction in these data
plot(colMeans(reacttime))
```

---

`ansari.test`  

Ansari-Bradley Test

Description

Performs the Ansari-Bradley two-sample test for a difference in scale parameters.

Usage

```r
ansari.test(x, ...)  

# Default S3 method:
ansari.test(x, y, alternative = c("two.sided", "less", "greater"),
             exact = NULL, conf.int = FALSE, conf.level = 0.95, ...)

# S3 method for class 'formula':
ansari.test(formula, data, subset, na.action, ...)
```

Arguments

- `x` numeric vector of data values.
- `y` numeric vector of data values.
alternative indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter.

exact a logical indicating whether an exact p-value should be computed.

conf.int a logical, indicating whether a confidence interval should be computed.

conf.level confidence level of the interval.

formula a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups.

data an optional data frame containing the variables in the model formula.

subset an optional vector specifying a subset of observations to be used.

na.action a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").

... further arguments to be passed to or from methods.

Details

Suppose that x and y are independent samples from distributions with densities \( f((t - m)/s)/s \) and \( f(t - m) \), respectively, where \( m \) is an unknown nuisance parameter and \( s \), the ratio of scales, is the parameter of interest. The Ansari-Bradley test is used for testing the null that \( s \) equals 1, the two-sided alternative being that \( s \neq 1 \) (the distributions differ only in variance), and the one-sided alternatives being \( s > 1 \) (the distribution underlying \( x \) has a larger variance, "greater") or \( s < 1 \) ("less").

By default (if exact is not specified), an exact p-value is computed if both samples contain less than 50 finite values and there are no ties. Otherwise, a normal approximation is used.

Optionally, a nonparametric confidence interval and an estimator for \( s \) are computed. If exact p-values are available, an exact confidence interval is obtained by the algorithm described in Bauer (1972), and the Hodges-Lehmann estimator is employed. Otherwise, the returned confidence interval and point estimate are based on normal approximations.

Value

A list with class "htest" containing the following components:

- statistic the value of the Ansari-Bradley test statistic.
- p.value the p-value of the test.
- null.value the ratio of scales \( s \) under the null, 1.
- alternative a character string describing the alternative hypothesis.
- method the string "Ansari-Bradley test".
- data.name a character string giving the names of the data.
- conf.int a confidence interval for the scale parameter. (Only present if argument conf.int = TRUE.)
- estimate an estimate of the ratio of scales. (Only present if argument conf.int = TRUE.)

Note

To compare results of the Ansari-Bradley test to those of the F test to compare two variances (under the assumption of normality), observe that \( s \) is the ratio of scales and hence \( s^2 \) is the ratio of variances (provided they exist), whereas for the F test the ratio of variances itself is the parameter of interest. In particular, confidence intervals are for \( s \) in the Ansari-Bradley test but for \( s^2 \) in the F test.
References


See Also

`fligner.test` for a rank-based (nonparametric) \( k \)-sample test for homogeneity of variances; `mood.test` for another rank-based two-sample test for a difference in scale parameters; `var.test` and `bartlett.test` for parametric tests for the homogeneity in variance.

Examples

```r
## Hollander & Wolfe (1973, p. 86f):
## Serum iron determination using Hyland control sera
ramsay <- c(111, 107, 100, 99, 102, 106, 109, 108, 104, 99,
            101, 96, 97, 102, 107, 113, 116, 113, 110, 98)
jung.parekh <- c(107, 108, 106, 98, 105, 103, 110, 105, 104,
ansari.test(ramsay, jung.parekh)

ansari.test(rnorm(10), rnorm(10, 0, 2), conf.int = TRUE)
```

---

**aov**

*Fit an Analysis of Variance Model*

Description

Fit an analysis of variance model by a call to `lm` for each stratum.

Usage

```r
aov(formula, data = NULL, projections = FALSE, qr = TRUE,
    contrasts = NULL, ...)
```

Arguments

- **formula**: A formula specifying the model.
- **data**: A data frame in which the variables specified in the formula will be found. If missing, the variables are searched for in the standard way.
- **projections**: Logical flag: should the projections be returned?
- **qr**: Logical flag: should the QR decomposition be returned?
- **contrasts**: A list of contrasts to be used for some of the factors in the formula. These are not used for any `Error` term, and supplying contrasts for factors only in the `Error` term will give a warning.
- **...**: Arguments to be passed to `lm`, such as `subset` or `na.action`.
Details

This provides a wrapper to \texttt{lm} for fitting linear models to balanced or unbalanced experimental designs.

The main difference from \texttt{lm} is in the way \texttt{print}, \texttt{summary} and so on handle the fit: this is expressed in the traditional language of the analysis of variance rather than that of linear models.

If the formula contains a single \texttt{Error} term, this is used to specify error strata, and appropriate models are fitted within each error stratum.

The formula can specify multiple responses.

Weights can be specified by a \texttt{weights} argument, but should not be used with an \texttt{Error} term, and are incompletely supported (e.g., not by \texttt{model.tables}).

Value

An object of class \texttt{c("aov", "lm")) or for multiple responses of class \texttt{c("maov", "aov", "mlm", "lm") or for multiple error strata of class "aovlist". There are \texttt{print} and \texttt{summary} methods available for these.

Note

\texttt{aov} is designed for balanced designs, and the results can be hard to interpret without balance: beware that missing values in the response(s) will likely lose the balance. If there are two or more error strata, the methods used are statistically inefficient without balance, and it may be better to use \texttt{lme}.

Balance can be checked with the \texttt{replications} function.

The default ‘contrasts’ in R are not orthogonal contrasts, and \texttt{aov} and its helper functions will work better with such contrasts: see the examples for how to select these.

Author(s)

The design was inspired by the S function of the same name described in Chambers \textit{et al.} (1992).

References


See Also

\texttt{lm}, \texttt{summary.aov}, \texttt{replications}, \texttt{alias}, \texttt{proj}, \texttt{model.tables}, \texttt{TukeyHSD}

Examples

```r
N <- c(0,1,0,1,1,0,0,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,0)
P <- c(1,1,0,0,0,1,0,1,1,1,0,0,0,1,0,1,1,0,0,1,0,1,1,0)
K <- c(1,0,0,1,1,0,0,1,0,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0)
yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,55.0,
          62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)
npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P),
                  K=factor(K), yield=yield)
```
## Set orthogonal contrasts.

```r
op <- options(contrasts=c("contr.helmert", "contr.poly"))
(npk.aov <- aov(yield ~ block + N * P * K, npk))
summary(npk.aov)
coefficients(npk.aov)
```

```r
## to show the effects of re-ordering terms contrast the two fits
aov(yield ~ block + N * P + K, npk)
aov(terms(yield ~ block + N * P + K, keep.order=TRUE), npk)
```

```r
## as a test, not particularly sensible statistically
npk.aovE <- aov(yield ~ N * P * K + Error(block), npk)
npk.aovE
summary(npk.aovE)
```

```r
options(op)# reset to previous
```

---

**approxfun**  
*Interpolation Functions*

### Description

Return a list of points which linearly interpolate given data points, or a function performing the linear (or constant) interpolation.

### Usage

```r
approx (x, y = NULL, xout, method="linear", n=50, yleft, yright, rule = 1, f = 0, ties = mean)
approxfun(x, y = NULL, method="linear", yleft, yright, rule = 1, f = 0, ties = mean)
```

### Arguments

- `x, y` vectors giving the coordinates of the points to be interpolated. Alternatively a single plotting structure can be specified: see `xy.coords`.
- `xout` an optional set of values specifying where interpolation is to take place.
- `method` specifies the interpolation method to be used. Choices are "linear" or "constant".
- `n` If `xout` is not specified, interpolation takes place at `n` equally spaced points spanning the interval `[min(x), max(x)]`.
- `yleft` the value to be returned when input `x` values are less than `min(x)`. The default is defined by the value of `rule` given below.
- `yright` the value to be returned when input `x` values are greater than `max(x)`. The default is defined by the value of `rule` given below.
- `rule` an integer describing how interpolation is to take place outside the interval `[min(x), max(x)]`. If `rule` is 1 then NAs are returned for such points and if it is 2, the value at the closest data extreme is used.
For method="constant" a number between 0 and 1 inclusive, indicating a compromise between left- and right-continuous step functions. If \( y_0 \) and \( y_1 \) are the values to the left and right of the point then the value is \( y_0 * (1-f) + y_1 * f \) so that \( f=0 \) is right-continuous and \( f=1 \) is left-continuous.

Handling of tied \( x \) values. Either a function with a single vector argument returning a single number result or the string "ordered".

The inputs can contain missing values which are deleted, so at least two complete \((x, y)\) pairs are required (for method = "linear", one otherwise). If there are duplicated (tied) \( x \) values and ties is a function it is applied to the \( y \) values for each distinct \( x \) value. Useful functions in this context include \texttt{mean}, \texttt{min}, and \texttt{max}. If ties="ordered" the \( x \) values are assumed to be already ordered. The first \( y \) value will be used for interpolation to the left and the last one for interpolation to the right.

approx returns a list with components \( x \) and \( y \), containing \( n \) coordinates which interpolate the given data points according to the method (and rule) desired.

The function approxfun returns a function performing (linear or constant) interpolation of the given data points. For a given set of \( x \) values, this function will return the corresponding interpolated values. This is often more useful than approx.

References


See Also

\texttt{spline} and \texttt{splinefun} for spline interpolation.

Examples

```r
x <- 1:10
y <- rnorm(10)
par(mfrow = c(2,1))
plot(x, y, main = "approx(.) and approxfun(.)")
points(approx(x, y), col = 2, pch = "+")
points(approx(x, y, method = "constant"), col = 4, pch = "+")

f <- approxfun(x, y)
curve(f(x), 0, 10, col = "green")
points(x, y)
is.function(fc <- approxfun(x, y, method = "const")) # TRUE
curve(fc(x), 0, 10, col = "darkblue", add = TRUE)

## Show treatment of 'ties':

x <- c(2,2:4,4,5,5,7,7,7)
y <- c(1:6, 5:4, 3:1)
approx(x,y, xout=x)$y # warning
(ay <- approx(x,y, xout=x, ties = "ordered")$y)
stopifnot(ay == c(2,2,3,6,6,6,4,1,1,1))
```
ar

Fit Autoregressive Models to Time Series

Description

Fit an autoregressive time series model to the data, by default selecting the complexity by AIC.

Usage

ar(x, aic = TRUE, order.max = NULL, method=c("yule-walker", "burg", "ols", "mle", "yw"), na.action, series, ...)

ar.burg(x, ...)  # Default S3 method:
ar.burg(x, aic = TRUE, order.max = NULL, na.action = na.fail, demean = TRUE, series, var.method = 1, ...)
  ## S3 method for class 'mts':
ar.burg(x, aic = TRUE, order.max = NULL, na.action = na.fail, demean = TRUE, series, var.method = 1, ...)

ar.yw(x, ...)  # Default S3 method:
ar.yw(x, aic = TRUE, order.max = NULL, na.action = na.fail, demean = TRUE, series, ...)
  ## S3 method for class 'mts':
ar.yw(x, aic = TRUE, order.max = NULL, na.action = na.fail, demean = TRUE, series, var.method = 1, ...)

ar.mle(x, aic = TRUE, order.max = NULL, na.action = na.fail, demean = TRUE, series, ...)
  ## S3 method for class 'ar':
predict(object, newdata, n.ahead = 1, se.fit = TRUE, ...)

Arguments

x A univariate or multivariate time series.
aic Logical flag. If TRUE then the Akaike Information Criterion is used to choose the order of the autoregressive model. If FALSE, the model of order order.max is fitted.
order.max Maximum order (or order) of model to fit. Defaults to $10 \log_{10}(N)$ where $N$ is the number of observations except for method="mle" where it is the minimum of this quantity and 12.
method  Character string giving the method used to fit the model. Must be one of the strings in the default argument (the first few characters are sufficient). Defaults to "yule-walker".

na.action  function to be called to handle missing values.

demean  should a mean be estimated during fitting?

series  names for the series. Defaults to `deparse(substitute(x))`.

var.method  the method to estimate the innovations variance (see Details).

...  additional arguments for specific methods.

object  a fit from `ar`.

newdata  data to which to apply the prediction.

n.ahead  number of steps ahead at which to predict.

se.fit  logical: return estimated standard errors of the prediction error?

Details

For definiteness, note that the AR coefficients have the sign in

\[ x_t - \mu = a_1(x_{t-1} - \mu) + \cdots + a_p(x_{t-p} - \mu) + e_t \]

\( ar \) is just a wrapper for the functions `ar.yw`, `ar.burg`, `ar.ols` and `ar.mle`.

Order selection is done by AIC if `aic` is true. This is problematic, as of the methods here only `ar.mle` performs true maximum likelihood estimation. The AIC is computed as if the variance estimate were the MLE, omitting the determinant term from the likelihood. Note that this is not the same as the Gaussian likelihood evaluated at the estimated parameter values. In `ar.yw` the variance matrix of the innovations is computed from the fitted coefficients and the autocovariance of \( x \).

`ar.burg` allows two methods to estimate the innovations variance and hence AIC. Method 1 is to use the update given by the Levinson-Durbin recursion (Brockwell and Davis, 1991, (8.2.6) on page 242), and follows S-PLUS. Method 2 is the mean of the sum of squares of the forward and backward prediction errors (as in Brockwell and Davis, 1996, page 145). Percival and Walden (1998) discuss both. In the multivariate case the estimated coefficients will depend (slightly) on the variance estimation method.

Remember that `ar` includes by default a constant in the model, by removing the overall mean of \( x \) before fitting the AR model, or (`ar.mle`) estimating a constant to subtract.

Value

For `ar` and its methods a list of class "ar" with the following elements:

- `order`  The order of the fitted model. This is chosen by minimizing the AIC if `aic=TRUE`, otherwise it is `order.max`.
- `ar`  Estimated autoregression coefficients for the fitted model.
- `var.pred`  The prediction variance: an estimate of the portion of the variance of the time series that is not explained by the autoregressive model.
- `x.mean`  The estimated mean of the series used in fitting and for use in prediction.
- `x.intercept`  (`ar.ols` only.) The intercept in the model for \( x - x.\text{mean} \).
- `aic`  The value of the `aic` argument.
n.used The number of observations in the time series.
order.max The value of the order.max argument.
partialacf The estimate of the partial autocorrelation function up to lag order.max.
resid residuals from the fitted model, conditioning on the first order observations. The first order residuals are set to NA. If x is a time series, so is resid.
method The value of the method argument.
series The name(s) of the time series.
frequency The frequency of the time series.
call The matched call.
asy.var.coef (univariate case, order > 0.) The asymptotic-theory variance matrix of the coefficient estimates.

For predict.ar, a time series of predictions, or if se.fit = TRUE, a list with components pred, the predictions, and se, the estimated standard errors. Both components are time series.

Note
Only the univariate case of ar.mle is implemented.
Fitting by method="mle" to long series can be very slow.

Author(s)
Martyn Plummer. Univariate case of ar.yw, ar.mle and C code for univariate case of ar.burg by B. D. Ripley.

References

See Also

*ar.ols*, *arima0* for ARMA models.

Examples

```
ar(lh)
ar(lh, method="burg")
ar(lh, method="ols")
ar(lh, FALSE, 4) # fit ar(4)

(sunspot.ar <- ar(sunspot.year))
predict(sunspot.ar, n.ahead=25)
## try the other methods too
```
## Burg is quite different here, as is OLS (see ar.ols)
ar(ts.union(BJsales, BJsales.lead), method="burg")

---

**ar.ols**

*Fit Autoregressive Models to Time Series by OLS*

**Description**

Fit an autoregressive time series model to the data by ordinary least squares, by default selecting the complexity by AIC.

**Usage**

```r
ar.ols(x, aic = TRUE, order.max = NULL, na.action = na.fail,
       demean = TRUE, intercept = demean, series, ...)
```

**Arguments**

- `x` A univariate or multivariate time series.
- `aic` Logical flag. If `TRUE` then the Akaike Information Criterion is used to choose the order of the autoregressive model. If `FALSE`, the model of order `order.max` is fitted.
- `order.max` Maximum order (or order) of model to fit. Defaults to $10 \log_{10}(N)$ where $N$ is the number of observations.
- `na.action` function to be called to handle missing values.
- `demean` should the AR model be for $x$ minus its mean?
- `intercept` should a separate intercept term be fitted?
- `series` names for the series. Defaults to `deparse(substitute(x))`.
- `...` further arguments to be passed to or from methods.

**Details**

`ar.ols` fits the general AR model to a possibly non-stationary and/or multivariate system of series $x$. The resulting unconstrained least squares estimates are consistent, even if some of the series are non-stationary and/or co-integrated. For definiteness, note that the AR coefficients have the sign in

$$x_t - \mu = a_0 + a_1(x_{t-1} - \mu) + \cdots + a_p(x_{t-p} - \mu) + \epsilon_t$$

where $a_0$ is zero unless `intercept` is true, and $\mu$ is the sample mean if `demean` is true, zero otherwise.

Order selection is done by AIC if `aic` is true. This is problematic, as `ar.ols` does not perform true maximum likelihood estimation. The AIC is computed as if the variance estimate (computed from the variance matrix of the residuals) were the MLE, omitting the determinant term from the likelihood. Note that this is not the same as the Gaussian likelihood evaluated at the estimated parameter values.

Some care is needed if `intercept` is true and `demean` is false. Only use this if the series are roughly centred on zero. Otherwise the computations may be inaccurate or fail entirely.
Value

A list of class "ar" with the following elements:

- **order**: The order of the fitted model. This is chosen by minimizing the AIC if `aic=TRUE`, otherwise it is `order.max`.
- **ar**: Estimated autoregression coefficients for the fitted model.
- **var.pred**: The prediction variance: an estimate of the portion of the variance of the time series that is not explained by the autoregressive model.
- **x.mean**: The estimated mean (or zero if `demean` is false) of the series used in fitting and for use in prediction.
- **x.intercept**: The intercept in the model for \( x - x.mean \), or zero if `intercept` is false.
- **aic**: The value of the `aic` argument.
- **n.used**: The number of observations in the time series.
- **order.max**: The value of the `order.max` argument.
- **partialacf**: NULL. For compatibility with `ar`.
- **resid**: residuals from the fitted model, conditioning on the first `order` observations. The first `order` residuals are set to NA. If `x` is a time series, so is `resid`.
- **method**: The character string "Unconstrained LS".
- **series**: The name(s) of the time series.
- **frequency**: The frequency of the time series.
- **call**: The matched call.
- **asy.se.coef**: The asymptotic-theory standard errors of the coefficient estimates.

Author(s)

Adrian Trapletti, Brian Ripley.

References


See Also

- `ar`

Examples

```r
ar(lh, method="burg")
ar.ols(lh)
ar.ols(lh, FALSE, 4) # fit ar(4)
ar.ols(ts.union(BJsales, BJsales.lead))
x <- diff(log(EuStockMarkets))
ar.ols(x, order.max=6, demean=FALSE, intercept=TRUE)
```
Fit an ARIMA model to a univariate time series.

Usage

arima(x, order = c(0, 0, 0),
    seasonal = list(order = c(0, 0, 0), period = NA),
    xreg = NULL, include.mean = TRUE, transform.pars = TRUE,
    fixed = NULL, init = NULL, method = c("CSS-ML", "ML", "CSS"),
    n.cond, optim.control = list(), kappa = 1e6)

Arguments

x
a univariate time series

order
A specification of the non-seasonal part of the ARIMA model: the three components \((p, d, q)\) are the AR order, the degree of differencing, and the MA order.

seasonal
A specification of the seasonal part of the ARIMA model, plus the period (which defaults to `frequency(x)`). This should be a list with components `order` and `period`, but a specification of just a numeric vector of length 3 will be turned into a suitable list with the specification as the `order`.

xreg
Optionally, a vector or matrix of external regressors, which must have the same number of rows as `x`.

include.mean
Should the ARIMA model include a mean term? The default is `TRUE` for undifferenced series, `FALSE` for differenced ones (where a mean would not affect the fit nor predictions).

transform.pars
Logical. If true, the AR parameters are transformed to ensure that they remain in the region of stationarity. Not used for `method = "CSS"`.

fixed
optional numeric vector of the same length as the total number of parameters. If supplied, only NA entries in `fixed` will be varied. `transform.pars = TRUE` will be overridden (with a warning) if any AR parameters are fixed. It may be wise to set `transform.pars = FALSE` when fixing MA parameters, especially near non-invertibility.

init
optional numeric vector of initial parameter values. Missing values will be filled in, by zeroes except for regression coefficients. Values already specified in `fixed` will be ignored.

method
Fitting method: maximum likelihood or minimize conditional sum-of-squares. The default (unless there are missing values) is to use conditional-sum-of-squares to find starting values, then maximum likelihood.

n.cond
Only used if fitting by conditional-sum-of-squares: the number of initial observations to ignore. It will be ignored if less than the maximum lag of an AR term.

optim.control
List of control parameters for `optim`.

kappa
the prior variance (as a multiple of the innovations variance) for the past observations in a differenced model. Do not reduce this.
Details

Different definitions of ARMA models have different signs for the AR and/or MA coefficients. The definition here has

\[ X_t = a_1 X_{t-1} + \cdots + a_p X_{t-p} + \epsilon_t + b_1 \epsilon_{t-1} + \cdots + b_q \epsilon_{t-q} \]

and so the MA coefficients differ in sign from those of S-PLUS. Further, if `include.mean` is true, this formula applies to \( X - m \) rather than \( X \). For ARIMA models with differencing, the differenced series follows a zero-mean ARMA model. If a `xreg` term is included, a linear regression (with a constant term if `include.mean` is true) is fitted with an ARMA model for the error term.

The variance matrix of the estimates is found from the Hessian of the log-likelihood, and so may only be a rough guide.

Optimization is done by `optim`. It will work best if the columns in `xreg` are roughly scaled to zero mean and unit variance, but does attempt to estimate suitable scalings.

Value

A list of class "Arima" with components:

- `coef` a vector of AR, MA and regression coefficients, which can be extracted by the `coef` method.
- `sigma2` the MLE of the innovations variance.
- `var.coef` the estimated variance matrix of the coefficients `coef`, which can be extracted by the `vcov` method.
- `loglik` the maximized log-likelihood (of the differenced data), or the approximation to it used.
- `arma` A compact form of the specification, as a vector giving the number of AR, MA, seasonal AR and seasonal MA coefficients, plus the period and the number of non-seasonal and seasonal differences.
- `aic` the AIC value corresponding to the log-likelihood. Only valid for `method = "ML"` fits.
- `residuals` the fitted innovations.
- `call` the matched call.
- `series` the name of the series `x`.
- `code` the convergence value returned by `optim`.
- `n.cond` the number of initial observations not used in the fitting.
- `model` A list representing the Kalman Filter used in the fitting. See `KalmanLike`.

Fitting methods

The exact likelihood is computed via a state-space representation of the ARIMA process, and the innovations and their variance found by a Kalman filter. The initialization of the differenced ARMA process uses stationarity and is based on Gardner et al. (1980). For a differenced process the non-stationary components are given a diffuse prior (controlled by `kappa`). Observations which are still controlled by the diffuse prior (determined by having a Kalman gain of at least \( 1e4 \)) are excluded from the likelihood calculations. (This gives comparable results to `arima0` in the absence of missing values, when the observations excluded are precisely those dropped by the differencing.)

Missing values are allowed, and are handled exactly in method "ML".
If `transform.pars` is true, the optimization is done using an alternative parametrization which is a variation on that suggested by Jones (1980) and ensures that the model is stationary. For an AR(p) model the parametrization is via the inverse tanh of the partial autocorrelations: the same procedure is applied (separately) to the AR and seasonal AR terms. The MA terms are not constrained to be invertible during optimization, but they will be converted to invertible form after optimization if `transform.pars` is true.

Conditional sum-of-squares is provided mainly for expositional purposes. This computes the sum of squares of the fitted innovations from observation `n.cond` on, (where `n.cond` is at least the maximum lag of an AR term), treating all earlier innovations to be zero. Argument `n.cond` can be used to allow comparability between different fits. The “part log-likelihood” is the first term, half the log of the estimated mean square. Missing values are allowed, but will cause many of the innovations to be missing.

When regressors are specified, they are orthogonalized prior to fitting unless any of the coefficients is fixed. It can be helpful to roughly scale the regressors to zero mean and unit variance.

**Note**

The results are likely to be different from S-PLUS’s `arima.mle`, which computes a conditional likelihood and does not include a mean in the model. Further, the convention used by `arima.mle` reverses the signs of the MA coefficients.

`arima` is very similar to `arima0` for ARMA models or for differenced models without missing values, but handles differenced models with missing values exactly. It is somewhat slower than `arima0`, particularly for seasonally differenced models.

**References**


**See Also**

`predict.Arima`, `arima.sim` for simulating from an ARIMA model, `tsdiag`, `arima0`, `ar`

**Examples**

```r
arima(lh, order = c(1,0,0))
arima(lh, order = c(3,0,0))
arima(lh, order = c(1,0,1))
arima(lh, order = c(3,0,0), method = "CSS")
arima(USAccDeaths, order = c(0,1,1), seasonal = list(order=c(0,1,1)))
arima(USAccDeaths, order = c(0,1,1), seasonal = list(order=c(0,1,1)),
      method = "CSS") # drops first 13 observations.
```
# for a model with as few years as this, we want full ML
arima(LakeHuron, order = c(2,0,0), xreg = time(LakeHuron)-1920)

## presidents contains NAs
## graphs in example(acf) suggest order 1 or 3
(fit1 <- arima(presidents, c(1, 0, 0)))
tsdig(fit1)
(fit3 <- arima(presidents, c(3, 0, 0))) # smaller AIC
tsdig(fit3)

---

arima.sim

Simulate from an ARIMA Model

**Description**
Simulate from an ARIMA model.

**Usage**

arima.sim(model, n, rand.gen = rnorm, innov = rand.gen(n, ...),
n.start = NA, ...)

**Arguments**

- **model**: A list with component `ar` and/or `ma` giving the AR and MA coefficients respectively. Optionally a component `order` can be used. An empty list gives an ARIMA(0, 0, 0) model, that is white noise.
- **n**: length of output series, before un-differencing.
- **rand.gen**: optional: a function to generate the innovations.
- **innov**: an optional times series of innovations. If not provided, `rand.gen` is used.
- **n.start**: length of “burn-in” period. If NA, the default, a reasonable value is computed.
- **...**: additional arguments for `rand.gen`. Most usefully, the standard deviation of the innovations generated by `rnorm` can be specified by `sd`.

**Details**

See `arima` for the precise definition of an ARIMA model.

The ARMA model is checked for stationarity.

ARIMA models are specified via the `order` component of `model`, in the same way as for `arima`. Other aspects of the `order` component are ignored, but inconsistent specifications of the MA and AR orders are detected. The un-differencing assumes previous values of zero, and to remind the user of this, those values are returned.

Random inputs for the “burn-in” period are generated by calling `rand.gen`.

**Value**

A time-series object of class "ts".
See Also

arima

Examples

arima.sim(n = 63, list(ar = c(0.8897, -0.4858), ma = c(-0.2279, 0.2488)),
          sd = sqrt(0.1796))
# mildly long-tailed
arima.sim(n = 63, list(ar=c(0.8897, -0.4858), ma=c(-0.2279, 0.2488)),
          rand.gen = function(n, ...) sqrt(0.1796) * rt(n, df = 5))

# An ARIMA simulation
ts.sim <- arima.sim(list(order = c(1,1,0), ar = 0.7), n = 200)
ts.plot(ts.sim)

Description

Fit an ARIMA model to a univariate time series, and forecast from the fitted model.

Usage

arima0(x, order = c(0, 0, 0),
      seasonal = list(order = c(0, 0, 0), period = NA),
      xreg = NULL, include.mean = TRUE, delta = 0.01,
      transform.pars = TRUE, fixed = NULL, init = NULL,
      method = c("ML", "CSS"), n.cond, optim.control = list())

## S3 method for class 'arima0':
predict(object, n.ahead = 1, newxreg, se.fit = TRUE, ...)

Arguments

x
  a univariate time series

order
  A specification of the non-seasonal part of the ARIMA model: the three components \((p, d, q)\) are the AR order, the degree of differencing, and the MA order.

seasonal
  A specification of the seasonal part of the ARIMA model, plus the period (which defaults to \texttt{frequency(x)})). This should be a list with components \texttt{order} and \texttt{period}, but a specification of just a numeric vector of length 3 will be turned into a suitable list with the specification as the \texttt{order}.

xreg
  Optionally, a vector or matrix of external regressors, which must have the same number of rows as \texttt{x}.

include.mean
  Should the ARIMA model include a mean term? The default is \texttt{TRUE} for undifferenced series, \texttt{FALSE} for differenced ones (where a mean would not affect the fit nor predictions).

delta
  A value to indicate at which point ‘fast recursions’ should be used. See the Details section.
transform.pars
Logical. If true, the AR parameters are transformed to ensure that they remain
in the region of stationarity. Not used for method = "CSS".

fixed
optional numeric vector of the same length as the total number of parameters.
If supplied, only NA entries in fixed will be varied. transform.pars = TRUE will be overridden (with a warning) if any ARMA parameters are fixed.

init
optional numeric vector of initial parameter values. Missing values will be filled
in, by zeroes except for regression coefficients. Values already specified in
fixed will be ignored.

method
Fitting method: maximum likelihood or minimize conditional sum-of-squares.

n.cond
Only used if fitting by conditional-sum-of-squares: the number of initial obser-
vations to ignore. It will be ignored if less than the maximum lag of an AR
term.

optim.control
List of control parameters for optim.

object
The result of an arima0 fit.

newxreg
New values of xreg to be used for prediction. Must have at least n.ahead
rows.

n.ahead
The number of steps ahead for which prediction is required.

se.fit
Logical: should standard errors of prediction be returned?

... arguments passed to or from other methods.

Details
Different definitions of ARMA models have different signs for the AR and/or MA coefficients. The
definition here has

\[ X_t = a_1 X_{t-1} + \cdots + a_p X_{t-p} + \epsilon_t + b_1 \epsilon_{t-1} + \cdots + b_q \epsilon_{t-q} \]

and so the MA coefficients differ in sign from those of S-PLUS. Further, if include.mean is
ture, this formula applies to \( X - m \) rather than \( X \). For ARIMA models with differencing, the
differenced series follows a zero-mean ARMA model.

The variance matrix of the estimates is found from the Hessian of the log-likelihood, and so may
only be a rough guide, especially for fits close to the boundary of invertibility.

Optimization is done by optim. It will work best if the columns in xreg are roughly scaled to
zero mean and unit variance, but does attempt to estimate suitable scalings.

Finite-history prediction is used. This is only statistically efficient if the MA part of the fit is
invertible, so predict.arima0 will give a warning for non-invertible MA models.

Value
For arima0, a list of class "arima0" with components:

coef a vector of AR, MA and regression coefficients,
sigma2 the MLE of the innovations variance.
var.coef the estimated variance matrix of the coefficients coef.
loglik the maximized log-likelihood (of the differenced data), or the approximation to
it used.
arima0

arma  A compact form of the specification, as a vector giving the number of AR, MA, seasonal AR and seasonal MA coefficients, plus the period and the number of non-seasonal and seasonal differences.

aic  the AIC value corresponding to the log-likelihood. Only valid for method = "ML" fits.

residuals  the fitted innovations.

call  the matched call.

series  the name of the series x.

convergence  the value returned by optim.

n.cond  the number of initial observations not used in the fitting.

For predict.arima0, a time series of predictions, or if se.fit = TRUE, a list with components pred, the predictions, and se, the estimated standard errors. Both components are time series.

Fitting methods

The exact likelihood is computed via a state-space representation of the ARMA process, and the innovations and their variance found by a Kalman filter based on Gardner et al. (1980). This has the option to switch to ‘fast recursions’ (assume an effectively infinite past) if the innovations variance is close enough to its asymptotic bound. The argument delta sets the tolerance: at its default value the approximation is normally negligible and the speed-up considerable. Exact computations can be ensured by setting delta to a negative value.

If transform.pars is true, the optimization is done using an alternative parametrization which is a variation on that suggested by Jones (1980) and ensures that the model is stationary. For an AR(p) model the parametrization is via the inverse tanh of the partial autocorrelations: the same procedure is applied (separately) to the AR and seasonal AR terms. The MA terms are also constrained to be invertible during optimization by the same transformation if transform.pars is true. Note that the MLE for MA terms does sometimes occur for MA polynomials with unit roots: such models can be fitted by using transform.pars = FALSE and specifying a good set of initial values (often obtainable from a fit with transform.pars = TRUE).

As from R 1.5.0 missing values are allowed, but any missing values will force delta to be ignored and full recursions used. Note that missing values will be propogated by differencing, so the procedure used in this function is not fully efficient in that case.

Conditional sum-of-squares is provided mainly for expositional purposes. This computes the sum of squares of the fitted innovations from observation n.cond on, (where n.cond is at least the maximum lag of an AR term), treating all earlier innovations to be zero. Argument n.cond can be used to allow comparability between different fits. The “part log-likelihood” is the first term, half the log of the estimated mean square. Missing values are allowed, but will cause many of the innovations to be missing.

When regressors are specified, they are orthogonalized prior to fitting unless any of the coefficients is fixed. It can be helpful to roughly scale the regressors to zero mean and unit variance.

Note

This is a preliminary version, and will be replaced by arima.

The standard errors of prediction exclude the uncertainty in the estimation of the ARMA model and the regression coefficients.

The results are likely to be different from S-PLUS’s arima.mle, which computes a conditional likelihood and does not include a mean in the model. Further, the convention used by arima.mle reverses the signs of the MA coefficients.
References


See Also

arima, ar, tsdiag

Examples

```r
## Not run: arima0(lh, order = c(1,0,0))
arima0(lh, order = c(3,0,0))
arima0(lh, order = c(1,0,1))
predict(arima0(lh, order = c(3,0,0)), n.ahead = 12)

arima0(lh, order = c(3,0,0), method = "CSS")

# for a model with as few years as this, we want full ML
(fit <- arima0(USAccDeaths, order = c(0,1,1),
              seasonal = list(order=c(0,1,1)), delta = -1))
predict(fit, n.ahead = 6)

arima0(LakeHuron, order = c(2,0,0), xreg = time(LakeHuron)-1920)
## Not run:
## presidents contains NAs
## graphs in example(acf) suggest order 1 or 3
(fit1 <- arima0(presidents, c(1, 0, 0), delta = -1))  # avoid warning
tsdiag(fit1)
(fit3 <- arima0(presidents, c(3, 0, 0), delta = -1))  # smaller AIC
tsdiag(fit3)
## End(Not run)
```

---

**ARMAacf**  
*Compute Theoretical ACF for an ARMA Process*

**Description**

Compute the theoretical autocorrelation function or partial autocorrelation function for an ARMA process.

**Usage**

```r
ARMAacf(ar = numeric(0), ma = numeric(0), lag.max = r, pacf = FALSE)
```
ARMAtoMA

Convert ARMA Process to Infinite MA Process

Description
Convert ARMA process to infinite MA process.

Usage
ARMAtoMA(ar = numeric(0), ma = numeric(0), lag.max)
as.hclust

Convert Objects to Class hclust

Description

Converts objects from other hierarchical clustering functions to class "hclust".

Usage

as.hclust(x, ...)

Arguments

x Hierarchical clustering object

... further arguments passed to or from other methods.

Details

Currently there is only support for converting objects of class "twins" as produced by the functions diana and agnes from the package cluster. The default method throws an error unless passed an "hclust" object.

Value

An object of class "hclust".
See Also

hclust, diana, agnes

Examples

```r
x <- matrix(rnorm(30), ncol=3)
hc <- hclust(dist(x), method="complete")

if(require(cluster, quietly=TRUE)) {# is a recommended package
  ag <- agnes(x, method="complete")
  hcag <- as.hclust(ag)
  ## The dendrograms order slightly differently:
  op <- par(mfrow=c(1,2))
  plot(hc); mtext("hclust", side=1)
  plot(hcag); mtext("agnes", side=1)
}
```
ave

Group Averages Over Level Combinations of Factors

Description

Subsets of \( x[] \) are averaged, where each subset consist of those observations with the same factor levels.

Usage

\[
\text{ave}(x, ..., \text{FUN} = \text{mean})
\]

Arguments

- **x**
  - A numeric.
- **...**
  - Grouping variables, typically factors, all of the same length as \( x \).
- **FUN**
  - Function to apply for each factor level combination.

Value

A numeric vector, say \( y \) of length \( \text{length}(x) \). If \(...\) is \( g1,g2 \), e.g., \( y[i] \) is equal to \( \text{FUN}(x[j], \text{for all } j \text{ with } g1[j] == g1[i] \text{ and } g2[j] == g2[i]) \).

See Also

- mean, median.

Examples

\[
\text{ave}(1:3) \# \text{no grouping } \rightarrow \text{grand mean}
\]

\[
\text{attach(warpbreaks)}
\]

\[
\text{ave(breaks, wool)}
\]

\[
\text{ave(breaks, tension)}
\]

\[
\text{ave(breaks, tension, } \text{FUN} = \text{function(x)mean(x, trim=.1)}
\]

\[
\text{plot(breaks, main =}
\]

"\text{ave(Warpbreaks) for wool } \times \text{ tension combinations}")

\[
\text{lines(ave(breaks, wool, tension ), type='s', col = "blue")}
\]

\[
\text{lines(ave(breaks, wool, tension, FUN=median), type='s', col = "green")}
\]

\[
\text{legend(40,70, c("mean","median"), lty=1,col=c("blue","green"), bg="gray90")}
\]

\[
\text{detach()}
\]
Bandwidth Selectors for Kernel Density Estimation

Description
Bandwidth selectors for gaussian windows in `density`.

Usage
```r
bw.nrd0(x)
bw.nrd(x)
bw.ucv(x, nb = 1000, lower = 0.1 * hmax, upper = hmax)
bw.bcv(x, nb = 1000, lower = 0.1 * hmax, upper = hmax)
bw.SJ(x, nb = 1000, lower = 0.1 * hmax, upper = hmax, method = c("ste", "dpi"))
```

Arguments
- `x`: A data vector.
- `nb`: number of bins to use.
- `lower`, `upper`: Range over which to minimize. The default is almost always satisfactory. `hmax` is calculated internally from a normal reference bandwidth.
- `method`: Either "ste" ("solve-the-equation") or "dpi" ("direct plug-in").

Details
`bw.nrd0` implements a rule-of-thumb for choosing the bandwidth of a Gaussian kernel density estimator. It defaults to 0.9 times the minimum of the standard deviation and the interquartile range divided by 1.34 times the sample size to the negative one-fifth power (= Silverman’s “rule of thumb”, Silverman (1986, page 48, eqn (3.31)) unless the quartiles coincide when a positive result will be guaranteed.

`bw.nrd` is the more common variation given by Scott (1992), using factor 1.06.

`bw.ucv` and `bw.bcv` implement unbiased and biased cross-validation respectively.

`bw.SJ` implements the methods of Sheather & Jones (1991) to select the bandwidth using pilot estimation of derivatives.

Value
A bandwidth on a scale suitable for the `bw` argument of `density`.

References
See Also

density.

bandwidth.nrd, ucv, bcv and width.SJ in package MASS, which are all scaled to the width argument of density and so give answers four times as large.

Examples

plot(density(precip, n = 1000))
rug(precip)
lines(density(precip, bw="nrd"), col = 2)
lines(density(precip, bw="ucv"), col = 3)
lines(density(precip, bw="bcv"), col = 4)
lines(density(precip, bw="SJ-ste"), col = 5)
lines(density(precip, bw="SJ-dpi"), col = 6)
legend(55, 0.035,
  legend = c("nrd", "nrd", "ucv", "bcv", "SJ-ste", "SJ-dpi"),
  col = 1:6, lty = 1)

bartlett.test

Bartlett Test of Homogeneity of Variances

Description

Performs Bartlett’s test of the null that the variances in each of the groups (samples) are the same.

Usage

bartlett.test(x, ...)

## Default S3 method:
bartlett.test(x, g, ...)

## S3 method for class 'formula':
bartlett.test(formula, data, subset, na.action, ...)

Arguments

x
  a numeric vector of data values, or a list of numeric data vectors representing the respective samples, or fitted linear model objects (inheriting from class "lm").

g
  a vector or factor object giving the group for the corresponding elements of x.
  Ignored if x is a list.

formula
  a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.

data
  an optional data frame containing the variables in the model formula.

subset
  an optional vector specifying a subset of observations to be used.

na.action
  a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").

...  
  further arguments to be passed to or from methods.
**Details**

If \( x \) is a list, its elements are taken as the samples or fitted linear models to be compared for homogeneity of variances. In this case, the elements must either all be numeric data vectors or fitted linear model objects, \( g \) is ignored, and one can simply use `bartlett.test(x)` to perform the test. If the samples are not yet contained in a list, use `bartlett.test(list(x, ...))`. Otherwise, \( x \) must be a numeric data vector, and \( g \) must be a vector or factor object of the same length as \( x \) giving the group for the corresponding elements of \( x \).

**Value**

A list of class "htest" containing the following components:

- `statistic` : Bartlett's K-squared test statistic.
- `parameter` : the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- `p.value` : the p-value of the test.
- `method` : the character string "Bartlett test of homogeneity of variances".
- `data.name` : a character string giving the names of the data.

**References**


**See Also**

- `var.test` for the special case of comparing variances in two samples from normal distributions;
- `fligner.test` for a rank-based (nonparametric) \( k \)-sample test for homogeneity of variances;
- `ansari.test` and `mood.test` for two rank based two-sample tests for difference in scale.

**Examples**

```r
plot(count ~ spray, data = InsectSprays)
bartlett.test(InsectSprays$count, InsectSprays$spray)
bartlett.test(count ~ spray, data = InsectSprays)
```

---

**Beta**

*The Beta Distribution*

**Description**

Density, distribution function, quantile function and random generation for the Beta distribution with parameters `shape1` and `shape2` (and optional non-centrality parameter `ncp`).

**Usage**

```r
dbeta(x, shape1, shape2, ncp=0, log = FALSE)
pbeta(q, shape1, shape2, ncp=0, lower.tail = TRUE, log.p = FALSE)
qbeta(p, shape1, shape2, lower.tail = TRUE, log.p = FALSE)
rbeta(n, shape1, shape2)
```
Arguments

x, q  vector of quantiles.

p  vector of probabilities.

n  number of observations. If length(n) > 1, the length is taken to be the number required.

shape1, shape2  positive parameters of the Beta distribution.

ncp  non-centrality parameter.

log, log.p  logical; if TRUE, probabilities p are given as log(p).

lower.tail  logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

The Beta distribution with parameters \( \text{shape1} = a \) and \( \text{shape2} = b \) has density

\[
 f(x) = \frac{\Gamma(a + b)}{\Gamma(a)\Gamma(b)} x^a (1 - x)^b
\]

for \( a > 0, b > 0 \) and \( 0 \leq x \leq 1 \) where the boundary values at \( x = 0 \) or \( x = 1 \) are defined as by continuity (as limits).

The mean is \( a/(a + b) \) and the variance is \( ab/((a + b)^2(a + b + 1)) \).

\( \text{pbeta} \) is closely related to the incomplete beta function. As defined by Abramowitz and Stegun

6.6.1

\( B_x(a, b) = \int_0^x t^{a-1}(1 - t)^{b-1} dt, \)

and 6.6.2 \( I_x(a, b) = B_x(a, b)/B(a, b) \) where \( B(a, b) = B_1(a, b) \) is the Beta function (\text{beta}).

\( I_x(a, b) \) is \text{pbeta}(x, a, b).

Value

dbeta gives the density, pbeta the distribution function, qbeta the quantile function, and rbeta generates random deviates.

References


See Also

\text{beta} for the Beta function, and \text{dgamma} for the Gamma distribution.

Examples

\[
x <- \text{seq}(0, 1, \text{length}=21)
dbeta(x, 1, 1)
pbeta(x, 1, 1)
\]
**binom.test**  

*Exact Binomial Test*

**Description**
Performs an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment.

**Usage**

```r
binom.test(x, n, p = 0.5,
            alternative = c("two.sided", "less", "greater"),
            conf.level = 0.95)
```

**Arguments**
- `x`: number of successes, or a vector of length 2 giving the numbers of successes and failures, respectively.
- `n`: number of trials; ignored if `x` has length 2.
- `p`: hypothesized probability of success.
- `alternative`: indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter.
- `conf.level`: confidence level for the returned confidence interval.

**Details**
Confidence intervals are obtained by a procedure first given in Clopper and Pearson (1934). This guarantees that the confidence level is at least `conf.level`, but in general does not give the shortest-length confidence intervals.

**Value**
A list with class "htest" containing the following components:
- `statistic`: the number of successes.
- `parameter`: the number of trials.
- `p.value`: the p-value of the test.
- `conf.int`: a confidence interval for the probability of success.
- `estimate`: the estimated probability of success.
- `null.value`: the probability of success under the null, `p`.
- `alternative`: a character string describing the alternative hypothesis.
- `method`: the character string "Exact binomial test".
- `data.name`: a character string giving the names of the data.
References


See Also

*prop.test* for a general (approximate) test for equal or given proportions.

Examples

```r
## Under (the assumption of) simple Mendelian inheritance, a cross
## between plants of two particular genotypes produces progeny 1/4 of
## which are "dwarf" and 3/4 of which are "giant", respectively.
## In an experiment to determine if this assumption is reasonable, a
## cross results in progeny having 243 dwarf and 682 giant plants.
## If "giant" is taken as success, the null hypothesis is that p =
binom.test(c(682, 243), p = 3/4)
binom.test(682, 682 + 243, p = 3/4)  # The same.
## => Data are in agreement with the null hypothesis.
```

Binomial

The Binomial Distribution

Description

Density, distribution function, quantile function and random generation for the binomial distribution with parameters `size` and `prob`.

Usage

```r
dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)
```

Arguments

- `x, q`  
  vector of quantiles.
- `p`  
  vector of probabilities.
- `n`  
  number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `size`  
  number of trials.
- `prob`  
  probability of success on each trial.
- `log, log.p`  
  logical; if TRUE, probabilities p are given as log(p).
- `lower.tail`  
  logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
Details

The binomial distribution with size = n and prob = p has density

\[ p(x) = \binom{n}{x} p^x (1 - p)^{n-x} \]

for \( x = 0, \ldots, n \).

If an element of x is not integer, the result of dbinom is zero, with a warning. \( p(x) \) is computed using Loader’s algorithm, see the reference below.

The quantile is defined as the smallest value \( x \) such that \( F(x) \geq p \), where \( F \) is the distribution function.

Value

dbinom gives the density, pbinom gives the distribution function, qbinom gives the quantile function and rbinom generates random deviates.

If size is not an integer, NaN is returned.

References

Catherine Loader (2000). Fast and Accurate Computation of Binomial Probabilities; manuscript available from http://cm.bell-labs.com/cm/ms/departments/sia/catherine/dbinom

See Also
dnbinom for the negative binomial, and dpois for the Poisson distribution.

Examples

```r
# Compute P(45 < X < 55) for X Binomial(100,0.5)
sum(dbinom(46:54, 100, 0.5))
## Using "log = TRUE" for an extended range :
n <- 2000
k <- seq(0, n, by = 20)
plot(k, dbinom(k, n, pi/10, log=TRUE), type='l', ylab="log density",
     main = "dbinom(*, log=TRUE) is better than log(dbinom(*))")
lines(k, log(dbinom(k, n, pi/10)), col='red', lwd=2)
## extreme points are omitted since dbinom gives 0.
mtext("dbinom(k, log=TRUE)", adj=0)
mtext("extended range", adj=0, line = -1, font=4)
mtext("log(dbinom(k))", col="red", adj=1)
```

biplot

Biplot of Multivariate Data

Description

Plot a biplot on the current graphics device.
Usage

biplot(x, ...)

## Default S3 method:
biplot(x, y, var.axes = TRUE, col, cex = rep(par("cex"), 2),
xlabs = NULL, ylabs = NULL, expand = 1,
xlim = NULL, ylim = NULL, arrow.len = 0.1,
main = NULL, sub = NULL, xlab = NULL, ylab = NULL, ...)

Arguments

x  The biplot, a fitted object. For biplot.default, the first set of points (a
two-column matrix), usually associated with observations.
y  The second set of points (a two-column matrix), usually associated with vari-
ables.
var.axes  If TRUE the second set of points have arrows representing them as (unscaled)
axes.
col  A vector of length 2 giving the colours for the first and second set of points
respectively (and the corresponding axes). If a single colour is specified it will be
used for both sets. If missing the default colour is looked for in the palette: if
there it and the next colour as used, otherwise the first two colours of the paletter
are used.
cex  The character expansion factor used for labelling the points. The labels can be
of different sizes for the two sets by supplying a vector of length two.
xlabs  A vector of character strings to label the first set of points: the default is to use
the row dimname of x, or 1:n is the dimname is NULL.
ylabs  A vector of character strings to label the second set of points: the default is to use
the row dimname of y, or 1:n is the dimname is NULL.
expand  An expansion factor to apply when plotting the second set of points relative to
the first. This can be used to tweak the scaling of the two sets to a physically
comparable scale.
arrow.len  The length of the arrow heads on the axes plotted in var.axes is true. The
arrow head can be suppressed by arrow.len = 0.
xlim, ylim  Limits for the x and y axes in the units of the first set of variables.
main, sub, xlab, ylab, ...  
graphical parameters.

Details

A biplot is plot which aims to represent both the observations and variables of a matrix of multivari-
ate data on the same plot. There are many variations on biplots (see the references) and perhaps the
most widely used one is implemented by biplot.princomp. The function biplot.default
merely provides the underlying code to plot two sets of variables on the same figure.

Graphical parameters can also be given to biplot.

Side Effects

a plot is produced on the current graphics device.
References


See Also

`biplot.princomp`, also for examples.

---

### biplot.princomp  

**Biplot for Principal Components**

#### Description

Produce a biplot (in the strict sense) from the output of `princomp` or `prcomp`.

#### Usage

```r
## S3 method for class 'prcomp':
biplot(x, choices = 1:2, scale = 1, pc.biplot = FALSE, ...)

## S3 method for class 'princomp':
biplot(x, choices = 1:2, scale = 1, pc.biplot = FALSE, ...)
```

#### Arguments

- `x`  
  an object of class "princomp".
- `choices`  
  length 2 vector specifying the components to plot. Only the default is a biplot in the strict sense.
- `scale`  
  The variables are scaled by `lambda ^ scale` and the observations are scaled by `lambda ^ (1-scale)` where `lambda` are the singular values as computed by `princomp`. Normally `0 <= scale <= 1`, and a warning will be issued if the specified `scale` is outside this range.
- `pc.biplot`  
  If true, use what Gabriel (1971) refers to as a "principal component biplot", with `lambda = 1` and observations scaled up by `sqrt(n)` and variables scaled down by `sqrt(n)`. Then inner products between variables approximate covariances and distances between observations approximate Mahalanobis distance.
- `...`  
  optional arguments to be passed to `biplot.default`.

#### Details

This is a method for the generic function `biplot`. There is considerable confusion over the precise definitions: those of the original paper, Gabriel (1971), are followed here. Gabriel and Odoroff (1990) use the same definitions, but their plots actually correspond to `pc.biplot = TRUE`.

#### Side Effects

A plot is produced on the current graphics device.
References


See Also

`biplot`, `princomp`.

Examples

```r
biplot(princomp(USArrests))
```

Birthday

Probability of coincidences

Description

Computes approximate answers to a generalised “birthday paradox” problem. `pbirthday` computes the probability of a coincidence and `qbirthday` computes the number of observations needed to have a specified probability of coincidence.

Usage

```r
qbirthday(prob = 0.5, classes = 365, coincident = 2)
pbirthday(n, classes = 365, coincident = 2)
```

Arguments

- `classes`: How many distinct categories the people could fall into
- `prob`: The desired probability of coincidence
- `n`: The number of people
- `coincident`: The number of people to fall in the same category

Details

The birthday paradox is that a very small number of people, 23, suffices to have a 50-50 chance that two of them have the same birthday. This function generalises the calculation to probabilities other than 0.5, numbers of coincident events other than 2, and numbers of classes other than 365.

This formula is approximate, as the example below shows. For `coincident=2` the exact computation is straightforward and may be preferable.

Value

- `qbirthday`: Number of people needed for a probability `prob` that `k` of them have the same one out of `classes` equiprobable labels.
- `pbirthday`: Probability of the specified coincidence
**Box.test**

**Box-Pierce and Ljung-Box Tests**

**Description**

Compute the Box–Pierce or Ljung–Box test statistic for examining the null hypothesis of independence in a given time series.

**Usage**

```r
Box.test(x, lag = 1, type = c("Box-Pierce", "Ljung-Box"))
```

**Arguments**

- `x` a numeric vector or univariate time series.
- `lag` the statistic will be based on lag autocorrelation coefficients.
- `type` test to be performed: partial matching is used.

**Value**

A list with class "htest" containing the following components:

- `statistic` the value of the test statistic.
- `parameter` the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- `p.value` the p-value of the test.

---

**References**


**Examples**

```r
## the standard version
qbirthday()
## same 4-digit PIN number
qbirthday(classes=10^4)
## 0.9 probability of three coincident birthdays
qbirthday(coincident=3, prob=0.9)
## Chance of 4 coincident birthdays in 150 people
pbirthday(150, coincident=4)
## Accuracy compared to exact calculation
x1 <- sapply(10:100, pbirthday)
x2 <- 1 - sapply(10:100, function(n) prod((365:(365-n+1))/rep(365, n)))
par(mfrow=c(2,2))
plot(x1, x2, xlab="approximate", ylab="exact")
abline(0, 1)
plot(x1, x1 - x2, xlab="approximate", ylab="error")
abline(h=0)
plot(x1, x2, log="xy", xlab="approximate", ylab="exact")
abline(0, 1)
plot(1-x1, 1-x2, log="xy", xlab="approximate", ylab="exact")
abline(0, 1)
```
method a character string indicating which type of test was performed.
data.name a character string giving the name of the data.

Note

Missing values are not handled.

Author(s)

A. Trapletti

References


Examples

```r
x <- rnorm(100)
Box.test(x, lag = 1)
Box.test(x, lag = 1, type="Ljung")
```

C

Sets Contrasts for a Factor

Description

Sets the "contrasts" attribute for the factor.

Usage

```r
C(object, contr, how.many, ...)
```

Arguments

- `object` a factor or ordered factor
- `contr` which contrasts to use. Can be a matrix with one row for each level of the factor or a suitable function like `contr.poly` or a character string giving the name of the function
- `how.many` the number of contrasts to set, by default one less than `nlevels(object)`
- `...` additional arguments for the function `contr`.

Details

For compatibility with S, `contr` can be `treatment`, `helmert`, `sum` or `poly` (without quotes) as shorthand for `contr.treatment` and so on.
**Value**

The factor object with the "contrasts" attribute set.

**References**


**See Also**

`contrasts`, `contr.sum`, etc.

**Examples**

```r
## reset contrasts to defaults
options(contrasts=c("contr.treatment", "contr.poly"))
attach(warpbreaks)
tens <- C(tension, poly, 1)
attributes(tens)
detach()
## tension SHOULD be an ordered factor, but as it is not we can use
aov(breaks ~ wool + tens + tension, data=warpbreaks)

## show the use of ... The default contrast is contr.treatment here
summary(lm(breaks ~ wool + C(tension, base=2), data=warpbreaks))

# following on from help(esoph)
model3 <- glm(cbind(ncases, ncontrols) ~ agegp + C(tobgp, , 1) +
               C(alcgp, , 1), data = esoph, family = binomial())
summary(model3)
```

---

**cancor**

*Canonical Correlations*

**Description**

Compute the canonical correlations between two data matrices.

**Usage**

```r
cancor(x, y, xcenter = TRUE, ycenter = TRUE)
```

**Arguments**

- `x` numeric matrix \((n \times p_1)\), containing the \(x\) coordinates.
- `y` numeric matrix \((n \times p_2)\), containing the \(y\) coordinates.
- `xcenter` logical or numeric vector of length \(p_1\), describing any centering to be done on the \(x\) values before the analysis. If TRUE (default), subtract the column means. If FALSE, do not adjust the columns. Otherwise, a vector of values to be subtracted from the columns.
- `ycenter` analogous to `xcenter`, but for the \(y\) values.
The canonical correlation analysis seeks linear combinations of the y variables which are well explained by linear combinations of the x variables. The relationship is symmetric as ‘well explained’ is measured by correlations.

A list containing the following components:

- cor: correlations.
- xcoef: estimated coefficients for the x variables.
- ycoef: estimated coefficients for the y variables.
- xcenter: the values used to adjust the x variables.
- ycenter: the values used to adjust the y variables.

References


See Also

qr, svd.

Examples

```r
pop <- LifeCycleSavings[, 2:3]
oec <- LifeCycleSavings[, -(2:3)]
cancor(pop, oec)

x <- matrix(rnorm(150), 50, 3)
y <- matrix(rnorm(250), 50, 5)
(cxy <- cancor(x, y))
all(abs(cor(x %*% cxy$xcoef, y %*% cxy$ycoef)[,1:3] - diag(cxy$cor)) < 1e-15)
all(abs(cor(x %*% cxy$xcoef) - diag(3)) < 1e-15)
all(abs(cor(y %*% cxy$ycoef) - diag(5)) < 1e-15)
```

---

**Case and Variable Names of Fitted Models**

**Description**

Simple utilities returning (non-missing) case names, and (non-eliminated) variable names.
Cauchy

Usage

case.names(object, ...)
## S3 method for class 'lm':
case.names(object, full = FALSE, ...)

variable.names(object, ...)
## S3 method for class 'lm':
variable.names(object, full = FALSE, ...)

Arguments

object an R object, typically a fitted model.
full logical; if TRUE, all names (including zero weights, ...) are returned.
... further arguments passed to or from other methods.

Value

A character vector.

See Also

lm

Examples

x <- 1:20
y <- x + (x/4 - 2)^3 + rnorm(20, s=3)
names(y) <- paste("O",x,sep=".")
ww <- rep(1,20); ww[13] <- 0
summary(lmxy <- lm(y ~ x + I(x^2)+I(x^3) + I((x-10)^2),
weights = ww), cor = TRUE)
variable.names(lmxy)
variable.names(lmxy, full= TRUE)# includes the last
case.names(lmxy)
case.names(lmxy, full = TRUE)# includes the 0-weight case

Cauchy

The Cauchy Distribution

Description

Density, distribution function, quantile function and random generation for the Cauchy distribution with location parameter location and scale parameter scale.

Usage

dcauchy(x, location = 0, scale = 1, log = FALSE)
pcauchy(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
qcauchy(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
rcauchy(n, location = 0, scale = 1)
Arguments

- x, q: vector of quantiles.
- p: vector of probabilities.
- n: number of observations. If \( \text{length}(n) > 1 \), the length is taken to be the number required.
- location, scale: location and scale parameters.
- log, log.p: logical; if TRUE, probabilities p are given as \( \log(p) \).
- lower.tail: logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

If location or scale are not specified, they assume the default values of 0 and 1 respectively.

The Cauchy distribution with location \( l \) and scale \( s \) has density

\[
f(x) = \frac{1}{\pi s} \left( 1 + \left( \frac{x - l}{s} \right)^2 \right)^{-1}
\]

for all \( x \).

Value

dcauchy, pcauchy, and qcauchy are respectively the density, distribution function and quantile function of the Cauchy distribution. rcauchy generates random deviates from the Cauchy.

References


See Also

dt for the t distribution which generalizes dcauchy(*, l = 0, s = 1).

Examples

dcauchy(-1:4)

chisq.test  Pearson’s Chi-squared Test for Count Data

Description

chisq.test performs chi-squared contingency table tests and goodness-of-fit tests.

Usage

chisq.test(x, y = NULL, correct = TRUE,
          p = rep(1/length(x), length(x)), rescale.p = FALSE,
          simulate.p.value = FALSE, B = 2000)
Arguments

x  a vector or matrix.
y  a vector; ignored if x is a matrix.
correct  a logical indicating whether to apply continuity correction when computing the test statistic.
p  a vector of probabilities of the same length of x. An error is given if any entry of p is negative.
rescale.p  a logical scalar; if TRUE then p is rescaled (if necessary) to sum to 1. If rescale.p is FALSE, and p does not sum to 1, an error is given.
simulate.p.value  a logical indicating whether to compute p-values by Monte Carlo simulation.
B  an integer specifying the number of replicates used in the Monte Carlo simulation.

details

If x is a matrix with one row or column, or if x is a vector and y is not given, then a "goodness-of-fit test" is performed ("x is treated as a one-dimensional contingency table"). The entries of x must be non-negative integers. In this case, the hypothesis tested is whether the population probabilities equal those in p, or are all equal if p is not given.

If x is a matrix with at least two rows and columns, it is taken as a two-dimensional contingency table. Again, the entries of x must be non-negative integers. Otherwise, x and y must be vectors or factors of the same length; incomplete cases are removed, the objects are coerced into factor objects, and the contingency table is computed from these. Then, Pearson’s chi-squared test of the null that the joint distribution of the cell counts in a 2-dimensional contingency table is the product of the row and column marginals is performed.

If simulate.p.value is FALSE, the p-value is computed from the asymptotic chi-squared distribution of the test statistic; continuity correction is only used in the 2-by-2 case if correct is TRUE. Otherwise, if simulate.p.value is TRUE, the p-value is computed by Monte Carlo simulation with B replicates.

In the contingency table case this is done by random sampling from the set of all contingency tables with given marginals, and works only if the marginals are positive. (A C translation of the algorithm of Patefield (1981) is used.)

In the goodness-of-fit case this is done by random sampling from the discrete distribution specified by p, each sample being of size n = sum(x). This simulation is done in raw R and is slow.

Value

A list with class "htest" containing the following components:

statistic  the value the chi-squared test statistic.
parameter  the degrees of freedom of the approximate chi-squared distribution of the test statistic, NA if the p-value is computed by Monte Carlo simulation.
p.value  the p-value for the test.
method  a character string indicating the type of test performed, and whether Monte Carlo simulation or continuity correction was used.
data.name  a character string giving the name(s) of the data.
observed  the observed counts.
expected the expected counts under the null hypothesis.
residuals the Pearson residuals, (observed - expected) / sqrt(expected).

References


Examples

```r
## Not really a good example
chisq.test(InsectSprays$count > 7, InsectSprays$spray)
# Prints test summary
chisq.test(InsectSprays$count > 7, InsectSprays$spray)$obs
# Counts observed
chisq.test(InsectSprays$count > 7, InsectSprays$spray)$exp
# Counts expected under the null

## Effect of simulating p-values
x <- matrix(c(12, 5, 7, 7), nc = 2)
chisq.test(x)$p.value # 0.4233
chisq.test(x, simulate.p.value = TRUE, B = 10000)$p.value
# around 0.29!

## Testing for population probabilities
## Case A. Tabulated data
x <- c(A = 20, B = 15, C = 25)
chisq.test(x)
chisq.test(as.table(x)) # the same
p <- c(89,37,30,28,2)
try{
  chisq.test(x, p = p) # gives an error
}
chisq.test(x, p = p, rescale.p = TRUE)
# works
p <- c(0.40,0.20,0.20,0.19,0.01)
# Expected count in category 5
# is 1.86 < 5 ==> chi square approx.
chisq.test(x, p = p)
# maybe doubtful, but is ok!
chisq.test(x, p = p,simulate.p.value = TRUE)

## Case B. Raw data
x <- trunc(5 * runif(100))
chisq.test(table(x)) # NOT 'chisq.test(x)'
```

Chisquare

The (non-central) Chi-Squared Distribution

Description

Density, distribution function, quantile function and random generation for the chi-squared ($\chi^2$) distribution with df degrees of freedom and optional non-centrality parameter ncp.
Usage

dchisq(x, df, ncp=0, log = FALSE)
pchisq(q, df, ncp=0, lower.tail = TRUE, log.p = FALSE)
qchisq(p, df, ncp=0, lower.tail = TRUE, log.p = FALSE)
rchisq(n, df, ncp=0)

Arguments

x, q  vector of quantiles.
p  vector of probabilities.
n  number of observations. If length(n) > 1, the length is taken to be the number required.
df  degrees of freedom (non-negative, but can be non-integer).
ncp  non-centrality parameter (non-negative). Note that ncp values larger than about 1417 are not allowed currently for pchisq and qchisq.
log, log.p  logical; if TRUE, probabilities p are given as log(p).
lower.tail  logical; if TRUE (default), probabilities are P[X ≤ x], otherwise, P[X > x].

Details

The chi-squared distribution with df = n > 0 degrees of freedom has density

\[ f_n(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2} \]

for \( x > 0 \). The mean and variance are \( n \) and \( 2n \).

The non-central chi-squared distribution with df = n degrees of freedom and non-centrality parameter ncp = \( \lambda \) has density

\[ f(x) = e^{-\lambda/2} \sum_{r=0}^{\infty} \frac{(\lambda/2)^r}{r!} f_{n+2r}(x) \]

for \( x \geq 0 \). For integer \( n \), this is the distribution of the sum of squares of \( n \) normals each with variance one, \( \lambda \) being the sum of squares of the normal means; further, \( E(X) = n + \lambda \), \( Var(X) = 2(n + 2 \ast \lambda) \), and \( E((X - E(X))^3) = 8(n + 3 \ast \lambda) \).

Note that the degrees of freedom df = \( n \), can be non-integer, and for non-centrality \( \lambda > 0 \), even \( n = 0 \); see Johnson, Kotz and Balakrishnan (1995, chapter 29).

Value

dchisq gives the density, pchisq gives the distribution function, qchisq gives the quantile function, and rchisq generates random deviates.

References


Johnson, Kotz and Balakrishnan (1995). Continuous Univariate Distributions, Vol 2; Wiley NY;
See Also

A central chi-squared distribution with \( n \) degrees of freedom is the same as a Gamma distribution with shape \( \alpha = n/2 \) and scale \( \sigma = 2 \). Hence, see \texttt{dgamma} for the Gamma distribution.

Examples

dchisq(1, df=1:3)
pchisq(1, df= 3)
pchisq(1, df= 3, ncp = 0:4)# includes the above

x <- 1:10
## Chi-squared(df = 2) is a special exponential distribution
all.equal(dchisq(x, df=2), dexp(x, 1/2))
all.equal(pchisq(x, df=2), pexp(x, 1/2))

## non-central RNG -- df=0 is ok for ncp > 0: Z0 has point mass at 0!
Z0 <- rchisq(100, df = 0, ncp = 2.)
graphics::stem(Z0)

## Not run:
## visual testing
## do P-P plots for 1000 points at various degrees of freedom
L <- 1.2; n <- 1000; pp <- ppoints(n)
op <- par(mfrow = c(3,3), mar = c(3,3,1,1)+.1, mgp = c(1.5,.6,0), oma = c(0,0,3,0))
for(df in 2^(4 * rnorm(9))) {
  plot(pp, sort(pchisq(rr <- rchisq(n,df=df, ncp=L), df=df, ncp=L)),
       ylab = "pchisq(rchisq(.),.)", pch=".")
  mtext(paste("df = ",formatC(df, digits = 4)), line = -2, adj = 0.05)
  abline(0,1,col=2)
}
mtext(expression("P-P plots : Noncentral \* \* \* \* \* \* chi^2 \*\*\* (n=1000, df=X, ncp= 1.2)"),
       cex = 1.5, font = 2, outer=TRUE)
par(op)
## End(Not run)

---

clearNames

Remove the Names from an Object

Description

This function sets the \texttt{names} attribute of \texttt{object} to \texttt{NULL} and returns the object.

Usage

clearNames(object)

Arguments

- \texttt{object} an object that may have a \texttt{names} attribute
Value

An object similar to object but without names.

Author(s)

Douglas Bates and Saikat DebRoy

See Also

setNames

Examples

lapply( women, mean )  # has a names attribute
clearNames( lapply( women, mean ) )  # removes the names

cmdscale  

Classical (Metric) Multidimensional Scaling

Description

Classical multidimensional scaling of a data matrix. Also known as principal coordinates analysis (Gower, 1966).

Usage

cmdscale(d, k = 2, eig = FALSE, add = FALSE, x.ret = FALSE)

Arguments

d  a distance structure such as that returned by dist or a full symmetric matrix containing the dissimilarities.

k  the dimension of the space which the data are to be represented in; must be in \{1, 2, \ldots, n - 1\}.

eig  indicates whether eigenvalues should be returned.

add  logical indicating if an additive constant \(c^*\) should be computed, and added to the non-diagonal dissimilarities such that all \(n - 1\) eigenvalues are non-negative.

x.ret  indicates whether the doubly centered symmetric distance matrix should be returned.

Details

Multidimensional scaling takes a set of dissimilarities and returns a set of points such that the distances between the points are approximately equal to the dissimilarities.

The functions isoMDS and sammon in package MASS provide alternative ordination techniques. When add = TRUE, an additive constant \(c^*\) is computed, and the dissimilarities \(d_{ij} + c^*\) are used instead of the original \(d_{ij}\)'s.

Whereas S (Becker et al., 1988) computes this constant using an approximation suggested by Torgerson, R uses the analytical solution of Cailliez (1983), see also Cox and Cox (1994).
Value

If `eig = FALSE` and `x.ret = FALSE` (default), a matrix with \( k \) columns whose rows give the coordinates of the points chosen to represent the dissimilarities.

Otherwise, a list containing the following components.

- **points**: a matrix with \( k \) columns whose rows give the coordinates of the points chosen to represent the dissimilarities.
- **eig**: the \( n - 1 \) eigenvalues computed during the scaling process if `eig` is true.
- **x**: the doubly centered distance matrix if `x.ret` is true.
- **GOF**: a numeric vector of length 2, equal to say \((g_1, g_2)\), where \( g_i = (\sum_{j=1}^{k} \lambda_j)/ (\sum_{j=1}^{n} T_i(\lambda_j)) \), where \( \lambda_j \) are the eigenvalues (sorted decreasingly), \( T_1(v) = |v| \), and \( T_2(v) = \max(v, 0) \).

References


See Also

*dist*. Also *isoMDS* and *sammon* in package *MASS*.

Examples

```r
loc <- cmdscale(eurodist)
x <- loc[,1]
y <- -loc[,2]
plot(x, y, type="n", xlab="", ylab="", main="cmdscale(eurodist)")
text(x, y, rownames(loc), cex=0.8)

cmdsE <- cmdscale(eurodist, k=20, add = TRUE, eig = TRUE, x.ret = TRUE)
str(cmdsE)
```
Description

`coef` is a generic function which extracts model coefficients from objects returned by modeling functions. `coefficients` is an alias for it.

Usage

```r
coef(object, ...) coefficients(object, ...)
```

Arguments

- `object`: an object for which the extraction of model coefficients is meaningful.
- `...`: other arguments.

Details

All object classes which are returned by model fitting functions should provide a `coef` method or use the default one. (Note that the method is for `coef` and not `coefficients`.)

Class "aov" has a `coef` method that does not report aliased coefficients (see `alias`).

Value

Coefficients extracted from the model object `object`.

References


See Also

`fitted.values` and `residuals` for related methods; `glm`, `lm` for model fitting.

Examples

```r
x <- 1:5; coef(lm(c(1:3,7,6) ~ x))
```
complete.cases

Find Complete Cases

Description

Return a logical vector indicating which cases are complete, i.e., have no missing values.

Usage

complete.cases(...)

Arguments

... a sequence of vectors, matrices and data frames.

Value

A logical vector specifying which observations/rows have no missing values across the entire sequence.

See Also

is.na, na.omit, na.fail.

Examples

x <- airquality[, -1] # x is a regression design matrix
y <- airquality[, 1] # y is the corresponding response

stopifnot(complete.cases(y) != is.na(y))
ok <- complete.cases(x,y)
sum(!ok) # how many are not "ok" ?
x <- x[ok,]
y <- y[ok]

confint

Confidence Intervals for Model Parameters

Description

Computes confidence intervals for one or more parameters in a fitted model. There is a default and a method for objects inheriting from class "lm".

Usage

confint(object, parm, level = 0.95, ...)

Arguments

object  a fitted model object.

parm  a specification of which parameters are to be given confidence intervals, either
a vector of numbers or a vector of names. If missing, all parameters are consid-
ered.

level  the confidence level required.

...  additional argument(s) for methods.

Details

confint is a generic function. The default method assumes asymptotic normality, and needs
suitable coef and vcov methods to be available. The default method can be called directly for
comparison with other methods.

For objects of class "lm" the direct formulae based on t values are used.

There are stub methods for classes "glm" and "nls" which invoke those in package MASS which
are based on profile likelihoods.

Value

A matrix (or vector) with columns giving lower and upper confidence limits for each parameter.
These will be labelled as (1-level)/2 and 1 - (1-level)/2 in % (by default 2.5% and 97.5%).

See Also

confint glm and confint nls in package MASS.

Examples

fit <- lm(100/mpg ~ disp + hp + wt + am, data=mtcars)
confint(fit)
confint(fit, "wt")

## from example glm (needs MASS to be present on the system)
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9); treatment <- gl(3,3)
glm.D93 <- glm(counts ~ outcome + treatment, family=poisson())
confint(glm.D93)
confint.default(glm.D93)  # based on asymptotic normality

---

constrOptim  Linearly constrained optimisation

Description

Minimise a function subject to linear inequality constraints using an adaptive barrier algorithm.

Usage

constrOptim(theta, f, grad, ui, ci, mu = 1e-04, control = list(),
method = if(is.null(grad)) "Nelder-Mead" else "BFGS",
outer.iterations = 100, outer.eps = 1e-05, ...)
Arguments

theta  Starting value: must be in the feasible region.
f    Function to minimise.
grad  Gradient of f.
ui    Constraints (see below).
ci    Constraints (see below).
mu    (Small) tuning parameter.
control  Passed to optim.
method  Passed to optim.
outer.iterations  Iterations of the barrier algorithm.
outer.eps  Criterion for relative convergence of the barrier algorithm.
...  Other arguments passed to optim, which will pass them to f and grad if it does not use them.

Details

The feasible region is defined by $u_i \times \theta - c_i \geq 0$. The starting value must be in the interior of the feasible region, but the minimum may be on the boundary.

A logarithmic barrier is added to enforce the constraints and then optim is called. The barrier function is chosen so that the objective function should decrease at each outer iteration. Minima in the interior of the feasible region are typically found quite quickly, but a substantial number of outer iterations may be needed for a minimum on the boundary.

The tuning parameter $\mu$ multiplies the barrier term. Its precise value is often relatively unimportant. As $\mu$ increases the augmented objective function becomes closer to the original objective function but also less smooth near the boundary of the feasible region.

Any optim method that permits infinite values for the objective function may be used (currently all but "L-BFGS-B"). The gradient function must be supplied except with method="Nelder-Mead".

As with optim, the default is to minimise and maximisation can be performed by setting control$fnscale to a negative value.

Value

As for optim, but with two extra components: barrier.value giving the value of the barrier function at the optimum and outer.iterations gives the number of outer iterations (calls to optim)

References

K. Lange Numerical Analysis for Statisticians. Springer 2001, p185ff

See Also

optim, especially method="L-BFGS-B" which does box-constrained optimisation.
## Contrast Matrices

### Description

Return a matrix of contrasts.

### Usage

- `contr.helmert(n, contrasts = TRUE)`
- `contr.poly(n, scores = 1:n, contrasts = TRUE)`
- `contr.sum(n, contrasts = TRUE)`
- `contr.treatment(n, base = 1, contrasts = TRUE)`
- `contr.SAS(n, contrasts = TRUE)`
Arguments

- `n`: a vector of levels for a factor, or the number of levels.
- `contrasts`: a logical indicating whether contrasts should be computed.
- `scores`: the set of values over which orthogonal polynomials are to be computed.
- `base`: an integer specifying which group is considered the baseline group. Ignored if `contrasts` is `FALSE`.

Details

These functions are used for creating contrast matrices for use in fitting analysis of variance and regression models. The columns of the resulting matrices contain contrasts which can be used for coding a factor with \( n \) levels. The returned value contains the computed contrasts. If the argument `contrasts` is `FALSE` a square indicator matrix (the dummy coding) is returned except for `contr.poly` (which include the 0-degree, i.e. constant, polynomial when `contrasts = FALSE`).

- `cont.helmert` returns Helmert contrasts, which contrast the second level with the first, the third with the average of the first two, and so on. `contr.poly` returns contrasts based on orthogonal polynomials. `contr.sum` uses ‘sum to zero contrasts’.

- `contr.treatment` contrasts each level with the baseline level (specified by `base`): the baseline level is omitted. Note that this does not produce ‘contrasts’ as defined in the standard theory for linear models as they are not orthogonal to the intercept.

- `contr.SAS` is a wrapper for `contr.treatment` that sets the base level to be the last level of the factor. The coefficients produced when using these contrasts should be equivalent to those produced by many (but not all) SAS procedures.

Value

A matrix with \( n \) rows and \( k \) columns, with \( k=n-1 \) if `contrasts` is `TRUE` and \( k=n \) if `contrasts` is `FALSE`.

References


See Also

- `contrasts.C`, and `aov`, `glm`, `lm`.

Examples

```r
(cH <- contr.helmert(4))
apply(cH, 2,sum) # column sums are 0!
crossprod(cH) # diagonal -- columns are orthogonal
contr.helmert(4, contrasts = FALSE) # just the 4 x 4 identity matrix

(cT <- contr.treatment(5))
all(crossprod(cT) == diag(4)) # TRUE: even orthonormal

(cP <- contr.SAS(5))
all(crossprod(cP) == diag(4)) # TRUE: even orthonormal
```
contrasts

(cP <- contr.poly(3))  # Linear and Quadratic
zapsmall(crossprod(cP), dig=15)  # orthonormal up to fuzz

contrasts

Get and Set Contrast Matrices

Description

Set and view the contrasts associated with a factor.

Usage

contrasts(x, contrasts = TRUE)
contrasts(x, how.many) <- value

Arguments

x
  a factor or a logical variable.
contrasts
  logical. See Details.
how.many
  How many contrasts should be made. Defaults to one less than the number of
evels of x. This need not be the same as the number of columns of ctr.
value
  either a numeric matrix whose columns give coefficients for contrasts in the
  levels of x, or the (quoted) name of a function which computes such matrices.

Details

If contrasts are not set for a factor the default functions from options("contrasts") are used.
A logical vector x is converted into a two-level factor with levels c(FALSE, TRUE) (regardless
of which levels occur in the variable).

The argument contrasts is ignored if x has a matrix contrasts attribute set. Otherwise if
contrasts = TRUE it is passed to a contrasts function such as contr.treatment and if
contrasts = FALSE an identity matrix is returned.

References


See Also

C, contr.helmert, contr.poly, contr.sum, contr.treatment; glm, aov, lm.

Examples

eexample(factor)
fff <- ff[, drop=TRUE]  # reduce to 5 levels.
contrasts(fff)  # treatment contrasts by default
contrasts(C(fff, sum))
contrasts(fff, contrasts = FALSE)  # the 5x5 identity matrix

contrasts(fff) <- contr.sum(5); contrasts(fff)  # set sum contrasts
contrasts(fff, 2) <- contr.sum(5); contrasts(fff)  # set 2 contrasts
# supply 2 contrasts, compute 2 more to make full set of 4.
contrasts(fff) <- contr.sum(5)[,1:2]; contrasts(fff)

---

**convolve**

**Fast Convolution**

**Description**

Use the Fast Fourier Transform to compute the several kinds of convolutions of two sequences.

**Usage**

```r
convolve(x, y, conj = TRUE, type = c("circular", "open", "filter"))
```

**Arguments**

- `x,y` numeric sequences of the same length to be convolved.
- `conj` logical; if TRUE, take the complex conjugate before back-transforming (default, and used for usual convolution).
- `type` character; one of "circular", "open", "filter" (beginning of word is ok). For circular, the two sequences are treated as circular, i.e., periodic. For open and filter, the sequences are padded with 0s (from left and right) first; "filter" returns the middle sub-vector of "open", namely, the result of running a weighted mean of `x` with weights `y`.

**Details**

The Fast Fourier Transform, `fft`, is used for efficiency.

The input sequences `x` and `y` must have the same length if circular is true.

Note that the usual definition of convolution of two sequences `x` and `y` is given by `convolve(x, rev(y), type = "o")`.

**Value**

If `r <- convolve(x,y, type = "open")` and `n <- length(x), m <- length(y)`, then

\[ r_k = \sum_i x_{k-m+i}y_i \]

where the sum is over all valid indices \( i, k = 1, \ldots, n + m - 1 \)

If `type == "circular"`, \( n = m \) is required, and the above is true for \( i, k = 1, \ldots, n \) when \( x_{j} := x_{n+j} \) for \( j < 1 \).

**References**


**See Also**

`fft`, `nextn`, and particularly `filter` (from the `stats` package) which may be more appropriate.
**Examples**

```r
x <- c(0,0,0,100,0,0,0)
y <- c(0,0,1, 2 ,1,0,0)/4
zapsmall(convolve(x,y)) # *NOT* what you first thought.
zapsmall(convolve(x, y[3:5], type="f")) # rather
x <- rnorm(50)
y <- rnorm(50)
# Circular convolution *has* this symmetry:
all.equal(convolve(x,y, conj = FALSE), rev(convolve(rev(y),x)))

n <- length(x <- -20:24)
y <- (x-10)^2/1000 + rnorm(x)/8

Han <- function(y) # Hanning
  convolve(y, c(1,2,1)/4, type = "filter")

plot(x,y, main="Using convolve(.) for Hanning filters")
lines(x[-c(1 , n) ], Han(y), col="red")
lines(x[-c(1:2, (n-1):n)], Han(Han(y)), lwd=2, col="dark blue")
```

**cophenetic**  
*Cophenetic Distances for a Hierarchical Clustering*

**Description**

Computes the cophenetic distances for a hierarchical clustering.

**Usage**

```r
cophenetic(x)
```

**Arguments**

- `x`  
an R object representing a hierarchical clustering. For the default method, an object of class `hclust` or with a method for `as.hclust()` such as `agnes`.

**Details**

The cophenetic distance between two observations that have been clustered is defined to be the intergroup dissimilarity at which the two observations are first combined into a single cluster. Note that this distance has many ties and restrictions.

It can be argued that a dendrogram is an appropriate summary of some data if the correlation between the original distances and the cophenetic distances is high. Otherwise, it should simply be viewed as the description of the output of the clustering algorithm.

`cophenetic` is a generic function. Support for classes which represent hierarchical clusterings (total indexed hierarchies) can be added by providing an `as.hclust()` or, more directly, a `cophenetic()` method for such a class.

The method for objects of class "dendrogram" requires that all leaves of the dendrogram object have non-null labels.
Value

An object of class dist.

Author(s)

Robert Gentleman

References


See Also
dist, hclust

Examples

d1 <- dist(USArrests)
hc <- hclust(d1, "ave")
d2 <- cophenetic(hc)
cor(d1,d2) # 0.7659

## Example from Sneath & Sokal, Fig. 5-29, p.279
d0 <- c(1,3.8,4.4,5.1, 4,4.2,5, 2.6,5.3, 5.4)
attributes(d0) <- list(Size = 5, diag=TRUE)
class(d0) <- "dist"
names(d0) <- letters[1:5]
d0
str(upgma <- hclust(d0, method = "average"))
plot(upgma, hang = -1)
#
(d.coph <- cophenetic(upgma))
cor(d0, d.coph) # 0.9911

Description

var, cov and cor compute the variance of x and the covariance or correlation of x and y if these are vectors. If x and y are matrices then the covariances (or correlations) between the columns of x and the columns of y are computed.

cov2cor scales a covariance matrix into the corresponding correlation matrix efficiently.

Usage

var(x, y = NULL, na.rm = FALSE, use)
cov(x, y = NULL, use = "all.obs",
method = c("pearson", "kendall", "spearman"))
**Arguments**

- **x**: a numeric vector, matrix or data frame.
- **y**: `NULL` (default) or a vector, matrix or data frame with compatible dimensions to `x`. The default is equivalent to `y = x` (but more efficient).
- **na.rm**: logical. Should missing values be removed?
- **use**: an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "all.obs", "complete.obs" or "pairwise.complete.obs".
- **method**: a character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson" (default), "kendall", or "spearman", can be abbreviated.
- **V**: symmetric numeric matrix, usually positive definite such as a covariance matrix.

**Details**

For `cov` and `cor` one must either give a matrix or data frame for `x` or give both `x` and `y`. `var` is just another interface to `cov`, where `na.rm` is used to determine the default for `use` when that is unspecified. If `na.rm` is TRUE then the complete observations (rows) are used (use = "complete") to compute the variance. Otherwise (use = "all"), `var` will give an error if there are missing values.

If `use` is "all.obs", then the presence of missing observations will produce an error. If `use` is "complete.obs" then missing values are handled by casewise deletion. Finally, if `use` has the value "pairwise.complete.obs" then the correlation between each pair of variables is computed using all complete pairs of observations on those variables. This can result in covariance or correlation matrices which are not positive semidefinite. "pairwise.complete.obs" only works with the "pearson" method for `cov` and `var`.

The denominator \( n - 1 \) is used which gives an unbiased estimator of the (co)variance for i.i.d. observations. These functions return `NA` when there is only one observation (whereas S-PLUS has been returning NaN), and fail if `x` has length zero.

For `cor()`, if `method` is "kendall" or "spearman", Kendall’s \( \tau \) or Spearman’s \( \rho \) statistic is used to estimate a rank-based measure of association. These are more robust and have been recommended if the data do not necessarily come from a bivariate normal distribution.

For `cov()`, a non-Pearson method is unusual but available for the sake of completeness. Note that "spearman" basically computes \( \text{cor}(\text{R}(x), \text{R}(y)) \) (or \( \text{cov}(.,.) \)) where \( \text{R}(u) := \text{rank}(u, \text{na.last}="keep") \). In the case of missing values, the ranks are calculated depending on the value of `use`, either based on complete observations, or based on pairwise completeness with reranking for each pair.

Prior to R 2.1.0, the ranking was done removing only cases that are missing on the variable itself. Scaling a covariance matrix into a correlation one can be achieved in many ways, mathematically most appealing by multiplication with a diagonal matrix from left and right, or more efficiently by using `sweep(.)`, `FUN = "/"` twice. The `cov2cor` function is even a bit more efficient, and provided mostly for didactical reasons.
Value

For \( r \leftarrow \text{cor}(*, \text{use} = \text{"all.obs"}) \), it is now guaranteed that \( \text{all}(r \leq 1) \).

References


See Also

- `cor.test` for confidence intervals (and tests).
- `cov.wt` for weighted covariance computation.
- `sd` for standard deviation (vectors).

Examples

```r
var(1:10)# 9.166667
var(1:5,1:5)# 2.5

## Two simple vectors
cor(1:10,2:11)# == 1

## Correlation Matrix of Multivariate sample:
(Cl <- cor(longley))
## Graphical Correlation Matrix:
symnum(Cl) # highly correlated

## Spearman's rho and Kendall's tau
symnum(clS <- cor(longley, method = "spearman"))
symnum(clK <- cor(longley, method = "kendall"))
## How much do they differ?
i <- lower.tri(Cl)
cor(cbind(P = Cl[i], S = clS[i], K = clK[i]))

## cov2cor() scales a covariance matrix by its diagonal
## to become the correlation matrix.
cov2cor # see the function definition (and learn ..)
stopifnot(all.equal(Cl, cov2cor(cov(longley))),
           all.equal(cor(longley, method="kendall"),
                     cov2cor(cov(longley, method="kendall"))))

##--- Missing value treatment:
C1 <- cov(swiss)
range(eigen(C1, only=TRUE)$val) # 6.19 1921
swM <- swiss
swM[1,2] <- swM[7,3] <- swM[25,5] <- NA # create 3 "missing"
try(cov(swM)) # Error: missing obs...
C2 <- cov(swM, use = "complete")
range(eigen(C2, only=TRUE)$val) # 6.46 1930
C3 <- cov(swM, use = "pairwise")
range(eigen(C3, only=TRUE)$val) # 6.19 1938

(scM <- symnum(cor(swM, method = "kendall", use = "complete")))
## Kendall's tau doesn't change much: identical symnum codings!
```
identical(scM, symnum(cor(swiss, method = "kendall")))

cor.test

Test for Association/Correlation Between Paired Samples

Description

Test for association between paired samples, using one of Pearson’s product moment correlation coefficient, Kendall’s τ or Spearman’s ρ.

Usage

cor.test(x, ...)  
## Default S3 method:  
cor.test(x, y,  
   alternative = c("two.sided", "less", "greater"),  
   method = c("pearson", "kendall", "spearman"),  
   exact = NULL, conf.level = 0.95, ...)  
## S3 method for class 'formula':  
cor.test(formula, data, subset, na.action, ...)

Arguments

x, y numeric vectors of data values. x and y must have the same length.

alternative indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. "greater" corresponds to positive association, "less" to negative association.

method a character string indicating which correlation coefficient is to be used for the test. One of "pearson", "kendall", or "spearman", can be abbreviated.

exact a logical indicating whether an exact p-value should be computed. Only used for Kendall’s τ. See the Details for the meaning of NULL (the default).

conf.level confidence level for the returned confidence interval. Currently only used for the Pearson product moment correlation coefficient if there are at least 4 complete pairs of observations.

formula a formula of the form ~ u + v, where each of u and v are numeric variables giving the data values for one sample. The samples must be of the same length.

data an optional data frame containing the variables in the model formula.

subset an optional vector specifying a subset of observations to be used.

na.action a function which indicates what should happen when the data contain NAs. Defaults togetOption("na.action").

... further arguments to be passed to or from methods.
The three methods each estimate the association between paired samples and compute a test of the value being zero. They use different measures of association, all in the range \([-1, 1]\) with 0 indicating no association. These are sometimes referred to as tests of no correlation, but that term is often confined to the default method.

If `method` is "pearson", the test statistic is based on Pearson's product moment correlation coefficient \(\text{cor}(x, y)\) and follows a t distribution with \(\text{length}(x) - 2\) degrees of freedom if the samples follow independent normal distributions. If there are at least 4 complete pairs of observation, an asymptotic confidence interval is given based on Fisher's Z transform.

If `method` is "kendall" or "spearman", Kendall's \(\tau\) or Spearman's \(\rho\) statistic is used to estimate a rank-based measure of association. These tests may be used if the data do not necessarily come from a bivariate normal distribution.

For Kendall's test, by default (if `exact` is NULL), an exact p-value is computed if there are less than 50 paired samples containing finite values and there are no ties. Otherwise, the test statistic is the estimate scaled to zero mean and unit variance, and is approximately normally distributed.

For Spearman's test, p-values are computed using algorithm AS 89.

### Value

A list with class "htest" containing the following components:

- `statistic` the value of the test statistic.
- `parameter` the degrees of freedom of the test statistic in the case that it follows a t distribution.
- `p.value` the p-value of the test.
- `estimate` the estimated measure of association, with name "cor", "tau", or "rho" corresponding to the method employed.
- `null.value` the value of the association measure under the null hypothesis, always 0.
- `alternative` a character string describing the alternative hypothesis.
- `method` a character string indicating how the association was measured.
- `data.name` a character string giving the names of the data.
- `conf.int` a confidence interval for the measure of association. Currently only given for Pearson's product moment correlation coefficient in case of at least 4 complete pairs of observations.

### References


### Examples

```r
## Assessment of tuna quality. We compare the Hunter L measure of
## lightness to the averages of consumer panel scores (recoded as
## integer values from 1 to 6 and averaged over 80 such values) in
## 9 lots of canned tuna.
```
## The alternative hypothesis of interest is that the Hunter L value is positively associated with the panel score.

```r
cor.test(x, y, method = "kendall", alternative = "greater")
```

```r
## => p=0.05972
```

```r
cor.test(x, y, method = "kendall", alternative = "greater",
exact = FALSE) # using large sample approximation
```

```r
## => p=0.04765
```

```r
## Compare this to

cor.test(x, y, method = "spearman", alternative = "g")
cor.test(x, y, alternative = "g")
```

## Formula interface.

```r
pairs(USJudgeRatings)
cor.test(~ CONT + INTG, data = USJudgeRatings)
```

---

### cov.wt

#### Weighted Covariance Matrices

**Description**

Returns a list containing estimates of the weighted covariance matrix and the mean of the data, and optionally of the (weighted) correlation matrix.

**Usage**

```r
cov.wt(x, wt = rep(1/nrow(x), nrow(x)), cor = FALSE, center = TRUE)
```

**Arguments**

- `x`: a matrix or data frame. As usual, rows are observations and columns are variables.
- `wt`: a non-negative and non-zero vector of weights for each observation. Its length must equal the number of rows of `x`.
- `cor`: A logical indicating whether the estimated correlation weighted matrix will be returned as well.
- `center`: Either a logical or a numeric vector specifying the centers to be used when computing covariances. If TRUE, the (weighted) mean of each variable is used, if FALSE, zero is used. If `center` is numeric, its length must equal the number of columns of `x`.

**Details**

The covariance matrix is divided by one minus the sum of squares of the weights, so if the weights are the default (1/n) the conventional unbiased estimate of the covariance matrix with divisor (n−1) is obtained. This differs from the behaviour in S-PLUS.
Value

A list containing the following named components:

- `cov`: the estimated (weighted) covariance matrix
- `center`: an estimate for the center (mean) of the data.
- `n.obs`: the number of observations (rows) in x.
- `wt`: the weights used in the estimation. Only returned if given as an argument.
- `cor`: the estimated correlation matrix. Only returned if cor is TRUE.

See Also

cov and var.

Description

Plots a cumulative periodogram.

Usage

cpgram(ts, taper = 0.1,
       main = paste("Series: ", deparse(substitute(ts))),
       ci.col = "blue")

Arguments

- `ts`: a univariate time series
- `taper`: proportion tapered in forming the periodogram
- `main`: main title
- `ci.col`: colour for confidence band.

Value

None.

Side Effects

Plots the cumulative periodogram in a square plot.

Note

From package MASS.

Author(s)

B.D. Ripley
Examples

```r
cutree

Examples

par(pty = "s", mfrow = c(1,2))
cpgram(lh)
lh.ar <- ar(lh, order.max = 9)
cpgram(lh.ar$resid, main = "AR(3) fit to lh")
cpgram(ldeaths)

cutree

Description

Cuts a tree, e.g., as resulting from `hclust`, into several groups either by specifying the desired number(s) of groups or the cut height(s).

Usage

cutree(tree, k = NULL, h = NULL)

Arguments

tree  
a tree as produced by `hclust`. `cutree()` only expects a list with components `merge`, `height`, and `labels`, of appropriate content each.

k     
an integer scalar or vector with the desired number of groups

h     
numeric scalar or vector with heights where the tree should be cut.

At least one of `k` or `h` must be specified, `k` overrides `h` if both are given.

Value

cutree returns a vector with group memberships if `k` or `h` are scalar, otherwise a matrix with group memberships is returned where each column corresponds to the elements of `k` or `h`, respectively (which are also used as column names).

References


See Also

`hclust`, `dendrogram` for cutting trees themselves.

Examples

```r
hc <- hclust(dist(USArrests))
cutree(hc, k=1:5)#k = 1 is trivial
cutree(hc, h=250)

## Compare the 2 and 3 grouping:
g24 <- cutree(hc, k = c(2,4))
table(g24,"2", g24,"4")
```
**decompose**  
*Classical Seasonal Decomposition by Moving Averages*

**Description**
Decompose a time series into seasonal, trend and irregular components using moving averages. Deals with additive or multiplicative seasonal component.

**Usage**
```r
decompose(x, type = c("additive", "multiplicative"), filter = NULL)
```

**Arguments**
- `x`: A time series.
- `type`: The type of seasonal component.
- `filter`: A vector of filter coefficients in reverse time order (as for AR or MA coefficients), used for filtering out the seasonal component. If `NULL`, a moving average with symmetric window is performed.

**Details**
The additive model used is:
\[ Y[t] = T[t] + S[t] + e[t] \]
The multiplicative model used is:
\[ Y[t] = T[t] \times S[t] + e[t] \]

**Value**
An object of class "decomposed.ts" with following components:
- `seasonal`: The seasonal component (i.e., the repeated seasonal figure)
- `figure`: The estimated seasonal figure only
- `trend`: The trend component
- `random`: The remainder part
- `type`: The value of `type`

**Note**
The function `stl` provides a much more sophisticated decomposition.

**Author(s)**
David Meyer (David.Meyer@wu-wien.ac.at)

**See Also**
- `stl`
**delete.response**

**Examples**

```r
m <- decompose(co2)
m$figure
plot(m)
```

**Description**

delete.response returns a `terms` object for the same model but with no response variable.
drop.terms removes variables from the right-hand side of the model. There is also a 
"[].terms" method to perform the same function (with keep.response=TRUE).
reformulate creates a formula from a character vector.

**Usage**

delete.response(termobj)

reformulate(termlabels, response = NULL)

drop.terms(termobj, dropx = NULL, keep.response = FALSE)

**Arguments**

termobj    A `terms` object

termlabels character vector giving the right-hand side of a model formula.

response character string, symbol or call giving the left-hand side of a model formula.

dropx vector of positions of variables to drop from the right-hand side of the model.

keep.response

Keep the response in the resulting object?

**Value**

delete.response and drop.terms return a `terms` object.
reformulate returns a formula.

**See Also**

terms

**Examples**

```r
ff <- y ~ z + x + w
tt <- terms(ff)
tt
delete.response(tt)
drop.terms(tt, 2:3, keep.response = TRUE)
tt[[-1]]
tt[2:3]
```
dendrapply

Apply a Function to All Nodes of a Dendrogram

Description

Apply function FUN to each node of a `dendrogram` recursively. When `y <- dendrapply(x, fn)` then `y` is a dendrogram of the same graph structure as `x` and each for each node, `y.node[j] <- FUN( x.node[j], ...)` (where `y.node[j]` is an (invalid!) notation for the j-th node of `y`.

Usage

dendrapply(X, FUN, ...)

Arguments

X
an object of class "dendrogram".

FUN
an R function to be applied to each dendrogram node, typically working on its 
`attributes` alone, returning an altered version of the same node.

... potential further arguments passed to FUN.

Value

Usually a dendrogram of the same (graph) structure as `X`. For that, the function must be conceptually of the form `FUN <- function(X) { attributes(X) <- .....; X }, i.e. returning the node with some attributes added or changed.

Note

this is still somewhat experimental, and suggestions for enhancements (or nice examples of usage) are very welcome.

Author(s)

Martin Maechler

See Also

`as.dendrogram`, `lapply` for applying a function to each component of a list.
Examples

```r
## a smallish simple dendrogram
dhc <- as.dendrogram(hc <- hclust(dist(USArrests), "ave"))
(dhc21 <- dhc[[2]][[1]])

## too simple:
dendrapply(dhc21, function(n) str(attributes(n)))

## toy example to set colored leaf labels:
local({
collab <- function(n) {
    if(is.leaf(n)) {
        a <- attributes(n)
        i <<- i+1
        attr(n, "nodePar") <-
            c(a$nodePar, list(lab.col = mycols[i], lab.font= i%%3))
    }
    n
}
mycols <- grDevices::rainbow(attr(dhc21,"members"))
i <<- 0
})
dL <- dendrapply(dhc21, collab)
op <- par(mfrow=2:1)
plot(dhc21)
plot(dL) ## --> colored labels!
par(op)
```

dendrogram

General Tree Structures

Description

Class "dendrogram" provides general functions for handling tree-like structures. It is intended as a replacement for similar functions in hierarchical clustering and classification/regression trees, such that all of these can use the same engine for plotting or cutting trees. The code is still in testing stage and the API may change in the future.

Usage

```r
as.dendrogram(object, ...)
## S3 method for class 'hclust':
as.dendrogram(object, hang = -1, ...)

## S3 method for class 'dendrogram':
plot(x, type = c("rectangle", "triangle"),
    center = FALSE,
    edge.root = is.leaf(x) || !is.null(attr(x,"edgetext")),
    nodePar = NULL, edgePar = list(),
    leaflab = c("perpendicular", "textlike", "none"),
    dLeaf = NULL, xlab = "", ylab = "", xaxt = "n", yaxt = "s",
    horiz = FALSE, frame.plot = FALSE, ...)
```
## S3 method for class 'dendrogram':
cut(x, h, ...)

## S3 method for class 'dendrogram':
print(x, digits, ...)

## S3 method for class 'dendrogram':
rev(x)

## S3 method for class 'dendrogram':
str(object, max.level = 0, digits.d = 3,
give.attr = FALSE, wid = getOption("width"),
nest.lev = 0, indent.str = ", stem = "--", ...)

is.leaf(object)

### Arguments

- **object**
  - any R object that can be made into one of class "dendrogram".

- **x**
  - object of class "dendrogram".

- **hang**
  - numeric scalar indicating how the height of leaves should be computed from the heights of their parents; see `plot.hclust`.

- **type**
  - type of plot.

- **center**
  - logical; if TRUE, nodes are plotted centered with respect to the leaves in the branch. Otherwise (default), plot them in the middle of all direct child nodes.

- **edge.root**
  - logical; if true, draw an edge to the root node.

- **nodePar**
  - a list of plotting parameters to use for the nodes (see `points`) or NULL by default which does not draw symbols at the nodes. The list may contain components named `pch`, `cex`, `col`, and/or `bg` each of which can have length two for specifying separate attributes for inner nodes and leaves.

- **edgePar**
  - a list of plotting parameters to use for the edge segments and labels (if there's an edge text). The list may contain components named `col`, `lty` and `lwd` (for the segments), `p.col`, `p.lwd`, and `p.lty` (for the polygon around the text) and `t.col` for the text color. As with `nodePar`, each can have length two for differentiating leaves and inner nodes.

- **lealab**
  - a string specifying how leaves are labeled. The default "perpendicular" write text vertically (by default). "textlike" writes text horizontally (in a rectangle), and "none" suppresses leaf labels.

- **dLeaf**
  - a number specifying the distance in user coordinates between the tip of a leaf and its label. If NULL as per default, 3/4 of a letter width or height is used.

- **horiz**
  - logical indicating if the dendrogram should be drawn horizontally or not.

- **frame.plot**
  - logical indicating if a box around the plot should be drawn, see `plot.default`.

- **h**
  - height at which the tree is cut.

- **..., xlab, ylab, xaxt, yaxt**
  - graphical parameters, or arguments for other methods.

- **digits**
  - integer specifying the precision for printing, see `print.default`.
max.level, digits.d, give.attr, wid, nest.lev, indent.str
arguments to str, see \texttt{str.default}(). Note that \texttt{give.attr = FALSE}
still shows height and members attributes for each node.

\texttt{stem} a string used for \texttt{str()} specifying the \texttt{stem} to use for each dendrogram branch.

### Details

Warning: This documentation is preliminary.

The dendrogram is directly represented as a nested list where each component corresponds to a branch of the tree. Hence, the first branch of tree $z$ is $z[[1]]$, the second branch of the corresponding subtree is $z[[1]][[2]]$ etc. Each node of the tree carries some information needed for efficient plotting or cutting as attributes, of which only members, height and leaf for leaves are compulsory:

- **members** total number of leaves in the branch
- **height** numeric non-negative height at which the node is plotted.
- **midpoint** numeric horizontal distance of the node from the left border (the leftmost leaf) of the branch (unit 1 between all leaves). This is used for \texttt{plot( \*, center=FALSE)}.
- **label** character; the label of the node
- **x.member** for \texttt{cut()}$upper$, the number of former members; more generally a substitute for the members component used for “horizontal” (when horiz = FALSE, else “vertical”) alignment.
- **edgetext** character; the label for the edge leading to the node
- **nodePar** a named list (of length-1 components) specifying node-specific attributes for \texttt{points} plotting, see the \texttt{nodePar} argument above.
- **edgePar** a named list (of length-1 components) specifying attributes for \texttt{segments} plotting of the edge leading to the node, and drawing of the edgetext if available, see the \texttt{edgePar} argument above.
- **leaf** logical, if \texttt{TRUE}, the node is a leaf of the tree.

\texttt{cut.dendrogram()} returns a list with components $\texttt{upper}$ and $\texttt{lower}$, the first is a truncated version of the original tree, also of class dendrogram, the latter a list with the branches obtained from cutting the tree, each a dendrogram.

There are \texttt{[[, print, and str} methods for "dendrogram" objects where the first one (extraction) ensures that selecting sub-branches keeps the class.

Objects of class "hclust" can be converted to class "dendrogram" using method \texttt{as.dendrogram}.

\texttt{rev.dendrogram} simply returns the dendrogram \texttt{x} with reversed nodes, see also \texttt{reorder.dendrogram}.

\texttt{is.leaf(object)} is logical indicating if \texttt{object} is a leaf (the most simple dendrogram).
\texttt{plotNode()} and \texttt{plotNodeLimit()} are helper functions.

### Note

When using \texttt{type = "triangle".center = TRUE} often looks better.

### See Also

\texttt{order.dendrogram} also on the \texttt{labels} method for dendrograms.
Examples

hc <- hclust(dist(USArrests), "ave")
(dend1 <- as.dendrogram(hc)) # "print()" method
str(dend1) # "str()" method
str(dend1, max = 2) # only the first two sub-levels

op <- par(mfrow= c(2,2), mar = c(5,2,1,4))
plot(dend1)
## "triangle" type and show inner nodes:
plot(dend1, nodePar=list(pch = c(1,NA), cex=0.8, lab.cex = 0.8),
     type = "t", center=TRUE)
plot(dend1, edgePar=list(col = 1:2, lty = 2:3), dLeaf=1, edge.root = TRUE)
plot(dend1, nodePar=list(pch = 2:1,cex=.4*2:1, col = 2:3), horiz=TRUE)

dend2 <- cut(dend1, h=70)
plot(dend2$upper)
## leafs are wrong horizontally:
plot(dend2$upper, nodePar=list(pch = c(1,7), col = 2:1))
## dend2$lower is *NOT* a dendrogram, but a list of .. :
plot(dend2$lower[[3]], nodePar=list(col=4), horiz = TRUE, type = "tr")
## "inner" and "leaf" edges in different type & color :
plot(dend2$lower[[2]], nodePar=list(col=1),# non empty list
     edgePar = list(lty=1:2, col=2:1), horiz=TRUE)
par(op)

str(d3 <- dend2$lower[[2]][[2]][[1]]

nP <- list(col=3:2, cex=c(2.0, 0.75), pch= 21:22, col = "light blue", "pink"),
      lab.col = "tomato")
plot(d3, nodePar= nP, edgePar = list(col="gray", lwd=2), horiz = TRUE)
addE <- function(n) {
  if(!is.leaf(n)) {
    attr(n, "edgePar") <- list(p.col="plum")
    attr(n, "edgetext") <- paste(attr(n,"members"),"members")
  }
  n
}
d3e <- dendrapply(d3, addE)
plot(d3e, nodePar= nP)
plot(d3e, nodePar= nP, leaflab = "textlike")

---

density

Kernel Density Estimation

Description

The (S3) generic function density computes kernel density estimates. Its default method does so
with the given kernel and bandwidth for univariate observations.

Usage

density(x, ...)
## Default S3 method:
density(x, bw = "nrd0", adjust = 1,
density

kernel = c("gaussian", "epanechnikov", "rectangular", "triangular", "biweight", "cosine", "optcosine"),
weights = NULL, window = kernel, width,
give.Rkern = FALSE,
n = 512, from, to, cut = 3, na.rm = FALSE, ...)

Arguments

x
the data from which the estimate is to be computed.

bw
the smoothing bandwidth to be used. The kernels are scaled such that this is the standard deviation of the smoothing kernel. (Note this differs from the reference books cited below, and from S-PLUS.)
bw can also be a character string giving a rule to choose the bandwidth. See bw.nrd.
The specified (or computed) value of bw is multiplied by adjust.

adjust
the bandwidth used is actually adjust*bw. This makes it easy to specify values like “half the default” bandwidth.

kernel, window
a character string giving the smoothing kernel to be used. This must be one of "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" or "optcosine", with default "gaussian", and may be abbreviated to a unique prefix (single letter).
"cosine" is smoother than "optcosine", which is the usual “cosine” kernel in the literature and almost MSE-efficient. However, "cosine" is the version used by S.

weights
numeric vector of non-negative observation weights, hence of same length as x. The default NULL is equivalent to weights = rep(1/nx, nx) where nx is the length of (the finite entries of) x[].

width
this exists for compatibility with S; if given, and bw is not, will set bw to width if this is a character string, or to a kernel-dependent multiple of width if this is numeric.

give.Rkern
logical; if true, no density is estimated, and the “canonical bandwidth” of the chosen kernel is returned instead.

n
the number of equally spaced points at which the density is to be estimated. When n > 512, it is rounded up to the next power of 2 for efficiency reasons (fft).

from, to
the left and right-most points of the grid at which the density is to be estimated.

cut
by default, the values of left and right are cut bandwidths beyond the extremes of the data. This allows the estimated density to drop to approximately zero at the extremes.

na.rm
logical; if TRUE, missing values are removed from x. If FALSE any missing values cause an error.

...
... further arguments for (non-default) methods.

Details

The algorithm used in density.default disperses the mass of the empirical distribution function over a regular grid of at least 512 points and then uses the fast Fourier transform to convolve this approximation with a discretized version of the kernel and then uses linear approximation to evaluate the density at the specified points.
The statistical properties of a kernel are determined by $\sigma_K^2 = \int t^2 K(t) dt$ which is always 1 for our kernels (and hence the bandwidth $bw$ is the standard deviation of the kernel) and $R(K) = \int K^2(t) dt$.

MSE-equivalent bandwidths (for different kernels) are proportional to $\sigma_K R(K)$ which is scale invariant and for our kernels equal to $R(K)$. This value is returned when `give.Rkern = TRUE`. See the examples for using exact equivalent bandwidths.

Infinite values in $x$ are assumed to correspond to a point mass at $\pm \infty$ and the density estimate is of the sub-density on $(-\infty, +\infty)$.

Value

If `give.Rkern` is true, the number $R(K)$, otherwise an object with class "density" whose underlying structure is a list containing the following components.

- `x` the $n$ coordinates of the points where the density is estimated.
- `y` the estimated density values.
- `bw` the bandwidth used.
- `n` the sample size after elimination of missing values.
- `call` the call which produced the result.
- `data.name` the deparsed name of the $x$ argument.
- `has.na` logical, for compatibility (always FALSE).

References


See Also

`bw.nrd`, `plot.density`, `hist`.

Examples

```r
plot(density(c(-20, rep(0, 98), 20)), xlim = c(-4, 4))# IQR = 0
# The Old Faithful geyser data
d <- density(faithful$eruptions, bw = "sj")
d
plot(d)

plot(d, type = "n")
polygon(d, col = "wheat")

## Missing values:
x <- xx <- faithful$eruptions
data.frame(x = x, xx = xx)
```

deriv

```
x[i.out <- sample(length(x), 10)] <- NA
doR <- density(x, bw = 0.15, na.rm = TRUE)
lines(doR, col = "blue")
points(xx[i.out], rep(0.01, 10))

## Weighted observations:
fe <- sort(faithful$eruptions) # has quite a few non-unique values
## use 'counts / n' as weights:
dw <- density(unique(fe), weights = table(fe)/length(fe), bw = d$bw)
str(dw) # smaller n: only 126, but identical estimate:
stopifnot(all.equal(d[1:3], dw[1:3]))

(kernels <- eval(formals(density.default)$kernel))

## show the kernels in the R parametrization
plot(density(0, bw = 1), xlab = "", main="R's density() kernels with bw = 1")
for(i in 2:length(kernels))
  lines(density(0, bw = 1, kern = kernels[i]), col = i)
legend(1.5,.4, legend = kernels, col = seq(kernels), lty = 1, cex = .8, y.int = 1)

## show the kernels in the S parametrization
plot(density(0, from=-1.2, to=1.2, width=2, kern="gaussian"), type="l",
ylim = c(0, 1), xlab="", main="R's density() kernels with width = 1")
for(i in 2:length(kernels))
  lines(density(0, width=2, kern = kernels[i]), col = i)
legend(0.6, 1.0, legend = kernels, col = seq(kernels), lty = 1)

##-------- Semi-advanced theoretic from here on ---------------------------
(RKs <- cbind(sapply(kernels, function(k)density(kern = k, give.Rkern = TRUE))))
100*round(RKs["epanechnikov",]/RKs, 4) ## Efficiencies

bw <- bw.SJ(precip) ## sensible automatic choice
plot(density(precip, bw = bw),
  main = "same sd bandwidths, 7 different kernels")
for(i in 2:length(kernels))
  lines(density(precip, bw = bw, kern = kernels[i]), col = i)

## Bandwidth Adjustment for "Exactly Equivalent Kernels"
h.f <- sapply(kernels, function(k)density(kern = k, give.Rkern = TRUE))
(h.f <- (h.f["gaussian"] / h.f)^ .2)
## -> 1, 1.01, .995, 1.007,... close to 1 => adjustment barely visible..
plot(density(precip, bw = bw),
  main = "equivalent bandwidths, 7 different kernels")
for(i in 2:length(kernels))
  lines(density(precip, bw = bw, adjust = h.f[i], kern = kernels[i]),
       col = i)
legend(55, 0.035, legend = kernels, col = seq(kernels), lty = 1)
```

---

deriv

Symbolic and Algorithmic Derivatives of Simple Expressions
Description
Compute derivatives of simple expressions, symbolically.

Usage
\[
\text{D (expr, name)}
\]
\[
\text{deriv(expr, namevec, function.arg, tag = ".expr", hessian = FALSE)}
\]
\[
\text{deriv3(expr, namevec, function.arg, tag = ".expr", hessian = TRUE)}
\]

Arguments
- **expr**  
expression or call to be differentiated.
- **name, namevec**  
character vector, giving the variable names (only one for \text{D}()) with respect to which derivatives will be computed.
- **function.arg**  
If specified, a character vector of arguments for a function return, or a function (with empty body) or TRUE, the latter indicating that a function with argument names namevec should be used.
- **tag**  
character; the prefix to be used for the locally created variables in result.
- **hessian**  
a logical value indicating whether the second derivatives should be calculated and incorporated in the return value.

Details
\text{D} is modelled after its S namesake for taking simple symbolic derivatives.
\text{deriv} is a generic function with a default and a \text{formula} method. It returns a \text{call} for computing the expr and its (partial) derivatives, simultaneously. It uses so-called "algorithmic derivatives". If function.arg is a function, its arguments can have default values, see the \text{fx} example below.
Currently, \text{deriv.formula} just calls \text{deriv.default} after extracting the expression to the right of \text{}.
\text{deriv3} and its methods are equivalent to \text{deriv} and its methods except that \text{hessian} defaults to TRUE for \text{deriv3}.

Value
\text{D} returns a call and therefore can easily be iterated for higher derivatives.
\text{deriv} and \text{deriv3} normally return an \text{expression} object whose evaluation returns the function values with a "gradient" attribute containing the gradient matrix. If \text{hessian} is TRUE the evaluation also returns a "hessian" attribute containing the Hessian array.
If function.arg is specified, \text{deriv} and \text{deriv3} return a function with those arguments rather than an expression.

References
See Also

`nlm` and `optim` for numeric minimization which could make use of derivatives.

Examples

```r
## formula argument :
dx2x <- deriv(~ x^2, "x") ; dx2x
## Not run:
expression(
  .value <- x^2
  .grad <- array(0, c(length(.value), 1), list(NULL, c("x"))
  .grad[, "x"] <- 2 * x
  attr(.value, "gradient") <- .grad
  .value
)
## End(Not run)
mode(dx2x)
x <- -1:2
eval(dx2x)

## Something 'tougher':
trig.exp <- expression(sin(cos(x + y^2)))
( D.sc <- D(trig.exp, "x") )
all.equal(D(trig.exp[[1]], "x"), D.sc)

( dxy <- deriv(trig.exp, c("x", "y")) )
y <- 1
eval(dxy)
eval(D.sc)

## function returned:
deriv(y ~ sin(cos(x) * y)), c("x","y"), func = TRUE)

## function with defaulted arguments:
(fx <- deriv(y ~ b0 + b1 * 2^(-x/th), c("b0", "b1", "th"),
function(b0, b1, th, x = 1:7){} ) )
fx(2,3,4)

## Higher derivatives
deriv3(y ~ b0 + b1 * 2^(-x/th), c("b0", "b1", "th"),
c("b0", "b1", "th", "x") )

## Higher derivatives:
DD <- function(expr,name, order = 1) {
  if(order < 1) stop("'order' must be >= 1")
  if(order == 1) D(expr, name)
  else DD(D(expr, name), name, order - 1)
}
DD(expression(sin(x^2)), "x", 3)
## showing the limits of the internal "simplify()" :
## Not run:
-sin(x^2) * (2 * x) * 2 + ((cos(x^2) * (2 * x) * (2 * x) + sin(x^2) * 2) * (2 * x) + sin(x^2) * (2 * x) * 2)
## End(Not run)
```
deviance  

---

**Model Deviance**

**Description**

Returns the deviance of a fitted model object.

**Usage**

```r
deviance(object, ...)```

**Arguments**

- `object`  
  an object for which the deviance is desired.
- `...`  
  additional optional argument.

**Details**

This is a generic function which can be used to extract deviances for fitted models. Consult the individual modeling functions for details on how to use this function.

**Value**

The value of the deviance extracted from the object `object`.

**References**


**See Also**

df.residual, extractAIC, glm, lm.

---

df.residual  

---

**Residual Degrees-of-Freedom**

**Description**

Returns the residual degrees-of-freedom extracted from a fitted model object.

**Usage**

```r
df.residual(object, ...)```

**Arguments**

- `object`  
  an object for which the degrees-of-freedom are desired.
- `...`  
  additional optional arguments.
**Details**

This is a generic function which can be used to extract residual degrees-of-freedom for fitted models. Consult the individual modeling functions for details on how to use this function.

The default method just extracts the `df.residual` component.

**Value**

The value of the residual degrees-of-freedom extracted from the object `x`.

**See Also**

`deviance`, `glm`, `lm`.

---

### diffinv

**Discrete Integration: Inverse of Differencing**

**Description**

Computes the inverse function of the lagged differences function `diff`.

**Usage**

```r
diffinv(x, ...)  
## Default S3 method:  
diffinv(x, lag = 1, differences = 1, xi, ...)  
## S3 method for class 'ts':  
diffinv(x, lag = 1, differences = 1, xi, ...)  
```

**Arguments**

- `x` a numeric vector, matrix, or time series.
- `lag` a scalar lag parameter.
- `differences` an integer representing the order of the difference.
- `xi` a numeric vector, matrix, or time series containing the initial values for the integrals. If missing, zeros are used.
- `...` arguments passed to or from other methods.

**Details**

`diffinv` is a generic function with methods for class "ts" and default for vectors and matrices.

Missing values are not handled.

**Value**

A numeric vector, matrix, or time series (the latter for the "ts" method) representing the discrete integral of `x`. 
Author(s)
A. Trapletti

See Also
diff

Examples
s <- 1:10
d <- diff(s)
diffinv(d, xi = 1)

dist
Distance Matrix Computation

Description
This function computes and returns the distance matrix computed by using the specified distance
measure to compute the distances between the rows of a data matrix.

Usage
dist(x, method = "euclidean", diag = FALSE, upper = FALSE, p = 2)
as.dist(m, diag = FALSE, upper = FALSE)

## Default S3 method:
as.dist(m, diag = FALSE, upper = FALSE)
## S3 method for class 'dist':
print(x, diag = NULL, upper = NULL,
digits = getOption("digits"), justify = "none", right = TRUE, ...)
## S3 method for class 'dist':
as.matrix(x)

Arguments
x a numeric matrix, data frame or "dist" object.
method the distance measure to be used. This must be one of "euclidean",
"maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given.
diag logical value indicating whether the diagonal of the distance matrix should be
printed by print.dist.
upper logical value indicating whether the upper triangle of the distance matrix should
be printed by print.dist.
p The power of the Minkowski distance.
m An object with distance information to be converted to a "dist" object. For
the default method, a "dist" object, or a matrix (of distances) or an object
which can be coerced to such a matrix using as.matrix(). (Only the lower
triangle of the matrix is used, the rest is ignored).

digits, justify
passed to format inside of print().
right, ... further arguments, passed to the (next) print method.

Details

Available distance measures are (written for two vectors \(x\) and \(y\)):

- **euclidean**: Usual square distance between the two vectors (2 norm).
- **maximum**: Maximum distance between two components of \(x\) and \(y\) (supremum norm)
- **manhattan**: Absolute distance between the two vectors (1 norm).
- **canberra**: \(\sum_i |x_i - y_i| / |x_i + y_i|\). Terms with zero numerator and denominator are omitted from the sum and treated as if the values were missing.
- **binary**: (aka asymmetric binary): The vectors are regarded as binary bits, so non-zero elements are ‘on’ and zero elements are ‘off’. The distance is the proportion of bits in which only one is on amongst those in which at least one is on.
- **minkowski**: The \(p\) norm, the \(p\)th root of the sum of the \(p\)th powers of the differences of the components.

Missing values are allowed, and are excluded from all computations involving the rows within which they occur. Further, when \(\text{Inf}\) values are involved, all pairs of values are excluded when their contribution to the distance gave \(\text{NaN}\) or \(\text{NA}\). If some columns are excluded in calculating a Euclidean, Manhattan, Canberra or Minkowski distance, the sum is scaled up proportionally to the number of columns used. If all pairs are excluded when calculating a particular distance, the value is \(\text{NA}\).

The "dist" method of \(\text{as.matrix()}\) and \(\text{as.dist()}\) can be used for conversion between objects of class "dist" and conventional distance matrices.

\(\text{as.dist()}\) is a generic function. Its default method handles objects inheriting from class "dist", or coercible to matrices using \(\text{as.matrix()}\). Support for classes representing distances (also known as dissimilarities) can be added by providing an \(\text{as.matrix()}\) or, more directly, an \(\text{as.dist}\) method for such a class.

Value

dist returns an object of class "dist".

The lower triangle of the distance matrix stored by columns in a vector, say do. If \(n\) is the number of observations, i.e., \(n \leftarrow \text{attr}(\text{do}, \"\text{Size}\")\), then for \(i < j <= n\), the dissimilarity between \((\text{row})i\) and \(j\) is \(\text{do}[n*(i-1) - i*(i-1)/2 + j-i]\). The length of the vector is \(n*(n-1)/2\), i.e., of order \(n^2\).

The object has the following attributes (besides "class" equal to "dist"):

- **Size** integer, the number of observations in the dataset.
- **Labels** optionally, contains the labels, if any, of the observations of the dataset.
- **Diag, Upper** logicals corresponding to the arguments \(\text{diag}\) and \(\text{upper}\) above, specifying how the object should be printed.
- **call** optionally, the \(\text{call}\) used to create the object.
- **method** optionally, the distance method used; resulting from \(\text{dist()}\), the \(\text{(match.arg()ed) method argument}\).
References


See Also

`daisy` in the `cluster` package with more possibilities in the case of mixed (continuous / categorical) variables, `hclust`.

Examples

```r
x <- matrix(rnorm(100), nrow=5)
dist(x)
dist(x, diag = TRUE)
dist(x, upper = TRUE)
m <- as.matrix(dist(x))
d <- as.dist(m)
stopifnot(d == dist(x))

## Use correlations between variables `as distance`

dd <- as.dist((1 - cor(USJudgeRatings))/2)
round(1000 * dd) # prints more nicely
plot(hclust(dd)) # to see a dendrogram of clustered variables

## example of binary and canberra distances.
x <- c(0, 0, 1, 1, 1)
y <- c(1, 0, 1, 1, 0)
dist(rbind(x,y), method= "binary")
## answer 0.4 = 2/5

dist(rbind(x,y), method= "canberra")
## answer 2 * (6/5)

## To find the names
labels(eurodist)

## Examples involving "Inf":

## 1)
x[6] <- Inf
(m2 <- rbind(x,y))
dist(m2, method="binary") # warning, answer 0.5 = 2/4

##These all give "Inf":
stopifnot(Inf == dist(m2, method= "euclidean"),
Inf == dist(m2, method= "maximum"),
Inf == dist(m2, method= "manhattan") )

## "Inf" is same as very large number:
x1 <- x; x1[6] <<- 1e100
stopifnot(dist(cbind(x ,y), method="canberra") ==
          print(dist(cbind(x1,y), method="canberra")))

## 2)
y[6] <- Inf #-> 6-th pair is excluded

dist(rbind(x,y), method="binary") # warning; 0.5
```
The `dummy.coef` function extracts coefficients in terms of the original levels of the coefficients rather than the coded variables.

**Usage**

```r
dummy.coef(object, ...)  
## S3 method for class 'lm':  
dummy.coef(object, use.na = FALSE, ...)  
## S3 method for class 'aovlist':  
dummy.coef(object, use.na = FALSE, ...)```

**Arguments**

- `object`: a linear model fit.
- `use.na`: logical flag for coefficients in a singular model. If `use.na` is true, undetermined coefficients will be missing; if false they will get one possible value.
- `...`: arguments passed to or from other methods.

**Details**

A fitted linear model has coefficients for the contrasts of the factor terms, usually one less in number than the number of levels. This function re-expresses the coefficients in the original coding; as the coefficients will have been fitted in the reduced basis, any implied constraints (e.g., zero sum for `contr.helmert` or `contr.sum` will be respected. There will be little point in using `dummy.coef` for `contr.treatment` contrasts, as the missing coefficients are by definition zero.

The method used has some limitations, and will give incomplete results for terms such as `poly(x, 2)`.

**Value**

A list giving for each term the values of the coefficients. For a multistratum `aov` model, such a list for each stratum.

**Warning**

This function is intended for human inspection of the output: it should not be used for calculations. Use coded variables for all calculations.

The results differ from S for singular values, where S can be incorrect.
See Also

aov, model.tables

Examples

options(contrasts = c("contr.helmert", "contr.poly"))
N <- c(0,1,0,1,1,1,0,0,0,1,0,1,0,1,1,0,0,1,0,1,0,1,0,1,0)
P <- c(1,1,0,0,1,0,1,1,1,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0)
K <- c(1,0,0,1,0,1,1,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0)
yield <- c(49.5, 62.8, 46.8, 57.0, 59.8, 58.5, 55.5, 56.0, 62.8, 55.8, 69.5,
55.0, 62.0, 48.8, 45.5, 44.2, 52.0, 51.5, 49.8, 48.8, 57.2, 59.0, 53.2, 56.0)
npk <- data.frame(block = gl(6,4), N = factor(N), P = factor(P),
                 K = factor(K), yield = yield)
npk.aov <- aov(yield ~ block + N*P*K, npk)
dummy.coef(npk.aov)
npk.aovE <- aov(yield ~ N*P*K + Error(block), npk)
dummy.coef(npk.aovE)

ecdf

Empirical Cumulative Distribution Function

Description

Compute or plot an empirical cumulative distribution function.

Usage

ecdf(x)

## S3 method for class 'ecdf':
plot(x, ..., ylab="Fn(x)", verticals = FALSE,
     col.01line = "gray70")

## S3 method for class 'ecdf':
print(x, digits = getOption("digits") - 2, ...)

Arguments

x numeric vector of “observations” in ecdf; for the methods, an object of class "ecdf", typically.
...
arguments to be passed to subsequent methods, i.e., plot.stepfun for the
plot method.
ylab label for the y-axis.
verticals see plot.stepfun.
col.01line numeric or character specifying the color of the horizontal lines at y = 0 and 1,
see colors.
digits number of significant digits to use, see print.
Details

The e.c.d.f. (empirical cumulative distribution function) $F_n$ is a step function with jumps $i/n$ at observation values, where $i$ is the number of tied observations at that value. Missing values are ignored.

For observations $x = (x_1, x_2, \ldots, x_n)$, $F_n$ is the fraction of observations less or equal to $t$, i.e.,

$$F_n(t) = {\#\{x_i \leq t\} / n} = \frac{1}{n} \sum_{i=1}^{n} 1[x_i \leq t].$$

The function `plot.ecdf` which implements the `plot` method for `ecdf` objects, is implemented via a call to `plot.stepfun`; see its documentation.

Value

For `ecdf`, a function of class "ecdf", inheriting from the "stepfun" class.

Warning

Prior to R 2.1.0, `ecdf` treated ties differently, so had multiple jumps of size $1/n$ at tied observations. This was not the most common definition, and could be very slow for large datasets with many ties.

Author(s)

Martin Maechler, ⟨maechler@stat.math.ethz.ch⟩.
Corrections by R-core.

See Also

`stepfun`, the more general class of step functions, `approxfun` and `splinefun`.

Examples

```r
##-- Simple didactical ecdf example:
Fn <- ecdf(rnorm(12))
Fn; summary(Fn)
12*Fn(knots(Fn)) == 1:12 ## == 1:12 if and only if there are no ties !

y <- round(rnorm(12),1); y[3] <- y[1]
Fn12 <- ecdf(y)
Fn12
print(knots(Fn12), dig=2)
12*Fn12(knots(Fn12)) ## =~ 1:12 if there were no ties
summary(Fn12)
summary.stepfun(Fn12)
print(ls.Fn12 <- ls(env= environment(Fn12)))
##[1] "f"  "method"  "n"  "x"  "y"  "yleft"  "yright"
12 * Fn12((-20:20)/10)

###----------------- Plotting --------------------------

op <- par(mfrow=c(3,1), mgp=c(1.5, 0.8,0), mar= .1+c(3,3,2,1))
```
### eff.aovlist

*Compute Efficiencies of Multistratum Analysis of Variance*

**Description**

Computes the efficiencies of fixed-effect terms in an analysis of variance model with multiple strata.

**Usage**

```r
eff.aovlist(aovlist)
```

**Arguments**

- `aovlist` The result of a call to `aov` with an `Error` term.

**Details**

Fixed-effect terms in an analysis of variance model with multiple strata may be estimable in more than one stratum, in which case there is less than complete information in each. The efficiency for a term is the fraction of the maximum possible precision (inverse variance) obtainable by estimating in just that stratum. Under the assumption of balance, this is the same for all contrasts involving that term.

This function is used to pick strata in which to estimate terms in `model.tables.aovlist` and `se.contrast.aovlist`.

In many cases terms will only occur in one stratum, when all the efficiencies will be one: this is detected and no further calculations are done.

The calculation used requires orthogonal contrasts for each term, and will throw an error if non-orthogonal contrasts (e.g. treatment contrasts or an unbalanced design) are detected.

**Value**

A matrix giving for each non-pure-error stratum (row) the efficiencies for each fixed-effect term in the model.
effects

References


See Also

`aov`, `model.tables.aovlist`, `se.contrast.aovlist`

Examples

```r
## An example of Yates (1932), a 2^3 design in 2 blocks replicated 4 times
Block <- gl(8, 4)
A <- factor(c(0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1))
B <- factor(c(0,0,1,1,0,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1))
C <- factor(c(0,1,1,0,1,0,0,1,0,0,1,1,0,0,1,1,0,1,0,1,0,1,0,1,0,1,1,1))
Yield <- c(101, 373, 398, 291, 312, 106, 265, 450, 106, 306, 324, 449,
          272, 89, 407, 338, 87, 324, 279, 471, 323, 128, 423, 334,
          131, 103, 445, 437, 324, 361, 302, 272)
aovdat <- data.frame(Block, A, B, C, Yield)
old <- getOption("contrasts")
options(contrasts=c("contr.helmert", "contr.poly"))
(fit <- aov(Yield ~ A*B*C + Error(Block), data = aovdat))
eff.aovlist(fit)
options(contrasts = old)
```

---

effects

### Effects from Fitted Model

Returns (orthogonal) effects from a fitted model, usually a linear model. This is a generic function, but currently only has a methods for objects inheriting from classes "lm" and "glm".

#### Usage

```r
effects(object, ...)
```

#### Arguments

- `object`: an R object; typically, the result of a model fitting function such as `lm`.
- `set.sign`: logical. If TRUE, the sign of the effects corresponding to coefficients in the model will be set to agree with the signs of the corresponding coefficients. Otherwise the sign is arbitrary.
- `...`: arguments passed to or from other methods.
Details

For a linear model fitted by `lm` or `aov`, the effects are the uncorrelated single-degree-of-freedom values obtained by projecting the data onto the successive orthogonal subspaces generated by the QR decomposition during the fitting process. The first \( r \) (the rank of the model) are associated with coefficients and the remainder span the space of residuals (but are not associated with particular residuals).

Empty models do not have effects.

Value

A (named) numeric vector of the same length as `residuals`, or a matrix if there were multiple responses in the fitted model, in either case of class "coef".

The first \( r \) rows are labelled by the corresponding coefficients, and the remaining rows are unlabelled. Note that in rank-deficient models the “corresponding” coefficients will be in a different order if pivoting occurred.

References


See Also

`coef`

Examples

```r
y <- c(1:3, 7, 5)
x <- c(1:3, 6:7)
( ee <- effects(lm(y ~ x)) )
c(round(ee - effects(lm(y+10 ~ I(x-3.8))),3)) # just the first is different
```

---

**embed**

*Embedding a Time Series*

Description

Embeds the time series \( x \) into a low-dimensional Euclidean space.

Usage

```r
embed (x, dimension = 1)
```

Arguments

- `x` : a numeric vector, matrix, or time series.
- `dimension` : a scalar representing the embedding dimension.

Details

Each row of the resulting matrix consists of sequences \( x[t], x[t-1], \ldots, x[t-dimension+1] \), where \( t \) is the original index of \( x \). If \( x \) is a matrix, i.e., \( x \) contains more than one variable, then \( x[t] \) consists of the \( t \)th observation on each variable.
Value

A matrix containing the embedded time series $x$.

Author(s)

A. Trapletti, B.D. Ripley

Examples

```r
x <- 1:10
embed(x, 3)
```

---

### expand.model.frame

Add new variables to a model frame

**Description**

Evaluates new variables as if they had been part of the formula of the specified model. This ensures that the same `na.action` and `subset` arguments are applied and allows, for example, $x$ to be recovered for a model using $\sin(x)$ as a predictor.

**Usage**

```r
expand.model.frame(model, extras,
                   envir = environment(formula(model)),
                   na.expand = FALSE)
```

**Arguments**

- `model` a fitted model
- `extras` one-sided formula or vector of character strings describing new variables to be added
- `envir` an environment to evaluate things in
- `na.expand` logical; see below

**Details**

If `na.expand=FALSE` then NA values in the extra variables will be passed to the `na.action` function used in `model`. This may result in a shorter data frame (with `na.omit`) or an error (with `na.fail`). If `na.expand=TRUE` the returned data frame will have precisely the same rows as `model.frame(model)`, but the columns corresponding to the extra variables may contain NA.

**Value**

A data frame.

**See Also**

- `model.frame`
- `predict`
Exponential

Examples

```r
model <- lm(log(Volume) ~ log(Girth) + log(Height), data=trees)
expand.model.frame(model, ~ Girth) # prints data.frame like

dd <- data.frame(x=1:5, y=rnorm(5), z=c(1,2,NA,4,5))
model <- glm(y ~ x, data=dd, subset=1:4, na.action=na.omit)
expand.model.frame(model, "z", na.expand=FALSE) # = default
expand.model.frame(model, "z", na.expand=TRUE)
```

Exponential

The Exponential Distribution

Description

Density, distribution function, quantile function and random generation for the exponential distribution with rate rate (i.e., mean 1/rate).

Usage

```r
dexp(x, rate = 1, log = FALSE)
pexp(q, rate = 1, lower.tail = TRUE, log.p = FALSE)
qexp(p, rate = 1, lower.tail = TRUE, log.p = FALSE)
rexp(n, rate = 1)
```

Arguments

- `x, q` vector of quantiles.
- `p` vector of probabilities.
- `n` number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `rate` vector of rates.
- `log, log.p` logical; if TRUE, probabilities p are given as log(p).
- `lower.tail` logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

Details

If `rate` is not specified, it assumes the default value of 1.

The exponential distribution with rate $\lambda$ has density

$$f(x) = \lambda e^{-\lambda x}$$

for $x \geq 0$.

Value

dexp gives the density, pexp gives the distribution function, qexp gives the quantile function, and rexp generates random deviates.
Note
The cumulative hazard \( H(t) = -\log(1 - F(t)) \) is \(-\text{pexp}(t, r, \text{lower} = \text{FALSE}, \text{log} = \text{TRUE})\).

References

See Also
exp for the exponential function, dgamma for the gamma distribution and dweibull for the Weibull distribution, both of which generalize the exponential.

Examples
\[
\text{dexp}(1) - \exp(-1) \rightarrow 0
\]

Description
Perform maximum-likelihood factor analysis on a covariance matrix or data matrix.

Usage
\[
factanal(x, \text{factors}, \text{data} = \text{NULL}, \text{covmat} = \text{NULL}, \text{n.obs} = \text{NA}, \text{subset}, \text{na.action}, \\
\text{start} = \text{NULL}, \text{scores} = \text{c}("\text{none}", "\text{regression}" , "\text{Bartlett}" ), \text{rotation} = "\text{varimax}" , \text{control} = \text{NULL}, \ldots)
\]

Arguments
- **x**: A formula or a numeric matrix or an object that can be coerced to a numeric matrix.
- **factors**: The number of factors to be fitted.
- **data**: A data frame, used only if \( x \) is a formula.
- **covmat**: A covariance matrix, or a covariance list as returned by \text{cov.wt}. Of course, correlation matrices are covariance matrices.
- **n.obs**: The number of observations, used if \text{covmat} is a covariance matrix.
- **subset**: A specification of the cases to be used, if \( x \) is used as a matrix or formula.
- **na.action**: The \text{na.action} to be used if \( x \) is used as a formula.
- **start**: NULL or a matrix of starting values, each column giving an initial set of unique-nesses.
- **scores**: Type of scores to produce, if any. The default is none, "regression" gives Thompson's scores, "Bartlett" given Bartlett's weighted least-squares scores. Partial matching allows these names to be abbreviated.
rotation character. "none" or the name of a function to be used to rotate the factors: it will be called with first argument the loadings matrix, and should return a list with component loadings giving the rotated loadings, or just the rotated loadings.

control A list of control values,

- nstart The number of starting values to be tried if start = NULL. Default 1.
- trace logical. Output tracing information? Default FALSE.
- lower The lower bound for uniquenesses during optimization. Should be > 0. Default 0.005.
- opt A list of control values to be passed to optim’s control argument.
- rotate a list of additional arguments for the rotation function.

... Components of control can also be supplied as named arguments to factanal.

Details

The factor analysis model is

\[ x = \Lambda f + e \]

for a \( p \)-element row-vector \( x \), a \( p \times k \) matrix of loadings, a \( k \)-element vector of scores and a \( p \)-element vector of errors. None of the components other than \( x \) is observed, but the major restriction is that the scores be uncorrelated and of unit variance, and that the errors be independent with variances \( \Phi \), the uniquenesses. Thus factor analysis is in essence a model for the covariance matrix of \( x \),

\[ \Sigma = \Lambda' \Lambda + \Psi \]

There is still some indeterminacy in the model for it is unchanged if \( \Lambda \) is replaced by \( G\Lambda \) for any orthogonal matrix \( G \). Such matrices \( G \) are known as rotations (although the term is applied also to non-orthogonal invertible matrices).

If covmat is supplied it is used. Otherwise \( x \) is used if it is a matrix, or a formula \( x \) is used with data to construct a model matrix, and that is used to construct a covariance matrix. (It makes no sense for the formula to have a response, and all the variables must be numeric.) Once a covariance matrix is found or calculated from \( x \), it is converted to a correlation matrix for analysis. The correlation matrix is returned as component correlation of the result.

The fit is done by optimizing the log likelihood assuming multivariate normality over the uniquenesses. (The maximizing loadings for given uniquenesses can be found analytically: Lawley & Maxwell (1971, p. 27).) All the starting values supplied in start are tried in turn and the best fit obtained is used. If start = NULL then the first fit is started at the value suggested by Jöreskog (1963) and given by Lawley & Maxwell (1971, p. 31), and then control$nstart - 1 other values are tried, randomly selected as equal values of the uniquenesses.

The uniquenesses are technically constrained to lie in \([0, 1]\), but near-zero values are problematical, and the optimization is done with a lower bound of control$lower, default 0.005 (Lawley & Maxwell, 1971, p. 32).

Scores can only be produced if a data matrix is supplied and used. The first method is the regression method of Thomson (1951), the second the weighted least squares method of Bartlett (1937, 8). Both are estimates of the unobserved scores \( f \). Thomson’s method regresses (in the population) the unknown \( f \) on \( x \) to yield

\[ \hat{f} = \Lambda'\Sigma^{-1}x \]

and then substitutes the sample estimates of the quantities on the right-hand side. Bartlett’s method minimizes the sum of squares of standardized errors over the choice of \( f \), given (the fitted) \( \Lambda \).
If \( x \) is a formula then the standard NA-handling is applied to the scores (if requested): see \texttt{napredict}.

**Value**

An object of class "\texttt{factanal}" with components

- \texttt{loadings}\hspace{1em}A matrix of loadings, one column for each factor. The factors are ordered in decreasing order of sums of squares of loadings, and given the sign that will make the sum of the loadings positive.
- \texttt{uniquenesses}\hspace{1em}The uniquenesses computed.
- \texttt{correlation}\hspace{1em}The correlation matrix used.
- \texttt{criteria}\hspace{1em}The results of the optimization: the value of the negative log-likelihood and information on the iterations used.
- \texttt{factors}\hspace{1em}The argument \texttt{factors}.
- \texttt{dof}\hspace{1em}The number of degrees of freedom of the factor analysis model.
- \texttt{method}\hspace{1em}The method: always "\texttt{mle}".
- \texttt{scores}\hspace{1em}If requested, a matrix of scores.
- \texttt{n.obs}\hspace{1em}The number of observations if available, or \texttt{NA}.
- \texttt{call}\hspace{1em}The matched call.
- \texttt{na.action}\hspace{1em}If relevant.
- \texttt{STATISTIC, PVAL}\hspace{1em}The significance-test statistic and P value, if if can be computed.

**Note**

There are so many variations on factor analysis that it is hard to compare output from different programs. Further, the optimization in maximum likelihood factor analysis is hard, and many other examples we compared had less good fits than produced by this function. In particular, solutions which are Heywood cases (with one or more uniquenesses essentially zero) are much often common than most texts and some other programs would lead one to believe.

**References**


**See Also**

\texttt{print.loadings}, \texttt{varimax}, \texttt{princomp}, \texttt{ability.cov}, \texttt{Harman23.cor}, \texttt{Harman74.cor}
Examples

# A little demonstration, v2 is just v1 with noise,
# and same for v4 vs. v3 and v6 vs. v5
# Last four cases are there to add noise
# and introduce a positive manifold (g factor)
v1 <- c(1,1,1,1,1,1,1,1,1,1,3,3,3,3,3,4,5,6)
v2 <- c(1,2,1,1,1,1,2,1,2,1,3,4,3,3,3,4,6,5)
v3 <- c(3,3,3,3,3,1,1,1,1,1,1,1,1,1,1,5,4,6)
v4 <- c(3,3,4,3,3,1,1,2,1,1,1,2,1,1,1,5,6,4)
v5 <- c(1,1,1,1,1,3,3,3,3,3,1,1,1,1,1,6,4,5)
v6 <- c(1,1,1,2,1,3,3,3,4,3,1,1,1,2,1,6,5,4)
m1 <- cbind(v1,v2,v3,v4,v5,v6)
cor(m1)
factanal(m1, factors=3) # varimax is the default
factanal(m1, factors=3, rotation="promax")
# The following shows the g factor as PC1
prcomp(m1)

## formula interface
factanal(~v1+v2+v3+v4+v5+v6, factors = 3,
    scores = "Bartlett")$scores

## a realistic example from Barthlomew (1987, pp. 61-65)
example(ability.cov)

factor.scope

Compute Allowed Changes in Adding to or Dropping from a Formula

Description

add.scope and drop.scope compute those terms that can be individually added to or dropped from a model while respecting the hierarchy of terms.

Usage

add.scope(terms1, terms2)

drop.scope(terms1, terms2)

factor.scope(factor, scope)

Arguments

terms1 

terms2 

the terms or formula for the base model.

the terms or formula for the upper (add.scope) or lower (drop.scope) scope. If missing for drop.scope it is taken to be the null formula, so all terms (except any intercept) are candidates to be dropped.

factor 

the "factor" attribute of the terms of the base object.

scope 

a list with one or both components drop and add giving the "factor" attribute of the lower and upper scopes respectively.
Details

factor.scope is not intended to be called directly by users.

Value

For add.scope and drop.scope a character vector of terms labels. For factor.scope, a list with components drop and add, character vectors of terms labels.

See Also

add1, drop1, aov, lm

Examples

add.scope(~ a + b + c + a:b, ~ (a + b + c)^3)
  # [1] "a:c"  "b:c"

drop.scope(~ a + b + c + a:b)
  # [1] "c"  "a:b"

family

Family Objects for Models

Description

Family objects provide a convenient way to specify the details of the models used by functions such as glm. See the documentation for glm for the details on how such model fitting takes place.

Usage

family(object, ...)

binomial(link = "logit")
gaussian(link = "identity")
Gamma(link = "inverse")
inverse.gaussian(link = "1/mu^2")
poisson(link = "log")
quasi(link = "identity", variance = "constant")
quasibinomial(link = "logit")
quasipoisson(link = "log")

Arguments

link a specification for the model link function. The gaussian family accepts the links "identity", "log" and "inverse"; the binomial family the links "logit", "probit", "cauchit", (corresponding to logistic, normal and Cauchy CDFs respectively) "log" and "cloglog" (complementary log-log); the Gamma family the links "inverse", "identity" and "log"; the poisson family the links "log", "identity", and "sqrt" and the inverse.gaussian family the links "1/mu^2", "inverse", "identity" and "log".
The quasi family allows the links "logit", "probit", "cloglog", "identity", "inverse", "log", "1/mu^2" and "sqrt". The function power can also be used to create a power link function for the quasi family.

**variance** for all families, other than quasi, the variance function is determined by the family. The quasi family will accept the specifications "constant", "mu(1-mu)", "mu", "mu^2" and "mu^3" for the variance function.

**object** the function family accesses the family objects which are stored within objects created by modelling functions (e.g., glm).

... further arguments passed to methods.

### Details

The quasibinomial and quasipoisson families differ from the binomial and poisson families only in that the dispersion parameter is not fixed at one, so they can “model” over-dispersion. For the binomial case see McCullagh and Nelder (1989, pp. 124–8). Although they show that there is (under some restrictions) a model with variance proportional to mean as in the quasi-binomial model, note that glm does not compute maximum-likelihood estimates in that model. The behaviour of S is closer to the quasi-variants.

### Author(s)

The design was inspired by S functions of the same names described in Hastie & Pregibon (1992).

### References


### See Also

*glm*, *power*.

### Examples

```r
nf <- gaussian() # Normal family
nf
str(nf)# internal STRucture

gf <- Gamma()
gf
str(gf)
g$f$linkinv
g$f$variance(-3:4) # = (.)^2

## quasipoisson. compare with example(glm)
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
```
The F Distribution

Description

Density, distribution function, quantile function and random generation for the F distribution with 
df1 and df2 degrees of freedom (and optional non-centrality parameter ncp).

Usage

df(x, df1, df2, log = FALSE)
pf(q, df1, df2, ncp=0, lower.tail = TRUE, log.p = FALSE)
qf(p, df1, df2, lower.tail = TRUE, log.p = FALSE)
rf(n, df1, df2)

Arguments

x, q  vector of quantiles.
p  vector of probabilities.
n  number of observations. If length(n) > 1, the length is taken to be the number required.
df1, df2  degrees of freedom. Inf is allowed.
ncp  non-centrality parameter.
log, log.p  logical; if TRUE, probabilities p are given as log(p).
lower.tail  logical; if TRUE (default), probabilities are P[X ≤ x], otherwise, P[X > x].
Details

The F distribution with \( df1 = n_1 \) and \( df2 = n_2 \) degrees of freedom has density

\[
f(x) = \frac{\Gamma(n_1/2 + n_2/2)}{\Gamma(n_1/2)\Gamma(n_2/2)} \left( \frac{n_1}{n_2} \right)^{n_1/2} x^{n_1/2-1} \left( 1 + \frac{n_1x}{n_2} \right)^{-(n_1+n_2)/2}
\]

for \( x > 0 \).

It is the distribution of the ratio of the mean squares of \( n_1 \) and \( n_2 \) independent standard normals, and hence of the ratio of two independent chi-squared variates each divided by its degrees of freedom. Since the ratio of a normal and the root mean-square of \( m \) independent normals has a Student’s \( t_m \) distribution, the square of a \( t_m \) variate has a F distribution on 1 and \( m \) degrees of freedom.

The non-central F distribution is again the ratio of mean squares of independent normals of unit variance, but those in the numerator are allowed to have non-zero means and \( ncp \) is the sum of squares of the means. See \texttt{Chisquare} for further details on non-central distributions.

Value

\( df \) gives the density, \( pf \) gives the distribution function \( qf \) gives the quantile function, and \( rf \) generates random deviates.

References


See Also

\texttt{dchisq} for chi-squared and \texttt{dt} for Student’s \( t \) distributions.

Examples

```r
## the density of the square of a \( t_m \) is 2 * dt(x, m)/(2 * x)
# check this is the same as the density of F_{1,m}
x <- seq(0.001, 5, len=100)
all.equal(df(x^2, 1, 5), dt(x, 5)/x)
## Identity: qf(2 * p - 1, 1, df)) == qt(p, df)^2) for p >= 1/2
p <- seq(1/2, .99, length=50); df <- 10
rel.err <- function(x,y) ifelse(x==y,0, abs(x-y)/mean(abs(c(x,y)))
quantile(rel.err(qf(2*p - 1, 1, df1=1, df2=df), qt(p, df)^2), .90)# ~= 7e-9
```

fft

Fast Discrete Fourier Transform

Description

Performs the Fast Fourier Transform of an array.

Usage

```r
fft(z, inverse = FALSE)
```

```r
mvfft(z, inverse = FALSE)
```
Arguments

z a real or complex array containing the values to be transformed.

inverse if TRUE, the unnormalized inverse transform is computed (the inverse has a + in the exponent of e, but here, we do not divide by 1/length(x)).

Value

When z is a vector, the value computed and returned by fft is the unnormalized univariate Fourier transform of the sequence of values in z. When z contains an array, fft computes and returns the multivariate (spatial) transform. If inverse is TRUE, the (unnormalized) inverse Fourier transform is returned, i.e., if y <- fft(z), then z is fft(y, inverse = TRUE) / length(y).

By contrast, mvfft takes a real or complex matrix as argument, and returns a similar shaped matrix, but with each column replaced by its discrete Fourier transform. This is useful for analyzing vector-valued series.

The FFT is fastest when the length of the series being transformed is highly composite (i.e., has many factors). If this is not the case, the transform may take a long time to compute and will use a large amount of memory.

References


See Also

convolve, nextn.

Examples

x <- 1:4
fft(x)
fft(fft(x), inverse = TRUE)/length(x)

filter Linear Filtering on a Time Series

Description

Applies linear filtering to a univariate time series or to each series separately of a multivariate time series.

Usage

filter(x, filter, method = c("convolution", "recursive"),
      sides = 2, circular = FALSE, init)
Arguments

- **x**: a univariate or multivariate time series.
- **filter**: a vector of filter coefficients in reverse time order (as for AR or MA coefficients).
- **method**: Either "convolution" or "recursive" (and can be abbreviated). If "convolution" a moving average is used: if "recursive" an autoregression is used.
- **sides**: for convolution filters only. If `sides=1` the filter coefficients are for past values only; if `sides=2` they are centred around lag 0. In this case the length of the filter should be odd, but if it is even, more of the filter is forward in time than backward.
- **circular**: for convolution filters only. If `TRUE`, wrap the filter around the ends of the series, otherwise assume external values are missing (NA).
- **init**: for recursive filters only. Specifies the initial values of the time series just prior to the start value, in reverse time order. The default is a set of zeros.

Details

Missing values are allowed in `x` but not in `filter` (where they would lead to missing values everywhere in the output).

Note that there is an implied coefficient 1 at lag 0 in the recursive filter, which gives

\[ y_i = x_i + f_1 y_{i-1} + \cdots + f_p y_{i-p} \]

No check is made to see if recursive filter is invertible: the output may diverge if it is not.

The convolution filter is

\[ y_i = f_1 x_{i+o} + \cdots + f_p x_{i+o-(p-1)} \]

where `o` is the offset: see `sides` for how it is determined.

Value

A time series object.

Note

`convolve`, `type="filter"`) uses the FFT for computations and so may be faster for long filters on univariate series, but it does not return a time series (and so the time alignment is unclear), nor does it handle missing values. `filter` is faster for a filter of length 100 on a series of length 1000, for example.

See Also

`convolve`, `arima.sim`

Examples

```r
x <- 1:100
filter(x, rep(1, 3))
filter(x, rep(1, 3), sides = 1)
filter(x, rep(1, 3), sides = 1, circular = TRUE)
filter(presidents, rep(1,3))
```
Description

Performs Fisher’s exact test for testing the null of independence of rows and columns in a contingency table with fixed marginals.

Usage

fisher.test(x, y = NULL, workspace = 200000, hybrid = FALSE, control = list(), or = 1, alternative = "two.sided", conf.int = TRUE, conf.level = 0.95)

Arguments

- `x`: either a two-dimensional contingency table in matrix form, or a factor object.
- `y`: a factor object; ignored if `x` is a matrix.
- `workspace`: an integer specifying the size of the workspace used in the network algorithm.
- `hybrid`: a logical indicating whether the exact probabilities (default) or a hybrid approximation thereof should be computed. In the hybrid case, asymptotic chi-squared probabilities are only used provided that the “Cochran” conditions are satisfied.
- `control`: a list with named components for low level algorithm control.
- `or`: the hypothesized odds ratio. Only used in the 2 by 2 case.
- `alternative`: indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. Only used in the 2 by 2 case.
- `conf.int`: logical indicating if a confidence interval should be computed (and returned).
- `conf.level`: confidence level for the returned confidence interval. Only used in the 2 by 2 case.

Details

If `x` is a matrix, it is taken as a two-dimensional contingency table, and hence its entries should be nonnegative integers. Otherwise, both `x` and `y` must be vectors of the same length. Incomplete cases are removed, the vectors are coerced into factor objects, and the contingency table is computed from these.

In the one-sided 2 by 2 cases, p-values are obtained directly using the hypergeometric distribution. Otherwise, computations are based on a C version of the FORTRAN subroutine FEXACT which implements the network developed by Mehta and Patel (1986) and improved by Clarkson, Fan & Joe (1993). The FORTRAN code can be obtained from http://www.netlib.org/toms/643. Note this fails (with an error message) when the entries of the table are too large.

In the 2 by 2 case, the null of conditional independence is equivalent to the hypothesis that the odds ratio equals one. Exact inference can be based on observing that in general, given all marginal totals fixed, the first element of the contingency table has a non-central hypergeometric distribution with non-centrality parameter given by the odds ratio (Fisher, 1935).
fisher.test

Value

A list with class "htest" containing the following components:

- **p.value**  the p-value of the test.
- **conf.int**  a confidence interval for the odds ratio. Only present in the 2 by 2 case.
- **estimate** an estimate of the odds ratio. Note that the *conditional* Maximum Likelihood Estimate (MLE) rather than the unconditional MLE (the sample odds ratio) is used. Only present in the 2 by 2 case.
- **null.value** the odds ratio under the null, or. Only present in the 2 by 2 case.
- **alternative** a character string describing the alternative hypothesis.
- **method**  the character string "Fisher’s Exact Test for Count Data".
- **data.name** a character string giving the names of the data.

References


See Also

cishq.test

Examples

```r
## Agresti (1990), p. 61f, Fisher’s Tea Drinker
## A British woman claimed to be able to distinguish whether milk or
## tea was added to the cup first. To test, she was given 8 cups of
tea, in four of which milk was added first. The null hypothesis
## is that there is no association between the true order of pouring
## and the women's guess, the alternative that there is a positive
## association (that the odds ratio is greater than 1).
TeaTasting <-
matrix(c(3, 1, 1, 3),
   nr = 2,
   dimnames = list(Guess = c("Milk", "Tea"),
                   Truth = c("Milk", "Tea")))
fisher.test(TeaTasting, alternative = "greater")
## => p=0.2429, association could not be established

## Fisher (1962), Convictions of like-sex twins in criminals
Convictions <-
matrix(c(2, 10, 15, 3),
```
```r
nr = 2,
dimnames =
  list(c("Dizygotic", "Monozygotic"),
       c("Convicted", "Not convicted"))
Convictions
fisher.test(Convictions, alternative = "less")
fisher.test(Convictions, conf.int = FALSE)
fisher.test(Convictions, conf.level = 0.95)$conf.int
fisher.test(Convictions, conf.level = 0.99)$conf.int
```

---

# fitted

## Extract Model Fitted Values

### Description

`fitted` is a generic function which extracts fitted values from objects returned by modeling functions. `fitted.values` is an alias for it.

All object classes which are returned by model fitting functions should provide a `fitted` method. (Note that the generic is `fitted` and not `fitted.values`.)

Methods can make use of `napredict` methods to compensate for the omission of missing values. The default, `lm` and `glm` methods do.

### Usage

```r
fitted(object, ...)
fitted.values(object, ...)
```

### Arguments

- `object` an object for which the extraction of model fitted values is meaningful.
- `...` other arguments.

### Value

Fitted values extracted from the object `x`.

### References


### See Also

`coefficients, glm, lm, residuals`. 
fivenum  

Tukey Five-Number Summaries

Description

Returns Tukey’s five number summary (minimum, lower-hinge, median, upper-hinge, maximum) for the input data.

Usage

fivenum(x, na.rm = TRUE)

Arguments

x numeric, maybe including NAs and ±Infs.
na.rm logical; if TRUE, all NA and NaNs are dropped, before the statistics are computed.

Value

A numeric vector of length 5 containing the summary information. See boxplot.stats for more details.

See Also

IQR, boxplot.stats, median, quantile, range.

Examples

fivenum(c(rnorm(100),-1:1/0))

fligner.test  

Fligner-Killeen Test of Homogeneity of Variances

Description

Performs a Fligner-Killeen (median) test of the null that the variances in each of the groups (samples) are the same.

Usage

fligner.test(x, ...)

## Default S3 method:
fligner.test(x, g, ...)

## S3 method for class 'formula':
fligner.test(formula, data, subset, na.action, ...)
**fligner.test**

Arguments

- **x**: a numeric vector of data values, or a list of numeric data vectors.
- **g**: a vector or factor object giving the group for the corresponding elements of x. Ignored if x is a list.
- **formula**: a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
- **data**: an optional data frame containing the variables in the model formula.
- **subset**: an optional vector specifying a subset of observations to be used.
- **na.action**: a function which indicates what should happen when the data contain NAs. Defaults to `getOption("na.action")`.
- **...**: further arguments to be passed to or from methods.

Details

If x is a list, its elements are taken as the samples to be compared for homogeneity of variances, and hence have to be numeric data vectors. In this case, g is ignored, and one can simply use `fligner.test(x)` to perform the test. If the samples are not yet contained in a list, use `fligner.test(list(x, ...))`.

Otherwise, x must be a numeric data vector, and g must be a vector or factor object of the same length as x giving the group for the corresponding elements of x.

The Fligner-Killeen (median) test has been determined in a simulation study as one of the many tests for homogeneity of variances which is most robust against departures from normality, see Conover, Johnson & Johnson (1981). It is a k-sample simple linear rank which uses the ranks of the absolute values of the centered samples and weights $a(i) = \text{qnorm}(1+i/(n+1))/2$. The version implemented here uses median centering in each of the samples (F-K:med $X^2$ in the reference).

Value

A list of class "htest" containing the following components:

- **statistic**: the Fligner-Killeen:med $X^2$ test statistic.
- **parameter**: the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- **p.value**: the p-value of the test.
- **method**: the character string "Fligner-Killeen test of homogeneity of variances".
- **data.name**: a character string giving the names of the data.

References


See Also

`ansari.test` and `mood.test` for rank-based two-sample test for a difference in scale parameters; `var.test` and `bartlett.test` for parametric tests for the homogeneity of variances.
Examples

```r
plot(count ~ spray, data = InsectSprays)
fligner.test(InsectSprays$count, InsectSprays$spray)
fligner.test(count ~ spray, data = InsectSprays)
## Compare this to bartlett.test()
```

---

### formula

**Model Formulae**

#### Description

The generic function *formula* and its specific methods provide a way of extracting formulae which have been included in other objects.

*as.formula* is almost identical, additionally preserving attributes when object already inherits from *"formula"*. The default value of the *env* argument is used only when the formula would otherwise lack an environment.

#### Usage

```r
formula(x, ...)
as.formula(object, env = parent.frame())
```

#### Arguments

- **x, object** R object.
- **...** further arguments passed to or from other methods.
- **env** the environment to associate with the result.

#### Details

The models fit by, e.g., the *lm* and *glm* functions are specified in a compact symbolic form. The ~ operator is basic in the formation of such models. An expression of the form $y \sim model$ is interpreted as a specification that the response $y$ is modelled by a linear predictor specified symbolically by *model*. Such a model consists of a series of terms separated by + operators. The terms themselves consist of variable and factor names separated by : operators. Such a term is interpreted as the interaction of all the variables and factors appearing in the term.

In addition to + and :, a number of other operators are useful in model formulae. The * operator denotes factor crossing: a*b interpreted as a+b+a:b. The ^ operator indicates crossing to the specified degree. For example (a+b+c)^2 is identical to (a+b+c)*(a+b+c) which in turn expands to a formula containing the main effects for a, b and c together with their second-order interactions. The %in% operator indicates that the terms on its left are nested within those on the right. For example a + b %in% a expands to the formula a + a:b. The - operator removes the specified terms, so that (a+b+c)^2 - a:b is identical to a + b + c + b:c + a:c. It can also used to remove the intercept term: $y \sim x - 1$ is a line through the origin. A model with no intercept can be also specified as $y \sim x + 0$ or $y \sim 0 + x$.

While formulae usually involve just variable and factor names, they can also involve arithmetic expressions. The formula $\log(y) \sim a + \log(x)$ is quite legal. When such arithmetic expressions involve operators which are also used symbolically in model formulae, there can be confusion between arithmetic and symbolic operator use.
To avoid this confusion, the function \( \text{I}() \) can be used to bracket those portions of a model formula where the operators are used in their arithmetic sense. For example, in the formula \( y \sim a + \text{I}(b+c) \), the term \( b+c \) is to be interpreted as the sum of \( b \) and \( c \).

As from R 1.8.0 variable names can be quoted by backticks 'like this' in formulae, although there is no guarantee that all code using formulae will accept such non-syntactic names.

When \text{formula} is called on a fitted model object, either a specific method is used (such as that for class "nls") of the default method. The default first looks for a "formula" component of the object (and evaluates it), then a "terms" component, then a formula parameter of the call (and evaluates its value) and finally a "formula" attribute.

Value

All the functions above produce an object of class "formula" which contains a symbolic model formula.

Environments

A formula object has an associated environment, and this environment (rather than the parent environment) is used by \text{model.frame} to evaluate variables that are not found in the supplied \text{data} argument.

Formulas created with the ~ operator use the environment in which they were created. Formulas created with \text{as.formula} will use the \text{env} argument for their environment. Pre-existing formulas extracted with \text{as.formula} will only have their environment changed if \text{env} is given explicitly.

References


See Also

\text{I}.

For formula manipulation: \text{terms}, and \text{all.vars}; for typical use: \text{lm}, \text{glm}, and \text{coplot}.

Examples

```r
class(fo <- y ~ x1*x2) # "formula"
fo
typeof(fo)# R internal : "language"
terms(fo)

environment(fo)
environment(as.formula("y ~ x"))
environment(as.formula("y ~ x",env=new.env()))

## Create a formula for a model with a large number of variables:
xnam <- paste("x", 1:25, sep="")
(fmla <- as.formula(paste("y ~ ", paste(xnam, collapse = "+"))))
```

```
formula.nls  Extract Model Formula from nls Object

Description

Returns the model used to fit object.

Usage

## S3 method for class 'nls':
formula(x, ...)

Arguments

x  an object inheriting from class "nls", representing a nonlinear least squares fit.
...  further arguments passed to or from other methods.

Value

a formula representing the model used to obtain object.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

nls, formula

Examples

fm1 <- nls(circumference ~ A/(1+exp((B-age)/C)), Orange,
           start = list(A=160, B=700, C = 350))
formula(fm1)

friedman.test  Friedman Rank Sum Test

Description

Performs a Friedman rank sum test with unreplicated blocked data.

Usage

friedman.test(y, ...)

## Default S3 method:
friedman.test(y, groups, blocks, ...)

## S3 method for class 'formula':
friedman.test(formula, data, subset, na.action, ...)

Examples

friedman.test(y, ...)

## Default S3 method:
friedman.test(y, groups, blocks, ...)

## S3 method for class 'formula':
friedman.test(formula, data, subset, na.action, ...)
friedman.test

Arguments

- **y**: either a numeric vector of data values, or a data matrix.
- **groups**: a vector giving the group for the corresponding elements of *y* if this is a vector; ignored if *y* is a matrix. If not a factor object, it is coerced to one.
- **blocks**: a vector giving the block for the corresponding elements of *y* if this is a vector; ignored if *y* is a matrix. If not a factor object, it is coerced to one.
- **formula**: a formula of the form *a ~ b | c*, where *a*, *b* and *c* give the data values and corresponding groups and blocks, respectively.
- **data**: an optional data frame containing the variables in the model formula.
- **subset**: an optional vector specifying a subset of observations to be used.
- **na.action**: a function which indicates what should happen when the data contain NAs. Defaults to `getOption("na.action")`.
- **...**: further arguments to be passed to or from methods.

Details

`friedman.test` can be used for analyzing unreplicated complete block designs (i.e., there is exactly one observation in *y* for each combination of levels of *groups* and *blocks*) where the normality assumption may be violated.

The null hypothesis is that apart from an effect of *blocks*, the location parameter of *y* is the same in each of the *groups*.

If *y* is a matrix, *groups* and *blocks* are obtained from the column and row indices, respectively. NA’s are not allowed in *groups* or *blocks*; if *y* contains NA’s, corresponding blocks are removed.

Value

A list with class "htest" containing the following components:

- **statistic**: the value of Friedman’s chi-squared statistic.
- **parameter**: the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- **p.value**: the p-value of the test.
- **method**: the character string "Friedman rank sum test".
- **data.name**: a character string giving the names of the data.

References


See Also

`quade.test`.
Examples

```r
## Comparison of three methods ("round out", "narrow angle", and
## "wide angle") for rounding first base. For each of 18 players
## and the three method, the average time of two runs from a point on
## the first base line 35ft from home plate to a point 15ft short of
## second base is recorded.
RoundingTimes <-
matrix(c(5.40, 5.50, 5.55,
        5.85, 5.70, 5.75,
        5.20, 5.60, 5.50,
        5.90, 5.85, 5.70,
        5.45, 5.55, 5.60,
        5.40, 5.40, 5.35,
        5.45, 5.50, 5.35,
        5.25, 5.15, 5.00,
        5.85, 5.80, 5.70,
        5.25, 5.20, 5.10,
        5.65, 5.55, 5.45,
        5.60, 5.35, 5.45,
        5.05, 5.00, 4.95,
        5.50, 5.50, 5.40,
        5.45, 5.55, 5.50,
        5.55, 5.55, 5.35,
        5.45, 5.50, 5.55,
        5.50, 5.45, 5.25,
        5.65, 5.60, 5.40,
        5.70, 5.65, 5.55,
        6.30, 6.30, 6.25),
nr = 22,
byrow = TRUE,
dimnames = list(1:22,
c("Round Out", "Narrow Angle", "Wide Angle")))
friedman.test(RoundingTimes)
## => strong evidence against the null that the methods are equivalent
## with respect to speed

wb <- aggregate(warpbreaks$breaks,
    by = list(w = warpbreaks$wool,
              t = warpbreaks$tension),
    FUN = mean)
w
friedman.test(wbx, wbw, wt)
friedman.test(x ~ w | t, data = wb)
```

ftable

**Flat Contingency Tables**

**Description**

Create “flat” contingency tables.
Usage

ftable(x, ...)

## Default S3 method:
ftable(..., exclude = c(NA, NaN), row.vars = NULL, col.vars = NULL)

Arguments

x, ...  
R objects which can be interpreted as factors (including character strings), or a list (or data frame) whose components can be so interpreted, or a contingency table object of class "table" or "ftable".

exclude  
values to use in the exclude argument of factor when interpreting non-factor objects.

row.vars  
a vector of integers giving the numbers of the variables, or a character vector giving the names of the variables to be used for the rows of the flat contingency table.

col.vars  
a vector of integers giving the numbers of the variables, or a character vector giving the names of the variables to be used for the columns of the flat contingency table.

Details

ftable creates “flat” contingency tables. Similar to the usual contingency tables, these contain the counts of each combination of the levels of the variables (factors) involved. This information is then re-arranged as a matrix whose rows and columns correspond to unique combinations of the levels of the row and column variables (as specified by row.vars and col.vars, respectively). The combinations are created by looping over the variables in reverse order (so that the levels of the “left-most” variable vary the slowest). Displaying a contingency table in this flat matrix form (via print.ftable, the print method for objects of class "ftable") is often preferable to showing it as a higher-dimensional array.

ftable is a generic function. Its default method, ftable.default, first creates a contingency table in array form from all arguments except row.vars and col.vars. If the first argument is of class "table", it represents a contingency table and is used as is; if it is a flat table of class "ftable", the information it contains is converted to the usual array representation using as.ftable. Otherwise, the arguments should be R objects which can be interpreted as factors (including character strings), or a list (or data frame) whose components can be so interpreted, which are cross-tabulated using table. Then, the arguments row.vars and col.vars are used to collapse the contingency table into flat form. If neither of these two is given, the last variable is used for the columns. If both are given and their union is a proper subset of all variables involved, the other variables are summed out.

When the arguments are R expressions interpreted as factors, additional arguments will be passed to table to control how the variable names are displayed; see the last example below.

Function ftable.formula provides a formula method for creating flat contingency tables.

Value

ftable returns an object of class "ftable", which is a matrix with counts of each combination of the levels of variables with information on the names and levels of the (row and columns) variables stored as attributes "row.vars" and "col.vars".
See Also

ftable.formula for the formula interface (which allows a data = . argument); read.ftable for information on reading, writing and coercing flat contingency tables; table for "ordinary" cross-tabulation; xtabs for formula-based cross-tabulation.

Examples

```r
## Start with a contingency table.
ftable(Titanic, row.vars = 1:3)
ftable(Titanic, row.vars = 1:2, col.vars = "Survived")
ftable(Titanic, row.vars = 2:1, col.vars = "Survived")

## Start with a data frame.
x <- ftable(mtcars[\c("cyl", "vs", "am", "gear")])
x
ftable(x, row.vars = c(2, 4))

## Start with expressions, use table()'s "dnn" to change labels
ftable(mtcars$cyl, mtcars$vs, mtcars$am, mtcars$gear, row.vars = c(2, 4),
  dnn = c("Cylinders", "V/S", "Transmission", "Gears"))
```

Description

Produce or manipulate a flat contingency table using formula notation.

Usage

```r
## S3 method for class 'formula':
ftable(formula, data = NULL, subset, na.action, ...)
```

Arguments

- `formula`: a formula object with both left and right hand sides specifying the column and row variables of the flat table.
- `data`: a data frame, list or environment containing the variables to be cross-tabulated, or a contingency table (see below).
- `subset`: an optional vector specifying a subset of observations to be used. Ignored if `data` is a contingency table.
- `na.action`: a function which indicates what should happen when the data contain NAs. Ignored if `data` is a contingency table.
- `...`: further arguments to the default `ftable` method may also be passed as arguments, see `ftable.default`.
Details

This is a method of the generic function `ftable`.

The left and right hand side of `formula` specify the column and row variables, respectively, of the flat contingency table to be created. Only the `+` operator is allowed for combining the variables. A `. ` may be used once in the formula to indicate inclusion of all the “remaining” variables.

If `data` is an object of class "table" or an array with more than 2 dimensions, it is taken as a contingency table, and hence all entries should be nonnegative. Otherwise, if it is not a flat contingency table (i.e., an object of class "ftable"), it should be a data frame or matrix, list or environment containing the variables to be cross-tabulated. In this case, `na.action` is applied to the data to handle missing values, and, after possibly selecting a subset of the data as specified by the `subset` argument, a contingency table is computed from the variables.

The contingency table is then collapsed to a flat table, according to the row and column variables specified by `formula`.

Value

A flat contingency table which contains the counts of each combination of the levels of the variables, collapsed into a matrix for suitably displaying the counts.

See Also

`ftable`, `ftable.default`, `table`.

Examples

Titanic

`x <- ftable(Survived ~ ., data = Titanic)`

`x`

`ftable(Sex ~ Class + Age, data = x)`

Gamma Dist

The Gamma Distribution

Description

Density, distribution function, quantile function and random generation for the Gamma distribution with parameters `shape` and `scale`.

Usage

`dgamma(x, shape = 1, scale = 1/rate, log = FALSE)`

`pgamma(q, shape = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)`

`qgamma(p, shape = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)`

`rgamma(n, shape = 1, scale = 1/rate)`
Arguments

\(x, q\) vector of quantiles.
\(p\) vector of probabilities.
\(n\) number of observations. If \(\text{length}(n) > 1\), the length is taken to be the number required.
\(\text{rate}\) an alternative way to specify the scale.
\(\text{shape, scale}\) shape and scale parameters.
\(\log, \log.p\) logical; if TRUE, probabilities \(p\) are given as \(\log(p)\).
\(\text{lower.tail}\) logical; if TRUE (default), probabilities are \(P[X \leq x]\), otherwise, \(P[X > x]\).

Details

If \(\text{scale}\) is omitted, it assumes the default value of 1.

The Gamma distribution with parameters \(\text{shape} = \alpha\) and \(\text{scale} = \sigma\) has density

\[f(x) = \frac{1}{\sigma^\alpha \Gamma(\alpha)} x^{\alpha - 1} e^{-x/\sigma}\]

for \(x > 0, \alpha > 0\) and \(\sigma > 0\). (Here \(\Gamma(\alpha)\) is the function implemented by R’s \texttt{gamma()} and defined in its help.)

The mean and variance are \(E(X) = \alpha \sigma\) and \(\text{Var}(X) = \alpha \sigma^2\).

The cumulative hazard \(H(t) = -\log(1 - F(t))\) is \(-\text{pgamma}(t, \ldots, \text{lower} = \text{FALSE}, \log = \text{TRUE})\).

Value

d\text{gamma} gives the density, p\text{gamma} gives the distribution function, q\text{gamma} gives the quantile function, and r\text{gamma} generates random deviates.

Note

The S parametrization is via \text{shape} and \text{rate}: S has no \text{scale} parameter.

p\text{gamma} is closely related to the incomplete gamma function. As defined by Abramowitz and Stegun 6.5.1

\[P(a, x) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} e^{-t} dt\]

\(P(a, x)\) is \text{pgamma}(x, a). Other authors (for example Karl Pearson in his 1922 tables) omit the normalizing factor, defining the incomplete gamma function as \text{pgamma}(x, a) \ast \text{gamma}(a).

As from R 2.1.0 \text{pgamma}() uses a new algorithm (mainly by Morten Welinder) which should be uniformly as accurate as AS 239.

References


See Also

gamma for the gamma function, dbeta for the Beta distribution and dchisq for the chi-squared distribution which is a special case of the Gamma distribution.

Examples

-\log(dgamma(1:4, shape=1))

p <- (1:9)/10
pgamma(qgamma(p,shape=2), shape=2)
1 - 1/exp(qgamma(p, shape=1))

Geometric

The Geometric Distribution

Description

Density, distribution function, quantile function and random generation for the geometric distribution with parameter prob.

Usage

dgeom(x, prob, log = FALSE)
pgeom(q, prob, lower.tail = TRUE, log.p = FALSE)
qgeom(p, prob, lower.tail = TRUE, log.p = FALSE)
rgeom(n, prob)

Arguments

x, q vector of quantiles representing the number of failures in a sequence of Bernoulli trials before success occurs.
p vector of probabilities.
n number of observations. If length(n) > 1, the length is taken to be the number required.
prob probability of success in each trial.
log, log.p logical; if TRUE, probabilities p are given as log(p).
lower.tail logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

The geometric distribution with prob = p has density

\[ p(x) = p(1 - p)^x \]

for \( x = 0, 1, 2, \ldots \)

If an element of x is not integer, the result of pgem is zero, with a warning.

The quantile is defined as the smallest value \( x \) such that \( F(x) \geq p \), where \( F \) is the distribution function.
Value

dgeom gives the density, pgeom gives the distribution function, qgeom gives the quantile function, and rgeom generates random deviates.

See Also
dnbinom for the negative binomial which generalizes the geometric distribution.

Examples

ggeom((1:9)/10, prob = .2)
Ni <- rgeom(20, prob = 1/4); table(factor(Ni, 0:max(Ni)))

getInitial

Get Initial Parameter Estimates

Description

This function evaluates initial parameter estimates for a nonlinear regression model. If data is a parameterized data frame or pframe object, its parameters attribute is returned. Otherwise the object is examined to see if it contains a call to a selfStart object whose initial attribute can be evaluated.

Usage

generic(object, data, ...)

Arguments

object a formula or a selfStart model that defines a nonlinear regression model
data a data frame in which the expressions in the formula or arguments to the selfStart model can be evaluated
... optional additional arguments

Value

A named numeric vector or list of starting estimates for the parameters. The construction of many selfStart models is such that these “starting” estimates are, in fact, the converged parameter estimates.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

nls, selfStart, selfStart.default, selfStart.formula

Examples

PurTrt <- Puromycin[ Puromycin$state == "treated", ]
getInitial( rate ~ SSmicmen( conc, Vm, K ), PurTrt )
glm

Fitting Generalized Linear Models

Description

glm is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage

```r
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart,
    offset, control = glm.control(...), model = TRUE,
    method = "glm.fit", x = FALSE, y = TRUE, contrasts = NULL, ...)
```

```r
glm.fit(x, y, weights = rep(1, nobs),
    start = NULL, etastart = NULL, mustart = NULL,
    offset = rep(0, nobs), family = gaussian(),
    control = glm.control(), intercept = TRUE)
```

```r
## S3 method for class 'glm':
weights(object, type = c("prior", "working"), ...)
```

Arguments

- `formula`: a symbolic description of the model to be fit. The details of model specification are given below.
- `family`: a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
- `data`: an optional data frame containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.
- `weights`: an optional vector of weights to be used in the fitting process.
- `subset`: an optional vector specifying a subset of observations to be used in the fitting process.
- `na.action`: a function which indicates what should happen when the data contain NAs. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The “factory-fresh” default is `na.omit`.
- `start`: starting values for the parameters in the linear predictor.
- `etastart`: starting values for the linear predictor.
- `mustart`: starting values for the vector of means.
- `offset`: this can be used to specify an a priori known component to be included in the linear predictor during fitting.
- `control`: a list of parameters for controlling the fitting process. See the documentation for `glm.control` for details.
model a logical value indicating whether model frame should be included as a component of the returned value.

method the method to be used in fitting the model. The default method "glm.fit" uses iteratively reweighted least squares (IWLS). The only current alternative is "model.frame" which returns the model frame and does no fitting.

x, y For glm: logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value.
For glm.fit: x is a design matrix of dimension n * p, and y is a vector of observations of length n.

contrasts an optional list. See the contrasts.arg of model.matrix.default.

object an object inheriting from class "glm".

type character, partial matching allowed. Type of weights to extract from the fitted model object.

intercept logical. Should an intercept be included in the null model?

... further arguments passed to or from other methods.

Details

A typical predictor has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response. For binomial models the response can also be specified as a factor (when the first level denotes failure and all others success) or as a two-column matrix with the columns giving the numbers of successes and failures. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with duplicates removed. The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a terms object as the formula.

A specification of the form first:second indicates the the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the cross of first and second. This is the same as first + second + first:second.

glm.fit and glm.fit.null are the workhorse functions: the former calls the latter for a null model (with no intercept).

If more than one of etastart, start and mustart is specified, the first in the list will be used.

All of weights, subset, offset, etastart and mustart are evaluated in the same way as variables in formula, that is first in data and then in the environment of formula.

Value

glm returns an object of class inheriting from "glm" which inherits from the class "lm". See later in this section.

The function summary (i.e., summary.glm) can be used to obtain or print a summary of the results and the function anova (i.e., anova.glm) to produce an analysis of variance table.

The generic accessor functions coefficients, effects, fitted.values and residuals can be used to extract various useful features of the value returned by glm.

weights extracts a vector of weights, one for each case in the fit (after subsetting and na.action).

An object of class "glm" is a list containing at least the following components:
coefficients a named vector of coefficients
residuals the working residuals, that is the residuals in the final iteration of the IWLS fit.
fitted.values the fitted mean values, obtained by transforming the linear predictors by the inverse of the link function.
rank the numeric rank of the fitted linear model.
family the family object used.
linear.predictors the linear fit on link scale.
deviance up to a constant, minus twice the maximized log-likelihood. Where sensible, the constant is chosen so that a saturated model has deviance zero.
aic Akaike’s An Information Criterion, minus twice the maximized log-likelihood plus twice the number of coefficients (so assuming that the dispersion is known).
null.deviance The deviance for the null model, comparable with deviance. The null model will include the offset, and an intercept if there is one in the model
iter the number of iterations of IWLS used.
weights the working weights, that is the weights in the final iteration of the IWLS fit.
prior.weights the case weights initially supplied.

df.residual the residual degrees of freedom.
df.null the residual degrees of freedom for the null model.
y the y vector used. (It is a vector even for a binomial model.)
converged logical. Was the IWLS algorithm judged to have converged?
boundary logical. Is the fitted value on the boundary of the attainable values?
call the matched call.
formula the formula supplied.
terms the terms object used.
data the data argument.
offset the offset vector used.
control the value of the control argument used.
method the name of the fitter function used, in R always "glm.fit".
contrasts (where relevant) the contrasts used.
xlevels (where relevant) a record of the levels of the factors used in fitting.

In addition, non-empty fits will have components qr, R and effects relating to the final weighted linear fit.

Objects of class "glm" are normally of class c("glm", "lm"), that is inherit from class "lm", and well-designed methods for class "lm" will be applied to the weighted linear model at the final iteration of IWLS. However, care is needed, as extractor functions for class "glm" such as residuals and weights do not just pick out the component of the fit with the same name.

If a binomial glm model is specified by giving a two-column response, the weights returned by prior.weights are the total numbers of cases (factored by the supplied case weights) and the component y of the result is the proportion of successes.
Author(s)

The original R implementation of glm was written by Simon Davies working for Ross Ihaka at the University of Auckland, but has since been extensively re-written by members of the R Core team. The design was inspired by the S function of the same name described in Hastie & Pregibon (1992).

References


See Also

anova.glm, summary.glm, etc. for glm methods, and the generic functions anova, summary, effects, fitted.values, and residuals. Further, lm for non-generalized linear models.
esoph, infert and predict.glm have examples of fitting binomial glms.

Examples

## Dobson (1990) Page 93: Randomized Controlled Trial:
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts))
glm.D93 <- glm(counts ~ outcome + treatment, family=poisson())
anova(glm.D93)
summary(glm.D93)

## an example with offsets from Venables & Ripley (2002, p.189)
## Not run:
## Need the anorexia data from a recent version of the package 'MASS':
library(MASS)
## End(Not run)
anorex.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
                 family = gaussian, data = anorexia)
summary(anorex.1)

# A Gamma example, from McCullagh & Nelder (1989, pp. 300-2)
clotting <- data.frame(
    u = c(5,10,15,20,30,40,60,80,100),
    lot1 = c(118,58,42,35,27,25,21,19,18),
    lot2 = c(69,35,26,21,18,16,13,12,12))
summary(glm(lot1 ~ log(u), data=clotting, family=Gamma))
summary(glm(lot2 ~ log(u), data=clotting, family=Gamma))

## Not run:
## for an example of the use of a terms object as a formula
demo(glm.vr)
## End(Not run)
glm.control

Auxiliary for Controlling GLM Fitting

Description

Auxiliary function as user interface for glm fitting. Typically only used when calling glm or glm.fit.

Usage

glm.control(epsilon = 1e-8, maxit = 25, trace = FALSE)

Arguments

epsilon positive convergence tolerance $\epsilon$; the iterations converge when $|\text{dev} - \text{dev}_{\text{old}}|/(|\text{dev}| + 0.1) < \epsilon$.

maxit integer giving the maximal number of IWLS iterations.

trace logical indicating if output should be produced for each iteration.

Details

If epsilon is small, it is also used as the tolerance for the least squares solution.

When trace is true, calls to cat produce the output for each IWLS iteration. Hence, options(digits = *) can be used to increase the precision, see the example.

Value

A list with the arguments as components.

References


See Also

glm.fit, the fitting procedure used by glm.

Examples

### A variation on example(glm) :

```r
## Annette Dobson's example ...
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
oo <- options(digits = 12) # to see more when tracing :
glm.D93X <- glm(counts ~ outcome + treatment, family=poisson(), trace = TRUE, epsilon = 1e-14)
options(oo)

coef(glm.D93X) # the last two are closer to 0 than in glm's glm.D93
# put less so than in R < 1.8.0 when the default was 1e-4
```
Accessing Generalized Linear Model Fits

Description

These functions are all methods for class glm or summary.glm objects.

Usage

```r
## S3 method for class 'glm':
family(object, ...)
```

```r
## S3 method for class 'glm':
residuals(object, type = c("deviance", "pearson", "working",
                         "response", "partial"), ...)
```

Arguments

- `object` an object of class glm, typically the result of a call to glm.
- `type` the type of residuals which should be returned. The alternatives are: "deviance" (default), "pearson", "working", "response", and "partial".
- `...` further arguments passed to or from other methods.

Details

The references define the types of residuals: Davison & Snell is a good reference for the usages of each.

The partial residuals are a matrix of working residuals, with each column formed by omitting a term from the model.

References


See Also

glm for computing glm.obj. anova.glm: the corresponding generic functions, summary.glm, coef, deviance, df.residual, effects, fitted, residuals.
**hclust**

Hierarchical Clustering

Description

Hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it.

Usage

```r
hclust(d, method = "complete", members=NULL)
```

```r
## S3 method for class 'hclust':
plot(x, labels = NULL, hang = 0.1,
    axes = TRUE, frame.plot = FALSE, ann = TRUE,
    main = "Cluster Dendrogram",
    sub = NULL, xlab = NULL, ylab = "Height", ...)
```

```r
plclust(tree, hang = 0.1, unit = FALSE, level = FALSE, hmin = 0,
    square = TRUE, labels = NULL, plot. = TRUE,
    axes = TRUE, frame.plot = FALSE, ann = TRUE,
    main = "", sub = NULL, xlab = NULL, ylab = "Height")
```

Arguments

d  a dissimilarity structure as produced by `dist`.

method  the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "ward", "single", "complete", "average", "mcquitty", "median" or "centroid".

members  `NULL` or a vector with length size of `d`. See the Details section.

x,tree  an object of the type produced by `hclust`.

hang  the fraction of the plot height by which labels should hang below the rest of the plot. A negative value will cause the labels to hang down from 0.

labels  A character vector of labels for the leaves of the tree. By default the row names or row numbers of the original data are used. If `labels=FALSE` no labels at all are plotted.

axes, frame.plot, ann  logical flags as in `plot.default`.

main, sub, xlab, ylab  character strings for `title`. `sub` and `xlab` have a non-NULL default when there's a `tree$call`.

...  Further graphical arguments.

unit  logical. If true, the splits are plotted at equally-spaced heights rather than at the height in the object.

hmin  numeric. All heights less than `hmin` are regarded as being `hmin`: this can be used to suppress detail at the bottom of the tree.

level, square, plot.  as yet unimplemented arguments of `plclust` for S-PLUS compatibility.
Details

This function performs a hierarchical cluster analysis using a set of dissimilarities for the \( n \) objects being clustered. Initially, each object is assigned to its own cluster and then the algorithm proceeds iteratively, at each stage joining the two most similar clusters, continuing until there is just a single cluster. At each stage distances between clusters are recomputed by the Lance–Williams dissimilarity update formula according to the particular clustering method being used.

A number of different clustering methods are provided. \textit{Ward’s} minimum variance method aims at finding compact, spherical clusters. The \textit{complete linkage} method finds similar clusters. The \textit{single linkage} method (which is closely related to the minimal spanning tree) adopts a ‘friends of friends’ clustering strategy. The other methods can be regarded as aiming for clusters with characteristics somewhere between the single and complete link methods. Note however, that methods "\textit{median}" and "\textit{centroid}" are \textit{not} leading to a \textit{monotone} distance measure, or equivalently the resulting dendrograms can have so called \textit{inversions} (which are hard to interpret).

If \texttt{members!}\texttt{=}\texttt{NULL}, then \( d \) is taken to be a dissimilarity matrix between clusters instead of dissimilarities between singletons and \texttt{members} gives the number of observations per cluster. This way the hierarchical cluster algorithm can be “started in the middle of the dendrogram”, e.g., in order to reconstruct the part of the tree above a cut (see examples). Dissimilarities between clusters can be efficiently computed (i.e., without \texttt{hclust} itself) only for a limited number of distance/linkage combinations, the simplest one being squared Euclidean distance and centroid linkage. In this case the dissimilarities between the clusters are the squared Euclidean distances between cluster means.

In hierarchical cluster displays, a decision is needed at each merge to specify which subtree should go on the left and which on the right. Since, for \( n \) observations there are \( n - 1 \) merges, there are \( 2^{(n-1)} \) possible orderings for the leaves in a cluster tree, or dendrogram. The algorithm used in \texttt{hclust} is to order the subtree so that the tighter cluster is on the left (the last, i.e., most recent, merge of the left subtree is at a lower value than the last merge of the right subtree). Single observations are the tightest clusters possible, and merges involving two observations place them in order by their observation sequence number.

Value

An object of class \texttt{hclust} which describes the tree produced by the clustering process. The object is a list with components:

\begin{itemize}
\item \texttt{merge} an \( n - 1 \) by 2 matrix. Row \( i \) of \texttt{merge} describes the merging of clusters at step \( i \) of the clustering. If an element \( j \) in the row is negative, then observation \( -j \) was merged at this stage. If \( j \) is positive then the merge was with the cluster formed at the (earlier) stage \( j \) of the algorithm. Thus negative entries in \texttt{merge} indicate agglomerations of singletons, and positive entries indicate agglomerations of non-singletons.
\item \texttt{height} a set of \( n - 1 \) non-decreasing real values. The clustering \textit{height}: that is, the value of the criterion associated with the clustering \texttt{method} for the particular agglomeration.
\item \texttt{order} a vector giving the permutation of the original observations suitable for plotting, in the sense that a cluster plot using this ordering and matrix \texttt{merge} will not have crossings of the branches.
\item \texttt{labels} labels for each of the objects being clustered.
\item \texttt{call} the call which produced the result.
\item \texttt{method} the cluster method that has been used.
\item \texttt{dist.method} the distance that has been used to create \( d \) (only returned if the distance object has a "\texttt{method}" attribute).
\end{itemize}
There are `print`, `plot` and `identify` (see `identify.hclust`) methods and the `rect.hclust()` function for `hclust` objects. The `plclust()` function is basically the same as the plot method, `plot.hclust`, primarily for back compatibility with S-plus. Its extra arguments are not yet implemented.

**Author(s)**

The `hclust` function is based on Fortran code contributed to STATLIB by F. Murtagh.

**References**


**See Also**

`identify.hclust`, `rect.hclust`, `cutree`, `dendrogram`, `kmeans`.

For the Lance–Williams formula and methods that apply it generally, see `agnes` from package `cluster`.

**Examples**

```r
hc <- hclust(dist(USArrests), "ave")
plot(hc)
plot(hc, hang = -1)

## Do the same with centroid clustering and squared Euclidean distance,
## cut the tree into ten clusters and reconstruct the upper part of the
## tree from the cluster centers.
hc <- hclust(dist(USArrests)^2, "cen")
memb <- cutree(hc, k = 10)
cent <- NULL
for(k in 1:10){
  cent <- rbind(cent, colMeans(USArrests[memb == k, , drop = FALSE]))
}
hc1 <- hclust(dist(cent)^2, method = "cen", members = table(memb))
opar <- par(mfrow = c(1, 2))
plot(hc, labels = FALSE, hang = -1, main = "Original Tree")
plot(hc1, labels = FALSE, hang = -1, main = "Re-start from 10 clusters")
par(opar)
```
Heatmap

Description

A heat map is a false color image (basically image(\(t(x)\))) with a dendrogram added to the left side and to the top. Typically, reordering of the rows and columns according to some set of values (row or column means) within the restrictions imposed by the dendrogram is carried out.

Usage

heatmap(x, Rowv=NULL, Colv=if(symph)"Rowv" else NULL, distfun = dist, hclustfun = hclust, reorderfun = function(d,w) reorder(d,w), add.expr, symm = FALSE, revC = identical(Colv, "Rowv"), scale=c("row", "column", "none"), na.rm = TRUE, margins = c(5, 5), ColSideColors, RowSideColors, cexRow = 0.2 + 1/log10(nr), cexCol = 0.2 + 1/log10(nc), labRow = NULL, labCol = NULL, main = NULL, xlab = NULL, ylab = NULL, keep.dendro = FALSE, verbose = getOption("verbose"), ...)

Arguments

x
numeric matrix of the values to be plotted.

Rowv
determines if and how the row dendrogram should be computed and reordered. Either a dendrogram or a vector of values used to reorder the row dendrogram or NA to suppress any row dendrogram (and reordering) or by default, NULL, see Details below.

Colv
determines if and how the column dendrogram should be reordered. Has the same options as the Rowv argument above and additionally when x is a square matrix, Colv = "Rowv" means that columns should be treated identically to the rows.

distfun
function used to compute the distance (dissimilarity) between both rows and columns. Defaults to dist.

hclustfun
function used to compute the hierarchical clustering when Rowv or Colv are not dendrograms. Defaults to hclust.

reorderfun
function(d,w) of dendrogram and weights for reordering the row and column dendrograms. The default uses reorder.dendrogram.

add.expr
expression that will be evaluated after the call to image. Can be used to add components to the plot.

symm
logical indicating if x should be treated symmetrically; can only be true when x is a square matrix.

revC
logical indicating if the column order should be reversed for plotting, such that e.g., for the symmetric case, the symmetry axis is as usual.

scale
character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. The default is "row" if symm false, and "none" otherwise.
heatmap

na.rm logical indicating whether NA’s should be removed.
margins numeric vector of length 2 containing the margins (see par(mar= *)) for column and row names, respectively.

ColSideColors (optional) character vector of length ncol(x) containing the color names for a horizontal side bar that may be used to annotate the columns of x.

RowSideColors (optional) character vector of length nrow(x) containing the color names for a vertical side bar that may be used to annotate the rows of x.
cexRow, cexCol positive numbers, used as cex.axis in for the row or column axis labeling. The defaults currently only use number of rows or columns, respectively.

labRow, labCol character vectors with row and column labels to use; these default torownames(x) or colnames(x), respectively.

main, xlab, ylab main, x- and y-axis titles; defaults to none.

keep.dendro logical indicating if the dendrogram(s) should be kept as part of the result (when Rowv and/or Colv are not NA).
verbose logical indicating if information should be printed.
...
additional arguments passed on to image e.g., col specifying the colors.

Details

If either Rowv or Colv are dendrograms they are honored (and not reordered). Otherwise, dendro-
grams are computed as dd <- as.dendrogram(hclustfun(distfun(X))) where X is either x or t(x).

If either is a vector (of “weights”) then the appropriate dendrogram is reordered according to the supplied values subject to the constraints imposed by the dendrogram, by reorder(dd, Rowv), in the row case. If either is missing, as by default, then the ordering of the corre-
sponding dendrogram is by the mean value of the rows/columns, i.e., in the case of rows, Rowv <-
rowMeans(x, na.rm=na.rm). If either is NULL, no reordering will be done for the corre-
sponding side.

By default (scale = "row") the rows are scaled to have mean zero and standard deviation one. There is some empirical evidence from genomic plotting that this is useful.

The default colors are not pretty. Consider using enhancements such as the RColorBrewer package, http://cran.r-project.org/src/contrib/PACKAGES.html#RColorBrewer.

Value

Invisibly, a list with components

rowInd row index permutation vector as returned by order.dendrogram.
colInd column index permutation vector.
Rowv the row dendrogram; only if input Rowv was not NA and keep.dendro is true.
Colv the column dendrogram; only if input Colv was not NA and keep.dendro is true.
Note

Unless Rowv = NA (or Colw = NA), the original rows and columns are reordered in any case to match the dendrogram, e.g., the rows by order.dendrogram(Rowv) where Rowv is the (possibly reorder() ed) row dendrogram.

heatmap() uses layout and draws the image in the lower right corner of a 2x2 layout. Consequently, it can not be used in a multi column/row layout, i.e., when par(mfrow = *) or (mfcol = *) has been called.

Author(s)

Andy Liaw, original; R. Gentleman, M. Maechler, W. Huber, revisions.

See Also

image, hclust

Examples

require(graphics)
x <- as.matrix(mtcars)
rc <- rainbow(nrow(x), start=0, end=.3)
cc <- rainbow(ncol(x), start=0, end=.3)
hv <- heatmap(x, col = cm.colors(256), scale="column",
RowSideColors = rc, ColSideColors = cc, margin=c(5,10),
xlab = "specification variables", ylab="Car Models",
main = "heatmap(<Mtcars data>, ..., scale = "column")")
str(hv) # the two re-ordering index vectors

## no column dendrogram (nor reordering) at all:
heatmap(x, Colv = NA, col = cm.colors(256), scale="column",
RowSideColors = rc, margin=c(5,10),
xlab = "specification variables", ylab="Car Models",
main = "heatmap(<Mtcars data>, ..., scale = "column")")

## "no nothing"
heatmap(x, Rowv = NA, Colv = NA, scale="column",
main = "heatmap(*, NA, NA) ~= image(t(x))")

round(Ca <- cor(attitude), 2)
symnum(Ca) # simple graphic
heatmap(Ca, symm = TRUE, margin=c(6,6))# with reorder()
heatmap(Ca, Rowv=FALSE, symm = TRUE, margin=c(6,6))# _NO_ reorder()

## For variable clustering, rather use distance based on cor():
symnum( cU <- cor(USJudgeRatings) )

hU <- heatmap(cU, Rowv = FALSE, symm = TRUE, col = topo.colors(16),
  distfun = function(c) as.dist(1 - c), keep.dendro = TRUE)

## The Correlation matrix with same reordering:
round(100 * cU[hU[[1]], hU[[2]]])

## The column dendrogram:
str(hU$Colv)
HoltWinters

Holt-Winters Filtering

Description
Computes Holt-Winters Filtering of a given time series. Unknown parameters are determined by minimizing the squared prediction error.

Usage
HoltWinters(x, alpha = NULL, beta = NULL, gamma = NULL, seasonal = c("additive", "multiplicative"), start.periods = 3, l.start = NULL, b.start = NULL, s.start = NULL, optim.start = c(alpha = 0.3, beta = 0.1, gamma = 0.1), optim.control = list())

Arguments
- x: An object of class ts
- alpha: alpha parameter of Holt-Winters Filter
- beta: beta parameter of Holt-Winters Filter. If set to 0, the function will do exponential smoothing.
- gamma: gamma parameter used for the seasonal component. If set to 0, an non-seasonal model is fitted.
- seasonal: Character string to select an "additive" (the default) or "multiplicative" seasonal model. The first few characters are sufficient. (Only takes effect if gamma is non-zero).
- start.periods: Start periods used in the autodetection of start values. Must be at least 3.
- l.start: Start value for level (a[0]).
- b.start: Start value for trend (b[0]).
- s.start: Vector of start values for the seasonal component (s[1][0]...s[p][0])
- optim.start: Vector with named components alpha, beta, and gamma containing the starting values for the optimizer. Only the values needed must be specified.
- optim.control: Optional list with additional control parameters passed to optim.

Details
The additive Holt-Winters prediction function (for time series with period length p) is

\[ \hat{Y}[t + h] = a[t] + hb[t] + s[t + 1 + (h - 1) \mod p], \]

where \(a[t], b[t]\) and \(s[t]\) are given by

\[ a[t] = \alpha(Y[t] - s[t - p]) + (1 - \alpha)(a[t - 1] + b[t - 1]) \]
\[ b[t] = \beta(a[t] - a[t - 1]) + (1 - \beta)b[t - 1] \]
\[ s[t + 1] = \gamma(Y[t] - a[t] - b[t]) + (1 - \gamma)s[t - p] \]
The multiplicative Holt-Winters prediction function (for time series with period length $p$) is

$$\hat{Y}[t + h] = (a[t] + hb[t]) \times s[t + 1 + (h - 1) \mod p],$$

where $a[t]$, $b[t]$ and $s[t]$ are given by

$$a[t] = \alpha (Y[t]/s[t-p]) + (1-\alpha)(a[t-1] + b[t-1])$$

$$b[t] = \beta (a[t] - a[t-1]) + (1-\beta)b[t-1]$$

$$s[t] = \gamma (Y[t]/a[t]) + (1-\gamma)s[t-p]$$

The function tries to find the optimal values of $\alpha$ and/or $\beta$ and/or $\gamma$ by minimizing the squared one-step prediction error if they are omitted.

For seasonal models, start values for $a$, $b$ and $s$ are detected by performing a simple decomposition in trend and seasonal component using moving averages (see function `decompose`) on the `start.periods` first periods (a simple linear regression on the trend component is used for starting level and trend.). For level/trend-models (no seasonal component), start values for $a$ and $b$ are $x[2]$ and $x[2] - x[1]$, respectively. For level-only models (ordinary exponential smoothing), the start value for $a$ is $x[1]$.

**Value**

An object of class "HoltWinters", a list with components:

- **fitted**: A multiple time series with one column for the filtered series as well as for the level, trend and seasonal components, estimated contemporaneously (that is at time $t$ and not at the end of the series).
- **x**: The original series
- **alpha**: alpha used for filtering
- **beta**: beta used for filtering
- **coefficients**: A vector with named components $a$, $b$, $s1$, ..., $sp$ containing the estimated values for the level, trend and seasonal components
- **seasonal**: The specified seasonal-parameter
- **SSE**: The final sum of squared errors achieved in optimizing
- **call**: The call used

**References**


P. R. Winters (1960) Forecasting sales by exponentially weighted moving averages, Management Science 6, 324–342.

**See Also**

`predict.HoltWinters`, `optim`
Examples

```r
## Seasonal Holt-Winters
(m <- HoltWinters(co2))
plot(m)
plot(fitted(m))

(m <- HoltWinters(AirPassengers, seasonal = "mul")."
plot(m)

## Non-Seasonal Holt-Winters
x <- uspop + rnorm(uspop, sd = 5)
m <- HoltWinters(x, gamma = 0)
plot(m)

## Exponential Smoothing
m2 <- HoltWinters(x, gamma = 0, beta = 0)
lines(fitted(m2)[,1], col = 3)
```

---

**Hypergeometric**

*The Hypergeometric Distribution*

**Description**

Density, distribution function, quantile function and random generation for the hypergeometric distribution.

**Usage**

```r
dhyper(x, m, n, k, log = FALSE)
phyper(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
qhyper(p, m, n, k, lower.tail = TRUE, log.p = FALSE)
rhyper(nn, m, n, k)
```

**Arguments**

- `x, q` vector of quantiles representing the number of white balls drawn without replacement from an urn which contains both black and white balls.
- `m` the number of white balls in the urn.
- `n` the number of black balls in the urn.
- `k` the number of balls drawn from the urn.
- `p` probability, it must be between 0 and 1.
- `nn` number of observations. If `length(nn) > 1`, the length is taken to be the number required.
- `log, log.p` logical; if TRUE, probabilities p are given as log(p).
- `lower.tail` logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
Details

The hypergeometric distribution is used for sampling without replacement. The density of this distribution with parameters $m$, $n$ and $k$ (named $Np$, $N - Np$, and $n$, respectively in the reference below) is given by

$$p(x) = \frac{\binom{m}{x}\binom{n}{k-x}}{\binom{m+n}{k}}$$

for $x = 0, \ldots, k$.

Value

dhyper gives the density, phyper gives the distribution function, qhyper gives the quantile function, and rhyper generates random deviates.

References


Examples

```r
m <- 10; n <- 7; k <- 8
x <- 0:(k+1)
rbind(phyper(x, m, n, k), dhyper(x, m, n, k))
all(phyper(x, m, n, k) == cumsum(dhyper(x, m, n, k))) # FALSE
## but error is very small:
signif(phyper(x, m, n, k) - cumsum(dhyper(x, m, n, k)), dig=3)
```

**identify.hclust** Identify Clusters in a Dendrogram

Description

`identify.hclust` reads the position of the graphics pointer when the (first) mouse button is pressed. It then cuts the tree at the vertical position of the pointer and highlights the cluster containing the horizontal position of the pointer. Optionally a function is applied to the index of data points contained in the cluster.

Usage

```r
## S3 method for class 'hclust':
identify(x, FUN = NULL, N = 20, MAXCLUSTER = 20, DEV.FUN = NULL, ...)
```

Arguments

- **x**: an object of the type produced by `hclust`.
- **FUN**: (optional) function to be applied to the index numbers of the data points in a cluster (see Details below).
- **N**: the maximum number of clusters to be identified.
- **MAXCLUSTER**: the maximum number of clusters that can be produced by a cut (limits the effective vertical range of the pointer).
DEV.FUN  (optional) integer scalar. If specified, the corresponding graphics device is made active before FUN is applied.

...  further arguments to FUN.

Details

By default clusters can be identified using the mouse and an invisible list of indices of the respective data points is returned.

If FUN is not NULL, then the index vector of data points is passed to this function as first argument, see the examples below. The active graphics device for FUN can be specified using DEV.FUN.

The identification process is terminated by pressing any mouse button other than the first, see also identify.

Value

Either a list of data point index vectors or a list of return values of FUN.

See Also

hclust, rect.hclust

Examples

```r
## Not run:
hca <- hclust(dist(USArrests))
plot(hca)
(x <- identify(hca))  ## Terminate with 2nd mouse button !!!!

hci <- hclust(dist(iris[,1:4]))
plot(hci)
identify(hci, function(k) print(table(iris[k,5])))

# open a new device (one for dendrogram, one for bars):
get(getOption("device"))()  # << make that narrow (& small) and *beside* 1st one
nD <- dev.cur()               # to be for the barplot
dev.set(dev.prev())# old one for dendrogram
plot(hci)

## select subtrees in dendrogram and *see* the species distribution:
identify(hci, function(k) barplot(table(iris[k,5]),col=2:4), DEV.FUN = nD)
## End(Not run)
```

influence.measures  Regression Deletion Diagnostics

Description

This suite of functions can be used to compute some of the regression (leave-one-out deletion) diagnostics for linear and generalized linear models discussed in Belsley, Kuh and Welsch (1980), Cook and Weisberg (1982), etc.
Usage

influence.measures(model)

rstandard(model, ...)  ## S3 method for class 'lm':
rstandard(model, infl = lm.influence(model, do.coef = FALSE),
          sd = sqrt(deviance(model)/df.residual(model)), ...)

rstandard(model, ...)  ## S3 method for class 'glm':
rstandard(model, infl = influence(model, do.coef = FALSE), ...)

drstudent(model, ...)  ## S3 method for class 'lm':
drstudent(model, infl = lm.influence(model, do.coef = FALSE),
           res = infl$wt.res, ...)

drstudent(model, ...)  ## S3 method for class 'glm':
drstudent(model, infl = influence(model, do.coef = FALSE), ...)

dffits(model, infl = , res = )

dfbeta(model, ...)  ## S3 method for class 'lm':
dfbeta(model, infl = lm.influence(model, do.coef = TRUE), ...)

dfbeta(model, ...)  ## S3 method for class 'lm':
dfbetas(model, ...)

covratio(model, infl = lm.influence(model, do.coef = FALSE),
          res = weighted.residuals(model))

cooks.distance(model, ...)  ## S3 method for class 'lm':
cooks.distance(model, infl = lm.influence(model, do.coef = FALSE),
                res = weighted.residuals(model),
                sd = sqrt(deviance(model)/df.residual(model)),
                hat = infl$hat, ...)

ccooks.distance(model, ...)  ## S3 method for class 'glm':
cooks.distance(model, infl = influence(model, do.coef = FALSE),
                res = infl$pear.res,
                dispersion = summary(model)$dispersion,
                hat = infl$hat, ...)

hatvalues(model, ...)  ## S3 method for class 'lm':
hatvalues(model, infl = lm.influence(model, do.coef = FALSE), ...)

hat(x, intercept = TRUE)

Arguments

model an R object, typically returned by lm or glm.
infl influence structure as returned by lm.influence or influence (the latter
The primary high-level function is `influence.measures` which produces a class "infl" object tabular display showing the DFBETAS for each model variable, DFFITS, covariance ratios, Cook's distances and the diagonal elements of the hat matrix. Cases which are influential with respect to any of these measures are marked with an asterisk.

The functions `dfbetas`, `dffits`, `covratio` and `cooks.distance` provide direct access to the corresponding diagnostic quantities. Functions `rstandard` and `rstudent` give the standardized and Studentized residuals respectively. (These re-normalize the residuals to have unit variance, using an overall and leave-one-out measure of the error variance respectively.)

Values for generalized linear models are approximations, as described in Williams (1987) (except that Cook's distances are scaled as $F$ rather than as chi-square values).

The optional `infl`, `res` and `sd` arguments are there to encourage the use of these direct access functions, in situations where, e.g., the underlying basic influence measures (from `lm.influence` or the generic `influence`) are already available.

Note that cases with `weights == 0` are dropped from all these functions, but that if a linear model has been fitted with `na.action = na.exclude`, suitable values are filled in for the cases excluded during fitting.

The function `hat()` exists mainly for S (version 2) compatibility; we recommend using `hatvalues()` instead.

For `hatvalues`, `dfbeta`, and `dfbetas`, the method for linear models also works for generalized linear models.

Author(s)

Several R core team members and John Fox, originally in his 'car' package.

References


See Also

influence (containing lm.influence).

Examples

## Analysis of the life-cycle savings data
## given in Belsley, Kuh and Welsch.
lm.SR <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)

inflm.SR <- influence.measures(lm.SR)
which(apply(inflm.SR$is.inf, 1, any)) # which observations 'are' influential
summary(inflm.SR) # only these
inflm.SR # all
plot(rstudent(lm.SR) ~ hatvalues(lm.SR)) # recommended by some

## The 'infl' argument is not needed, but avoids recomputation:
rs <- rstandard(lm.SR)
iflSR <- influence(lm.SR)
identical(rs, rstandard(lm.SR, infl = iflSR))
## to "see" the larger values:
1000 * round(dfbetas(lm.SR, infl = iflSR), 3)

## Huber's data [Atkinson 1985]
xh <- c(-4:0, 10)
yh <- c(2.48, .73, -.04, -1.44, -1.32, 0)
summary(lmH <- lm(yh ~ xh))
(im <- influence.measures(lmH))
plot(xh,yh, main = "Huber's data: L.S. line and influential obs.")
abline(lmH); points(xh[im$is.inf], yh[im$is.inf], pch=20, col=2)

integrate

Integration of One-Dimensional Functions

Description

Adaptive quadrature of functions of one variable over a finite or infinite interval.

Usage

integrate(f, lower, upper, subdivisions=100, 
  rel.tol = .Machine$double.eps^0.25, abs.tol = rel.tol, 
  stop.on.error = TRUE, keep.xy = FALSE, aux = NULL, ...)

Arguments

f an R function taking a numeric first argument and returning a numeric vector of
the same length. Returning a non-finite element will generate an error.

lower, upper the limits of integration. Can be infinite.

subdivisions the maximum number of subintervals.

rel.tol relative accuracy requested.

abs.tol absolute accuracy requested.
stop.on.error

logical. If true (the default) an error stops the function. If false some errors will give a result with a warning in the message component.

keep.xy

unused. For compatibility with S.
aux

unused. For compatibility with S.

... additional arguments to be passed to f. Remember to use argument names not matching those of integrate(.)!

Details

If one or both limits are infinite, the infinite range is mapped onto a finite interval.

For a finite interval, globally adaptive interval subdivision is used in connection with extrapolation by the Epsilon algorithm.

d rel.tol cannot be less than max(50*Machine$double.eps, 0.5e-28) if abs.tol <= 0.

Value

A list of class "integrate" with components

value the final estimate of the integral.
abs.error estimate of the modulus of the absolute error.
subdivisions the number of subintervals produced in the subdivision process.
message "OK" or a character string giving the error message.
call the matched call.

Note

Like all numerical integration routines, these evaluate the function on a finite set of points. If the function is approximately constant (in particular, zero) over nearly all its range it is possible that the result and error estimate may be seriously wrong.

When integrating over infinite intervals do so explicitly, rather than just using a large number as the endpoint. This increases the chance of a correct answer – any function whose integral over an infinite interval is finite must be near zero for most of that interval.

References

Based on QUADPACK routines dqags and dqagi by R. Piessens and E. deDoncker-Kapenga, available from Netlib.

See

See Also

The function adapt in the adapt package on CRAN, for multivariate integration.
interaction.plot

Examples

```r
integrate(dnorm, -1.96, 1.96)
integrate(dnorm, -Inf, Inf)

## a slowly-convergent integral
integrand <- function(x) {1/((x+1)*sqrt(x))}
integrate(integrand, lower = 0, upper = Inf)

## don't do this if you really want the integral from 0 to Inf
integrate(integrand, lower = 0, upper = 10)
integrate(integrand, lower = 0, upper = 100000)
integrate(integrand, lower = 0, upper = 1000000, stop.on.error = FALSE)

try(integrate(function(x) 2, 0, 1)) ## no vectorizable function
integrate(function(x) rep(2, length(x)), 0, 1) ## correct

## integrate can fail if misused
integrate(dnorm,0,2)
integrate(dnorm,0,20)
integrate(dnorm,0,200)
integrate(dnorm,0,2000)
integrate(dnorm,0,20000) ## fails on many systems
integrate(dnorm,0,Inf) ## works
```

interaction.plot  Two-way Interaction Plot

Description

Plots the mean (or other summary) of the response for two-way combinations of factors, thereby illustrating possible interactions.

Usage

```r
interaction.plot(x.factor, trace.factor, response, fun = mean,
                 type = c("l", "p", "b"), legend = TRUE,
                 trace.label = deparse(substitute(trace.factor)),
                 fixed = FALSE,
                 xlab = deparse(substitute(x.factor)), ylab = ylabel,
                 ylim = range(cells, na.rm=TRUE),
                 lty = nc:1, col = 1, pch = c(1:9, 0, letters),
                 xpd = NULL, leg.bg = par("bg"), leg.bty = "n",
                 xtick = FALSE, xaxt = par("xaxt"), axes = TRUE, ...)
```

Arguments

- `x.factor`: a factor whose levels will form the x axis.
- `trace.factor`: another factor whose levels will form the traces.
- `response`: a numeric variable giving the response.
- `fun`: the function to compute the summary. Should return a single real value.
- `type`: the type of plot: lines or points.
interaction.plot

legend logical. Should a legend be included?
trace.label overall label for the legend.
fixed logical. Should the legend be in the order of the levels of trace.factor or in the order of the traces at their right-hand ends?
xlab, ylab the x and y label of the plot each with a sensible default.
ylim numeric of length 2 giving the y limits for the plot.
lty line type for the lines drawn, with sensible default.
col the color to be used for plotting.
pch a vector of plotting symbols or characters, with sensible default.
xpd determines clipping behaviour for the legend used, see par(xpd). Per default, the legend is not clipped at the figure border.
leg.bg, leg.bty arguments passed to legend().
xtick logical. Should tick marks be used on the x axis?
xaxt, axes, ... graphics parameters to be passed to the plotting routines.

Details

By default the levels of x.factor are plotted on the x axis in their given order, with extra space left at the right for the legend (if specified). If x.factor is an ordered factor and the levels are numeric, these numeric values are used for the x axis.

The response and hence its summary can contain missing values. If so, the missing values and the line segments joining them are omitted from the plot (and this can be somewhat disconcerting).

The graphics parameters xlab, ylab, ylim, lty, col and pch are given suitable defaults (and xlim and xaxs are set and cannot be overriden). The defaults are to cycle through the line types, use the foreground colour, and to use the symbols 1:9, 0, and the capital letters to plot the traces.

Note

Some of the argument names and the precise behaviour are chosen for S-compatibility.

References


Examples

attach(ToothGrowth)
interaction.plot(dose, supp, len, fixed=TRUE)
dose <- ordered(dose)
interaction.plot(dose, supp, len, fixed=TRUE, col = 2:3, leg.bty = "o")
interaction.plot(dose, supp, len, fixed=TRUE, col = 2:3, type = "p")
detach()

with(OrchardSprays, {
interaction.plot(treatment, rowpos, decrease)
interaction.plot(rowpos, treatment, decrease, cex.axis=0.8)
## order the rows by their mean effect

attach(OrchardSprays)
interaction.plot(treatment, rowpos, decrease)
interaction.plot(rowpos, treatment, decrease, cex.axis=0.8)
## order the rows by their mean effect
rowpos <- factor(rowpos, levels=sort.list(tapply(decrease, rowpos, mean)))
interaction.plot(rowpos, treatment, decrease, col = 2:9, lty = 1)
}

with(esoph, {
  interaction.plot(agegp, alcgp, ncases/ncontrols, main = "'esoph' Data")
  interaction.plot(agegp, tobgp, ncases/ncontrols, trace.label="tobacco",
                  fixed=TRUE, xaxt = "n")
})
## deal with NAs:
esoph[66,] # second to last age group: 65-74
esophNA <- esoph; esophNA$ncases[66] <- NA
with(esophNA, {
  interaction.plot(agegp, alcgp, ncases/ncontrols, col= 2:5)
    # doesn't show *last* group either
  interaction.plot(agegp, alcgp, ncases/ncontrols, col= 2:5, type = "b")
    ## alternative take non-NA's {"cheating"
  interaction.plot(agegp, alcgp, ncases/ncontrols, col= 2:5,
                  fun = function(x) mean(x, na.rm=TRUE),
                  sub = "function(x) mean(x, na.rm=TRUE)")
})
rm(esophNA) # to clear up

---

**IQR**

**The Interquartile Range**

**Description**

computes interquartile range of the \( x \) values.

**Usage**

\[
\text{IQR}(x, \text{na.rm} = \text{FALSE})
\]

**Arguments**

- **x**: a numeric vector.
- **na.rm**: logical. Should missing values be removed?

**Details**

Note that this function computes the quartiles using the `quantile` function rather than following Tukey's recommendations, i.e., \( \text{IQR}(x) = \text{quantile}(x, 3/4) - \text{quantile}(x, 1/4) \).

For normally \( N(m,1) \) distributed \( X \), the expected value of \( \text{IQR}(X) \) is \( 2 \times \text{qnorm}(3/4) = 1.3490 \), i.e., for a normal-consistent estimate of the standard deviation, use \( \text{IQR}(x) / 1.349 \).

**References**


**See Also**

`fivenum`, `mad` which is more robust, `range`, `quantile`. 
**is.empty.model**

**Check if a Model is Empty**

**Description**

R model notation allows models with no intercept and no predictors. These require special handling internally. `is.empty.model()` checks whether an object describes an empty model.

**Usage**

```r
is.empty.model(x)
```

**Arguments**

- `x` (A terms object or an object with a `terms` method.)

**Value**

TRUE if the model is empty

**See Also**

`lm`, `glm`

**Examples**

```r
y <- rnorm(20)
is.empty.model(y ~ 0)
is.empty.model(y ~ -1)
is.empty.model(lm(y ~ 0))
```

---

**isoreg**

**Isotonic / Monotone Regression**

**Description**

Compute the isotonic (monotonely increasing nonparametric) least squares regression which is piecewise constant.

**Usage**

```r
isoreg(x, y = NULL)
```

**Arguments**

- `x`, `y` (in `isoreg`, coordinate vectors of the regression points. Alternatively a single “plotting” structure can be specified: see `xy.coords`.)

---
Details

The algorithm determines the convex minorant \( m(x) \) of the cumulative data (i.e., \( \text{cumsum}(y) \)) which is piecewise linear and the result is \( m'(x) \), a step function with level changes at locations where the convex \( m(x) \) touches the cumulative data polygon and changes slope.

`as.stepfun()` returns a `stepfun` object which can be more parsimonious.

Value

`isoreg()` returns an object of class `isoreg` which is basically a list with components

- \( x \): original (constructed) abscissa values.
- \( y \): corresponding y values.
- \( yf \): fitted values corresponding to ordered \( x \) values.
- \( yc \): cumulative y values corresponding to ordered \( x \) values.
- \( iKnots \): integer vector giving indices where the fitted curve jumps, i.e., where the convex minorant has kinks.
- \( isOrd \): logical indicating if original \( x \) values were ordered increasingly already.
- \( ord \): if(!isOrd): integer permutation \( \text{order}(x) \) of original \( x \).
- \( call \): the `call` to `isoreg()` used.

Note

The code should be improved to accept `weights` additionally and solve the corresponding weighted least squares problem.

“Patches are welcome!”

References


See Also

the plotting method `plot.isoreg` with more examples; `isoMDS()` from the MASS package internally uses isotonic regression.

Examples

```r
(ir <- isoreg(c(1,0,4,3,3,5,4,2,0)))
plot(ir, plot.type = "row")

(ir3 <- isoreg(y3 <- c(1,0,4,3,3,5,4,2, 3)))# last "3", not "0"
(fi3 <- as.stepfun(ir3))
(ir4 <- isoreg(1:10, y4 <- c(5, 9, 1:2, 5:8, 3, 8)))
cat("R^2 =", formatC(sum(residuals(ir4)^2) / (9*var(y4)), dig=2),"n")
```
KalmanLike

Kalman Filtering

Description

Use Kalman Filtering to find the (Gaussian) log-likelihood, or for forecasting or smoothing.

Usage

KalmanLike(y, mod, nit = 0, fast=TRUE)
KalmanRun(y, mod, nit = 0, fast=TRUE)
KalmanSmooth(y, mod, nit = 0)
KalmanForecast(n.ahead = 10, mod, fast=TRUE)
makeARIMA(phi, theta, Delta, kappa = 1e6)

Arguments

y a univariate time series.
mod A list describing the state-space model: see Details.
nit The time at which the initialization is computed. nit = 0 implies that the initialization is for a one-step prediction, so \( P_n \) should not be computed at the first step.
n.ahead The number of steps ahead for which prediction is required.
phi, theta numeric vectors of length \( \geq 0 \) giving AR and MA parameters.
Delta vector of differencing coefficients, so an ARMA model is fitted to \( y[t] - \Delta[1] \times y[t-1] - \ldots \).
kappa the prior variance (as a multiple of the innovations variance) for the past observations in a differenced model.
fast If TRUE the mod object may be modified.

Details

These functions work with a general univariate state-space model with state vector \( a \), transitions \( a \leftarrow T \cdot a + R \cdot e \sim N(0, \kappa Q) \) and observation equation \( y = Z' \cdot a + \text{eta}, (\text{eta} \equiv \eta), \eta \sim N(0, \kappa h) \). The likelihood is a profile likelihood after estimation of \( \kappa \).

The model is specified as a list with at least components

- **T** the transition matrix
- **Z** the observation coefficients
- **h** the observation variance
- **V** \( \text{RQR}' \)
- **a** the current state estimate
- **P** the current estimate of the state uncertainty matrix
- **Pn** the estimate at time \( t - 1 \) of the state uncertainty matrix

KalmanSmooth is the workhorse function for tsSmooth.
makeARIMA constructs the state-space model for an ARIMA model.
Value

For KalmanLike, a list with components Lik (the log-likelihood less some constants) and s2, the estimate of of $\kappa$.

For KalmanRun, a list with components values, a vector of length 2 giving the output of KalmanLike, resid (the residuals) and states, the contemporaneous state estimates, a matrix with one row for each time.

For KalmanSmooth, a list with two components. Component smooth is a n by p matrix of state estimates based on all the observations, with one row for each time. Component var is a n by p by p array of variance matrices.

For KalmanForecast, a list with components pred, the predictions, and var, the unscaled variances of the prediction errors (to be multiplied by s2).

For makeARIMA, a model list including components for its arguments.

Warning

These functions are designed to be called from other functions which check the validity of the arguments passed, so very little checking is done.

In particular, KalmanLike alters the objects passed as the elements a, P and Pn of mod, so these should not be shared. Use fast=FALSE to prevent this.

References


See Also

arima, StructTS, tsSmooth.

Description

kernapply computes the convolution between an input sequence and a specific kernel.

Usage

kernapply(x, k, circular = FALSE, ...)
kernapply(k1, k2)

Arguments

k, k1, k2 smoothing "tskernel" objects.

x an input vector, matrix, or time series to be smoothed.

circular a logical indicating whether the input sequence to be smoothed is treated as circular, i.e., periodic.

... arguments passed to or from other methods.
**kernel**  

**Value**  
A smoothed version of the input sequence.

**Author(s)**  
A. Trapletti

**See Also**  
kernel, convolve, filter, spectrum

**Examples**  
```r  
## see 'kernel' for examples
```

---

### Description

The "tskernel" class is designed to represent discrete symmetric normalized smoothing kernels. These kernels can be used to smooth vectors, matrices, or time series objects.

There are print, plot and [ methods for these kernel objects.

### Usage

```r  
kernel(coef, m, r, name)  
df.kernel(k)  
bandwidth.kernel(k)  
is.tskernel(k)  
## S3 method for class 'tskernel':  
plot(x, type = "h", xlab = "k", ylab = "W[k]",  
     main = attr(x,"name"), ...)  
```

### Arguments

- `coef`: the upper half of the smoothing kernel coefficients (including coefficient zero)  
  or the name of a kernel (currently "daniell", "dirichlet", "fejer" or 
  "modified.daniell").
- `m`: the kernel dimension(s). When `m` has length larger than one, it means the convolution of kernels of dimension `m[j]`, for `j` in 1:length(`m`). Currently this is supported only for the named "daniell" kernels.
- `name`: the name the kernel will be called.
- `r`: the kernel order for a Fejer kernel.
- `k,x`: a "tskernel" object.
- `type, xlab, ylab, main, ...`: arguments passed to `plot.default`. 

---
Details

kernel is used to construct a general kernel or named specific kernels. The modified Daniell kernel halves the end coefficients (as used by S-PLUS).

The \{ method allows natural indexing of kernel objects with indices in \((-m) : m\). The normalization is such that for \(k \leftarrow \text{kernel}(*)\), \(\sum(k_{-k\cdot m} : k\cdot m)\) is one.

df.kernel returns the “equivalent degrees of freedom” of a smoothing kernel as defined in Brockwell and Davies (1991), page 362, and bandwidth.kernel returns the equivalent bandwidth as defined in Bloomfield (1976), p. 201, with a continuity correction.

Value

kernel() returns an object of class "tskernel" which is basically a list with the two components coef and the kernel dimension \(m\). An additional attribute is "name".

Author(s)

A. Trapletti; modifications by B.D. Ripley

References


See Also

cernapply

Examples

```r
## Demonstrate a simple trading strategy for the
## financial time series German stock index DAX.
x <- EuStockMarkets[,1]
k1 <- kernel("daniell", 50) # a long moving average
k2 <- kernel("daniell", 10) # and a short one
plot(k1)
plot(k2)
x1 <- kernapply(x, k1)
x2 <- kernapply(x, k2)
plot(x)
lines(x1, col = "red") # go long if the short crosses the long upwards
lines(x2, col = "green") # and go short otherwise

## More interesting kernels
kd <- kernel("daniell", c(3,3))
kd # note the unusual indexing
kd[-2:2]
plot(kernel("fejer", 100, r=6))
plot(kernel("modified.daniell", c(7,5,3)))

# Reproduce example 10.4.3 from Brockwell and Davies (1991)
spectrum(sunspot.year, kernel=kernel("daniell", c(11,7,3)), log="no")
```
**kmeans**

**K-Means Clustering**

**Description**

Perform k-means clustering on a data matrix.

**Usage**

```r
kmeans(x, centers, iter.max = 10, nstart = 1,
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"))
```

**Arguments**

- `x` A numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).
- `centers` Either the number of clusters or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in `x` is chosen as the initial centres.
- `iter.max` The maximum number of iterations allowed.
- `nstart` If `centers` is a number, how many random sets should be chosen?
- `algorithm` character: may be abbreviated.

**Details**

The data given by `x` is clustered by the $k$-means method, which aims to partition the points into $k$ groups such that the sum of squares from points to the assigned cluster centres is minimized. At the minimum, all cluster centres are at the mean of their Voronoi sets (the set of data points which are nearest to the cluster centre).

The algorithm of Hartigan and Wong (1979) is used by default. Note that some authors use $k$-means to refer to a specific algorithm rather than the general method: most commonly the algorithm given by MacQueen (1967) but sometimes that given by Lloyd (1957) and Forgy (1965). The Hartigan–Wong algorithm generally does a better job than either of those, but trying several random starts is often recommended.

Except for the Lloyd–Forgy method, $k$ clusters will always be returned if a number is specified. If an initial matrix of centres is supplied, it is possible that no point will be closest to one or more centres, which is currently an error for the Hartigan–Wong method.

**Value**

An object of class "kmeans" which is a list with components:

- `cluster` A vector of integers indicating the cluster to which each point is allocated.
- `centers` A matrix of cluster centres.
- `withinss` The within-cluster sum of squares for each cluster.
- `size` The number of points in each cluster.

There is a `print` method for this class.
References


Examples

```r
# a 2-dimensional example
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
            matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
colnames(x) <- c("x", "y")
(cl <- kmeans(x, 2))
plot(x, col = cl$cluster)
points(cl$centers, col = 1:2, pch = 8, cex=2)

## random starts do help here with too many clusters
(cl <- kmeans(x, 5, nstart = 25))
plot(x, col = cl$cluster)
points(cl$centers, col = 1:5, pch = 8)
```

---

**kruskal.test**  
*Kruskal-Wallis Rank Sum Test*

**Description**

Performs a Kruskal-Wallis rank sum test.

**Usage**

```r
kruskal.test(x, ...)
```

## Default S3 method: 
```
kruksal.test(x, g, ...)
```

## S3 method for class 'formula':
```
kruksal.test(formula, data, subset, na.action, ...)
```

**Arguments**

- **x**: a numeric vector of data values, or a list of numeric data vectors.
- **g**: a vector or factor object giving the group for the corresponding elements of x. Ignored if x is a list.
- **formula**: a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
kruskal.test

data  an optional data frame containing the variables in the model formula.
subset an optional vector specifying a subset of observations to be used.
na.action a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
... further arguments to be passed to or from methods.

Details

kruskal.test performs a Kruskal-Wallis rank sum test of the null that the location parameters of the distribution of x are the same in each group (sample). The alternative is that they differ in at least one.

If x is a list, its elements are taken as the samples to be compared, and hence have to be numeric data vectors. In this case, g is ignored, and one can simply use kruskal.test(x) to perform the test. If the samples are not yet contained in a list, use kruskal.test(list(x, ...)).

Otherwise, x must be a numeric data vector, and g must be a vector or factor object of the same length as x giving the group for the corresponding elements of x.

Value

A list with class "htest" containing the following components:

- statistic the Kruskal-Wallis rank sum statistic.
- parameter the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- p.value the p-value of the test.
- method the character string "Kruskal-Wallis rank sum test".
- data.name a character string giving the names of the data.

References


See Also

The Wilcoxon rank sum test (wilcox.test) as the special case for two samples; lm together with anova for performing one-way location analysis under normality assumptions; with Student’s t test (t.test) as the special case for two samples.

Examples

```r
## Hollander & Wolfe (1973), 116.
## Mucociliary efficiency from the rate of removal of dust in normal
## subjects, subjects with obstructive airway disease, and subjects
## with asbestosis.
x <- c(2.9, 3.0, 2.5, 2.6, 3.2) # normal subjects
y <- c(3.8, 2.7, 4.0, 2.4) # with obstructive airway disease
z <- c(2.8, 3.4, 3.7, 2.2, 2.0) # with asbestosis
kruskal.test(list(x, y, z))
## Equivalently,
x <- c(x, y, z)
g <- factor(rep(1:3, c(5, 4, 5)),
```
labels = c("Normal subjects",
          "Subjects with obstructive airway disease",
          "Subjects with asbestosis")

kruskal.test(x, g)

## Formula interface.
boxplot(Ozone ~ Month, data = airquality)
kruskal.test(Ozone ~ Month, data = airquality)

---

**ks.test**

**Kolmogorov-Smirnov Tests**

**Description**

Performs one or two sample Kolmogorov-Smirnov tests.

**Usage**

```r
ks.test(x, y, ..., alternative = c("two.sided", "less", "greater"),
        exact = NULL)
```

**Arguments**

- `x` a numeric vector of data values.
- `y` either a numeric vector of data values, or a character string naming a distribution function.
- `...` parameters of the distribution specified (as a character string) by `y`.
- `alternative` indicates the alternative hypothesis and must be one of "two.sided" (default), "less", or "greater". You can specify just the initial letter.
- `exact` NULL or a logical indicating whether an exact p-value should be computed. See Details for the meaning of NULL. Not used for the one-sided two-sample case.

**Details**

If `y` is numeric, a two-sample test of the null hypothesis that `x` and `y` were drawn from the same continuous distribution is performed.

Alternatively, `y` can be a character string naming a continuous distribution function. In this case, a one-sample test is carried out of the null that the distribution function which generated `x` is distribution `y` with parameters specified by `...`.

The presence of ties generates a warning, since continuous distributions do not generate them.

The possible values "two.sided", "less" and "greater" of `alternative` specify the null hypothesis that the true distribution function of `x` is equal to, not less than or not greater than the hypothesized distribution function (one-sample case) or the distribution function of `y` (two-sample case), respectively.

Exact p-values are not available for the one-sided two-sample case, or in the case of ties. `exact = NULL` (the default), an exact p-value is computed if the sample size if less than 100 in the one-sample case, and if the product of the sample sizes is less than 10000 in the two-sample case. Otherwise, asymptotic distributions are used whose approximations may be inaccurate in small samples. In the one-sample two-sided case, exact p-values are obtained as described in Marsaglia,
Tsang & Wang (2003). The formula of Birnbaum & Tingey (1951) is used for the one-sample one-sided case.

If a single-sample test is used, the parameters specified in . . . must be pre-specified and not estimated from the data. There is some more refined distribution theory for the KS test with estimated parameters (see Durbin, 1973), but that is not implemented in ks.test.

Value

A list with class "htest" containing the following components:

- statistic: the value of the test statistic.
- p.value: the p-value of the test.
- alternative: a character string describing the alternative hypothesis.
- method: a character string indicating what type of test was performed.
- data.name: a character string giving the name(s) of the data.

References


Durbin, J. (1973) Distribution theory for tests based on the sample distribution function. SIAM.


See Also

- shapiro.test which performs the Shapiro-Wilk test for normality.

Examples

    x <- rnorm(50)
    y <- runif(30)
    # Do x and y come from the same distribution?
    ks.test(x, y)
    # Does x come from a shifted gamma distribution with shape 3 and rate 2?
    ks.test(x+2, "pgamma", 3, 2) # two-sided, exact
    ks.test(x+2, "pgamma", 3, 2, exact = FALSE)
    ks.test(x+2, "pgamma", 3, 2, alternative = "gr")

ksmooth

Kernel Regression Smoother

Description

The Nadaraya-Watson kernel regression estimate.
Usage

ksmooth(x, y, kernel = c("box", "normal"), bandwidth = 0.5,
        range.x = range(x), n.points = max(100, length(x)), x.points)

Arguments

x         input x values
y         input y values
kernel    the kernel to be used.
bandwidth the bandwidth. The kernels are scaled so that their quartiles (viewed as probability densities) are at ± 0.25 * bandwidth.
range.x   the range of points to be covered in the output.
n.points  the number of points at which to evaluate the fit.
x.points  points at which to evaluate the smoothed fit. If missing, n.points are chosen uniformly to cover range.x.

Value

A list with components

x         values at which the smoothed fit is evaluated. Guaranteed to be in increasing order.
y         fitted values corresponding to x.

Note

This function is implemented purely for compatibility with S, although it is nowhere near as slow as the S function. Better kernel smoothers are available in other packages.

Examples

with(cars, {
  plot(speed, dist)
  lines(ksmooth(speed, dist, "normal", bandwidth=2), col=2)
  lines(ksmooth(speed, dist, "normal", bandwidth=5), col=3)
})

---

lag       Lag a Time Series

Description

Compute a lagged version of a time series, shifting the time base back by a given number of observations.

Usage

lag(x, ...)

## Default S3 method:
lag(x, k = 1, ...)

lag

Lag a Time Series
**Arguments**

- **x** A vector or matrix or univariate or multivariate time series
- **k** The number of lags (in units of observations).
- ... further arguments to be passed to or from methods.

**Details**

Vector or matrix arguments `x` are coerced to time series.

`lag` is a generic function; this page documents its default method.

**Value**

A time series object.

**Note**

Note the sign of `k`: a series lagged by a positive `k` starts *earlier*.

**References**


**See Also**

`diff`, `deltat`

**Examples**

```r
lag(ldeaths, 12) # starts one year earlier
```

---

**Description**

Plot time series against lagged versions of themselves. Helps visualizing “auto-dependence” even when auto-correlations vanish.

**Usage**

```r
lag.plot(x, lags = 1, layout = NULL, set.lags = 1:lags,
        main = NULL, asp = 1,
        font.main=par("font.main"), cex.main=par("cex.main"),
        diag = TRUE, diag.col = "gray", type = "p", oma = NULL,
        ask = NULL, do.lines = (n <= 150), labels = do.lines, ...)
```
Arguments

- **x**: time-series (univariate or multivariate)
- **lags**: number of lag plots desired, see arg `set.lags`
- **layout**: the layout of multiple plots, basically the `mfrow` `par()` argument. The default uses about a square layout (see `n2mfrow` such that all plots are on one page.
- **set.lags**: positive integer vector allowing to specify the set of lags used; defaults to `1:lags`
- **main**: character with a main header title to be done on the top of each page.
- **asp**: Aspect ratio to be fixed, see `plot.default`
- **font.main, cex.main**: attributes for the title, see `par()`
- **diag**: logical indicating if the x=y diagonal should be drawn.
- **diag.col**: color to be used for the diagonal if `diag`.
- **type**: plot type to be used, but see `plot.ts` about its restricted meaning.
- **oma**: outer margins, see `par`
- **ask**: logical; if true, the user is asked before a new page is started.
- **do.lines**: logical indicating if lines should be drawn.
- **labels**: logical indicating if labels should be used.
- **...**: Further arguments to `plot.ts`

Note

It is more flexible and has different default behaviour than the S version. We use `main =` instead of `head =` for internal consistency.

Author(s)

Martin Maechler

See Also

- `plot.ts` which is the basic work horse.

Examples

```r
lag.plot(nhtemp, 8, diag.col = "forest green")
lag.plot(nhtemp, 5, main="Average Temperatures in New Haven")
## ask defaults to TRUE when we have more than one page:
lag.plot(nhtemp, 6, layout = c(2,1), asp = NA,
       main = "New Haven Temperatures", col.main = "blue")

## Multivariate (but non-stationary! ...)
lag.plot(freeny.x, lag = 3)
## Not run:
no lines for long series :
lag.plot(sqrt(sunspots), set = c(1:4, 9:12), pch = ".", col = "gold")
## End(Not run)
```
Robust Line Fitting

Description

Fit a line robustly as recommended in Exploratory Data Analysis.

Usage

```r
line(x, y)
```

Arguments

- `x`, `y` the arguments can be any way of specifying x-y pairs.

Value

An object of class "tukeyline".
Methods are available for the generic functions `coef`, `residuals`, `fitted`, and `print`.

References


See Also

`lm`.

Examples

```r
plot(cars)
(z <- line(cars))
abline(coef(z))
## Tukey-Anscombe Plot :
plot(residuals(z) ~ fitted(z), main = deparse(z$call))
```

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```r
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```
Arguments

**formula**
a symbolic description of the model to be fit. The details of model specification
are given below.

**data**
an optional data frame containing the variables in the model. If not found in
data, the variables are taken from `environment(formula)`, typically the
environment from which `lm` is called.

**subset**
an optional vector specifying a subset of observations to be used in the fitting
process.

**weights**
an optional vector of weights to be used in the fitting process. If specified,
weighted least squares is used with weights `weights` (that is, minimizing
`sum(w*e^2)`); otherwise ordinary least squares is used.

**na.action**
a function which indicates what should happen when the data contain NAs. The
default is set by the `na.action` setting of `options`, and is `na.fail` if that
is unset. The “factory-fresh” default is `na.omit`. Another possible value is
`NULL`, no action.

**method**
the method to be used; for fitting, currently only `method = "qr"` is sup-
ported; `method = "model.frame"` returns the model frame (the same as
with `model = TRUE`, see below).

**model, x, y, qr**
logicals. If TRUE the corresponding components of the fit (the model frame, the
model matrix, the response, the QR decomposition) are returned.

**singular.ok**
logical. If FALSE (the default in S but not in R) a singular fit is an error.

**contrasts**
an optional list. See the `contrasts.arg` of `model.matrix.default`.

**offset**
this can be used to specify an a priori known component to be included in the
linear predictor during fitting. An `offset` term can be included in the formula
instead or as well, and if both are specified their sum is used.

**...**
additional arguments to be passed to the low level regression fitting functions
(see below).

Details

Models for `lm` are specified symbolically. A typical model has the form `response ~ terms`
where `response` is the (numeric) response vector and `terms` is a series of terms which specifies a
linear predictor for `response`. A terms specification of the form `first + second` indicates all
the terms in `first` together with all the terms in `second` with duplicates removed. A specification
of the form `first:second` indicates the set of terms obtained by taking the interactions of all
terms in `first` with all terms in `second`. The specification `first*second` indicates the cross
of `first` and `second`. This is the same as `first + second + first:second`.

If `response` is a matrix a linear model is fitted separately by least-squares to each column of the
matrix.

See `model.matrix` for some further details. The terms in the formula will be re-ordered so that
main effects come first, followed by the interactions, all second-order, all third-order and so on: to
avoid this pass a `terms` object as the formula.

A formula has an implied intercept term. To remove this use either `y ~ x - 1` or `y ~ 0 + x`.
See `formula` for more details of allowed formulae.

`lm` calls the lower level functions `lm.fit`, etc, see below, for the actual numerical computations.
For programming only, you may consider doing likewise.

All of `weights`, `subset` and `offset` are evaluated in the same way as variables in `formula`,
that is first in `data` and then in the environment of `formula`. 
**Value**

`lm` returns an object of class "lm" or for multiple responses of class `c("mlm", "lm")`. The functions `summary` and `anova` are used to obtain and print a summary and analysis of variance table of the results. The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

An object of class "lm" is a list containing at least the following components:

- `coefficients` a named vector of coefficients
- `residuals` the residuals, that is response minus fitted values.
- `fitted.values` the fitted mean values.
- `rank` the numeric rank of the fitted linear model.
- `weights` (only for weighted fits) the specified weights.
- `df.residual` the residual degrees of freedom.
- `call` the matched call.
- `terms` the `terms` object used.
- `contrasts` (only where relevant) the contrasts used.
- `xlevels` (only where relevant) a record of the levels of the factors used in fitting.
- `offset` the offset used (missing if none were used).
- `y` if requested, the response used.
- `x` if requested, the model matrix used.
- `model` if requested (the default), the model frame used.

In addition, non-null fits will have components `assign`, `effects` and (unless not requested) `qr` relating to the linear fit, for use by extractor functions such as `summary` and `effects`.

**Using time series**

Considerable care is needed when using `lm` with time series.

Unless `na.action = NULL`, the time series attributes are stripped from the variables before the regression is done. (This is necessary as omitting NAs would invalidate the time series attributes, and if NAs are omitted in the middle of the series the result would no longer be a regular time series.)

Even if the time series attributes are retained, they are not used to line up series, so that the time shift of a lagged or differenced regressor would be ignored. It is good practice to prepare a data argument by `ts.intersect(...)`, then apply a suitable `na.action` to that data frame and call `lm` with `na.action = NULL` so that residuals and fitted values are time series.

**Note**

Offsets specified by `offset` will not be included in predictions by `predict.lm`, whereas those specified by an offset term in the formula will be.

**Author(s)**

The design was inspired by the S function of the same name described in Chambers (1992). The implementation of model formula by Ross Ihaka was based on Wilkinson & Rogers (1973).
References


See Also

`summary.lm` for summaries and `anova.lm` for the ANOVA table; `aov` for a different interface.

The generic functions `coef, effects, residuals, fitted, vcov.`

`predict.lm` (via `predict`) for prediction, including confidence and prediction intervals;

`confint` for confidence intervals of `parameters`.

`lm.influence` for regression diagnostics, and `glm` for `generalized` linear models.

The underlying low level functions, `lm.fit` for plain, and `lm.wfit` for weighted regression fitting.

Examples

```r
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2,10,20, labels=c("Ctl","Trt"))
weight <- c(ctl, trt)

anova(lm.D9 <- lm(weight ~ group))

summary(lm.D90 <- lm(weight ~ group - 1)) # omitting intercept
summary(resid(lm.D9) - resid(lm.D90)) # residuals almost identical

opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(lm.D9, las = 1) # Residuals, Fitted, ...
par(opar)

# model frame :
stopifnot(identical(lm(weight ~ group, method = "model.frame"),
                    model.frame(lm.D9)))
```

---

**lm.fit**  
*Fitter Functions for Linear Models*

**Description**

These are the basic computing engines called by `lm` used to fit linear models. These should usually not be used directly unless by experienced users.

**Usage**

```r
lm.fit (x, y, offset = NULL, method = "qr", tol = 1e-7, singular.ok = TRUE, ...)
```

```r
lm.wfit(x, y, w, offset = NULL, method = "qr", tol = 1e-7, singular.ok = TRUE, ...)
```
Arguments

x design matrix of dimension \( n \times p \).
y vector of observations of length \( n \), or a matrix with \( n \) rows.
w vector of weights (length \( n \)) to be used in the fitting process for the \( w \)fit functions. Weighted least squares is used with weights \( w \), i.e., \( \sum(w \times e^2) \) is minimized.
offset numeric of length \( n \). This can be used to specify an \( a \ priori \) known component to be included in the linear predictor during fitting.
method currently, only method="qr" is supported.
tol tolerance for the \( qr \) decomposition. Default is 1e-7.
singular.ok logical. If FALSE, a singular model is an error.
... currently disregarded.

Value

a list with components

coefficients p vector
residuals n vector or matrix
fitted.values n vector or matrix
effects (not null fits): vector of orthogonal single-df effects. The first rank of them correspond to non-aliased coefficients, and are named accordingly.
weights n vector — only for the \( \ast \)fit\( \ast \) functions.
rack integer, giving the rank
df.residual degrees of freedom of residuals
qr (not null fits) the QR decomposition, see \( qr \).

See Also

\( lm \) which you should use for linear least squares regression, unless you know better.

Examples

set.seed(129)
n <- 7; p <- 2
X <- matrix(rnorm(n * p), n,p) # no intercept!
y <- rnorm(n)
w <- rnorm(n)^2
str(lmw <- lm.wfit(x=X, y=y, w=w))
str(lm. <- lm.fit (x=X, y=y))
Description

This function provides the basic quantities which are used in forming a wide variety of diagnostics for checking the quality of regression fits.

Usage

\[
\text{influence(model, \ldots)}
\]

## S3 method for class 'lm':
\[
\text{influence(model, do.coef = TRUE, \ldots)}
\]

## S3 method for class 'glm':
\[
\text{influence(model, do.coef = TRUE, \ldots)}
\]

\[
\text{lm.influence(model, do.coef = TRUE)}
\]

Arguments

- **model**: an object as returned by \texttt{lm}.
- **do.coef**: logical indicating if the changed \texttt{coefficients} (see below) are desired. These need \(O(n^2 p)\) computing time.
- **\ldots**: further arguments passed to or from other methods.

Details

The \texttt{influence.measures()} and other functions listed in \textbf{See Also} provide a more user oriented way of computing a variety of regression diagnostics. These all build on \texttt{lm.influence}.

An attempt is made to ensure that computed hat values that are probably one are treated as one, and the corresponding rows in \texttt{sigma} and \texttt{coefficients} are NaN. (Dropping such a case would normally result in a variable being dropped, so it is not possible to give simple drop-one diagnostics.)

Value

A list containing the following components of the same length or number of rows \(n\), which is the number of non-zero weights. Cases omitted in the fit are omitted unless a \texttt{na.action} method was used (such as \texttt{na.exclude}) which restores them.

- **hat**: a vector containing the diagonal of the “hat” matrix.
- **coefficients** (unless do.coef is false) a matrix whose i-th row contains the change in the estimated coefficients which results when the i-th case is dropped from the regression. Note that aliased coefficients are not included in the matrix.
- **sigma**: a vector whose i-th element contains the estimate of the residual standard deviation obtained when the i-th case is dropped from the regression.
- **wt.res**: a vector of weighted (or for class \texttt{glm} rather \texttt{deviance}) residuals.
Note

The coefficients returned by the R version of lm.influence differ from those computed by S. Rather than returning the coefficients which result from dropping each case, we return the changes in the coefficients. This is more directly useful in many diagnostic measures. Since these need \(O(n^2p)\) computing time, they can be omitted by do.coef = FALSE.

Note that cases with weights == 0 are dropped (contrary to the situation in S).

If a model has been fitted with na.action=na.exclude (see na.exclude), cases excluded in the fit are considered here.

References

See the list in the documentation for influence.measures.


See Also

summary.lm for summary and related methods;
influence.measures,
hat for the hat matrix diagonals,
dfbetas, dffits, covratio, cooks.distance, lm.

Examples

```r
## Analysis of the life-cycle savings data
## given in Belsley, Kuh and Welsch.
summary(lm.SR <- lm(sr ~ pop15 + pop75 + dpi + ddpi,
                   data = LifeCycleSavings),
       corr = TRUE)
str(lmI <- lm.influence(lm.SR))

## For more "user level" examples, use example(influence.measures)
```

Description

All these functions are methods for class "lm" objects.

Usage

```r
## S3 method for class 'lm':
family(object, ...)

## S3 method for class 'lm':
formula(x, ...)

## S3 method for class 'lm':
residuals(object,
          type = c("working", "response", "deviance", "pearson",
```

Arguments

object, x an object inheriting from class `lm`, usually the result of a call to `lm` or `aov`.
...
  further arguments passed to or from other methods.
type the type of residuals which should be returned.

Details

The generic accessor functions `coef`, `effects`, `fitted` and `residuals` can be used to extract various useful features of the value returned by `lm`.

The working and response residuals are “observed - fitted”. The deviance and pearson residuals are weighted residuals, scaled by the square root of the weights used in fitting. The partial residuals are a matrix with each column formed by omitting a term from the model. In all these, zero weight cases are never omitted (as opposed to the standardized `rstudent` residuals, and the `weighted.residuals`).

The "lm" method for generic `labels` returns the term labels for estimable terms, that is the names of the terms with an least one estimable coefficient.

References


See Also

The model fitting function `lm`, `anova.lm`.
`coef`, `deviance`, `df.residual`, `effects`, `fitted`, `glm` for generalized linear models, `influence` (etc on that page) for regression diagnostics, `weighted.residuals`, `residuals`, `residuals.glm`, `summary.lm`.

Examples

```r
##-- Continuing the lm(.) example:
coef(lm.D90)# the bare coefficients

## The 2 basic regression diagnostic plots [plot.lm(.) is preferred]
plot(resid(lm.D90), fitted(lm.D90))# Tukey-Anscombe's
abline(h=0, lty=2, col = 'gray')

qqnorm(residuals(lm.D90))
```
loadings

Print Loadings in Factor Analysis

Description

Extract or print loadings in factor analysis (or principal components analysis).

Usage

```r
loadings(x)

## S3 method for class 'loadings':
print(x, digits = 3, cutoff = 0.1, sort = FALSE, ...)

## S3 method for class 'factanal':
print(x, digits = 3, ...)
```

Arguments

- `x` an object of class "factanal" or "princomp" or the `loadings` component of such an object.
- `digits` number of decimal places to use in printing uniquenesses and loadings.
- `cutoff` loadings smaller than this (in absolute value) are suppressed.
- `sort` logical. If true, the variables are sorted by their importance on each factor. Each variable with any loading larger than 0.5 (in modulus) is assigned to the factor with the largest loading, and the variables are printed in the order of the factor they are assigned to, then those unassigned.
- `...` further arguments for other methods, such as `cutoff` and `sort` for `print.factanal`.

See Also

`factanal`, `princomp`

loess

Local Polynomial Regression Fitting

Description

Fit a polynomial surface determined by one or more numerical predictors, using local fitting.

Usage

```r
loess(formula, data, weights, subset, na.action, model = FALSE,
span = 0.75, enp.target, degree = 2,
parametric = FALSE, drop.square = FALSE, normalize = TRUE,
family = c("gaussian", "symmetric"),
method = c("loess", "model.frame"),
control = loess.control(...), ...)
```
Arguments

- **formula**: a formula specifying the numeric response and one to four numeric predictors (best specified via an interaction, but can also be specified additively).
- **data**: an optional data frame within which to look first for the response, predictors and weights.
- **weights**: optional weights for each case.
- **subset**: an optional specification of a subset of the data to be used.
- **na.action**: the action to be taken with missing values in the response or predictors. The default is given by `getOption("na.action")`.
- **model**: should the model frame be returned?
- **span**: the parameter $\alpha$ which controls the degree of smoothing.
- **enp.target**: an alternative way to specify `span`, as the approximate equivalent number of parameters to be used.
- **degree**: the degree of the polynomials to be used, up to 2.
- **parametric**: should any terms be fitted globally rather than locally? Terms can be specified by name, number or as a logical vector of the same length as the number of predictors.
- **drop.square**: for fits with more than one predictor and `degree=2`, should the quadratic term (and cross-terms) be dropped for particular predictors? Terms are specified in the same way as for `parametric`.
- **normalize**: should the predictors be normalized to a common scale if there is more than one? The normalization used is to set the 10% trimmed standard deviation to one. Set to false for spatial coordinate predictors and others known to be a common scale.
- **family**: if "gaussian" fitting is by least-squares, and if "symmetric" a re-descending M estimator is used with Tukey’s biweight function.
- **method**: fit the model or just extract the model frame.
- **control**: control parameters: see `loess.control`.
- **...**: control parameters can also be supplied directly.

Details

Fitting is done locally. That is, for the fit at point $x$, the fit is made using points in a neighbourhood of $x$, weighted by their distance from $x$ (with differences in 'parametric' variables being ignored when computing the distance). The size of the neighbourhood is controlled by $\alpha$ (set by `span` or `enp.target`). For $\alpha < 1$, the neighbourhood includes proportion $\alpha$ of the points, and these have tricubic weighting (proportional to $\left(1 - \left(\text{dist}/\maxdist\right)^3\right)^3$. For $\alpha > 1$, all points are used, with the ‘maximum distance’ assumed to be $\alpha^{1/p}$ times the actual maximum distance for $p$ explanatory variables.

For the default family, fitting is by (weighted) least squares. For family="symmetric" a few iterations of an M-estimation procedure with Tukey’s biweight are used. Be aware that as the initial value is the least-squares fit, this need not be a very resistant fit.

It can be important to tune the control list to achieve acceptable speed. See `loess.control` for details.

Value

An object of class "loess".
**Note**

As this is based on the cloess package available at netlib, it is similar to but not identical to the loess function of S. In particular, conditioning is not implemented.

The memory usage of this implementation of loess is roughly quadratic in the number of points, with 1000 points taking about 10Mb.

**Author(s)**


**References**


**See Also**

loess.control, predict.loess.

cloess, the ancestor of loess (with different defaults!).

**Examples**

```r
   cars.lo <- loess(dist ~ speed, cars)
predict(cars.lo, data.frame(speed = seq(5, 30, 1)), se = TRUE)
# to allow extrapolation
cars.lo2 <- loess(dist ~ speed, cars,
   control = loess.control(surface = "direct"))
predict(cars.lo2, data.frame(speed = seq(5, 30, 1)), se = TRUE)
```

---

**Description**

Set control parameters for loess fits.

**Usage**

```r
   loess.control(surface = c("interpolate", "direct"),
      statistics = c("approximate", "exact"),
      trace.hat = c("exact", "approximate"),
      cell = 0.2, iterations = 4, ...)
```

**Arguments**

- `surface` should be fitted surface be computed exactly or via interpolation from a kd tree?
- `statistics` should the statistics be computed exactly or approximately? Exact computation can be very slow.
should the trace of the smoother matrix be computed exactly or approximately? It is recommended to use the approximation for more than about 1000 data points. if interpolation is used this controls the accuracy of the approximation via the maximum number of points in a cell in the kd tree. Cells with more than \( \text{floor}(n \times \text{span} \times \text{cell}) \) points are subdivided. the number of iterations used in robust fitting. further arguments which are ignored.

A list with components

- surface
- statistics
- trace.hat
- cell
- iterations

with meanings as explained under ‘Arguments’.

The Logistic Distribution

Density, distribution function, quantile function and random generation for the logistic distribution with parameters location and scale.

Usage

```r
dlogis(x, location = 0, scale = 1, log = FALSE)
plogis(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
qlogis(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
rlogis(n, location = 0, scale = 1)
```

Arguments

- `x, q` vector of quantiles.
- `p` vector of probabilities.
- `n` number of observations. If \( \text{length}(n) > 1 \), the length is taken to be the number required.
- `location, scale` location and scale parameters.
- `log, log.p` logical; if TRUE, probabilities \( p \) are given as \( \log(p) \).
- `lower.tail` logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
Details

If location or scale are omitted, they assume the default values of 0 and 1 respectively.

The Logistic distribution with location = \( \mu \) and scale = \( \sigma \) has distribution function

\[
F(x) = \frac{1}{1 + e^{-(x-\mu)/\sigma}}
\]

and density

\[
f(x) = \frac{1}{\sigma} \frac{e^{(x-\mu)/\sigma}}{(1 + e^{(x-\mu)/\sigma})^2}
\]

It is a long-tailed distribution with mean \( \mu \) and variance \( \pi^2/3 \sigma^2 \).

Value

dlogis gives the density, plogis gives the distribution function, qlogis gives the quantile function, and rlogis generates random deviates.

Note

qlogis(p) is the same as the well known 'logit' function, \( \text{logit}(p) = \log(p/(1-p)) \), and plogis(x) has consequently been called the "inverse logit".

The distribution function is a rescaled hyperbolic tangent, \( \text{plogis}(x) = (1 + \tanh(x/2))/2 \), and it is called sigmoid function in contexts such as neural networks.

References


Examples

\[
\text{var(rlogis(4000, 0, s = 5))} \quad \text{# approximately (+/- 3)}
\]
\[
\pi^2/3 \times 5^2
\]
Arguments

object any object from which a log-likelihood value, or a contribution to a log-likelihood value, can be extracted.

... some methods for this generic function require additional arguments.

REML an optional logical value. If TRUE the restricted log-likelihood is returned, else, if FALSE, the log-likelihood is returned. Defaults to FALSE.

Details

For a glm fit the family does not have to specify how to calculate the log-likelihood, so this is based on the family’s function to compute the AIC. For gaussian, Gamma and inverse.gaussian families it assumed that the dispersion of the GLM is estimated and has been included in the AIC, and for all other families it is assumed that the dispersion is known.

Note that this procedure is not completely accurate for the gamma and inverse gaussian families, as the estimate of dispersion used is not the MLE.

Value

Returns an object, say r, of class logLik which is a number with attributes, attr(r, "df") (degrees of freedom) giving the number of parameters in the model. There’s a simple print method for logLik objects.

The details depend on the method function used; see the appropriate documentation.

Author(s)

Jose Pinheiro and Douglas Bates

References

For \texttt{logLik.lm}:


See Also

\texttt{logLik.gls}, \texttt{logLik.lme}, in package \texttt{nlme}, etc.

Examples

```r
x <- 1:5
lmx <- lm(x ~ 1)
logLik(lmx) # using print.logLik() method
str(logLik(lmx))

## lm method
(fm1 <- lm(rating ~ ., data = attitude))
logLik(fm1)
logLik(fm1, REML = TRUE)

res <- try(data(Orthodont, package="nlme"))
if(!inherits(res, "try-error")) {
  fml <- lm(distance ~ Sex * age, Orthodont)
  print(logLik(fml))
```
print(logLik(fm1, REML = TRUE))

loglin  Fitting Log-Linear Models

Description

loglin is used to fit log-linear models to multidimensional contingency tables by Iterative Proportional Fitting.

Usage

loglin(table, margin, start = rep(1, length(table)), fit = FALSE, eps = 0.1, iter = 20, param = FALSE, print = TRUE)

Arguments

table  a contingency table to be fit, typically the output from table.

margin  a list of vectors with the marginal totals to be fit. (Hierarchical) log-linear models can be specified in terms of these marginal totals which give the “maximal” factor subsets contained in the model. For example, in a three-factor model, list(c(1, 2), c(1, 3)) specifies a model which contains parameters for the grand mean, each factor, and the 1-2 and 1-3 interactions, respectively (but no 2-3 or 1-2-3 interaction), i.e., a model where factors 2 and 3 are independent conditional on factor 1 (sometimes represented as ’[12][13]’).

The names of factors (i.e., names(dimnames(table))) may be used rather than numeric indices.

start  a starting estimate for the fitted table. This optional argument is important for incomplete tables with structural zeros in table which should be preserved in the fit. In this case, the corresponding entries in start should be zero and the others can be taken as one.

fit  a logical indicating whether the fitted values should be returned.

eps  maximum deviation allowed between observed and fitted margins.

iter  maximum number of iterations.

param  a logical indicating whether the parameter values should be returned.

print  a logical. If TRUE, the number of iterations and the final deviation are printed.

Details

The Iterative Proportional Fitting algorithm as presented in Haberman (1972) is used for fitting the model. At most iter iterations are performed, convergence is taken to occur when the maximum deviation between observed and fitted margins is less than eps. All internal computations are done in double precision; there is no limit on the number of factors (the dimension of the table) in the model.

Assuming that there are no structural zeros, both the Likelihood Ratio Test and Pearson test statistics have an asymptotic chi-squared distribution with df degrees of freedom.

Package MASS contains loglm, a front-end to loglin which allows the log-linear model to be specified and fitted in a formula-based manner similar to that of other fitting functions such as lm or glm.
Lognormal

Value

A list with the following components.

- `lrt` the Likelihood Ratio Test statistic.
- `pearson` the Pearson test statistic (X-squared).
- `df` the degrees of freedom for the fitted model. There is no adjustment for structural zeros.
- `margin` list of the margins that were fit. Basically the same as the input `margin`, but with numbers replaced by names where possible.
- `fit` An array like `table` containing the fitted values. Only returned if `fit` is `TRUE`.
- `param` A list containing the estimated parameters of the model. The “standard” constraints of zero marginal sums (e.g., zero row and column sums for a two factor parameter) are employed. Only returned if `param` is `TRUE`.

Author(s)

Kurt Hornik

References


See Also

table

Examples

```r
## Model of joint independence of sex from hair and eye color.
fm <- loglin(HairEyeColor, list(c(1, 2), c(1, 3), c(2, 3)))
fm
1 - pchisq(fm$lrt, fm$df)
## Model with no three-factor interactions fits well.
```

Lognormal

The Log Normal Distribution

Description

Density, distribution function, quantile function and random generation for the log normal distribution whose logarithm has mean equal to `meanlog` and standard deviation equal to `sdlog`. 
Usage

dlnorm(x, meanlog = 0, sdlog = 1, log = FALSE)
plnorm(q, meanlog = 0, sdlog = 1, lower.tail = TRUE, log.p = FALSE)
qlnorm(p, meanlog = 0, sdlog = 1, lower.tail = TRUE, log.p = FALSE)
rlnorm(n, meanlog = 0, sdlog = 1)

Arguments

x, q  vector of quantiles.
p  vector of probabilities.
n  number of observations. If length(n) > 1, the length is taken to be the number required.
meanlog, sdlog  mean and standard deviation of the distribution on the log scale with default values of 0 and 1 respectively.
log, log.p  logical; if TRUE, probabilities p are given as log(p).
lower.tail  logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

The log normal distribution has density

\[
f(x) = \frac{1}{\sqrt{2\pi\sigma x}} e^{-(\log(x) - \mu)^2 / 2\sigma^2}
\]

where \( \mu \) and \( \sigma \) are the mean and standard deviation of the logarithm. The mean is \( E(X) = \exp(\mu + 1/2\sigma^2) \), and the variance \( Var(X) = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1) \) and hence the coefficient of variation is \( \sqrt{\exp(\sigma^2) - 1} \) which is approximately \( \sigma \) when that is small (e.g., \( \sigma < 1/2 \)).

Value

dlnorm gives the density, plnorm gives the distribution function, qlnorm gives the quantile function, and rlnorm generates random deviates.

Note

The cumulative hazard \( H(t) = -\log(1 - F(t)) \) is \(-plnorm(t, r, lower = FALSE, log = TRUE)\).

References


See Also

dnorm for the normal distribution.

Examples

dlnorm(1) == dnorm(0)
lowess

Scatter Plot Smoothing

Description

This function performs the computations for the LOWESS smoother (see the reference below). lowess returns a list containing components x and y which give the coordinates of the smooth. The smooth should be added to a plot of the original points with the function lines.

Usage

lowess(x, y = NULL, f = 2/3, iter = 3, delta = 0.01 * diff(range(xy$x[o])))

Arguments

x, y vectors giving the coordinates of the points in the scatter plot. Alternatively a single plotting structure can be specified.
f the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness.
iter the number of robustifying iterations which should be performed. Using smaller values of iter will make lowess run faster.
delta values of x which lie within delta of each other are replaced by a single value in the output from lowess. Defaults to 1/100th of the range of x.

References


See Also

loess, a newer formula based version of lowess (with different defaults!).

Examples

plot(cars, main = "lowess(cars)")
lines(lowess(cars), col = 2)
lines(lowess(cars, f=.2), col = 3)
legend(5, 120, c(paste("f = ", c("2/3", ".2"))), lty = 1, col = 2:3)
**ls.diag**

*Compute Diagnostics for ‘lsfit’ Regression Results*

**Description**

Computes basic statistics, including standard errors, t- and p-values for the regression coefficients.

**Usage**

```r
ls.diag(ls.out)
```

**Arguments**

- `ls.out` Typically the result of `lsfit()`

**Value**

A list with the following numeric components.

- `std.dev` The standard deviation of the errors, an estimate of $\sigma$.
- `hat` diagonal entries $h_{ii}$ of the hat matrix $H$
- `std.res` standardized residuals
- `stud.res` studentized residuals
- `cooks` Cook’s distances
- `dfits` DFITS statistics
- `correlation` correlation matrix
- `std.err` standard errors of the regression coefficients
- `cov.scaled` Scaled covariance matrix of the coefficients
- `cov.unscaled` Unscaled covariance matrix of the coefficients

**References**


**See Also**

`hat` for the hat matrix diagonals, `ls.print`, `lm.influence`, `summary.lm`, `anova`.

**Examples**

```r
##-- Using the same data as the lm(.) example:
lsD9 <- lsfit(x = as.numeric(gl(2, 10, 20)), y = weight)
dlsD9 <- ls.diag(lsD9)
str(dlsD9, give.attr=FALSE)
abs(1 - sum(dlsD9$hat) / 2) < 10*.Machine$double.eps # sum(h.ii) = p
plot(dlsD9$hat, dlsD9$stud.res, xlim=c(0,0.11))
abline(h = 0, lty = 2, col = "lightgray")
```
**ls.print**  
*Print ‘lsfit’ Regression Results*

**Description**

Computes basic statistics, including standard errors, t- and p-values for the regression coefficients and prints them if `print.it` is TRUE.

**Usage**

```r
ls.print(ls.out, digits = 4, print.it = TRUE)
```

**Arguments**

- `ls.out` Typically the result of `lsfit()`
- `digits` The number of significant digits used for printing
- `print.it` a logical indicating whether the result should also be printed

**Value**

A list with the components

- `summary` The ANOVA table of the regression
- `coef.table` matrix with regression coefficients, standard errors, t- and p-values

**Note**

Usually, you’d rather use `summary(lm(...))` and `anova(lm(...))` for obtaining similar output.

**See Also**

- `ls.diag`, `lsfit`, also for examples; `lm`, `lm.influence` which usually are preferable.

---

**lsfit**  
*Find the Least Squares Fit*

**Description**

The least squares estimate of $\beta$ in the model

$$Y = X\beta + \epsilon$$

is found.

**Usage**

```r
lsfit(x, y, wt = NULL, intercept = TRUE, tolerance = 1e-07, yname = NULL)
```
**Arguments**

- **x**: a matrix whose rows correspond to cases and whose columns correspond to variables.
- **y**: the responses, possibly a matrix if you want to fit multiple left hand sides.
- **wt**: an optional vector of weights for performing weighted least squares.
- **intercept**: whether or not an intercept term should be used.
- **tolerance**: the tolerance to be used in the matrix decomposition.
- **yname**: names to be used for the response variables.

**Details**

If weights are specified then a weighted least squares is performed with the weight given to the \( j \)th case specified by the \( j \)th entry in \( wt \).

If any observation has a missing value in any field, that observation is removed before the analysis is carried out. This can be quite inefficient if there is a lot of missing data.

The implementation is via a modification of the LINPACK subroutines which allow for multiple left-hand sides.

**Value**

A list with the following named components:

- **coef**: the least squares estimates of the coefficients in the model (\( \beta \) as stated above).
- **residuals**: residuals from the fit.
- **intercept**: indicates whether an intercept was fitted.
- **qr**: the QR decomposition of the design matrix.

**References**


**See Also**


**Examples**

```r
##-- Using the same data as the lm(.) example:
lsD9 <- lsfit(x = unclass(gl(2,10)), y = weight)
ls.print(lsD9)
```
Median Absolute Deviation

Description

Compute the median absolute deviation, i.e., the (lo-/hi-) median of the absolute deviations from the median, and (by default) adjust by a factor for asymptotically normal consistency.

Usage

```
mad(x, center = median(x), constant = 1.4826, na.rm = FALSE, 
    low = FALSE, high = FALSE)
```

Arguments

- `x`: a numeric vector.
- `center`: Optionally, the centre: defaults to the median.
- `constant`: scale factor.
- `na.rm`: if TRUE then NA values are stripped from x before computation takes place.
- `low`: if TRUE, compute the “lo-median”, i.e., for even sample size, do not average the two middle values, but take the smaller one.
- `high`: if TRUE, compute the “hi-median”, i.e., take the larger of the two middle values for even sample size.

Details

The actual value calculated is `constant * cMedian(abs(x - center))` with the default value of `center` being `median(x)`, and `cMedian` being the usual, the “low” or “high” median, see the arguments description for `low` and `high` above.

The default `constant = 1.4826` (approximately `1/Φ−1(3/4) = 1/qnorm(3/4)`) ensures consistency, i.e.,

\[ E[\text{mad}(X_1, \ldots, X_n)] = \sigma \]

for \(X_i\) distributed as \(N(\mu, \sigma^2)\) and large \(n\).

If `na.rm` is TRUE then NA values are stripped from \(x\) before computation takes place. If this is not done then an NA value in \(x\) will cause `mad` to return NA.

See Also

`IQR` which is simpler but less robust, `median`, `var`.

Examples

```
mad(c(1:9))
print(mad(c(1:9), constant=1))
mad(c(1:8,100), constant=1) # = 2 ; TRUE
x <- c(1,2,3, 5,7,8)
sort(abs(x - median(x)))
c(mad(x, co=1), mad(x, co=1, lo = TRUE), mad(x, co=1, hi = TRUE))
```
Description

Returns the Mahalanobis distance of all rows in x and the vector \( \mu = \text{center} \) with respect to \( \Sigma = \text{cov} \). This is (for vector x) defined as

\[
D^2 = (x - \mu)' \Sigma^{-1} (x - \mu)
\]

Usage

\[
\text{mahalanobis}(x, \text{center}, \text{cov}, \text{inverted}=\text{FALSE}, \ldots)
\]

Arguments

- **x**: vector or matrix of data with, say, \( p \) columns.
- **center**: mean vector of the distribution or second data vector of length \( p \).
- **cov**: covariance matrix \((p \times p)\) of the distribution.
- **inverted**: logical. If True, \( \text{cov} \) is supposed to contain the inverse of the covariance matrix.
- **\ldots**: passed to \text{solve} for computing the inverse of the covariance matrix (if inverted is false).

See Also

cov, var

Examples

\[
\begin{array}{l}
\text{ma} \leftarrow \text{cbind}(1:6, 1:3) \\
(S \leftarrow \text{var(ma)}) \\
\text{mahalanobis}(c(0,0), 1:2, S) \\
\end{array}
\]

\[
\begin{array}{l}
x \leftarrow \text{matrix}(\text{rnorm}(100*3), \text{ncol} = 3) \\
\text{stopifnot(mahalanobis(x, 0, diag(ncol(x))) == rowSums(x*x))} \\
\text{## Here, } D^2 = \text{usual Euclidean distances} \\
\text{Sx} \leftarrow \text{cov(x)} \\
\text{D2} \leftarrow \text{mahalanobis(x, colMeans(x), Sx)} \\
\text{plot(density(D2, bw=.5), main="Mahalanobis distances, n=100, p=3")}; \text{rug(D2)} \\
\text{qqplot(qchisq(ppoints(100), df=3), D2,} \\
\text{main = expression("Q-Q plot of Mahalanobis" * ~D^2 *} \\
\text{" vs. quantiles of" * ~ chi(3)^2))} \\
\text{abline(0, 1, col = 'gray')} \\
\end{array}
\]
make.link

Create a Link for GLM families

Description

This function is used with the family functions in glm(). Given a link, it returns a link function, an inverse link function, the derivative $d\mu/d\eta$ and a function for domain checking.

Usage

make.link(link)

Arguments

- link character or numeric; one of "logit", "probit", "cloglog", "identity", "log", "sqrt", "1/mu^2", "inverse", or number, say $\lambda$ resulting in power link $= \mu^\lambda$.

Value

A list with components

- linkfun Link function function(mu)
- linkinv Inverse link function function(eta)
- mu.eta Derivative function(eta) $d\mu/d\eta$
- valideta function(eta)[ TRUE if all of eta is in the domain of linkinv ].

See Also

glm, family.

Examples

str(make.link("logit"))

l2 <- make.link(2)
l2$linkfun(0:3)# 0 1 4 9
l2$mu.eta(eta= 1:2)#= 1/(2*sqrt(eta))

makepredictcall

Utility Function for Safe Prediction

Description

A utility to help model.frame.default create the right matrices when predicting from models with terms like poly or ns.

Usage

makepredictcall(var, call)
manova

Arguments

  var        A variable.
  call       The term in the formula, as a call.

Details

This is a generic function with methods for poly, bs and ns: the default method handles scale. If model.frame.default encounters such a term when creating a model frame, it modifies the predvars attribute of the terms supplied to replace the term with one that will work for predicting new data. For example makepredictcall.ns adds arguments for the knots and intercept.

To make use of this, have your model-fitting function return the terms attribute of the model frame, or copy the predvars attribute of the terms attribute of the model frame to your terms object.

To extend this, make sure the term creates variables with a class, and write a suitable method for that class.

Value

A replacement for call for the predvars attribute of the terms.

See Also

model.frame, poly, scale; bs and ns in package splines, cars

Examples

## using poly: this did not work in R < 1.5.0
fm <- lm(weight ~ poly(height, 2), data = women)
plot(women, xlab = "Height (in)", ylab = "Weight (lb)")
ht <- seq(57, 73, len = 200)
lines(ht, predict(fm, data.frame(height=ht)))

## see also example(cars)

## see bs and ns for spline examples.

---

manova  

*Multivariate Analysis of Variance*

Description

A class for the multivariate analysis of variance.

Usage

  manova(...)

Arguments

...  Arguments to be passed to aov.
Details

Class "manova" differs from class "aov" in selecting a different summary method. Function manova calls aov and then add class "manova" to the result object for each stratum.

Value

See aov and the comments in Details here.

Note

manova does not support multistratum analysis of variance, so the formula should not include an Error term.

References


See Also

aov, summary.manova, the latter containing examples.

mantelhaen.test  Cochran-Mantel-Haenszel Chi-Squared Test for Count Data

Description

Performs a Cochran-Mantel-Haenszel chi-squared test of the null that two nominal variables are conditionally independent in each stratum, assuming that there is no three-way interaction.

Usage

mantelhaen.test(x, y = NULL, z = NULL, alternative = c("two.sided", "less", "greater"), correct = TRUE, exact = FALSE, conf.level = 0.95)

Arguments

x

either a 3-dimensional contingency table in array form where each dimension is at least 2 and the last dimension corresponds to the strata, or a factor object with at least 2 levels.

y

a factor object with at least 2 levels; ignored if x is an array.

z

a factor object with at least 2 levels identifying to which stratum the corresponding elements in x and y belong; ignored if x is an array.

alternative

indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. Only used in the 2 by 2 by K case.

correct

a logical indicating whether to apply continuity correction when computing the test statistic. Only used in the 2 by 2 by K case.
exact  a logical indicating whether the Mantel-Haenszel test or the exact conditional test (given the strata margins) should be computed. Only used in the 2 by 2 by \( K \) case.

conf.level  confidence level for the returned confidence interval. Only used in the 2 by 2 by \( K \) case.

Details

If \( x \) is an array, each dimension must be at least 2, and the entries should be nonnegative integers. NA's are not allowed. Otherwise, \( x, y \) and \( z \) must have the same length. Triples containing NA's are removed. All variables must take at least two different values.

Value

A list with class "htest" containing the following components:

statistic  Only present if no exact test is performed. In the classical case of a 2 by 2 by \( K \) table (i.e., of dichotomous underlying variables), the Mantel-Haenszel chi-squared statistic; otherwise, the generalized Cochran-Mantel-Haenszel statistic.

parameter  the degrees of freedom of the approximate chi-squared distribution of the test statistic (1 in the classical case). Only present if no exact test is performed.

p.value  the p-value of the test.

conf.int  a confidence interval for the common odds ratio. Only present in the 2 by 2 by \( K \) case.

estimate  an estimate of the common odds ratio. If an exact test is performed, the conditional Maximum Likelihood Estimate is given; otherwise, the Mantel-Haenszel estimate. Only present in the 2 by 2 by \( K \) case.

null.value  the common odds ratio under the null of independence, 1. Only present in the 2 by 2 by \( K \) case.

alternative  a character string describing the alternative hypothesis. Only present in the 2 by 2 by \( K \) case.

method  a character string indicating the method employed, and whether or not continuity correction was used.

data.name  a character string giving the names of the data.

Note

The asymptotic distribution is only valid if there is no three-way interaction. In the classical 2 by 2 by \( K \) case, this is equivalent to the conditional odds ratios in each stratum being identical. Currently, no inference on homogeneity of the odds ratios is performed.

See also the example below.

References


Examples

## Agresti (1990), pages 231--237, Penicillin and Rabbits
## Investigation of the effectiveness of immediately injected or 1.5
## hours delayed penicillin in protecting rabbits against a lethal
## injection with beta-hemolytic streptococci.
Rabbits <-
array(c(0, 0, 6, 5,
       3, 0, 3, 6,
       6, 2, 0, 4,
       5, 1, 6, 0,
       2, 5, 0, 0),
dim = c(2, 2, 5),
dimnames = list(
    Delay = c("None", "1.5h"),
    Response = c("Cured", "Died"),
    Penicillin.Level = c("1/8", "1/4", "1/2", "1", "4")))

Rabbits
## Classical Mantel-Haenszel test
mantelhaen.test(Rabbits)
## => p = 0.047, some evidence for higher cure rate of immediate
## injection
## Exact conditional test
mantelhaen.test(Rabbits, exact = TRUE)
## => p = 0.040
## Exact conditional test for one-sided alternative of a higher
## cure rate for immediate injection
mantelhaen.test(Rabbits, exact = TRUE, alternative = "greater")
## => p = 0.020

## UC Berkeley Student Admissions
mantelhaen.test(UCBAdmissions)
## No evidence for association between admission and gender
## when adjusted for department. However,
apply(UCBAdmissions, 3, function(x) (x[1,1]*x[2,2])/(x[1,2]*x[2,1]))
## This suggests that the assumption of homogeneous (conditional)
## odds ratios may be violated. The traditional approach would be
## using the Woolf test for interaction:
woolf <- function(x) {
  x <- x + 1 / 2
  k <- dim(x)[3]
  or <- apply(x, 3, function(x) (x[1,1]*x[2,2])/(x[1,2]*x[2,1]))
  w <- apply(x, 3, function(x) 1 / sum(1 / x))
  1 - pchisq(sum(w * (log(or) - weighted.mean(log(or), w)) ^ 2), k - 1)
}
woolf(UCBAdmissions)
## => p = 0.003, indicating that there is significant heterogeneity.
## (And hence the Mantel-Haenszel test cannot be used.)

## Job Satisfaction example.
Satisfaction <-
as.table(array(c(1, 2, 0, 0, 3, 3, 1, 2,
               11, 17, 8, 4, 2, 3, 5, 2,
               1, 0, 0, 0, 3, 0, 1,
               2, 5, 7, 9, 1, 1, 2, 6),
dim = c(4, 4, 2),
      dimnames = list(X1 = c("Yes", "No"),
                      X2 = c("Unemployed", "Employed"),
                      Gender = c("Female", "Male")))

Satisfaction
## => p = 0.020, and thus some evidence for an association between
## finding a job upon graduation and gender, controlling job
## satisfaction.
## Exact conditional test
mantelhaen.test(Satisfaction, exact = TRUE)
## => p = 0.002
## Exact conditional test for one-sided alternative of a higher
## proportion finding a job upon graduation for females
mantelhaen.test(Satisfaction, exact = TRUE, alternative = "greater")
## => p = 0.001

## UC Berkeley Stat Student Admissions
mantelhaen.test(UCBStatAdmissions)
## => p = 0.003, indicating that there is significant heterogeneity.
## (And hence the Mantel-Haenszel test cannot be used.)
dimnames =
list(Income =
  c("<5000", "5000-15000",
  "15000-25000",">25000"),
  "Job Satisfaction" =
  c("V_D", "L_S", "M_S", "V_S"),
  Gender = c("Female", "Male")))

## (Satisfaction categories abbreviated for convenience.)
ftable(. ~ Gender + Income, Satisfaction)
## Table 7.8 in Agresti (2002), p. 288.
mantelhaen.test(Satisfaction)

---

**Mauchley's test of sphericity**

**Description**
Tests whether a Wishart-distributed covariance matrix (or transformation thereof) is proportional to a given matrix.

**Usage**
```r
## S3 methods for class 'SSD' or 'mlm'
mauchley.test(object, Sigma = diag(nrow = p),
               T = Thin.row(proj(M) - proj(X)), M = diag(nrow = p), X = ~0,
               idata = data.frame(index = seq(length = p)), ...)
```

**Arguments**
- `object`: object of class `SSD` or `mlm`
- `Sigma`: Matrix to be proportional to
- `T`: Transformation matrix. By default computed from `M` and `X`
- `M`: Formula or matrix describing the outer projection (see below)
- `X`: Formula or matrix describing the inner projection (see below)
- `idata`: Data frame describing intra-block design
- `...`: For consistency with generic

**Details**
Mauchley's test test for whether a covariance matrix can be assumed to be proportional to a given matrix.

It is common to transform the observations prior to testing. This typically involves transformation to intra-block differences, but more complicated within-block designs can be encountered, making more elaborate transformations necessary. A transformation matrix `T` can be given directly or specified as the difference between two projections onto the spaces spanned by `M` and `X`, which in turn can be given as matrices or as model formulas with respect to `idata` (the tests will be invariant to parametrization of the quotient space `M/X`).

The common use of this test is in repeated measurements designs, with `X=~1`. This is almost, but not quite the same as testing for compound symmetry in the untransformed covariance matrix.
Value
An object of class "htest"

Note
The p-value differs slightly from that of SAS because a second order term is included in the asymptotic approximation.

References

See Also
SSD, anova.mlm

Examples
example(SSD)  # Brings in the mlmfit and reacttime objects

### traditional test of intrasubj. contrasts
mauchley.test(mlmfit, X=~1)

### tests using intra-subject 3x2 design
idata <- data.frame(deg=gl(3,1,6, labels=c(0,4,8)),
noise=gl(2,3,6, labels=c("A","P")))
mauchley.test(mlmfit, X = ~ deg + noise, idata = idata)
mauchley.test(mlmfit, M = ~ deg + noise, X = ~ noise, idata=idata)

mcnemar.test  
McNemar's Chi-squared Test for Count Data

Description
Performs McNemar's chi-squared test for symmetry of rows and columns in a two-dimensional contingency table.

Usage
mcnemar.test(x, y = NULL, correct = TRUE)

Arguments
x either a two-dimensional contingency table in matrix form, or a factor object.
y a factor object; ignored if x is a matrix.
correct a logical indicating whether to apply continuity correction when computing the test statistic.
Details

The null is that the probabilities of being classified into cells \([i, j]\) and \([j, i]\) are the same. If \(x\) is a matrix, it is taken as a two-dimensional contingency table, and hence its entries should be nonnegative integers. Otherwise, both \(x\) and \(y\) must be vectors of the same length. Incomplete cases are removed, the vectors are coerced into factor objects, and the contingency table is computed from these.

Continuity correction is only used in the 2-by-2 case if \(\text{correct}\) is \(\text{TRUE}\).

Value

A list with class "htest" containing the following components:

- **statistic**: the value of McNemar's statistic.
- **parameter**: the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- **p.value**: the p-value of the test.
- **method**: a character string indicating the type of test performed, and whether continuity correction was used.
- **data.name**: a character string giving the name(s) of the data.

References


Examples

```r
## Presidential Approval Ratings.
## Approval of the President's performance in office in two surveys,
## one month apart, for a random sample of 1600 voting-age Americans.
Performance <- matrix(c(794, 86, 150, 570),
                      nr = 2,
                      dimnames = list("1st Survey" = c("Approve", "Disapprove"),
                                      "2nd Survey" = c("Approve", "Disapprove")))
Performance
mcnemar.test(Performance)
## => very strong association between the two successive ratings
```

**median**

**Median Value**

Description

Compute the sample median of the vector of values given as its argument.

Usage

```r
median(x, na.rm = FALSE)
```
Arguments

\texttt{x} \quad a \text{ numeric vector containing the values whose median is to be computed.}
\texttt{na.rm} \quad a \text{ logical value indicating whether NA values should be stripped before the computation proceeds.}

References


See Also

\texttt{quantile} for general quantiles.

Examples

\begin{verbatim}
median(1:4) # = 2.5 [even number]
median(c(1:3,100,1000)) # = 3 [odd, robust]
\end{verbatim}

\texttt{medpolish Median Polish of a Matrix}

Description

Fits an additive model using Tukey’s \textit{median polish} procedure.

Usage

\begin{verbatim}
medpolish(x, eps = 0.01, maxiter = 10, trace.iter = TRUE, na.rm = FALSE)
\end{verbatim}

Arguments

\texttt{x} \quad a \text{ numeric matrix.}
\texttt{eps} \quad \text{real number greater than 0. A tolerance for convergence: see Details.}
\texttt{maxiter} \quad \text{the maximum number of iterations}
\texttt{trace.iter} \quad \text{logical. Should progress in convergence be reported?}
\texttt{na.rm} \quad \text{logical. Should missing values be removed?}

Details

The model fitted is additive (constant + rows + columns). The algorithm works by alternately removing the row and column medians, and continues until the proportional reduction in the sum of absolute residuals is less than \texttt{eps} or until there have been \texttt{maxiter} iterations. The sum of absolute residuals is printed at each iteration of the fitting process, if \texttt{trace.iter} is \texttt{TRUE}. If \texttt{na.rm} is \texttt{FALSE} the presence of any NA value in \texttt{x} will cause an error, otherwise NA values are ignored.

\texttt{medpolish} returns an object of class \texttt{medpolish} (see below). There are printing and plotting methods for this class, which are invoked via the generics \texttt{print} and \texttt{plot}.
Value

An object of class medpolish with the following named components:

- **overall**: the fitted constant term.
- **row**: the fitted row effects.
- **col**: the fitted column effects.
- **residuals**: the residuals.
- **name**: the name of the dataset.

References


See Also

`median`, `aov` for a mean instead of median decomposition.

Examples

```r
## Deaths from sport parachuting; from ABC of EDA, p.224:

deaths <-
    rbind(c(14,15,14),
          c( 7, 4, 7),
          c( 8, 2,10),
          c(15, 9,10),
          c( 0, 2, 0))

dimnames(deaths) <- list(c("1-24", "25-74", "75-199", "200++", "NA"),
                         paste(1973:1975))

deaths

(med.d <- medpolish(deaths))
plot(med.d)

## Check decomposition:

all(deaths == med.d$overall + outer(med.d$row,med.d$col, "+") + med.d$resid)
```

---

**model.extract**

*Extract Components from a Model Frame*

**Description**

Returns the response, offset, subset, weights or other special components of a model frame passed as optional arguments to `model.frame`.

**Usage**

```r
model.extract(frame, component)  
model.offset(x)  
model.response(data, type = "any")  
model.weights(x)
```
model.frame

Arguments

frame, x, data
A model frame.

component
literal character string or name. The name of a component to extract, such as
"weights", "subset".

type
One of "any", "numeric", "double". Using the either of latter two co-
erces the result to have storage mode "double".

Details

model.extract is provided for compatibility with S, which does not have the more specific
functions. It is also useful to extract e.g. the etastart and mustart components of a glm fit.
model.offset and model.response are equivalent to model.frame(, "offset")
and model.frame(, "response") respectively.
model.weights is slightly different from model.frame(, "weights") in not naming the
vector it returns.

Value
The specified component of the model frame, usually a vector.

See Also

model.frame, offset

Examples

a <- model.frame(cbind(ncases, ncontrols) ~ agegp + tobgp + alcgp, data=esoph)
model.extract(a, "response")
stopifnot(model.extract(a, "response") == model.response(a))

a <- model.frame(ncases/(ncases+ncontrols) ~ agegp + tobgp + alcgp,
data = esoph, weights = ncases+ncontrols)
model.response(a)
model.extract(a, "weights")

a <- model.frame(cbind(ncases, ncontrols) ~ agegp,
something = tobgp, data = esoph)
names(a)
stopifnot(model.extract(a, "something") == esoph$tobgp)

model.frame

Extracting the “Environment” of a Model Formula

Description

model.frame (a generic function) and its methods return a data.frame with the variables
needed to use formula and any ... arguments.
Usage

```r
model.frame(formula, ...)  
```

## Default S3 method:
```r
model.frame(formula, data = NULL,  
            subset = NULL, na.action = na.fail,  
            drop.unused.levels = FALSE, xlev = NULL, ...)
```

## S3 method for class 'aovlist':
```r
model.frame(formula, data = NULL, ...)
```

## S3 method for class 'glm':
```r
model.frame(formula, ...)
```

## S3 method for class 'lm':
```r
model.frame(formula, ...)
```

Arguments

- `formula`: a model `formula` or `terms` object or an R object.
- `data`: data.frame, list, environment or object coercible to data.frame containing the variables in `formula`. Neither a matrix nor an array will be accepted.
- `subset`: a specification of the rows to be used: defaults to all rows. This can be any valid indexing vector (see `[.data.frame`) for the rows of data or if that is not supplied, a data frame made up of the variables used in `formula`.
- `na.action`: how NAs are treated. The default is first, any `na.action` attribute of `data`, second a `na.action` setting of `options`, and third `na.fail` if that is unset. The "factory-fresh" default is `na.omit`. Another possible value is `NULL`.
- `drop.unused.levels`: should factors have unused levels dropped? Defaults to `FALSE`.
- `xlev`: a named list of character vectors giving the full set of levels to be assumed for each factor.
- `...`: further arguments such as `data`, `na.action`, `subset`. Any additional arguments such as `offset` and `weights` which reach the default method are used to create further columns in the model frame, with parenthesised names such as "(offset)".

Details

Exactly what happens depends on the class and attributes of the object `formula`. If this is an object of fitted-model class such as "lm", the method will either returned the saved model frame used when fitting the model (if any, often selected by argument `model = TRUE`) or pass the call used when fitting on to the default method. The default method itself can cope with rather standard model objects such as those of classes "lqs" and "ppr" from package MASS if no other arguments are supplied.

The rest of this section applies only to the default method.

If either `formula` or `data` is already a model frame (a data frame with a "terms" attribute and the other is missing, the model frame is returned. Unless `formula` is a terms object, `terms` is
called on it. (If you wish to use the keep.order argument of terms.formula, pass a terms object rather than a formula.)

Row names for the model frame are taken from the data argument if present, then from the names of the response in the formula (or rownames if it is a matrix), if there is one.

All the variables in formula, subset and in ... are looked for first in data and then in the environment of formula (see the help for formula() for further details) and collected into a data frame. Then the subset expression is evaluated, and it is is used as a row index to the data frame. Then the na.action function is applied to the data frame (and may well add attributes). The levels of any factors in the data frame are adjusted according to the drop.unused.levels and xlev arguments.

Unless na.action = NULL, time-series attributes will be removed from the variables found (since they will be wrong if NAs are removed).

Value

A data.frame containing the variables used in formula plus those specified ....

References


See Also

model.matrix for the “design matrix”, formula for formulas and expand.model.frame for model.frame manipulation.

Examples

data.class(model.frame(dist ~ speed, data = cars))
**model.matrix**

**Arguments**

`object`  
an object of an appropriate class. For the default method, a model formula or terms object.

`data`  
a data frame created with `model.frame`. If another sort of object, `model.frame` is called first.

`contrasts.arg`  
A list, whose entries are contrasts suitable for input to the `contrasts` replacement function and whose names are the names of columns of `data` containing factors.

`xlev`  
to be used as argument of `model.frame` if `data` has no "terms" attribute.

`...`  
further arguments passed to or from other methods.

**Details**

`model.matrix` creates a design matrix from the description given in `terms(object)`, using the data in `data` which must contain variables with the same names as would be created by a call to `model.frame(object)` or, more precisely, by evaluating `attr(terms(object), "variables")`. If it is a data frame, there may be other columns and the order of columns is not important.

If `contrasts.arg` is specified for a factor it overrides the default factor coding for that variable and any "contrasts" attribute set by `C` or `contrasts`.

In an interaction term, the variable whose levels vary fastest is the first one to appear in the formula (and not in the term), so in `~ a + b + b:a` the interaction will have a varying fastest.

By convention, if the response variable also appears on the right-hand side of the formula it is dropped (with a warning), although interactions involving the term are retained.

**Value**

The design matrix for a regression model with the specified formula and data.

There is an attribute "assign", an integer vector with an entry for each column in the matrix giving the term in the formula which gave rise to the column.

If there are any factors in terms in the model, there is an attribute "contrasts", a named list with an entry for each factor. This specifies the contrasts that would be used in terms in which the factor is coded by contrasts (in some terms dummy coding may be used), either as a character vector naming a function or as a numeric matrix.

**References**


**See Also**

`model.frame, model.extract, terms`

**Examples**

```r
ff <- log(Volume) ~ log(Height) + log(Girth)
str(m <- model.frame(ff, trees))
mat <- model.matrix(ff, m)
```
dd <- data.frame(a = gl(3,4), b = gl(4,1,12))  # balanced 2-way
options("contrasts")
model.matrix(~ a + b, dd)
model.matrix(~ a + b, dd, contrasts = list(a="contr.sum"))
model.matrix(~ a + b, dd, contrasts = list(a="contr.sum", b="contr.poly"))
m.orth <- model.matrix(~a+b, dd, contrasts = list(a="contr.helmert"))
crossprod(m.orth)  # m.orth is ALMOST orthogonal

---

model.tables  
*Compute Tables of Results from an Aov Model Fit*

**Description**

Computes summary tables for model fits, especially complex `aov` fits.

**Usage**

model.tables(x, ...)

```r
## S3 method for class 'aov':
model.tables(x, type = "effects", se = FALSE, cterms, ...)

## S3 method for class 'aovlist':
model.tables(x, type = "effects", se = FALSE, ...)
```

**Arguments**

- `x` a model object, usually produced by `aov`
- `type` type of table: currently only "effects" and "means" are implemented.
- `se` should standard errors be computed?
- `cterms` A character vector giving the names of the terms for which tables should be computed. The default is all tables.
- `...` further arguments passed to or from other methods.

**Details**

For `type = "effects"` give tables of the coefficients for each term, optionally with standard errors.

For `type = "means"` give tables of the mean response for each combinations of levels of the factors in a term.

**Value**

An object of class "tables.aov", as list which may contain components

- `tables` A list of tables for each requested term.
- `n` The replication information for each term.
- `se` Standard error information.
**Warning**

The implementation is incomplete, and only the simpler cases have been tested thoroughly. Weighted `aov` fits are not supported.

**See Also**

`aov`, `proj`, `replications`, `TukeyHSD`, `se.contrast`

**Examples**

```r
N <- c(0,1,0,1,1,1,0,0,1,1,0,1,1,0,1,1,0,0,1,1,0,1,1,0,0)
P <- c(1,1,0,0,0,1,1,0,0,1,1,1,0,0,1,1,0,1,1,1,0,1,1,0,1)
K <- c(1,0,0,1,0,1,1,0,0,1,1,0,0,1,0,1,0,0,0,1,1,0,1,0,0)
yield <- c(49.5, 62.8, 46.8, 57.0, 59.8, 58.5, 55.5, 56.0, 62.8, 55.8, 69.5,
           55.0, 62.0, 48.8, 45.5, 44.2, 52.0, 51.5, 49.8, 48.8, 57.2, 59.0, 53.2, 56.0)

npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P),
                   K=factor(K), yield=yield)
options(contrasts=c("contr.helmert", "contr.treatment"))
npk.aov <- aov(yield ~ block + N*P*K, npk)
model.tables(npk.aov, "means", se = TRUE)

# as a test, not particularly sensible statistically
npk.aovE <- aov(yield ~ N*P*K + Error(block), npk)
model.tables(npk.aovE, se=TRUE)
model.tables(npk.aovE, "means")
```

---

### monthplot

**Plot a Seasonal or other Subseries from a Time Series**

These functions plot seasonal (or other) subseries of a time series. For each season (or other category), a time series is plotted.

**Usage**

```r
monthplot(x, 

## S3 method for class 'stl':
monthplot(x, labels = NULL, ylab = choice, choice = "seasonal", 

## S3 method for class 'StructTS':
monthplot(x, labels = NULL, ylab = choice, choice = "sea", 

## S3 method for class 'ts':
monthplot(x, labels = NULL, times = time(x), phase = cycle(x),
          ylab = deparse(substitute(x)), 

## Default S3 method:
monthplot(x, labels = 1:12,
```
ylab = deparse(substitute(x)),
times = 1:length(x),
phase = (times - 1)%%length(labels) + 1, base = mean,
axes = TRUE, type = c("l", "h"), box = TRUE, add = FALSE,
...)

Arguments

x
Time series or related object.
labels
Labels to use for each “season”.
ylab
y label.
times
Time of each observation.
phase
Indicator for each “season”.
base
Function to use for reference line for subseries.
choice
Which series of an stl or StructTS object?
...
Arguments to be passed to the default method or graphical parameters.
axes
Should axes be drawn (ignored if add=TRUE)?
type
Type of plot. The default is to join the points with lines, and "h" is for histogram-like vertical lines.
box
Should a box be drawn (ignored if add=TRUE)?
add
Should thus just add on an existing plot.

Details

These functions extract subseries from a time series and plot them all in one frame. The ts, stl, and StructTS methods use the internally recorded frequency and start and finish times to set the scale and the seasons. The default method assumes observations come in groups of 12 (though this can be changed).

If the labels are not given but the phase is given, then the labels default to the unique values of the phase. If both are given, then the phase values are assumed to be indices into the labels array, i.e., they should be in the range from 1 to length(labels).

Value

These functions are executed for their side effect of drawing a seasonal subseries plot on the current graphical window.

Author(s)

Duncan Murdoch

References


See Also

ts, stl, StructTS
mood.test

Examples

### The CO2 data

```r
fit <- stl(log(co2), s.window = 20, t.window = 20)
plot(fit)
```  
```r
op <- par(mfrow = c(2,2))
monthplot(co2, ylab = "data", cex.axis = 0.8)
monthplot(fit, choice = "seasonal", cex.axis = 0.8)
monthplot(fit, choice = "trend", cex.axis = 0.8)
monthplot(fit, choice = "remainder", type = "h", cex.axis = 0.8)
par(op)
```  
```r
## The CO2 data, grouped quarterly
quarter <- (cycle(co2) - 1) %% 3
monthplot(co2, phase = quarter)
```  
```r
## see also JohnsonJohnson
```  
---

### mood.test

**Mood Two-Sample Test of Scale**

#### Description

Performs Mood’s two-sample test for a difference in scale parameters.

#### Usage

```r
mood.test(x, ...)
```  
```r
## Default S3 method:
mood.test(x, y, alternative = c("two.sided", "less", "greater"), ...)
```  
```r
## S3 method for class 'formula':
mood.test(formula, data, subset, na.action, ...)
```  
#### Arguments

- **x, y**: numeric vectors of data values.
- **alternative**: indicates the alternative hypothesis and must be one of "two.sided" (default), "greater" or "less" all of which can be abbreviated.
- **formula**: a formula of the form `lhs ~ rhs` where `lhs` is a numeric variable giving the data values and `rhs` a factor with two levels giving the corresponding groups.
- **data**: an optional data frame containing the variables in the model formula.
- **subset**: an optional vector specifying a subset of observations to be used.
- **na.action**: a function which indicates what should happen when the data contain NAs. Defaults to `getOption("na.action")`.
- **...**: further arguments to be passed to or from methods.
Details

The underlying model is that the two samples are drawn from \( f(x - l) \) and \( f((x - l)/s)/s \), respectively, where \( l \) is a common location parameter and \( s \) is a scale parameter.

The null hypothesis is \( s = 1 \).

There are more useful tests for this problem.

Value

A list with class "htest" containing the following components:

- `statistic`: the value of the test statistic.
- `p.value`: the p-value of the test.
- `alternative`: a character string describing the alternative hypothesis.
- `method`: the character string "Mood two-sample test of scale".
- `data.name`: a character string giving the names of the data.

References


See Also

- `fligner.test` for a rank-based (nonparametric) k-sample test for homogeneity of variances;
- `ansari.test` for another rank-based two-sample test for a difference in scale parameters;
- `var.test` and `bartlett.test` for parametric tests for the homogeneity in variance.

Examples

```r
## Same data as for the Ansari-Bradley test:
## Serum iron determination using Hyland control sera
ramsay <- c(111, 107, 100, 99, 102, 106, 109, 108, 104, 99,
            101, 96, 97, 102, 107, 113, 116, 113, 110, 98)
jung.parekh <- c(107, 108, 106, 98, 105, 103, 110, 105, 104,
mood.test(ramsay, jung.parekh)
## Compare this to ansari.test(ramsay, jung.parekh)
```

Multinomial

The Multinomial Distribution

Description

Generate multinomially distributed random number vectors and compute multinomial “density” probabilities.

Usage

```r
rmultinom(n, size, prob)
dmultinom(x, size = NULL, prob, log = FALSE)
```
**Multinomial**

**Arguments**

- `x` vector of length `K` of integers in `0:size`.
- `n` number of random vectors to draw.
- `size` integer, say `N`, specifying the total number of objects that are put into `K` boxes in the typical multinomial experiment. For `dmultinom`, it defaults to `sum(x)`.
- `prob` numeric non-negative vector of length `K`, specifying the probability for the `K` classes; is internally normalized to sum 1.
- `log` logical; if TRUE, log probabilities are computed.

**Details**

If `x` is a `K`-component vector, `dmultinom(x, prob)` is the probability

\[ P(X_1 = x_1, \ldots, X_K = x_K) = C \times \prod_{j=1}^{K} \pi_j^{x_j} \]

where `C` is the “multinomial coefficient” \( C = N!/\prod_{j=1}^{K} x_j! \) and \( N = \sum_{j=1}^{K} x_j \).

By definition, each component \( X_j \) is binomially distributed as `Bin(size, prob[j])` for \( j = 1, \ldots, K \).

The `rmultinom()` algorithm draws binomials from `Bin(n_j, P_j)` sequentially, where \( n_1 = N \) (\( N := size \)), \( P_1 = \pi_1 \) (\( \pi \) is prob scaled to sum 1), and for \( j \geq 2 \), recursively \( n_j = N - \sum_{k=1}^{j-1} n_k \) and \( P_j = \pi_j / (1 - \sum_{k=1}^{j-1} \pi_k) \).

**Value**

For `rmultinom()`, an integer \( K \times n \) matrix where each column is a random vector generated according to the desired multinomial law, and hence summing to `size`. Whereas the transposed result would seem more natural at first, the returned matrix is more efficient because of column-wise storage.

**Note**

`dmultinom` is currently not vectorized at all and has no C interface (API); this may be amended in the future.

**See Also**

`rbinom` which is a special case conceptually.

**Examples**

```r
rmultinom(10, size = 12, prob=c(0.1,0.2,0.8))

pr <- c(1,3,6,10) # normalization not necessary for generation
rmultinom(10, 20, prob = pr)

## all possible outcomes of Multinom(N = 3, K = 3)
X <- t(as.matrix(expand.grid(0:3, 0:3))); X <- X[, colSums(X) <= 3]
X <- rbind(X, 3:3 - colSums(X)); dimnames(X) <- list(letters[1:3], NULL)
X
round(apply(X, 2, function(x) dmultinom(x, prob = c(1,2,5))), 3)
```
na.action  NA Action

Description

na.action is a generic function, and na.action.default its default method.

Usage

na.action(object, ...)

Arguments

object any object whose NA action is given.
... further arguments special methods could require.

Value

The “NA action” which should be applied to object whenever NAs are not desired.

References


See Also

options("na.action").na.omit.na.fail, also for na.exclude, na.pass.

Examples

na.action(c(1, NA))

na.contiguous  Find Longest Contiguous Stretch of non-NA

Description

Find the longest consecutive stretch of non-missing values in a time series object. (In the event of a
tie, the first such stretch.)

Usage

na.contiguous(object, ...)

Arguments

object a univariate or multivariate time series.
... further arguments passed to or from other methods.
Value

A time series without missing values. The class of object will be preserved.

See Also

na.omit and na.omit.ts; na.fail

Examples

na.contiguous(presidents)

Description

These generic functions are useful for dealing with NAs in e.g., data frames. na.fail returns the object if it does not contain any missing values, and signals an error otherwise. na.omit returns the object with incomplete cases removed. na.pass returns the object unchanged.

Usage

na.fail(object, ...)
na.omit(object, ...)
na.exclude(object, ...)
na.pass(object, ...)

Arguments

object an R object, typically a data frame
...

further arguments special methods could require.

Details

At present these will handle vectors, matrices and data frames comprising vectors and matrices (only).

If na.omit removes cases, the row numbers of the cases form the "na.action" attribute of the result, of class "omit".

na.exclude differs from na.omit only in the class of the "na.action" attribute of the result, which is "exclude". This gives different behaviour in functions making use of naresid and napredict: when na.exclude is used the residuals and predictions are padded to the correct length by inserting NAs for cases omitted by na.exclude.

References


See Also

na.action; options with argument na.action for setting “NA actions”; and lm and glm for functions using these. na.contiguous as alternative for time series.
Examples

DF <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA))
na.omit(DF)
na.omit(m)
stopifnot(all(na.omit(1:3) == 1:3))  # does not affect objects with no NA's
try(na.fail(DF)) #> Error: missing values in ...

options("na.action")

---

naprint  
Adjust for Missing Values

Description

Use missing value information to report the effects of an na.action.

Usage

naprint(x, ...)

Arguments

x  
An object produced by an na.action function.

...  
further arguments passed to or from other methods.

Details

This is a generic function, and the exact information differs by method. naprint.omit reports the number of rows omitted: naprint.default reports an empty string.

Value

A character string providing information on missing values, for example the number.

---

naresid  
Adjust for Missing Values

Description

Use missing value information to adjust residuals and predictions.

Usage

naresid(omit, x, ...)
napredict(omit, x, ...)
Arguments

omit an object produced by an `na.action` function, typically the "na.action" attribute of the result of `na.omit` or `na.exclude`.

x a vector, data frame, or matrix to be adjusted based upon the missing value information.

... further arguments passed to or from other methods.

Details

These are utility functions used to allow `predict` and `resid` methods for modelling functions to compensate for the removal of NAs in the fitting process. They are used by the default, "lm" and "glm" methods, and by further methods in packages MASS, rpart and survival.

The default methods do nothing. The default method for the `na.exclude` action is to pad the object with NAs in the correct positions to have the same number of rows as the original data frame.

Currently `naresid` and `napredict` are identical, but future methods need not be. `naresid` is used for residuals, and `napredict` for fitted values and predictions.

Value

These return a similar object to x.

Note

Packages rpart and survival5 used to contain versions of these functions that had an `na.omit` action equivalent to that now used for `na.exclude`.

NegBinomial

The Negative Binomial Distribution

Description

Density, distribution function, quantile function and random generation for the negative binomial distribution with parameters size and prob.

Usage

```r
dnbinom(x, size, prob, mu, log = FALSE)
pnbinom(q, size, prob, mu, lower.tail = TRUE, log.p = FALSE)
qnbinom(p, size, prob, mu, lower.tail = TRUE, log.p = FALSE)
rnbinom(n, size, prob, mu)
```

Arguments

- x vector of (non-negative integer) quantiles.
- q vector of quantiles.
- p vector of probabilities.
- n number of observations. If `length(n) > 1`, the length is taken to be the number required.
NegBinomial

size  target for number of successful trials, or dispersion parameter (the shape parameter of the gamma mixing distribution).
prob  probability of success in each trial.
mu    alternative parametrization via mean: see Details
log, log.p  logical; if TRUE, probabilities p are given as log(p).
lower.tail  logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

The negative binomial distribution with \( \text{size} = n \) and \( \text{prob} = p \) has density

\[
p(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!} p^n (1-p)^x
\]

for \( x = 0, 1, 2, \ldots \).

This represents the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached.

A negative binomial distribution can arise as a mixture of Poisson distributions with mean distributed as a \( \text{Gamma}(\text{size}) \) distribution with scale parameter \( (1 - \text{prob})/\text{prob} \) and shape parameter \( \text{size} \). (This definition allows non-integer values of \( \text{size} \).) In this model \( \text{prob} = \text{scale}/(1+\text{scale}) \), and the mean is \( \text{size} \times (1 - \text{prob})/\text{prob} \).

The alternative parametrization (often used in ecology) is by the mean \( \mu \), and \( \text{size} \), the dispersion parameter, where \( \text{prob} = \text{size}/(\text{size}+\mu) \). The variance is \( \mu + \mu^2/\text{size} \) in this parametrization or \( n(1-p)/p^2 \) in the first one.

If an element of \( x \) is not integer, the result of \( \text{dnbinom} \) is zero, with a warning.

The quantile is defined as the smallest value \( x \) such that \( F(x) \geq p \), where \( F \) is the distribution function.

Value

\( \text{dnbinom} \) gives the density, \( \text{pnbinom} \) gives the distribution function, \( \text{qbinom} \) gives the quantile function, and \( \text{rbinom} \) generates random deviates.

See Also

\( \text{dbinom} \) for the binomial, \( \text{dpois} \) for the Poisson and \( \text{dgeom} \) for the geometric distribution, which is a special case of the negative binomial.

Examples

\[
x <- 0:11
\]
\[
\text{dnbinom}(x, \text{size} = 1, \text{prob} = 1/2) * 2^(1 + x) \# == 1
\]
\[
126 / \text{dnbinom}(0:8, \text{size} = 2, \text{prob} = 1/2) \#- theoretically integer
\]
\[
## Cumulative ('p') = Sum of discrete prob.s ('d'); Relative error :
summary(1 - cumsum(dnbinom(x, size = 2, prob = 1/2)) / pnbinom(x, size = 2, prob = 1/2))
\]
\[
x <- 0:15
\]
\[
\text{size} <- (1:20)/4
\]
\[
\text{persp}(x,\text{size}, \text{dnb} <- \text{outer}(x,\text{size},\text{function}(x,s)\text{dnbinom}(x,s, pr= 0.4)), \\
\text{xlab} = "x", \text{ylab} = "s", \text{zlab}="density", \text{theta} = 150)
\]
title(tit <- "negative binomial density(x,s, pr = 0.4) vs. x & s")

image(x, size, log10(dnb), main= paste("log [",tit,"]"))
contour(x, size, log10(dnb), add=TRUE)

## Alternative parametrization
x1 <- rbinom(500, mu = 4, size = 1)
x2 <- rbinom(500, mu = 4, size = 10)
x3 <- rbinom(500, mu = 4, size = 100)
h1 <- hist(x1, breaks = 20, plot = FALSE)
h2 <- hist(x2, breaks = h1$breaks, plot = FALSE)
h3 <- hist(x3, breaks = h1$breaks, plot = FALSE)
barplot(rbind(h1$counts, h2$counts, h3$counts),
       beside = TRUE, col = c("red","blue","cyan"),
       names.arg = round(h1$breaks[-length(h1$breaks)]))

---

nextn

### Highly Composite Numbers

**Description**

`nextn` returns the smallest integer, greater than or equal to `n`, which can be obtained as a product of powers of the values contained in `factors`. `nextn` is intended to be used to find a suitable length to zero-pad the argument of `fft` so that the transform is computed quickly. The default value for `factors` ensures this.

**Usage**

`nextn(n, factors = c(2, 3, 5))`

**Arguments**

- `n` an integer.
- `factors` a vector of positive integer factors.

**See Also**

`convolve`, `fft`.

**Examples**

```r
nextn(1001) # 1024
table(sapply(599:630, nextn))
```
Description

This function carries out a minimization of the function $f$ using a Newton-type algorithm. See the references for details.

Usage

```r
nlm(f, p, hessian = FALSE, typsize=rep(1, length(p)), fscale=1,
    print.level = 0, ndigit=12, gradtol = 1e-6,
    stepmax = max(1000 * sqrt(sum((p/typsize)^2)), 1000),
    steptol = 1e-6, iterlim = 100, check.analyticals = TRUE, ...)
```

Arguments

- **f** the function to be minimized. If the function value has an attribute called `gradient` or both `gradient` and `hessian` attributes, these will be used in the calculation of updated parameter values. Otherwise, numerical derivatives are used. `deriv` returns a function with suitable `gradient` attribute. This should be a function of a vector of the length of `p` followed by any other arguments specified by the `...` argument.
- **p** starting parameter values for the minimization.
- **hessian** if `TRUE`, the hessian of $f$ at the minimum is returned.
- **typsize** an estimate of the size of each parameter at the minimum.
- **fscale** an estimate of the size of $f$ at the minimum.
- **print.level** this argument determines the level of printing which is done during the minimization process. The default value of 0 means that no printing occurs, a value of 1 means that initial and final details are printed and a value of 2 means that full tracing information is printed.
- **ndigit** the number of significant digits in the function $f$.
- **gradtol** a positive scalar giving the tolerance at which the scaled gradient is considered close enough to zero to terminate the algorithm. The scaled gradient is a measure of the relative change in $f$ in each direction $p[i]$ divided by the relative change in $p[i]$.
- **stepmax** a positive scalar which gives the maximum allowable scaled step length. `stepmax` is used to prevent steps which would cause the optimization function to overflow, to prevent the algorithm from leaving the area of interest in parameter space, or to detect divergence in the algorithm. `stepmax` would be chosen small enough to prevent the first two of these occurrences, but should be larger than any anticipated reasonable step.
- **steptol** a positive scalar providing the minimum allowable relative step length.
- **iterlim** a positive integer specifying the maximum number of iterations to be performed before the program is terminated.
- **check.analyticals** a logical scalar specifying whether the analytic gradients and Hessians, if they are supplied, should be checked against numerical derivatives at the initial parameter values. This can help detect incorrectly formulated gradients or Hessians.
- **...** additional arguments to `f`. 
Details

If a gradient or hessian is supplied but evaluates to the wrong mode or length, it will be ignored if `check.analyticals = TRUE` (the default) with a warning. The hessian is not even checked unless the gradient is present and passes the sanity checks.

From the three methods available in the original source, we always use method “1” which is line search.

Value

A list containing the following components:

- **minimum**: the value of the estimated minimum of \( f \).
- **estimate**: the point at which the minimum value of \( f \) is obtained.
- **gradient**: the gradient at the estimated minimum of \( f \).
- **hessian**: the hessian at the estimated minimum of \( f \) (if requested).
- **code**: an integer indicating why the optimization process terminated.
  - 1: relative gradient is close to zero, current iterate is probably solution.
  - 2: successive iterates within tolerance, current iterate is probably solution.
  - 3: last global step failed to locate a point lower than `estimate`. Either `estimate` is an approximate local minimum of the function or `steptol` is too small.
  - 4: iteration limit exceeded.
  - 5: maximum step size `stepmax` exceeded five consecutive times. Either the function is unbounded below, becomes asymptotic to a finite value from above in some direction or `steptol` is too small.
- **iterations**: the number of iterations performed.

References


See Also

- `optim`, `optimize` for one-dimensional minimization and `unroot` for root finding. `deriv` to calculate analytical derivatives.
- For nonlinear regression, `nls` may be better.

Examples

```r
f <- function(x) sum((x-1:length(x))^2)
nlm(f, c(10,10))
nlm(f, c(10,10), print.level = 2)
str(nlm(f, c(5), hessian = TRUE))

f <- function(x, a) sum((x-a)^2)
nlm(f, c(10,10), a=c(3,5))
f <- function(x, a)
```
nlminb

Optimization using PORT routines

Description

Unconstrained and constrained optimization using PORT routines.

Usage

nlminb(start, objective, gradient = NULL, hessian = NULL,
      scale = 1, control = list(), lower = -Inf, upper = Inf, ...)

Arguments

start numeric vector, initial values for the parameters to be optimized
objective function to be minimized. Must return a scalar value (possibly NA/Inf). The first
argument to objective is the vector of parameters to be optimized, whose
initial values are supplied through start. Further arguments (fixed during the
course of the optimization) to objective may be specified as well (see ...).
gradient optional function that takes the same arguments as objective and evaluates
the gradient of objective at its first argument. Must return a vector as long
as start.
hessian optional function that takes the same arguments as objective and evaluates
the hessian of objective at its first argument. Must return a square matrix of
order length(start). Only the lower triangle is used.
scale See PORT documentation (or leave alone)
control a list of control parameters. See below for details.
lower, upper vector of lower and upper bounds, replicated to be as long as start. If unspec-
ified, all parameters are assumed to be unconstrained.
... further arguments to be supplied to objective

Value

A list with components:

par The best set of parameters found.
objective The value of objective corresponding to par.
convergence An integer code. 0 indicates successful convergence.
message A character string giving any additional information returned by the optimizer,
or NULL. For details, see PORT documentation.
iterations Number of iterations performed.
evaluations Number of objective function and gradient function evaluations

---

{ res <- sum((x-a)^2)
  attr(res, "gradient") <- 2*(x-a)
  res
}

nlm(f, c(10,10), a=c(3,5))

## more examples, including the use of derivatives.
## Not run: demo(nlm)
Control parameters

Possible names in the control list and their default values are:

**eval.max** Maximum number of evaluations of the objective function allowed. Defaults to 200.

**iter.max** Maximum number of iterations allowed. Defaults to 150.

**trace** The value of the objective function and the parameters is printed every trace’th iteration.

  Defaults to 0 which indicates no trace information is to be printed.

**abs.tol** Absolute tolerance. Defaults to 1e-20.

**rel.tol** Relative tolerance. Defaults to 1e-10.

**x.tol** X tolerance. Defaults to 1.5e-8.

**step.min** Minimum step size. Defaults to 2.2e-14.

Author(s)

Douglas Bates, Deepayan Sarkar

References

http://netlib.bell-labs.com/netlib/port/

Examples

```r
x <- rnbinom(100, mu = 10, size = 10)
hdev <- function(par) {
  -sum(dnbinom(x, mu = par[1], size = par[2], log = TRUE))
}
nlminb(c(9, 12), hdev)
nlminb(c(20, 20), hdev, lower = 0, upper = Inf)
nlminb(c(20, 20), hdev, lower = 0.001, upper = Inf)
```

```r
## slightly modified from the S-PLUS help page for nlminb
# this example minimizes a sum of squares with known solution y
sumsq <- function(x, y) {sum((x-y)^2)}
y <- rep(1,5)
x0 <- rnorm(length(y))
nlminb( start = x0, obj = sumsq, y = y)
# now use bounds with a y that has some components outside the bounds
y <- c(0, 2, 0, -2, 0)
nlminb( start = x0, obj = sumsq, lower = -1, upper = 1, y = y)
# try using the gradient
sumsq.g <- function(x,y) 2*(x-y)
nlminb( start = x0, obj = sumsq, grad = sumsq.g, lo = -1, up = 1, y = y)
# now use the hessian, too
sumsq.h <- function(x,y) diag(2, nrow = length(x))
nlminb(st = x0, obj = sumsq, grad = sumsq.g, hes = sumsq.h, lo = -1, up = 1, y = y)
```

```r
## Rest lifted from optim help page
fr <- function(x) { ## Rosenbrock Banana function
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) { ## Gradient of 'fr'
```
\begin{verbatim}
x1 <- x[1]
x2 <- x[2]
c <- c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
       200 * (x2 - x1 * x1))
}
nlminb(c(-1.2,1), fr)
nlminb(c(-1.2,1), fr, grr)

flb <- function(x)
  
  { p <- length(x);
    sum(c(1, rep(4, p-1)) * (x - c(1, x[-p])^2)^2)
  }

## 25-dimensional box constrained
## par[24] is *not* at boundary
nlminb(rep(3, 25), flb,
       lower=rep(2, 25),
       upper=rep(4, 25))
\end{verbatim}

\section*{nls \hfill Nonlinear Least Squares}

\subsection*{Description}
Determine the nonlinear least-squares estimates of the nonlinear model parameters and return a class \code{nls} object.

\subsection*{Usage}
\code{nls(formula, data, start, control, algorithm,}
trace, subset, weights, na.action, model,
lower, upper, \ldots)}

\subsection*{Arguments}
\begin{itemize}
  \item \code{formula} a nonlinear model formula including variables and parameters
  \item \code{data} an optional data frame in which to evaluate the variables in \code{formula}
  \item \code{start} a named list or named numeric vector of starting estimates
  \item \code{control} an optional list of control settings. See \code{nls.control} for the names of the settable control values and their effect.
  \item \code{algorithm} character string specifying the algorithm to use. The default algorithm is a Gauss-Newton algorithm. Other possible values are "plinear" for the Golub-Pereyra algorithm for partially linear least-squares models and "port" for the "nl2sol" algorithm from the Port package.
  \item \code{trace} logical value indicating if a trace of the iteration progress should be printed. Default is \code{FALSE}. If \code{TRUE} the residual sum-of-squares and the parameter values are printed at the conclusion of each iteration. When the "plinear" algorithm is used, the conditional estimates of the linear parameters are printed after the nonlinear parameters. When the "port" algorithms are used the objective function value printed is half the residual sum-of-squares.
  \item \code{subset} an optional vector specifying a subset of observations to be used in the fitting process.
  \item \code{weights} an optional numeric vector of (fixed) weights. When present, the objective function is weighted least squares. \textit{not yet implemented}
\end{itemize}
na.action: a function which indicates what should happen when the data contain NAs.

model: logical. If true, the model frame is returned as part of the object. Default is FALSE.

lower, upper: vector of lower and upper bounds, replicated to be as long as start. If unspecified, all parameters are assumed to be unconstrained. Bounds can only be used with the "port" algorithm. They are ignored, with a warning, if given for other algorithms.

...: Additional optional arguments. None are used at present.

Details

Do not use \texttt{nls} on artificial "zero-residual" data.

The \texttt{nls} function uses a relative-offset convergence criterion that compares the numerical imprecision at the current parameter estimates to the residual sum-of-squares. This performs well on data of the form

\[ y = f(x, \theta) + \epsilon \]

(with \texttt{var(\epsilon)} > 0). It fails to indicate convergence on data of the form

\[ y = f(x, \theta) \]

because the criterion amounts to comparing two components of the round-off error. If you wish to test \texttt{nls} on artificial data please add a noise component, as shown in the example below.

An \texttt{nls} object is a type of fitted model object. It has methods for the generic functions \texttt{coef}, \texttt{formula}, \texttt{resid}, \texttt{print}, \texttt{summary}, \texttt{AIC}, \texttt{fitted} and \texttt{vcov}.

Variables in \texttt{formula} are looked for first in \texttt{data}, then the environment of \texttt{formula} (added in 1.9.1) and finally along the search path. Functions in \texttt{formula} are searched for first in the environment of \texttt{formula} (added in 1.9.1) and then along the search path.

Value

A list of

\texttt{m}: an \texttt{nlsModel} object incorporating the model

\texttt{data}: the expression that was passed to \texttt{nls} as the data argument. The actual data values are present in the environment of the \texttt{m} component.

Author(s)

Douglas M. Bates and Saikat DebRoy

References


\url{http://netlib.bell-labs.com/netlib/port/} for the Port library

See Also

\texttt{nlsModel}
Examples

DNase1 <- subset(DNase, Run == 1)
## using a selfStart model
fm1DNase1 <- nls( density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)
supply( fm1DNase1 )
## using conditional linearity
fm2DNase1 <- nls( density ~ 1/(1 + exp((xmid - log(conc))/scal)),
                  data = DNase1,
                  start = list(xmid = 0, scal = 1),
                  alg = "plinear", trace = TRUE)
supply( fm2DNase1 )
## without conditional linearity
fm3DNase1 <- nls( density ~ Asym/(1 + exp((xmid - log(conc))/scal)),
                  data = DNase1,
                  start = list(Asym = 3, xmid = 0, scal = 1),
                  trace = TRUE)
supply( fm3DNase1 )
## using the nl2sol algorithm
fm4DNase1 <- nls( density ~ Asym/(1 + exp((xmid - log(conc))/scal)),
                  data = DNase1,
                  start = list(Asym = 3, xmid = 0, scal = 1),
                  trace = TRUE, algorithm = "port")
supply(fm4DNase1)
## weighted nonlinear regression
Treated <- Puromycin[Puromycin$state == "treated", ]
weighted.MM <- function(resp, conc, Vm, K)
{
  ## Purpose: exactly as white book p.451 -- RHS for nls()
  ## Weighted version of Michaelis-Menten model
  ## ---------------------------------------------------------
  ## Arguments: 'y', 'x' and the two parameters (see book)
  ## ---------------------------------------------------------
  ## Author: Martin Maechler, Date: 23 Mar 2001, 18:48

  pred <- (Vm * conc)/(K + conc)
  (resp - pred) / sqrt(pred)
}

Pur.wt <- nls( ~ weighted.MM(rate, conc, Vm, K), data = Treated,
         start = list(Vm = 200, K = 0.1),
         trace = TRUE)
## The two examples below show that you can fit a model to
## artificial data with noise but not to artificial data
## without noise.

x = 1:10
y = x
## perfect fit
yeps = y + rnorm(length(y), sd = 0.01)
## added noise
nls(yeps ~ a + b*x, start = list(a = 0.12345, b = 0.54321),
    trace = TRUE)
## Not run:
nls(y ~ a + b*x, start = list(a = 0.12345, b = 0.54321),
    trace = TRUE)
## End(Not run)
nls.control

Control the Iterations in nls

Description

Allow the user to set some characteristics of the nls nonlinear least squares algorithm.

Usage

nls.control(maxiter = 50, tol = 1e-05, minFactor = 1/1024)

Arguments

- **maxiter**: A positive integer specifying the maximum number of iterations allowed.
- **tol**: A positive numeric value specifying the tolerance level for the relative offset convergence criterion.
- **minFactor**: A positive numeric value specifying the minimum step-size factor allowed on any step in the iteration. The increment is calculated with a Gauss-Newton algorithm and successively halved until the residual sum of squares has been decreased or until the step-size factor has been reduced below this limit.

Value

A list with exactly three components:

- **maxiter**
- **tol**
- **minFactor**

with meanings as explained under ‘Arguments’.

Author(s)

Douglas Bates and Saikat DebRoy

References


See Also

nls

Examples

nls.control(minFactor = 1/2048)
Create an nlsModel Object

This is the constructor for nlsModel objects, which are function closures for several functions in a list. The closure includes a nonlinear model formula, data values for the formula, as well as parameters and their values.

Usage

nlsModel(form, data, start)

Arguments

form a nonlinear model formula
data a data frame or a list in which to evaluate the variables from the model formula
start a named list or named numeric vector of starting estimates for the parameters in the model

Details

An nlsModel object is primarily used within the nls function. It encapsulates the model, the data, and the parameters in an environment and provides several methods to access characteristics of the model. It forms an important component of the object returned by the nls function.

See nls for where elements of the formula form are looked for. In normal use all the variables will be in data.

Value

The value is a list of functions that share a common environment.

resid returns the residual vector evaluated at the current parameter values
fitted returns the fitted responses and their gradient at the current parameter values
formula returns the model formula
deviance returns the residual sum-of-squares at the current parameter values
gradient returns the gradient of the model function at the current parameter values
conv returns the relative-offset convergence criterion evaluated at the current parameter values
incr returns the parameter increment calculated according to the Gauss-Newton formula
setPars a function with one argument, pars. It sets the parameter values for the nlsModel object and returns a logical value denoting a singular gradient array.
getPars returns the current value of the model parameters as a numeric vector
getAllPars returns the current value of the model parameters as a numeric vector
getEnv returns the environment shared by these functions, which contains copies of all the variables in data and has as parent the environment of form.
trace the function that is called at each iteration if tracing is enabled
Rmat the upper triangular factor of the gradient array at the current parameter values
predict takes as argument newdata,a data.frame and returns the predicted re-
sponse for newdata.

Author(s)
Douglas M. Bates and Saikat DebRoy

References

See Also
nls

Examples
DNase1 <- DNase[ DNase$Run == 1, ]
mod <-
nlsModel(density ~ SSlogis( log(conc), Asym, xmid, scal ),
DNase1, list( Asym = 3, xmid = 0, scal = 1 ))
mod$getPars()  # returns the parameters as a list
mod$deviance() # returns the residual sum-of-squares
mod$resid()    # returns the residual vector and the gradient
mod$incr()    # returns the suggested increment
mod$setPars( unlist(mod$getPars()) + mod$incr() )  # set new parameter values
mod$getPars()  # check the parameters have changed
mod$deviance() # see if the parameter increment was successful
mod$trace()   # check the tracing
mod$Rmat()    # R matrix from the QR decomposition of the gradient

---

**NLSstAsymptotic**

*Fit the Asymptotic Regression Model*

Description
Fits the asymptotic regression model, in the form $b_0 + b_1 \times (1 - \exp(-\exp(lrc) \times x))$ to the $xy$ data. This can be used as a building block in determining starting estimates for more complicated models.

Usage
NLSstAsymptotic(xy)

Arguments
xy a sortedXyData object
Value

A numeric value of length 3 with components labelled b0, b1, and lrc. b0 is the estimated intercept on the y-axis, b1 is the estimated difference between the asymptote and the y-intercept, and lrc is the estimated logarithm of the rate constant.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

SSasymp

Examples

Lob.329 <- Loblolly[ Loblolly$Seed == "329", ]
NLSstAsymptotic(sortedXyData(expression(age), expression(height), Lob.329 ))

NLSstClosestX

Inverse Interpolation

Description

Use inverse linear interpolation to approximate the x value at which the function represented by xy is equal to yval.

Usage

NLSstClosestX(xy, yval)

Arguments

xy a sortedXyData object
yval a numeric value on the y scale

Value

A single numeric value on the x scale.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

sortedXyData, NLSstLfAsymptote, NLSstRtAsymptote, selfStart

Examples

DNase.2 <- DNase[ DNase$Run == "2", ]
DN.srt <- sortedXyData( expression(log(conc)), expression(density), DNase.2 )
NLSstClosestX( DN.srt, 1.0 )
NLSstLfAsymptote  

Horizontal Asymptote on the Left Side

Description

Provide an initial guess at the horizontal asymptote on the left side (i.e., small values of \( x \)) of the graph of \( y \) versus \( x \) from the \( xy \) object. Primarily used within initial functions for self-starting nonlinear regression models.

Usage

\[
\text{NLSstLfAsymptote}(xy)
\]

Arguments

xy  
a sortedXyData object

Value

A single numeric value estimating the horizontal asymptote for small \( x \).

Author(s)

Jose Pinheiro and Douglas Bates

See Also

\( \text{sortedXyData, NLSstClosestX, NLSstRtAsymptote, selfStart} \)

Examples

\[
\begin{align*}
\text{DNase.2 } & \leftarrow \text{DNase[ DNase$Run } = \text{ "2"}, ] \\
\text{DN.srt } & \leftarrow \text{sortedXyData( expression(log(conc)), expression(density), DNase.2 )} \\
\text{NLSstLfAsymptote( DN.srt )}
\end{align*}
\]

NLSstRtAsymptote  

Horizontal Asymptote on the Right Side

Description

Provide an initial guess at the horizontal asymptote on the right side (i.e., large values of \( x \)) of the graph of \( y \) versus \( x \) from the \( xy \) object. Primarily used within initial functions for self-starting nonlinear regression models.

Usage

\[
\text{NLSstRtAsymptote}(xy)
\]

Arguments

xy  
a sortedXyData object
Value

A single numeric value estimating the horizontal asymptote for large x.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

sortedXyData, NLSstClosestX, NLSstRtAsymptote, selfStart

Examples

DNase.2 <- DNase[ DNase$Run == "2", ]
DN.srt <- sortedXyData( expression(log(conc)), expression(density), DNase.2 )
NLSstRtAsymptote( DN.srt )

Normal

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution
with mean equal to mean and standard deviation equal to sd.

Usage

dnorm(x, mean=0, sd=1, log = FALSE)
pnorm(q, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean=0, sd=1)

Arguments

x, q

vector of quantiles.

p

vector of probabilities.

n

number of observations. If length(n) > 1, the length is taken to be the number required.

mean

vector of means.

sd

vector of standard deviations.

log, log.p

logical; if TRUE, probabilities p are given as log(p).

lower.tail

logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).
Details

If `mean` or `sd` are not specified they assume the default values of 0 and 1, respectively.

The normal distribution has density

\[ f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]

where \(\mu\) is the mean of the distribution and \(\sigma\) the standard deviation.

`qnorm` is based on Wichura's algorithm AS 241 which provides precise results up to about 16 digits.

Value

dnorm gives the density, pnorm gives the distribution function, qnorm gives the quantile function, and rnorm generates random deviates.

References


See Also

runif and .Random.seed about random number generation, and dlnorm for the Lognormal distribution.

Examples

dnorm(0) == 1/ sqrt(2*pi)
dnorm(1) == exp(-1/2)/ sqrt(2*pi)
dnorm(l) == 1/ sqrt(2*pi*exp(l))

## Using "log = TRUE" for an extended range :
par(mfrow=c(2,1))
plot(function(x)dnorm(x, log=TRUE), -60, 50, main = "log { Normal density }")
curve(log(dnorm(x)), add=TRUE, col="red",lwd=2)

## if you want the so-called 'error function'
erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1
## (see Abrahamowitz and Stegun 29.2.29)
erfc <- function(x) 2 * pnorm(x * sqrt(2), lower=FALSE)
### numericDeriv

**Evaluate derivatives numerically**

**Description**

`numericDeriv` numerically evaluates the gradient of an expression.

**Usage**

```r
class (numericDeriv
  expr, theta, rho = parent.frame()
```

**Arguments**

- **expr**: The expression to be differentiated. The value of this expression should be a numeric vector.
- **theta**: A character vector of names of variables used in `expr`.
- **rho**: An environment containing all the variables needed to evaluate `expr`.

**Details**

This is a front end to the C function `numeric_deriv`, which is described in *Writing R Extensions*.

**Value**

The value of `eval(expr, envir = rho)` plus a matrix attribute called `gradient`. The columns of this matrix are the derivatives of the value with respect to the variables listed in `theta`.

**Author(s)**

Saikat DebRoy (saikat@stat.wisc.edu)

**Examples**

```r
myenv <- new.env()
assign("mean", 0., env = myenv)
assign("sd", 1., env = myenv)
assign("x", seq(-3., 3., len = 31), env = myenv)
numericDeriv(quote(pnorm(x, mean, sd)), c("mean", "sd"), myenv)
```

### offset

**Include an Offset in a Model Formula**

**Description**

An offset is a term to be added to a linear predictor, such as in a generalised linear model, with known coefficient 1 rather than an estimated coefficient.

**Usage**

```r
offset(object)
```
oneway.test

Arguments

object    An offset to be included in a model frame

Details

There can be more than one offset in a model formula, but – is not supported for offset terms (and is equivalent to +).

Value

The input value.

See Also

model.offset, model.frame.

For examples see glm and Insurance in package MASS.

oneway.test  Test for Equal Means in a One-Way Layout

Description

Test whether two or more samples from normal distributions have the same means. The variances are not necessarily assumed to be equal.

Usage

oneway.test(formula, data, subset, na.action, var.equal = FALSE)

Arguments

formula    a formula of the form lhs ~ rhs where lhs gives the sample values and rhs the corresponding groups.
data    an optional data frame containing the variables in the model formula.
subset    an optional vector specifying a subset of observations to be used.
na.action    a function which indicates what should happen when the data contain NAs. Defaults togetOption("na.action").
var.equal    a logical variable indicating whether to treat the variances in the samples as equal. If TRUE, then a simple F test for the equality of means in a one-way analysis of variance is performed. If FALSE, an approximate method of Welch (1951) is used, which generalizes the commonly known 2-sample Welch test to the case of arbitrarily many samples.
Value

A list with class "htest" containing the following components:

- **statistic**: the value of the test statistic.
- **parameter**: the degrees of freedom of the exact or approximate F distribution of the test statistic.
- **p.value**: the p-value of the test.
- **method**: a character string indicating the test performed.
- **data.name**: a character string giving the names of the data.

References


See Also

The standard t test (`t.test`) as the special case for two samples; the Kruskal-Wallis test `kruskal.test` for a nonparametric test for equal location parameters in a one-way layout.

Examples

```r
## Not assuming equal variances
oneway.test(extra ~ group, data = sleep)
## Assuming equal variances
oneway.test(extra ~ group, data = sleep, var.equal = TRUE)
## which gives the same result as
anova(lm(extra ~ group, data = sleep))
```

---

`optim`  
*General-purpose Optimization*

Description

General-purpose optimization based on Nelder–Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization and simulated annealing.

Usage

```r
optim(par, fn, gr = NULL,  
method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN"),  
lower = -Inf, upper = Inf,  
control = list(), hessian = FALSE, ...)
```
Arguments

par
Initial values for the parameters to be optimized over.

fn
A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.

gr
A function to return the gradient for the "BFGS", "CG" and "L-BFGS-B" methods. If it is NULL, a finite-difference approximation will be used. For the "SANN" method it specifies a function to generate a new candidate point. If it is NULL a default Gaussian Markov kernel is used.

method
The method to be used. See Details.

lower, upper
Bounds on the variables for the "L-BFGS-B" method.

control
A list of control parameters. See Details.

hessian
Logical. Should a numerically differentiated Hessian matrix be returned?

...
Further arguments to be passed to fn and gr.

Details

By default this function performs minimization, but it will maximize if control$fnscale is negative.

The default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions.

Method "BFGS" is a quasi-Newton method (also known as a variable metric algorithm), specifically that published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno. This uses function values and gradients to build up a picture of the surface to be optimized.

Method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak–Ribiere or Beale–Sorenson updates). Conjugate gradient methods will generally be more fragile that the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.

Method "L-BFGS-B" is that of Byrd et al. (1995) which allows box constraints, that is each variable can be given a lower and/or upper bound. The initial value must satisfy the constraints. This uses a limited-memory modification of the BFGS quasi-Newton method. If non-trivial bounds are supplied, this method will be selected, with a warning.

Nocedal and Wright (1999) is a comprehensive reference for the previous three methods.

Method "SANN" is by default a variant of simulated annealing given in Belisle (1992). Simulated-annealing belongs to the class of stochastic global optimization methods. It uses only function values but is relatively slow. It will also work for non-differentiable functions. This implementation uses the Metropolis function for the acceptance probability. By default the next candidate point is generated from a Gaussian Markov kernel with scale proportional to the actual temperature. If a function to generate a new candidate point is given, method "SANN" can also be used to solve combinatorial optimization problems. Temperatures are decreased according to the logarithmic cooling schedule as given in Belisle (1992, p. 890). Note that the "SANN" method depends critically on the settings of the control parameters. It is not a general-purpose method but can be very useful in getting to a good value on a very rough surface.

Function fn can return NA or Inf if the function cannot be evaluated at the supplied value, but the initial value must have a computable finite value of fn. (Except for method "L-BFGS-B" where the values should always be finite.)
optim can be used recursively, and for a single parameter as well as many.

The control argument is a list that can supply any of the following components:

**trace** Non-negative integer. If positive, tracing information on the progress of the optimization is produced. Higher values may produce more tracing information: for method "L-BFGS-B" there are six levels of tracing. (To understand exactly what these do see the source code: higher levels give more detail.)

**fnscale** An overall scaling to be applied to the value of \( f_n \) and \( q \) during optimization. If negative, turns the problem into a maximization problem. Optimization is performed on \( f_n(par)/fnscale \).

**parscale** A vector of scaling values for the parameters. Optimization is performed on \( par/parscale \) and these should be comparable in the sense that a unit change in any element produces about a unit change in the scaled value.

**ndeps** A vector of step sizes for the finite-difference approximation to the gradient, on \( par/parscale \) scale. Defaults to \( 1e^{-3} \).

**maxit** The maximum number of iterations. Defaults to 100 for the derivative-based methods, and 500 for "Nelder-Mead". For "SANN" \( maxit \) gives the total number of function evaluations. There is no other stopping criterion. Defaults to 10000.

**abstol** The absolute convergence tolerance. Only useful for non-negative functions, as a tolerance for reaching zero.

**reltol** Relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor of \( reltol \times (abs(val) + reltol) \) at a step. Defaults to \( \sqrt{\text{.Machine$double.eps}} \), typically about \( 1e^{-8} \).

**alpha, beta, gamma** Scaling parameters for the "Nelder-Mead" method. \( alpha \) is the reflection factor (default 1.0), \( beta \) the contraction factor (0.5) and \( gamma \) the expansion factor (2.0).

**REPORT** The frequency of reports for the "BFGS" and "L-BFGS-B" methods if \( control$trace \) is positive. Defaults to every 10 iterations.

**type** for the conjugate-gradients method. Takes value 1 for the Fletcher–Reeves update, 2 for Polak–Ribiere and 3 for Beale–Sorenson.

**lmm** is an integer giving the number of BFGS updates retained in the "L-BFGS-B" method. It defaults to 5.

**factr** controls the convergence of the "L-BFGS-B" method. Convergence occurs when the reduction in the objective is within this factor of the machine tolerance. Default is \( 1e7 \), that is a tolerance of about \( 1e^{-8} \).

**pgtol** helps controls the convergence of the "L-BFGS-B" method. It is a tolerance on the projected gradient in the current search direction. This defaults to zero, when the check is suppressed.

**temp** controls the "SANN" method. It is the starting temperature for the cooling schedule. Defaults to 10.

**tmax** is the number of function evaluations at each temperature for the "SANN" method. Defaults to 10.

**Value**

A list with components:

- **par** The best set of parameters found.
- **value** The value of \( fn \) corresponding to \( par \).
counts

A two-element integer vector giving the number of calls to fn and gr respectively. This excludes those calls needed to compute the Hessian, if requested, and any calls to fn to compute a finite-difference approximation to the gradient.

convergence

An integer code. 0 indicates successful convergence. Error codes are

1 indicates that the iteration limit maxit had been reached.
10 indicates degeneracy of the Nelder–Mead simplex.
51 indicates a warning from the "L-BFGS-B" method; see component message for further details.
52 indicates an error from the "L-BFGS-B" method; see component message for further details.

message

A character string giving any additional information returned by the optimizer, or NULL.

hessian

Only if argument hessian is true. A symmetric matrix giving an estimate of the Hessian at the solution found. Note that this is the Hessian of the unconstrained problem even if the box constraints are active.

Note

optim will work with one-dimensional pars, but the default method does not work well (and will warn). Use optimize instead.

The code for methods "Nelder–Mead", "BFGS" and "CG" was based originally on Pascal code in Nash (1990) that was translated by p2c and then hand-optimized. Dr Nash has agreed that the code can be made freely available.

The code for method "L-BFGS-B" is based on Fortran code by Zhu, Byrd, Lu-Chen and Nocedal obtained from Netlib (file 'opt/lbfgs_bcm.shar': another version is in 'toms/778').

The code for method "SANN" was contributed by A. Trapletti.

References


See Also

nlm, optimize, constrOptim
**Examples**

```r
fr <- function(x) {  ## Rosenbrock Banana function
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) {  ## Gradient of 'fr'
  x1 <- x[1]
  x2 <- x[2]
  c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
     200 * (x2 - x1 * x1))
}
optim(c(-1.2,1), fr)
optim(c(-1.2,1), fr, grr, method = "BFGS")
optim(c(-1.2,1), fr, NULL, method = "BFGS", hessian = TRUE)
optim(c(-1.2,1), fr, grr, method = "CG")
optim(c(-1.2,1), fr, grr, method = "CG", control=list(type=2))
optim(c(-1.2,1), fr, grr, method = "L-BFGS-B")
flb <- function(x)  
{ p <- length(x); sum(c(1, rep(4, p-1)) * (x - c(1, x[-p])^2)^2) }
## 25-dimensional box constrained
optim(rep(3, 25), flb, NULL, "L-BFGS-B",
       lower=rep(2, 25), upper=rep(4, 25)) # par[24] is *not* at boundary
## "wild" function , global minimum at about -15.81515
fw <- function (x)
  10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80
plot(fw, -50, 50, n=1000, main = "optim() minimising 'wild function'")
res <- optim(50, fw, method="SANN",
             control=list(maxit=20000, temp=20, parscale=20))
res
## Now improve locally
(r2 <- optim(res$par, fw, method="BFGS"))
points(r2$par, r2$val, pch = 8, col = "red", cex = 2)
## Combinatorial optimization: Traveling salesman problem
library(stats) # normally loaded
eurodistmat <- as.matrix(eurodist)
distance <- function(sq) {  # Target function
  sq2 <- embed(sq, 2)
  return(sum(eurodistmat[cbind(sq2[,2],sq2[,1]))))
}
genseq <- function(sq) {  # Generate new candidate sequence
  idx <- seq(2, NROW(eurodistmat)-1, by=1)
  changepoints <- sample(idx, size=2, replace=FALSE)
  tmp <- sq[changepoints[1]]
  sq[changepoints[1]] <- sq[changepoints[2]]
  sq[changepoints[2]] <- tmp
  return(sq)
}
```
```r
# Initial sequence
sq <- c(1,2:NROW(eurodistmat),1)

# Chosen to get a good solution quickly
res <- optim(sq, distance, genseq, method="SANN",
control = list(maxit=6000, temp=2000, trace=TRUE))

loc <- cmdscale(eurodist)
rx <- range(x <- loc[,1])
ry <- range(y <- -loc[,2])
tspin <- loc[sq]
tspres <- loc[res$par,]
s <- seq(NROW(tspres)-1)

plot(x, y, type="n", asp=1, xlab="", ylab="",
main="initial solution of traveling salesman problem")
arrows(tspin[s,1], -tpinit[s,2], tspinit[s+1,1], tspinit[s+1,2],
angle=10, col="green")
text(x, y, names(eurodist), cex=0.8)

plot(x, y, type="n", asp=1, xlab="", ylab="",
main="optim() 'solving' traveling salesman problem")
arrows(tspres[s,1], -tspres[s,2], tspres[s+1,1], tspres[s+1,2],
angle=10, col="red")
text(x, y, names(eurodist), cex=0.8)
```

---

### optimize

**One Dimensional Optimization**

**Description**

The function `optimize` searches the interval from `lower` to `upper` for a minimum or maximum of the function `f` with respect to its first argument.

`optimise` is an alias for `optimize`.

**Usage**

```r
optimize(f = , interval = , lower = min(interval),
upper = max(interval), maximum = FALSE,
tol = .Machine$double.eps^0.25, ...)
```

```r
optimise(f = , interval = , lower = min(interval),
upper = max(interval), maximum = FALSE,
tol = .Machine$double.eps^0.25, ...)
```

**Arguments**

- `f`: the function to be optimized. The function is either minimized or maximized over its first argument depending on the value of `maximum`.
- `interval`: a vector containing the end-points of the interval to be searched for the minimum.
- `lower`: the lower end point of the interval to be searched.
upper
maximum
tol

the upper end point of the interval to be searched.
the desired accuracy.

logical. Should we maximize or minimize (the default)?

... additional arguments to \( f \).

Details

The method used is a combination of golden section search and successive parabolic interpolation. Convergence is never much slower than that for a Fibonacci search. If \( f \) has a continuous second derivative which is positive at the minimum (which is not at \( \text{lower} \) or \( \text{upper} \)), then convergence is superlinear, and usually of the order of about 1.324.

The function \( f \) is never evaluated at two points closer together than \( \epsilon|x_0| + (\text{tol}/3) \), where \( \epsilon \) is approximately \( \sqrt{\text{.Machine}$double.eps} \) and \( x_0 \) is the final abscissa \( \text{optimize()}\$\text{minimum} \).

If \( f \) is a unimodal function and the computed values of \( f \) are always unimodal when separated by at least \( \epsilon|x| + (\text{tol}/3) \), then \( x_0 \) approximates the abscissa of the global minimum of \( f \) on the interval \( \text{lower}, \text{upper} \) with an error less than \( \epsilon|x_0| + \text{tol} \).

If \( f \) is not unimodal, then \( \text{optimize()} \) may approximate a local, but perhaps non-global, minimum to the same accuracy.

The first evaluation of \( f \) is always at \( x_1 = a + (1 - \phi)(b - a) \) where \( (a, b) = (\text{lower}, \text{upper}) \) and \( \phi = (\sqrt{5} - 1)/2 = 0.61803.. \) is the golden section ratio. Almost always, the second evaluation is at \( x_2 = a + \phi(b - a) \). Note that a local minimum inside \( [x_1, x_2] \) will be found as solution, even when \( f \) is constant in there, see the last example.

It uses a C translation of Fortran code (from Netlib) based on the Algol 60 procedure \text{localmin} \) given in the reference.

Value

A list with components \( \text{minimum} \) (or \( \text{maximum} \)) and \( \text{objective} \) which give the location of the minimum (or maximum) and the value of the function at that point.

References


See Also

\text{nlm}, \text{uniroot}.

Examples

```r
f <- function (x,a) (x-a)^2
xmin <- optimize(f, c(0, 1), tol = 0.0001, a = 1/3)

## See where the function is evaluated:
optimize(function(x) x^2*(print(x)-1), l=0, u=10)

## "wrong" solution with unlucky interval and piecewise constant f():
f <- function(x) ifelse(x > -1, ifelse(x < 4, exp(-1/abs(x - 1)), 10), 10)
fp <- function(x) { print(x); f(x) }
```
plot(f, -2,5, ylim = 0:1, col = 2)
optimize(fp, c(-4, 20))# doesn't see the minimum
optimize(fp, c(-7, 20))# ok

order.dendrogram

Ordering or Labels of the Leaves in a Dendrogram

Description

Theses functions return the order (index) or the "label" attribute for the leaves in a dendrogram. These indices can then be used to access the appropriate components of any additional data.

Usage

order.dendrogram(x)

## S3 method for class 'dendrogram':
labels(object, ...)

Arguments

x, object a dendrogram (see as.dendrogram).
...
additional arguments

Details

The indices or labels for the leaves in left to right order are retrieved.

Value

A vector with length equal to the number of leaves in the dendrogram is returned. From r <- order.dendrogram(), each element is the index into the original data (from which the dendrogram was computed).

Author(s)

R. Gentleman (order.dendrogram and Martin Maechler (labels.dendrogram).

See Also

reorder.dendrogram.

Examples

set.seed(123)
x <- rnorm(10)
hc <- hclust(dist(x))
hc$order
dd <- as.dendrogram(hc)
order.dendrogram(dd) ## the same :
stopifnot(hc$order == order.dendrogram(dd))
d2 <- as.dendrogram(hclust(dist(USArrests)))
labels(d2) ## in this case the same as
stopifnot(labels(d2) == rownames(USArrests)[order.dendrogram(d2)])

p.adjust

Adjust P-values for Multiple Comparisons

Description

Given a set of p-values, returns p-values adjusted using one of several methods.

Usage

p.adjust(p, method = p.adjust.methods, n = length(p))

p.adjust.methods
# c("holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none")

Arguments

p vector of p-values (possibly with NAs).
method correction method
n number of comparisons, must be at least length(p); only set this (to non-default) when you know what you are doing!

Details

The adjustment methods include the Bonferroni correction ("bonferroni") in which the p-values are multiplied by the number of comparisons. Less conservative corrections are also included by Holm (1979) ("holm"), Hochberg (1988) ("hochberg"), Hommel (1988) ("hommel"), Benjamini & Hochberg (1995) ("BH"), and Benjamini & Yekutieli (2001) ("BY"), respectively. A pass-through option ("none") is also included. The set of methods are contained in the p.adjust.methods vector for the benefit of methods that need to have the method as an option and pass it on to p.adjust.

The first four methods are designed to give strong control of the family wise error rate. There seems no reason to use the unmodified Bonferroni correction because it is dominated by Holm’s method, which is also valid under arbitrary assumptions.

Hochberg’s and Hommel’s methods are valid when the hypothesis tests are independent or when they are non-negatively associated (Sarkar, 1998; Sarkar and Chang, 1997). Hommel’s method is more powerful than Hochberg’s, but the difference is usually small and the Hochberg p-values are faster to compute.

The "BH" and "BY" method of Benjamini, Hochberg, and Yekutieli control the false discovery rate, the expected proportion of false discoveries amongst the rejected hypotheses. The false discovery rate is a less stringent condition than the family wise error rate, so these methods are more powerful than the others.

Note that you can set n larger than length(p) which means the unobserved p-values are assumed to be greater than all the observed p for "bonferroni" and "holm" methods and equal to 1 for the other methods.
Value

A vector of corrected p-values (same length as p).

References


See Also

pairwise.* functions such as `pairwise.t.test`.

Examples

```r
set.seed(123)
x <- rnorm(50, m=c(rep(0,25),rep(3,25)))
p <- 2*pnorm( sort(-abs(x)))
round(p, 3)
round(p.adjust(p), 3)
round(p.adjust(p, "BH"), 3)

## or all of them at once (dropping the "fdr" alias):
p.adj <- sapply(p.adjust.M, function(meth) p.adjust(p, meth))
round(p.adj, 3)

## or a bit nicer:
noquote(apply(p.adj, 2, format.pval, digits = 3))

## and a graphic:
matplot(p, p.adj, ylab="p.adjust(p, meth)", type = "l", asp=1, lty=1:6,
main = "P-value adjustments")
legend(.7,.6, p.adjust.M, col=1:6, lty=1:6)

## Can work with NA's:
pN <- p; iN <- c(46,47); pN[iN] <- NA
```
pN.a <- sapply(p.adjust.M, function(meth) p.adjust(pN, meth))
## The smallest 20 P-values all affected by the NA's:
round((pN.a / p.adj)[1:20, , 4])

---

pairwise.prop.test  Pairwise comparisons for proportions

Description

Calculate pairwise comparisons between pairs of proportions with correction for multiple testing

Usage

pairwise.prop.test(x, n, p.adjust.method = p.adjust.methods, ...)

Arguments

x  
Vector of counts of successes or a matrix with 2 columns giving the counts of successes and failures, respectively.

n  
Vector of counts of trials; ignored if x is a matrix.

p.adjust.method  
Method for adjusting p values (see p.adjust)

...  
Additional arguments to pass to prop.test

Value

Object of class "pairwise.htest"

See Also

prop.test, p.adjust

Examples

smokers <- c( 83, 90, 129, 70 )
patients <- c( 86, 93, 136, 82 )
pairwise.prop.test(smokers, patients)
pairwise.t.test  Pairwise t tests

Description
Calculate pairwise comparisons between group levels with corrections for multiple testing

Usage
pairwise.t.test(x, g, p.adjust.method = p.adjust.methods,
                pool.sd = TRUE, ...)

Arguments
x    Response vector
  g    Grouping vector or factor
  p.adjust.method  Method for adjusting p values (see p.adjust)
  pool.sd  Switch to allow/disallow the use of a pooled SD
  ...     Additional arguments to pass to t.test

Value
Object of class "pairwise.htest"

See Also
t.test, p.adjust

Examples
attach(airquality)
Month <- factor(Month, labels = month.abb[5:9])
pairwise.t.test(Ozone, Month)
pairwise.t.test(Ozone, Month, p.adj = "bonf")
pairwise.t.test(Ozone, Month, pool.sd = FALSE)
detach()

pairwise.table  Tabulate p values for pairwise comparisons

Description
Creates table of p values for pairwise comparisons with corrections for multiple testing.

Usage
pairwise.table(compare.levels, level.names, p.adjust.method)
Arguments

compare.levels
  Function to compute (raw) p value given indices i and j
level.names  Names of the group levels
p.adjust.method
  Method for multiple testing adjustment

Details

Functions that do multiple group comparisons create separate compare.levels functions (assumed to be symmetrical in i and j) and passes them to this function.

Value

Table of p values in lower triangular form.

See Also

pairwise.t.test, et al.
Examples

attach(airquality)
Month <- factor(Month, labels = month.abb[5:9])
## These give warnings because of ties :
pairwise.wilcox.test(Ozone, Month)
pairwise.wilcox.test(Ozone, Month, p.adj = "bonf")
detach()

plot.acf

Plot Autocovariance and Autocorrelation Functions

Description

Plot method for objects of class "acf".

Usage

## S3 method for class 'acf':
plot(x, ci = 0.95, type = "h", xlab = "Lag", ylab = NULL,
     ylim = NULL, main = NULL,
     ci.col = "blue", ci.type = c("white", "ma"),
     max.mfrow = 6, ask = Ngps > 1 && dev.interactive(),
     mar = if(nser > 2) c(3,2,2,0.8) else par("mar"),
     oma = if(nser > 2) c(1,1.2,1,1) else par("oma"),
     mgp = if(nser > 2) c(1.5,0.6,0) else par("mgp"),
     xpd = par("xpd"), cex.main = if(nser > 2) 1 else par("cex.main"),
     verbose = getOption("verbose"),
     ...)

Arguments

x an object of class "acf".

coverage probability for confidence interval. Plotting of the confidence interval
is suppressed if ci is zero or negative.

type the type of plot to be drawn, default to histogram like vertical lines.
xlab the x label of the plot.
ylab the y label of the plot.
ylim numeric of length 2 giving the y limits for the plot.
main overall title for the plot.

colour to plot the confidence interval lines.

should the confidence limits assume a white noise input or for lag k an MA(k-1)
input?

max.mfrow positive integer; for multivariate x indicating how many rows and columns of
plots should be put on one page, using par(mfrow = c(m,m)).
ask logical; if TRUE, the user is asked before a new page is started.

mar, oma, mgp, xpd, cex.main

graphics parameters as in par(*), by default adjusted to use smaller than de-
fault margins for multivariate x only. xpd = NA used to be the default for R
version <= 1.4.0.
verbose logical. Should R report extra information on progress?
...

Note
The confidence interval plotted in plot.acf is based on an uncorrelated series and should be treated with appropriate caution. Using ci.type = "ma" may be less potentially misleading.

See Also
acf which calls plot.acf by default.

Examples

z4 <- ts(matrix(rnorm(400), 100, 4), start=c(1961, 1), frequency=12)
z7 <- ts(matrix(rnorm(700), 100, 7), start=c(1961, 1), frequency=12)
acf(z4)
acf(z7, max.mfrow = 7)# squeeze on 1 page
acf(z7) # multi-page
plot.HoltWinters  
Plot function for HoltWinters objects

Description

Produces a chart of the original time series along with the fitted values. Optionally, predicted values (and their confidence bounds) can also be plotted.

Usage

```r
## S3 method for class 'HoltWinters':
plot(x, predicted.values = NA, intervals = TRUE,
     separator = TRUE, col = 1, col.predicted = 2,
     col.intervals = 4, col.separator = 1, lty = 1,
     lty.predicted = 1, lty.intervals = 1, lty.separator = 3,
     ylab = "Observed / Fitted", main = "Holt-Winters filtering",
     ylim = NULL, ...) # S3 method for class 'HoltWinters':
```

Arguments

- `x` Object of class "HoltWinters"
- `predicted.values` Predicted values as returned by `predict.HoltWinters`
- `intervals` If `TRUE`, the prediction intervals are plotted (default).
- `separator` If `TRUE`, a separating line between fitted and predicted values is plotted (default).
- `col, lty` Color/line type of original data (default: black solid).
- `col.predicted, lty.predicted` Color/line type of fitted and predicted values (default: red solid).
- `col.intervals, lty.intervals` Color/line type of prediction intervals (default: blue solid).
- `col.separator, lty.separator` Color/line type of observed/predicted values separator (default: black dashed).
- `ylab` Label of the y-axis.
- `main` Main title.
- `ylim` Limits of the y-axis. If `NULL`, the range is chosen such that the plot contains the original series, the fitted values, and the predicted values if any.
- `...` Other graphics parameters.

Author(s)

David Meyer (David.Meyer@wu-wien.ac.at)

References


The `plot` method for R objects of class `isoreg`.

Usage

```r
## S3 method for class 'isoreg':
plot(x, plot.type = c("single", "row.wise", "col.wise"),
     main = paste("Isotonic regression", deparse(x$call)),
     main2 = "Cumulative Data and Convex Minorant",
     xlab = "x0", ylab = "x$y",
     par.fit = list(col = "red", cex = 1.5, pch = 13, lwd = 1.5),
     mar = if (both) 0.1 + c(3.5, 2.5, 1, 1) else par("mar"),
     mgp = if (both) c(1.6, 0.7, 0) else par("mgp"),
     grid = length(x$x) < 12, ...)
```

Arguments

- `x`: an `isoreg` object.
- `plot.type`: character indicating which type of plot is desired. The first (default) only draws the data and the fit, where the others add a plot of the cumulative data and fit.
- `main`: main title of plot, see `title`.
- `main2`: title for second (cumulative) plot.
- `xlab, ylab`: x- and y- axis annotation.
- `par.fit`: a list of arguments (for `points` and `lines`) for drawing the fit.
- `mar, mgp`: graphical parameters, see `par`, mainly for the case of two plots.
- `grid`: logical indicating if grid lines should be drawn. If true, `grid()` is used for the first plot, where as vertical lines are drawn at “touching” points for the cumulative plot.
- `...`: further arguments passed to and from methods.

See Also

- `isoreg` for computation of `isoreg` objects.
- `HoltWinters`, `predict.HoltWinters`
Examples

example(isoreg) # for the examples there

## 'same' plot as above, "proving" that only ranks of 'x' are important
plot(isoreg(2^(1:9), c(1,0,4,3,3,5,4,2,0)), plot.t = "row", log = "x")

plot(ir3, plot.type = "row", ylab = "y3")
plot(isoreg(y3 - 4), plot.t="r", ylab = "y3 - 4")
plot(ir4, plot.type = "ro", ylab = "y4", xlab = "x - 1:n")

## experiment a bit with these (C-c C-j):
plot(isoreg(sample(9), y3), plot.type="row")
plot(isoreg(sample(9), y3), plot.type="col.wise")

plot(ir <- isoreg(sample(10), sample(10, replace = TRUE)), plot.t = "r")

plot.lm

--

Plot Diagnostics for an lm Object

Description

Six plots (selectable by which) are currently available: a plot of residuals against fitted values, a Scale-Location plot of $\sqrt{\text{residuals}}$ against fitted values, a Normal Q-Q plot, a plot of Cook’s distances versus row labels, a plot of residuals against leverages, and a plot of Cook’s distances against leverage/(1-leverage). By default, the first three and 5 are provided.

Usage

## S3 method for class 'lm':
plot(x, which = c(1:3,5),
caption = c("Residuals vs Fitted", "Normal Q-Q", 
"Scale-Location", "Cook's distance", 
"Residuals vs Leverage", "Cook's distance vs Leverage"), 
panel = if(add.smooth) panel.smooth else points, 
sub.caption = NULL, main = ",
ask = prod(par("mfcol")) < length(which) && dev.interactive(), 
...,
id.n = 3, labels.id = names(residuals(x)), cex.id = 0.75, 
qqline = TRUE, cook.levels = c(0.5, 1.0), 
add.smooth = getOption("add.smooth"), label.pos = c(4,2))

Arguments

x  lm object, typically result of lm or glm.
which  if a subset of the plots is required, specify a subset of the numbers 1:6.
caption  captions to appear above the plots
panel  panel function. The useful alternative to points, panel.smooth can be chosen by add.smooth = TRUE.
sub.caption  common title—above figures if there are multiple; used as sub (s.title) otherwise. If NULL, as by default, a possible shortened version of deparse(x$call) is used.
main  title to each plot—in addition to the above caption.
ask  logical; if TRUE, the user is asked before each plot, see \texttt{par}(ask=.)
...  other parameters to be passed through to plotting functions.
id.n  number of points to be labelled in each plot, starting with the most extreme.
labels.id  vector of labels, from which the labels for extreme points will be chosen. \texttt{NULL} uses observation numbers.
cex.id  magnification of point labels.
qqline  logical indicating if a \texttt{qqline}() should be added to the normal Q-Q plot.
cook.levels  levels of Cook’s distance at which to draw contours.
add.smooth  logical indicating if a smoother should be added to most plots; see also \texttt{panel} above.
label.pos  position of labels, for the left half and right half of the graph respectively, for plots 1-3.

**Details**

\texttt{sub.caption}—by default the function call—is shown as a subtitle (under the x-axis title) on each plot when plots are on separate pages, or as a subtitle in the outer margin (if any) when there are multiple plots per page.

The “Scale-Location” plot, also called “Spread-Location” or “S-L” plot, takes the square root of the absolute residuals in order to diminish skewness ($\sqrt{|E|}$ is much less skewed than $|E|$ for Gaussian zero-mean $E$).

The ‘S-L’, the Q-Q, and the Residual-Leverage plot, use standardized residuals which have identical variance (under the hypothesis). They are given as $R_i/(s \times \sqrt{1 - h_{ii}})$ where $h_{ii}$ are the diagonal entries of the hat matrix, \texttt{influence}() $\hat{\text{hat}}$, see also \texttt{hat}.

The Residual-Leverage plot shows contours of equal Cook’s distance, for values of \texttt{cook.levels}, by default 0.5 and 1. If the leverages are constant, as typically in a balanced \texttt{aov} situation, the plot uses factor level combinations instead of the leverages for the x-axis.

In the Cook’s distance vs leverage/(1-leverage) plot, contours of standardized residuals that are equal in magnitude are lines through the origin. The contour lines are labeled with the magnitudes.

**Author(s)**

John Maindonald and Martin Maechler.

**References**


**See Also**

termplot, \texttt{lm.influence}, \texttt{cooks.distance}, \texttt{hatvalues}. 
Examples

```r
## Analysis of the life-cycle savings data
## given in Belsley, Kuh and Welsch.
plot(lm.SR <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings))

## 4 plots on 1 page; allow room for printing model formula in outer margin:
par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
plot(lm.SR)
plot(lm.SR, id.n = NULL) # no id's
plot(lm.SR, id.n = 5, labels.id = NULL) # 5 id numbers

## Was default in R <= 2.1.x: Cook's distances instead of Residual-Leverage plot
plot(lm.SR, which = 1:4)

## Fit a smooth curve, where applicable:
plot(lm.SR, panel = panel.smooth)
## Gives a smoother curve
plot(lm.SR, panel = function(x,y) panel.smooth(x, y, span = 1))

par(mfrow = c(2, 1)) # same oma as above
plot(lm.SR, which = 1:2, sub.caption = "Saving Rates, n=50, p=5")
```

---

**plot.ppr**  
*Plot Ridge Functions for Projection Pursuit Regression Fit*

**Description**

Plot ridge functions for projection pursuit regression fit.

**Usage**

```r
## S3 method for class 'ppr':
plot(x, ask, type = "o", ...)  
```

**Arguments**

- `x`  
  A fit of class "ppr" as produced by a call to `ppr`.
- `ask`  
  the graphics parameter `ask`: see `par` for details. If set to `TRUE` will ask between the plot of each cross-section.
- `type`  
  the type of line to draw
- `...`  
  further graphical parameters

**Value**

None

**Side Effects**

A series of plots are drawn on the current graphical device, one for each term in the fit.
plot.profile.nls

See Also
ppr, par

Examples

attach(rock)
areal <- area/10000; peril <- peri/10000
par(mfrow=c(3,2))# maybe: , pty="s")
rock.ppr <- ppr(log(perm) ~ areal + peril + shape,
               data = rock, nterms = 2, max.terms = 5)
plot(rock.ppr, main="ppr(log(perm)~ ., nterms=2, max.terms=5)"
plot(update(rock.ppr, bass=5), main = "update(..., bass = 5)"
plot(update(rock.ppr, sm.method="gcv", gcvpen=2),
       main = "update(..., sm.method="gcv", gcvpen=2)"
detach()

Description
Displays a series of plots of the profile t function and interpolated confidence intervals for the parameters in a nonlinear regression model that has been fit with nls and profiled with profile.nls.

Usage

## S3 method for class 'profile.nls':
plot(x, levels, conf = c(99, 95, 90, 80, 50)/100, 
nseg = 50, absVal = TRUE, ...)

Arguments

x an object of class "profile.nls"
levels levels, on the scale of the absolute value of a t statistic, at which to interpolate intervals. Usually conf is used instead of giving levels explicitly.
conf a numeric vector of confidence levels for profile-based confidence intervals on the parameters. Defaults to c(0.99, 0.95, 0.90, 0.80, 0.50).
nseg an integer value giving the number of segments to use in the spline interpolation of the profile t curves. Defaults to 50.
absVal a logical value indicating whether or not the plots should be on the scale of the absolute value of the profile t. Defaults to TRUE.
... other arguments to the plot function can be passed here.

Author(s)
Douglas M. Bates and Saikat DebRoy

References
plot.spec

See Also
nls, profile, profile.nls

Examples

# obtain the fitted object
fm1 <- nls(demand ~ SSasympOrig( Time, A, lrc ), data = BOD)
# get the profile for the fitted model
pr1 <- profile( fm1 )
opar <- par(mfrow = c(2,2), oma = c(1.1, 0, 1.1, 0), las = 1)
plot(pr1, conf = c(95, 90, 80, 50)/100)
plot(pr1, conf = c(95, 90, 80, 50)/100, absVal = FALSE)
mtext("Confidence intervals based on the profile sum of squares",
     side = 3, outer = TRUE)
mtext("BOD data - confidence levels of 50%, 80%, 90% and 95%",
     side = 1, outer = TRUE)
par(opar)

plot.spec

Plotting Spectral Densities

Description

Plotting method for objects of class "spec". For multivariate time series it plots the marginal spectra of the series or pairs plots of the coherency and phase of the cross-spectra.

Usage

## S3 method for class 'spec':
plot(x, add = FALSE, ci = 0.95, log = c("yes", "dB", "no"),
     xlab = "frequency", ylab = NULL, type = "l",
     ci.col = "blue", ci.lty = 3,
     main = NULL, sub = NULL,
     plot.type = c("marginal", "coherency", "phase"),
     ...
)

plot.spec.phase(x, ci = 0.95,
    xlab = "frequency", ylab = "phase",
    ylim = c(-pi, pi), type = "l",
    main = NULL, ci.col = "blue", ci.lty = 3, ...
)

plot.spec.coherency(x, ci = 0.95,
    xlab = "frequency",
    ylab = "squared coherency",
    ylim = c(0, 1), type = "l",
    main = NULL, ci.col = "blue", ci.lty = 3, ...
)

Arguments

x an object of class "spec".
add logical. If TRUE, add to already existing plot. Only valid for plot.type = "marginal".
ci

coverage probability for confidence interval. Plotting of the confidence bar/limits is omitted unless ci is strictly positive.

log

If "dB", plot on log10 (decibel) scale (as S-PLUS), otherwise use conventional log scale or linear scale. Logical values are also accepted. The default is "yes" unless options(ts.S.compat = TRUE) has been set, when it is "dB". Only valid for plot.type = "marginal".

xlab

the x label of the plot.

ylab

the y label of the plot. If missing a suitable label will be constructed.

type

the type of plot to be drawn, defaults to lines.

ci.col

colour for plotting confidence bar or confidence intervals for coherency and phase.

ci.lty

line type for confidence intervals for coherency and phase.

main

overall title for the plot. If missing, a suitable title is constructed.

sub

a sub title for the plot. Only used for plot.type = "marginal". If missing, a description of the smoothing is used.

plot.type

For multivariate time series, the type of plot required. Only the first character is needed.

ylim, ...

Graphical parameters.

See Also

spectrum

plot.stepfun

Plot Step Functions

Description

Method of the generic plot for stepfun objects and utility for plotting piecewise constant functions.

Usage

## S3 method for class 'stepfun':
plot(x, xval, xlim, ylim,
    xlab = "x", ylab = "f(x)", main = NULL,
    add = FALSE, verticals = TRUE, do.points = TRUE,
    pch = par("pch"),
    col.points = par("col"), cex.points = par("cex"),
    col.hor = par("col"), col.vert = par("col"),
    lty = par("lty"), lwd = par("lwd"), ...)

## S3 method for class 'stepfun':
lines(x, ...)
plot.stepfun

Arguments

- **x**: an R object inheriting from "stepfun".
- **xval**: numeric vector of abscissa values at which to evaluate x. Defaults to knots(x) restricted to xlim.
- **xlim, ylim**: numeric(2) each; range of x or y values to use. Both have sensible defaults.
- **xlab, ylab**: labels of x and y axis.
- **main**: main title.
- **add**: logical; if TRUE only add to an existing plot.
- **verticals**: logical; if TRUE, draw vertical lines at steps.
- **do.points**: logical; if TRUE, also draw points at the (xlim restricted) knot locations.
- **pch**: character; point character if do.points.
- **col.points**: character or integer code; color of points if do.points.
- **cex.points**: numeric; character expansion factor if do.points.
- **col.hor**: color of horizontal lines.
- **col.vert**: color of vertical lines.
- **lty, lwd**: line type and thickness for all lines.
- **...**: further arguments of plot(.), or if (add) segments(.).

Value

A list with two components

- **t**: abscissa (x) values, including the two outermost ones.
- **y**: y values ‘in between’ the t[].

Author(s)


See Also

dcdf for empirical distribution functions as special step functions, approxfun and splinefun.

Examples

```r
y0 <- c(1,2,4,3)
sfun0 <- stepfun(1:3, y0, f = 0)
sfun.2 <- stepfun(1:3, y0, f = .2)
sfun1 <- stepfun(1:3, y0, right = TRUE)

tt <- seq(0,3, by=0.1)
op <- par(mfrow=c(2,2))
plot(sfun0); plot(sfun0, xval=tt, add=TRUE, col.h="bisque")
plot(sfun.2);plot(sfun.2,xval=tt, add=TRUE, col.h="orange")
plot(sfun1);lines(sfun1, xval=tt, col.h="coral")
##-- This is revealing :
plot(sfun0, verticals= FALSE,
main = "stepfun(x, y0, f=f) for f = 0, .2, 1")
for(i in 1:3)
lines(list(sfun0,sfun.2,stepfun(1:3,y0,f = i))[[i]], col.h=i, col.v=i)
```
legend(2.5, 1.9, paste("f =", c(0,0.2,1)), col=1:3, lty=1, y.inter=1); par(op)

# Extend and/or restrict 'viewport':
plot(sfun0, xlim = c(0,5), ylim = c(0, 3.5),
     main = "plot(stepfun(*), xlim= . , ylim = .)"
)
##-- this works too (automatic call to ecdf(.)):
plot.stepfun(rt(50, df=3), col.vert = "gray20")

---

plot.ts

Plotting Time-Series Objects

Description

Plotting method for objects inheriting from class "ts".

Usage

## S3 method for class 'ts':
plot(x, y = NULL, plot.type = c("multiple", "single"),
     xy.labels, xy.lines, panel = lines, nc, yax.flip = FALSE,
     mar.multi = c(0, 5.1, 0, if(yax.flip) 5.1 else 2.1),
     oma.multi = c(6, 0, 5, 0), axes = TRUE, ...)

## S3 method for class 'ts':
lines(x, ...)

Arguments

x, y
time series objects, usually inheriting from class "ts".

plot.typefor multivariate time series, should the series be plotted separately (with a common time axis) or on a single plot?

xy.labelslogical, indicating if text() labels should be used for an x-y plot, or character, supplying a vector of labels to be used. The default is to label for up to 150 points, and not for more.

xy.lineslogical, indicating if lines should be drawn for an x-y plot. Defaults to the value of xy.labels if that is logical, otherwise to TRUE.

panela function(x, col, bg, pch, type, ...) which gives the action to be carried out in each panel of the display for plot.type="multiple". The default is lines.

ncthe number of columns to use when type="multiple". Defaults to 1 for up to 4 series, otherwise to 2.

yax.fliplogical indicating if the y-axis (ticks and numbering) should flip from side 2 (left) to 4 (right) from series to series when type="multiple".

mar.multi, oma.multi
the (default) par settings for plot.type="multiple". Modify with care!

axeslogical indicating if x- and y- axes should be drawn.

...additional graphical arguments, see plot, plot.default and par.
Details
If y is missing, this function creates a time series plot, for multivariate series of one of two kinds depending on plot.type.
If y is present, both x and y must be univariate, and a “scatter” plot y ~ x will be drawn, enhanced by using text if xy.labels is TRUE or character, and lines if xy.lines is TRUE.

See Also
ts for basic time series construction and access functionality.

Examples
## Multivariate
z <- ts(matrix(rt(200 * 8, df = 3), 200, 8), start=c(1961, 1), frequency=12)
plot(z, yax.flip = TRUE)

## A phase plot:
plot(nhtemp, c(nhtemp[-1], NA), cex = .8, col="blue",
     main = "Lag plot of New Haven temperatures")
## a clearer way to do this would be
## Not run:
plot(nhtemp, lag(nhtemp, 1), cex = .8, col="blue",
     main = "Lag plot of New Haven temperatures")
## End(Not run)

## xy.lines and xy.labels are FALSE for large series:
plot(lag(sunspots, 1), sunspots, pch = ".")

SMI <- EuStockMarkets[, "SMI"]
plot(lag(SMI, 1), SMI, pch = ".")
plot(lag(SMI, 20), SMI, pch = ".", log = "xy",
     main = "4 weeks lagged SMI stocks -- log scale", xy.lines= TRUE)

Poisson

The Poisson Distribution

Description
Density, distribution function, quantile function and random generation for the Poisson distribution with parameter lambda.

Usage
dpois(x, lambda, log = FALSE)
ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)
rpois(n, lambda)
Arguments

- **x**: vector of (non-negative integer) quantiles.
- **q**: vector of quantiles.
- **p**: vector of probabilities.
- **n**: number of random values to return.
- **lambda**: vector of positive means.
- **log**, **log.p**: logical; if TRUE, probabilities **p** are given as log(p).
- **lower.tail**: logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

The Poisson distribution has density

\[
p(x) = \frac{\lambda^x e^{-\lambda}}{x!}
\]

for \( x = 0, 1, 2, \ldots \). The mean and variance are \( E(X) = Var(X) = \lambda \).

If an element of **x** is not integer, the result of **dpois** is zero, with a warning. \( p(x) \) is computed using Loader’s algorithm, see the reference in **dbinom**.

The quantile is left continuous: \( qgeom(q, \text{prob}) \) is the largest integer \( x \) such that \( P(X \leq x) < q \).

Setting **lower.tail** = FALSE allows to get much more precise results when the default, **lower.tail** = TRUE would return 1, see the example below.

Value

**dpois** gives the (log) density, **ppois** gives the (log) distribution function, **qpois** gives the quantile function, and **rpois** generates random deviates.

See Also

**dbinom** for the binomial and **dnbinom** for the negative binomial distribution.

Examples

- \(-\log(dpois(0:7, \text{lambda}=1) * \text{gamma}(1+ 0:7)) \approx 1\)
- \(\text{Ni} \leftarrow \text{rpois}(50, \text{lam}= 4); \text{table(factor(Ni, 0:max(Ni)))}\)
- \(1 - \text{ppois}(10*(15:25), \text{lambda}=100) \quad \# \text{becomes 0 (cancellation)}\)
- \(\text{ppois}(10*(15:25), \text{lambda}=100, \text{lower}=\text{FALSE}) \quad \# \text{no cancellation}\)

\[
\text{par(mfrow = c(2, 1))}\]
\[
\text{x} \leftarrow \text{seq(-0.01, 5, 0.01)}\]
\[
\text{plot(x, ppois(x, 1), type="s", ylab="F(x)", main="Poisson(1) CDF")}\]
\[
\text{plot(x, pbinom(x, 100, 0.01),type="s", ylab="F(x)",}\]
\[
\text{main="Binomial(100, 0.01) CDF")}\]

power.t.test

Power calculations for one and two sample t tests

Description

Compute power of test, or determine parameters to obtain target power.

Usage

power.t.test(n = NULL, delta = NULL, sd = 1, sig.level = 0.05, power = NULL, type = c("two.sample", "one.sample", "paired"), alternative = c("two.sided", "one.sided"), strict = FALSE)

Arguments

n Number of observations (per group)
delta True difference in means
sd Standard deviation
sig.level Significance level (Type I error probability)
power Power of test (1 minus Type II error probability)
type Type of t test
alternative One- or two-sided test
strict Use strict interpretation in two-sided case

Details

Exactly one of the parameters n, delta, power, sd, and sig.level must be passed as NULL, and that parameter is determined from the others. Notice that the last two have non-NULL defaults so NULL must be explicitly passed if you want to compute them.

If strict = TRUE is used, the power will include the probability of rejection in the opposite direction of the true effect, in the two-sided case. Without this the power will be half the significance level if the true difference is zero.

Value

Object of class "power.htest", a list of the arguments (including the computed one) augmented with method and note elements.

Note

uniroot is used to solve power equation for unknowns, so you may see errors from it, notably about inability to bracket the root when invalid arguments are given.

Author(s)

Peter Dalgaard. Based on previous work by Claus Ekstrøm
See Also
t.test, unroot

Examples

```r
power.t.test(n = 20, delta = 1)
power.t.test(power = .90, delta = 1)
power.t.test(power = .90, delta = 1, alt = "one.sided")
```

---

power

Create a Power Link Object

Description

Creates a link object based on the link function $\eta = \mu^\lambda$.

Usage

```r
power(lambda = 1)
```

Arguments

- `lambda`: a real number.

Details

If `lambda` is non-negative, it is taken as zero, and the log link is obtained. The default $\lambda = 1$ gives the identity link.

Value

A list with components `linkfun`, `linkinv`, `mu.eta`, and `valideta`. See `make.link` for information on their meaning.

References


See Also

`make.link.family`

To raise a number to a power, see `Arithmetic`.

To calculate the power of a test, see various functions in the `stats` package, e.g., `power.t.test`.

Examples

```r
power()
quasi(link=power(1/3))[[c("linkfun", "linkinv")]]
```
**Description**

Computes the Phillips-Perron test for the null hypothesis that x has a unit root against a stationary alternative.

**Usage**

```r
PP.test(x, lshort = TRUE)
```

**Arguments**

- `x` a numeric vector or univariate time series.
- `lshort` a logical indicating whether the short or long version of the truncation lag parameter is used.

**Details**

The general regression equation which incorporates a constant and a linear trend is used and the corrected t-statistic for a first order autoregressive coefficient equals one is computed. To estimate \( \sigma^2 \) the Newey-West estimator is used. If `lshort` is `TRUE`, then the truncation lag parameter is set to `trunc(4 * (n/100)^0.25)`, otherwise `trunc(12 * (n/100)^0.25)` is used. The p-values are interpolated from Table 4.2, page 103 of Banerjee et al. (1993).

Missing values are not handled.

**Value**

A list with class "htest" containing the following components:

- `statistic` the value of the test statistic.
- `parameter` the truncation lag parameter.
- `p.value` the p-value of the test.
- `method` a character string indicating what type of test was performed.
- `data.name` a character string giving the name of the data.

**Author(s)**

A. Trapletti

**References**


ppoints

 Ordinates for Probability Plotting

Examples

```r
x <- rnorm(1000)
PP.test(x)
y <- cumsum(x) # has unit root
PP.test(y)
```

ppoints  Ordinates for Probability Plotting

Description

Generates the sequence of “probability” points \((1:m - a)/(m + (1-a)-a)\) where \(m\) is either \(n\), if \(\text{length}(n)==1\), or \(\text{length}(n)\).

Usage

```r
ppoints(n, a = ifelse(n <= 10, 3/8, 1/2))
```

Arguments

- **n**: either the number of points generated or a vector of observations.
- **a**: the offset fraction to be used; typically in \((0, 1)\).

Details

If \(0 < a < 1\), the resulting values are within \((0, 1)\) (excluding boundaries). In any case, the resulting sequence is symmetric in \([0, 1]\), i.e., \(p + \text{rev}(p) == 1\).

ppoints() is used in `qqplot` and `qqnorm` to generate the set of probabilities at which to evaluate the inverse distribution.

References


See Also

`qqplot`, `qqnorm`.

Examples

```r
ppoints(4) # the same as ppoints(1:4)
ppoints(10)
ppoints(10, a=1/2)
```
**Description**

Fit a projection pursuit regression model.

**Usage**

```r
ppr(x, ...)  
## S3 method for class 'formula':  
ppr(formula, data, weights, subset, na.action,  
    contrasts = NULL, ..., model = FALSE)  
## Default S3 method:  
ppr(x, y, weights = rep(1,n),  
    ww = rep(1,q), nterms, max.terms = nterms, optlevel = 2,  
    sm.method = c("supsmu", "spline", "gcvspline"),  
    bass = 0, span = 0, df = 5, gcvpen = 1, ...)  
```

**Arguments**

- `formula` a formula specifying one or more numeric response variables and the explanatory variables.
- `x` numeric matrix of explanatory variables. Rows represent observations, and columns represent variables. Missing values are not accepted.
- `y` numeric matrix of response variables. Rows represent observations, and columns represent variables. Missing values are not accepted.
- `nterms` number of terms to include in the final model.
- `data` data frame from which variables specified in `formula` are preferentially to be taken.
- `weights` a vector of weights \( w_i \) for each case.
- `ww` a vector of weights for each response, so the fit criterion is the sum over case \( i \) and responses \( j \) of \( w_i \) \( w_j \) \((y_{ij} - \text{fit}_{ij})^2\) divided by the sum of \( w_i \).
- `subset` an index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
- `na.action` a function to specify the action to be taken if \( \text{NAs} \) are found. The default action is given by `getOption("na.action")`. (NOTE: If given, this argument must be named.)
- `contrasts` the contrasts to be used when any factor explanatory variables are coded.
- `max.terms` maximum number of terms to choose from when building the model.
- `optlevel` integer from 0 to 3 which determines the thoroughness of an optimization routine in the SMART program. See the **Details** section.
sm.method the method used for smoothing the ridge functions. The default is to use Friedman's super smoother supsmu. The alternatives are to use the smoothing spline code underlying smooth.spline, either with a specified (equivalent) degrees of freedom for each ridge functions, or to allow the smoothness to be chosen by GCV.

bass super smoother bass tone control used with automatic span selection (see supsmu); the range of values is 0 to 10, with larger values resulting in increased smoothing.

span super smoother span control (see supsmu). The default, 0, results in automatic span selection by local cross validation. span can also take a value in \(0, 1\].

df if \textbf{sm.method} is "spline" specifies the smoothness of each ridge term via the requested equivalent degrees of freedom.

gcvpen if \textbf{sm.method} is "gcvspline" this is the penalty used in the GCV selection for each degree of freedom used.

... arguments to be passed to or from other methods.

model logical. If true, the model frame is returned.

Details

The basic method is given by Friedman (1984), and is essentially the same code used by S-PLUS’s ppreg. This code is extremely sensitive to the compiler used.

The algorithm first adds up to max.terms ridge terms one at a time; it will use less if it is unable to find a term to add that makes sufficient difference. It then removes the least "important" term at each step until nterms terms are left.

The levels of optimization (argument \textbf{optlevel}) differ in how thoroughly the models are refitted during this process. At level 0 the existing ridge terms are not refitted. At level 1 the projection directions are not refitted, but the ridge functions and the regression coefficients are. Levels 2 and 3 refit all the terms and are equivalent for one response; level 3 is more careful to re-balance the contributions from each regressor at each step and so is a little less likely to converge to a saddle point of the sum of squares criterion.

Value

A list with the following components, many of which are for use by the method functions.

call the matched call

p the number of explanatory variables (after any coding)

q the number of response variables

mu the argument nterms

ml the argument max.terms

gof the overall residual (weighted) sum of squares for the selected model

gofn the overall residual (weighted) sum of squares against the number of terms, up to max.terms. Will be invalid (and zero) for less than nterms.

df the argument df

edf if \textbf{sm.method} is "spline" or "gcvspline" the equivalent number of degrees of freedom for each ridge term used.

xnames the names of the explanatory variables

ynames the names of the response variables
alpha  a matrix of the projection directions, with a column for each ridge term
beta   a matrix of the coefficients applied for each response to the ridge terms: the rows are the responses and the columns the ridge terms
yb     the weighted means of each response
ys     the overall scale factor used: internally the responses are divided by ys to have unit total weighted sum of squares.
fitted.values the fitted values, as a matrix if q > 1.
residuals the residuals, as a matrix if q > 1.
smod   internal work array, which includes the ridge functions evaluated at the training set points.
model  (only if model=TRUE) the model frame.

References


See Also

plot.ppr, supsmu, smooth.spline

Examples

# Note: your numerical values may differ
attach(rock)
areal <- area/10000; peril <- peri/10000
rock.ppr <- ppr(log(perm) ~ areal + peril + shape,
               data = rock, nterms = 2, max.terms = 5)
rock.ppr
# Call:
# ppr.formula(formula = log(perm) ~ areal + peril + shape, data = rock,
#              nterms = 2, max.terms = 5)
# # Goodness of fit:
# # 2 terms 3 terms 4 terms 5 terms
# # 8.737806 5.289517 4.745799 4.490378

summary(rock.ppr)
# .... (same as above)
# ....

# Projection direction vectors:
# term 1  term 2
# areal  0.34357179  0.37071027
# peril -0.93781471 -0.61923542
# shape  0.04961846  0.69218595

# Coefficients of ridge terms:
# term 1  term 2
prcomp

Principal Components Analysis

Description

Performs a principal components analysis on the given data matrix and returns the results as an object of class prcomp.

Usage

prcomp(x, ...)

## S3 method for class 'formula':
prcomp(formula, data = NULL, subset, na.action, ...)

## Default S3 method:
prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE, tol = NULL, ...)

## S3 method for class 'prcomp':
predict(object, newdata, ...)

Arguments

formula a formula with no response variable.
data an optional data frame containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset an optional vector used to select rows (observations) of the data matrix x.
na.action a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The “factory-fresh” default is na.omit.
... arguments passed to or from other methods. If x is a formula one might specify scale. or tol.
x a numeric or complex matrix (or data frame) which provides the data for the principal components analysis.
retx a logical value indicating whether the rotated variables should be returned.
center a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.
scale.

A logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of \( x \) can be supplied. The value is passed to scale.

tol

A value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to \( \text{tol} \) times the standard deviation of the first component.) With the default null setting, no components are omitted. Other settings for \( \text{tol} \) could be \( \text{tol} = 0 \) or \( \text{tol} = \sqrt{\text{.Machine}\$\text{double}\$.\text{eps}} \), which would omit essentially constant components.

object

Object of class inheriting from \("prcomp"\)

newdata

An optional data frame or matrix in which to look for variables with which to predict. If omitted, the scores are used. If the original fit used a formula or a data frame or a matrix with column names, newdata must contain columns with the same names. Otherwise it must contain the same number of columns, to be used in the same order.

Details

The calculation is done by a singular value decomposition of the (centered and possibly scaled) data matrix, not by using eigen on the covariance matrix. This is generally the preferred method for numerical accuracy. The print method for the these objects prints the results in a nice format and the plot method produces a scree plot.

Value

prcomp returns a list with class \("prcomp"\) containing the following components:

sdev

The standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix).

rotation

The matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors). The function princomp returns this in the element loadings.

x

If retx is true the value of the rotated data (the centred (and scaled if requested) data multiplied by the rotation matrix) is returned. Hence, \( \text{cov}(x) \) is the diagonal matrix \( \text{diag}(\text{sdev}^2) \).

center, scale

The centering and scaling used, or FALSE.

Note

The signs of the columns of the rotation matrix are arbitrary, and so may differ between different programs for PCA, and even between different builds of R.

References


See Also

biplot.prcomp, princomp.cor, cov, svd, eigen.

Examples

```r
## the variances of the variables in the
## USArrests data vary by orders of magnitude, so scaling is appropriate
prcomp(USArrests)  # inappropriate
prcomp(USArrests, scale = TRUE)
prcomp(~ Murder + Assault + Rape, data = USArrests, scale = TRUE)
plot(prcomp(USArrests))
summary(prcomp(USArrests, scale = TRUE))
biplot(prcomp(USArrests, scale = TRUE))
```

Description

`predict` is a generic function for predictions from the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

Usage

```r
predict (object, ...)  
```

Arguments

- `object` a model object for which prediction is desired.
- `...` additional arguments affecting the predictions produced.

Details

Most prediction methods which similar to fitting linear models have an argument `newdata` specifying the first place to look for explanatory variables to be used for prediction. Some considerable attempts are made to match up the columns in `newdata` to those used for fitting, for example that they are of comparable types and that any factors have the same level set in the same order (or can be transformed to be so).

Time series prediction methods in package `stats` have an argument `n.ahead` specifying how many time steps ahead to predict.

Many methods have a logical argument `se.fit` saying if standard errors are to returned.

Value

The form of the value returned by `predict` depends on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

References

predict.Arima

See Also

Examples
## All the "predict" methods found
## NB most of the methods in the standard packages are hidden.
for(fn in methods("predict"))
  try({
    f <- eval(substitute(getAnywhere(fn)$objs[[1]]), list(fn = fn))
    cat(fn, ":
	", deparse(args(f)), "\n"), silent = TRUE)

predict.Arima Forecast from ARIMA fits

Description
Forecast from models fitted by arima.

Usage
## S3 method for class 'Arima':
predict(object, n.ahead = 1, newxreg = NULL,
    se.fit = TRUE, ...)

Arguments

object The result of an arima fit.
n.ahead The number of steps ahead for which prediction is required.
newxreg New values of xreg to be used for prediction. Must have at least n.ahead rows.
se.fit Logical: should standard errors of prediction be returned?
... arguments passed to or from other methods.

Details
Finite-history prediction is used, via KalmanForecast. This is only statistically efficient if the MA part of the fit is invertible, so predict.Arima will give a warning for non-invertible MA models.

The standard errors of prediction exclude the uncertainty in the estimation of the ARMA model and the regression coefficients. According to Harvey (1993, pp. 58–9) the effect is small.

Value
A time series of predictions, or if se.fit = TRUE, a list with components pred, the predictions, and se, the estimated standard errors. Both components are time series.
References


See Also

arima

Examples

predict(arima(lh, order = c(3,0,0)), n.ahead = 12)

(fit <- arima(USAccDeaths, order = c(0,1,1),
seasonal = list(order=c(0,1,1))))
predict(fit, n.ahead = 6)

predict.glm

Predict Method for GLM Fits

Description

Obtains predictions and optionally estimates standard errors of those predictions from a fitted generalized linear model object.

Usage

## S3 method for class 'glm':
predict(object, newdata = NULL,
type = c("link", "response", "terms"),
se.fit = FALSE, dispersion = NULL, terms = NULL,
na.action = na.pass, ...)

Arguments

object
newdata
type
se.fit
dispersion

a fitted object of class inheriting from "glm".
optionally, a data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
the type of prediction required. The default is on the scale of the linear predictors; the alternative "response" is on the scale of the response variable. Thus for a default binomial model the default predictions are of log-odds (probabilities on logit scale) and type = "response" gives the predicted probabilities. The "terms" option returns a matrix giving the fitted values of each term in the model formula on the linear predictor scale. The value of this argument can be abbreviated.
logical switch indicating if standard errors are required.
the dispersion of the GLM fit to be assumed in computing the standard errors. If omitted, that returned by summary applied to the object is used.
terms with type="terms" by default all terms are returned. A character vector specifies which terms are to be returned.

na.action function determining what should be done with missing values in newdata. The default is to predict NA.

... further arguments passed to or from other methods.

Value

If se = FALSE, a vector or matrix of predictions. If se = TRUE, a list with components

- fit Predictions
- se.fit Estimated standard errors
- residual.scale A scalar giving the square root of the dispersion used in computing the standard errors.

Note

Variables are first looked for in newdata and then searched for in the usual way (which will include the environment of the formula used in the fit). As from R 2.0.0 a warning will be given if the variables found are not of the same length as those in newdata if it was supplied.

See Also

glm, SafePrediction

Examples

```r
## example from Venables and Ripley (2002, pp. 190-2.)
ldose <- rep(0:5, 2)
numdead <- c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16)
sex <- factor(rep(c("M", "F"), c(6, 6)))
SF <- cbind(numdead, numalive=20-numdead)
budworm.lg <- glm(SF ~ sex*ldose, family=binomial)
summary(budworm.lg)

plot(c(1,32), c(0,1), type = "n", xlab = "dose", ylab = "prob", log = "x")
text(2^ldose, numdead/20, as.character(sex))
ld <- seq(0, 5, 0.1)
lines(2^ld, predict(budworm.lg, data.frame(ldose=ld, sex=factor(rep("M", length(ld))), levels=levels(sex)), type = "response"))
lines(2^ld, predict(budworm.lg, data.frame(ldose=ld, sex=factor(rep("F", length(ld))), levels=levels(sex)), type = "response"))
```
predict.HoltWinters

prediction function for fitted Holt-Winters models

Description

Computes predictions and prediction intervals for models fitted by the Holt-Winters method.

Usage

### S3 method for class 'HoltWinters':

predict(object, n.ahead=1, prediction.interval = FALSE, level = 0.95, ...)

Arguments

- **object**: An object of class *HoltWinters*.
- **n.ahead**: Number of future periods to predict.
- **prediction.interval**: logical. If TRUE, the lower and upper bounds of the corresponding prediction intervals are computed.
- **level**: Confidence level for the prediction interval.
- **...**: arguments passed to or from other methods.

Value

A time series of the predicted values. If prediction intervals are requested, a multiple time series is returned with columns *fit*, *lwr* and *upr* for the predicted values and the lower and upper bounds respectively.

Author(s)

David Meyer (David.Meyer@wu-wien.ac.at)

References


See Also

*HoltWinters*

Examples

```r
m <- HoltWinters(co2)
p <- predict(m, 50, prediction.interval = TRUE)
plot(m, p)
```
predict.lm

Predict method for Linear Model Fits

Description

Predicted values based on linear model object

Usage

```r
## S3 method for class 'lm':
predict(object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
        interval = c("none", "confidence", "prediction"),
        level = 0.95, type = c("response", "terms"),
        terms = NULL, na.action = na.pass, ...)
```

Arguments

- `object`: Object of class inheriting from "lm"
- `newdata`: An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
- `se.fit`: A switch indicating if standard errors are required.
- `scale`: Scale parameter for std.err. calculation
- `df`: Degrees of freedom for scale
- `interval`: Type of interval calculation
- `level`: Tolerance/confidence level
- `type`: Type of prediction (response or model term)
- `terms`: If type="terms", which terms (default is all terms)
- `na.action`: function determining what should be done with missing values in newdata. The default is to predict NA.
- `...`: further arguments passed to or from other methods.

Details

`predict.lm` produces predicted values, obtained by evaluating the regression function in the frame `newdata` (which defaults to `model.frame(object)`). If the logical `se.fit` is TRUE, standard errors of the predictions are calculated. If the numeric argument `scale` is set (with optional `df`), it is used as the residual standard deviation in the computation of the standard errors, otherwise this is extracted from the model fit. Setting `interval` specifies computation of confidence or prediction (tolerance) intervals at the specified `level`, sometimes referred to as narrow vs. wide intervals.

If the fit is rank-deficient, some of the columns of the design matrix will have been dropped. Prediction from such a fit only makes sense if `newdata` is contained in the same subspace as the original data. That cannot be checked accurately, so a warning is issued.
predict.loess

Value

predict.lm produces a vector of predictions or a matrix of predictions and bounds with column names fit, lwr, and upr if interval is set. If se.fit is TRUE, a list with the following components is returned:

- **fit**: vector or matrix as above
- **se.fit**: standard error of predicted means
- **residual.scale**: residual standard deviations
- **df**: degrees of freedom for residual

Note

Variables are first looked for in newdata and then searched for in the usual way (which will include the environment of the formula used in the fit). As from R 2.0.0 a warning will be given if the variables found are not of the same length as those in newdata if it was supplied.

Offsets specified by offset in the fit by lm will not be included in predictions, whereas those specified by an offset term in the formula will be.

See Also

The model fitting function lm, predict, SafePrediction

Examples

```r
## Predictions
x <- rnorm(15)
y <- x + rnorm(15)
predict(lm(y ~ x))
new <- data.frame(x = seq(-3, 3, 0.5))
predict(lm(y ~ x), new, se.fit = TRUE)
pred.w.plim <- predict(lm(y ~ x), new, interval="prediction")
pred.w.clim <- predict(lm(y ~ x), new, interval="confidence")
matplot(new$x,cbind(pred.w.clim, pred.w.plim[,2:1]),
        lty=c(1,2,2,3,3), type="l", ylab="predicted y")
```

predict.loess  
*Predict Loess Curve or Surface*

Description

Predictions from a loess fit, optionally with standard errors.

Usage

```r
## S3 method for class 'loess'
predict(object, newdata = NULL, se = FALSE, ...)
```
**predict.loess**

**Arguments**

- `object`: an object fitted by `loess`.
- `newdata`: an optional data frame in which to look for variables with which to predict. If missing, the original data points are used.
- `se`: should standard errors be computed?
- `...`: arguments passed to or from other methods.

**Details**

The standard errors calculation is slower than prediction.

When the fit was made using `surface="interpolate"` (the default), `predict.loess` will not extrapolate – so points outside an axis-aligned hypercube enclosing the original data will have missing (NA) predictions and standard errors.

**Value**

If `se = FALSE`, a vector giving the prediction for each row of `newdata` (or the original data). If `se = TRUE`, a list containing components

- `fit`: the predicted values.
- `se`: an estimated standard error for each predicted value.
- `residual.scale`: the estimated scale of the residuals used in computing the standard errors.
- `df`: an estimate of the effective degrees of freedom used in estimating the residual scale, intended for use with t-based confidence intervals.

If `newdata` was the result of a call to `expand.grid`, the predictions (and s.e.’s if requested) will be an array of the appropriate dimensions.

**Note**

Variables are first looked for in `newdata` and then searched for in the usual way (which will include the environment of the formula used in the fit). As from R 2.0.0 a warning will be given if the variables found are not of the same length as those in `newdata` if it was supplied.

**Author(s)**

B.D. Ripley, based on the `cloess` package of Cleveland, Grosse and Shyu.

**See Also**

- `loess`

**Examples**

```r
  cars.lo <- loess(dist ~ speed, cars)
predict(cars.lo, data.frame(speed=seq(5, 30, 1)), se=TRUE)
  # to get extrapolation
cars.lo2 <- loess(dist ~ speed, cars,
                 control=loess.control(surface="direct"))
predict(cars.lo2, data.frame(speed=seq(5, 30, 1)), se=TRUE)
```
Description

`predict.nls` produces predicted values, obtained by evaluating the regression function in the frame `newdata`. If the logical `se.fit` is TRUE, standard errors of the predictions are calculated. If the numeric argument `scale` is set (with optional `df`), it is used as the residual standard deviation in the computation of the standard errors, otherwise this is extracted from the model fit. Setting `intervals` specifies computation of confidence or prediction (tolerance) intervals at the specified `level`.

At present `se.fit` and `interval` are ignored.

Usage

```r
## S3 method for class 'nls':
predict(object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
        interval = c("none", "confidence", "prediction"),
        level = 0.95, ...)
```

Arguments

- `object` An object that inherits from class `nls`.
- `newdata` A named list or data frame in which to look for variables with which to predict. If `newdata` is missing the fitted values at the original data points are returned.
- `se.fit` A logical value indicating if the standard errors of the predictions should be calculated. Defaults to FALSE. At present this argument is ignored.
- `scale` A numeric scalar. If it is set (with optional `df`), it is used as the residual standard deviation in the computation of the standard errors, otherwise this information is extracted from the model fit. At present this argument is ignored.
- `df` A positive numeric scalar giving the number of degrees of freedom for the scale estimate. At present this argument is ignored.
- `interval` A character string indicating if prediction intervals or a confidence interval on the mean responses are to be calculated. At present this argument is ignored.
- `level` A numeric scalar between 0 and 1 giving the confidence level for the intervals (if any) to be calculated. At present this argument is ignored.
- `...` Additional optional arguments. At present no optional arguments are used.

Value

`predict.nls` produces a vector of predictions. When implemented, `interval` will produce a matrix of predictions and bounds with column names `fit`, `lwr`, and `upr`. When implemented, if `se.fit` is TRUE, a list with the following components will be returned:

- `fit` vector or matrix as above
- `se.fit` standard error of predictions
- `residual.scale` residual standard deviations
- `df` degrees of freedom for residual
predict.smooth.spline

Note
Variables are first looked for in newdata and then searched for in the usual way (which will include the environment of the formula used in the fit). As from R 2.0.0 a warning will be given if the variables found are not of the same length as those in newdata if it was supplied.

See Also
The model fitting function nls.predict.

Examples

```r
fm <- nls(demand ~ SSasympOrig(Time, A, lrc), data = BOD)
predict(fm)  # fitted values at observed times
## Form data plot and smooth line for the predictions
opar <- par(las = 1)
plot(demand ~ Time, data = BOD, col = 4,
     main = "BOD data and fitted first-order curve",
     xlim = c(0,7), ylim = c(0, 20) )
tt <- seq(0, 8, length = 101)
lines(tt, predict(fm, list(Time = tt)))
par(opar)
```

predict.smooth.spline

Predict from Smoothing Spline Fit

Description
Predict a smoothing spline fit at new points, return the derivative if desired. The predicted fit is linear beyond the original data.

Usage

```r
## S3 method for class 'smooth.spline':
predict(object, x, deriv = 0, ...)
```

Arguments

- `object`: a fit from `smooth.spline`
- `x`: the new values of `x`
- `deriv`: integer; the order of the derivative required.
- `...`: further arguments passed to or from other methods.

Value

A list with components

- `x`: The input `x`
- `y`: The fitted values or derivatives at `x`
preplot

Pre-computations for a Plotting Object

Description

Compute an object to be used for plots relating to the given model object.

Usage

preplot(object, ...)

Arguments

object a fitted model object.

... additional arguments for specific methods.
Details

Only the generic function is currently provided in base R, but some add-on packages have methods. Principally here for S compatibility.

Value

An object set up to make a plot that describes object.

Description

princomp performs a principal components analysis on the given numeric data matrix and returns the results as an object of class princomp.

Usage

princomp(x, ...)

## S3 method for class 'formula':
princomp(formula, data = NULL, subset, na.action, ...)

## Default S3 method:
princomp(x, cor = FALSE, scores = TRUE, covmat = NULL,
subset = rep(TRUE, nrow(as.matrix(x))), ...)

## S3 method for class 'princomp':
predict(object, newdata, ...)

Arguments

formula a formula with no response variable.
data an optional data frame containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset an optional vector used to select rows (observations) of the data matrix x.
na.action a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The “factory-fresh” default is na.omit.
x a numeric matrix or data frame which provides the data for the principal components analysis.
cor a logical value indicating whether the calculation should use the correlation matrix or the covariance matrix.
scores a logical value indicating whether the score on each principal component should be calculated.
covmat a covariance matrix, or a covariance list as returned by cov.wt (and cov.mve or cov.mcd from package MASS). If supplied, this is used rather than the covariance matrix of x.
... arguments passed to or from other methods. If \( x \) is a formula one might specify \cor\ or \scores\.

**object**  
Object of class inheriting from "princomp"

**newdata**  
An optional data frame or matrix in which to look for variables with which to predict. If omitted, the scores are used. If the original fit used a formula or a data frame or a matrix with column names, \newdata\ must contain columns with the same names. Otherwise it must contain the same number of columns, to be used in the same order.

**Details**

`princomp` is a generic function with "formula" and "default" methods.

The calculation is done using `eigen` on the correlation or covariance matrix, as determined by `cor`. This is done for compatibility with the S-PLUS result. A preferred method of calculation is to use `svd` on \( x \), as is done in `prcomp`.

Note that the default calculation uses divisor \( N \) for the covariance matrix.

The `print` method for the these objects prints the results in a nice format and the `plot` method produces a scree plot (`screeplot`). There is also a `biplot` method.

If \( x \) is a formula then the standard NA-handling is applied to the scores (if requested): see `napredict`.

`princomp` only handles so-called R-mode PCA, that is feature extraction of variables. If a data matrix is supplied (possibly via a formula) it is required that there are at least as many units as variables. For Q-mode PCA use `prcomp`.

**Value**

`princomp` returns a list with class "princomp" containing the following components:

**sdev**  
the standard deviations of the principal components.

**loadings**  
the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors). This is of class "loadings": see `loadings` for its `print` method.

**center**  
the means that were subtracted.

**scale**  
the scalings applied to each variable.

**n.obs**  
the number of observations.

**scores**  
if `scores = TRUE`, the scores of the supplied data on the principal components. These are non-null only if \( x \) was supplied, and if `covmat` was also supplied if it was a covariance list.

**call**  
the matched call.

**na.action**  
If relevant.

**Note**

The signs of the columns of the loadings and scores are arbitrary, and so may differ between different programs for PCA, and even between different builds of \( R \).

**References**


See Also

summary.princomp, screeplot, biplot.princomp, prcomp, cor, cov, eigen.

Examples

## The variances of the variables in the
## USArrests data vary by orders of magnitude, so scaling is appropriate
(pc.cr <- princomp(USArrests))   # inappropriate
princomp(USArrests, cor = TRUE) # == prcomp(USArrests, scale=TRUE)
## Similar, but different:
## The standard deviations differ by a factor of sqrt(49/50)

summary(pc.cr <- princomp(USArrests, cor = TRUE))
loadings(pc.cr)  ## note that blank entries are small but not zero
plot(pc.cr)     # shows a screeplot.
biplot(pc.cr)

## Formula interface
princomp(~ ., data = USArrests, cor = TRUE)
# NA-handling
USArrests[1, 2] <- NA
pc.cr <- princomp(~ Murder + Assault + UrbanPop,
                  data = USArrests, na.action=na.exclude, cor = TRUE)
pc.cr$scores

print.power.htest  Print method for power calculation object

Description

Print object of class "power.htest" in nice layout.

Usage

## S3 method for class 'power.htest':
print(x, ...)

Arguments

x                         Object of class "power.htest".
...
            further arguments to be passed to or from methods.

Details

A power.htest object is just a named list of numbers and character strings, supplemented with method and note elements. The method is displayed as a title, the note as a footnote, and the remaining elements are given in an aligned ‘name = value’ format.

Value

none
print.ts

Printing Time-Series Objects

Description

Print method for time series objects.

Usage

## S3 method for class 'ts':
print(x, calendar, ...)

Arguments

- **x**: a time series object.
- **calendar**: enable/disable the display of information about month names, quarter names or year when printing. The default is TRUE for a frequency of 4 or 12, FALSE otherwise.
- **...**: additional arguments to print.

Details

This is the print methods for objects inheriting from class "ts".

See Also

print.ts.

Examples

```r
print(ts(1:10, freq = 7, start = c(12, 2)), calendar = TRUE)
```
printCoefmat

Print Coefficient Matrices

Description

Utility function to be used in “higher level” print methods, such as print.summary.lm, print.summary.glm and print.anova. The goal is to provide a flexible interface with smart defaults such that often, only x needs to be specified.

Usage

printCoefmat(x, digits=max(3,getOption("digits") - 2),
  signif.stars=getOption("show.signif.stars"),
  signif.legend=signif.stars,
  dig.tst=max(1, min(5, digits - 1)),
  cs.ind=1:k, tst.ind=k + 1, zap.ind=integer(0),
  P.values=NULL,
  has.Pvalue=nc >= 4 &&
    substr(colnames(x)[nc],1,3) == "Pr(",
  eps.Pvalue=.Machine$double.eps,
  na.print="NA", ...)

Arguments

x
  a numeric matrix like object, to be printed.
digits
  minimum number of significant digits to be used for most numbers.
signif.stars
  logical; if TRUE, P-values are additionally encoded visually as “significance stars” in order to help scanning of long coefficient tables. It defaults to the show.signif.stars slot of options.
signif.legend
  logical; if TRUE, a legend for the “significance stars” is printed provided signif.stars=TRUE.
dig.tst
  minimum number of significant digits for the test statistics, see tst.ind.
cs.ind
  indices (integer) of column numbers which are (like) coefficients and standard errors to be formatted together.
tst.ind
  indices (integer) of column numbers for test statistics.
zap.ind
  indices (integer) of column numbers which should be formatted by zapsmall, i.e., by “zapping” values close to 0.
P.values
  logical or NULL; if TRUE, the last column of x is formatted by format.pval as P values. If P.values = NULL, the default, it is set to TRUE only if options("show.coef.Pvalue") is TRUE and x has at least 4 columns and the last column name of x starts with "Pr("
has.Pvalue
  logical; if TRUE, the last column of x contains P values; in that case, it is printed if and only if P.values (above) is true.
eps.Pvalue
  number,
na.print
  a character string to code NA values in printed output.
...
  further arguments for print.
profile

Generic Function for Profiling Models

Description
Investigates behavior of objective function near the solution represented by fitted. See documentation on method functions for further details.

Usage
profile(fitted, ...)

Arguments
fitted the original fitted model object.
... additional parameters. See documentation on individual methods.

Value
A list with an element for each parameter being profiled. See the individual methods for further details.

See Also
profile.nls, profile.glm in package MASS,...
For profiling code, see Rprof.
Method for Profiling nls Objects

Description

Investigates behavior of the log-likelihood function near the solution represented by fitted.

Usage

```r
## S3 method for class 'nls':
profile(fitted, which = 1:npar, maxpts = 100, alphamax = 0.01,
delta.t = cutoff/5, ...)
```

Arguments

- `fitted`: the original fitted model object.
- `which`: the original model parameters which should be profiled. By default, all parameters are profiled.
- `maxpts`: maximum number of points to be used for profiling each parameter.
- `alphamax`: maximum significance level allowed for the profile t-statistics.
- `delta.t`: suggested change on the scale of the profile t-statistics. Default value chosen to allow profiling at about 10 parameter values.
- `...`: further arguments passed to or from other methods.

Details

The profile t-statistics is defined as the square root of change in sum-of-squares divided by residual standard error with an appropriate sign.

Value

A list with an element for each parameter being profiled. The elements are data-frames with two variables

- `par.vals`: a matrix of parameter values for each fitted model.
- `tau`: The profile t-statistics.

Author(s)

Douglas M. Bates and Saikat DebRoy

References


See Also

`nls`, `profile`, `profiler.nls`, `plot.profile.nls`
Examples

```r
# obtain the fitted object
fm1 <- nls(demand ~ SSasympOrig(Time, A, lrc), data = BOD)
# get the profile for the fitted model
pr1 <- profile(fm1)
# profiled values for the two parameters
pr1$A
pr1$lrc
```

profiler

Constructor for Profiler Objects for Nonlinear Models

Description

Create a profiler object for the model object fitted.

Usage

```r
profiler(fitted, ...)
```

Arguments

- `fitted`: the original fitted model object.
- `...`: Additional parameters. See documentation on individual methods.

Value

An object of class "profiler" which is a list with function elements

- `getFittedPars()`: the parameters in fitted
- `setDefault(varying, params)`: this is used for changing the default settings for profiling. In absence of both parameters, the default is set to the original fitted parameters with all parameters varying. The arguments are
  - `varying`: a logical, integer or character vector giving parameters to be varied.
  - `params`: the default value at which profiling is to take place.
- `getProfile(varying, params)`: this can be used in conjunction with `setDefault` without any arguments. Alternatively, the parameters to be varied and the values for fixed parameters can be specified using the arguments. The arguments are
  - `varying`: a logical vector giving parameters to be varied. This can be omitted if `params` is a named list or numeric vector.
  - `params`: values for parameters to be held fixed.
  It returns a list with elements
  - `parameters`: the parameter values for the profiled optimum.
  - `fstat`: a profile statistics. See individual methods for details.
  - `varying`: a logical vector indicating parameters which were varied.
Author(s)
Douglas M. Bates and Saikat DebRoy

See Also
profiler.nls, profile

Examples

# see documentation on individual methods

profiler.nls  Constructor for Profiler Objects from nls Objects

Description
Create a profiler object for the model object fitted of class nls.

Usage

## S3 method for class 'nls':
profiler(fitted, ...)

Arguments

fitted  the original fitted model object of class nls.
...
Additional parameters. None are used.

Value
An object of class profiler.nls which is a list with function elements
getFittedModel()
  the nlsModel object corresponding to fitted
getFittedPars()
See documentation for profiler
setDefault(varying, params)
  See documentation for profiler
getProfile(varying, params)
  In the returned list, fstat is the ratio of change in sum-of-squares and the
  residual standard error.
  For other details, see documentation for profiler

Warning
When using setDefault and getProfile together, the internal state of the fitted model may get
changed. So after completing the profiling for a parameter, the internal states should be restored by a
call to setDefault without any arguments. For example see below or the source for profile.nls.

Author(s)
Douglas M. Bates and Saikat DebRoy
References


See Also

nls, nlsModel, profiler, profile.nls

Examples

```r
## obtain the fitted object
fm1 <- nls(demand ~ SSasympOrig( Time, A, lrc ), data = BOD)
## get the profile for the fitted model
prof1 <- profiler( fm1 )
## profile with A fixed at 16.0
prof1$getProfile(c(FALSE, TRUE), 16.0)
## vary lrc
prof1$setDefault(varying = c(FALSE, TRUE))
## fix A at 14.0 and starting estimate of lrc at -0.2
prof1$setDefault(params = c(14.0, -0.2))
## and get the profile
prof1$getProfile()
## finally, set defaults back to original estimates
prof1$setDefault()
```

proj

Projections of Models

Description

proj returns a matrix or list of matrices giving the projections of the data onto the terms of a linear model. It is most frequently used for aov models.

Usage

```r
proj(object, ...)
```

## S3 method for class 'aov':
proj(object, onedf = FALSE, unweighted.scale = FALSE, ...)

## S3 method for class 'aovlist':
proj(object, onedf = FALSE, unweighted.scale = FALSE, ...)

## Default S3 method:
proj(object, onedf = TRUE, ...)

## S3 method for class 'lm':
proj(object, onedf = FALSE, unweighted.scale = FALSE, ...)
Arguments

object An object of class "lm" or a class inheriting from it, or an object with a similar structure including in particular components qr and effects.

onedf A logical flag. If TRUE, a projection is returned for all the columns of the model matrix. If FALSE, the single-column projections are collapsed by terms of the model (as represented in the analysis of variance table).

unweighted.scale If the fit producing object used weights, this determines if the projections correspond to weighted or unweighted observations.

... Swallow and ignore any other arguments.

Details

A projection is given for each stratum of the object, so for aov models with an Error term the result is a list of projections.

Value

A projection matrix or (for multi-stratum objects) a list of projection matrices.

Each projection is a matrix with a row for each observation and either a column for each term (onedf = FALSE) or for each coefficient (onedf = TRUE). Projection matrices from the default method have orthogonal columns representing the projection of the response onto the column space of the Q matrix from the QR decomposition. The fitted values are the sum of the projections, and the sum of squares for each column is the reduction in sum of squares from fitting that column (after those to the left of it).

The methods for lm and aov models add a column to the projection matrix giving the residuals (the projection of the data onto the orthogonal complement of the model space).

Strictly, when onedf = FALSE the result is not a projection, but the columns represent sums of projections onto the columns of the model matrix corresponding to that term. In this case the matrix does not depend on the coding used.

Author(s)

The design was inspired by the S function of the same name described in Chambers et al. (1992).

References


See Also

aov, lm, model.tables

Examples

N <- c(0,1,0,1,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,0,0,1)
P <- c(1,1,0,0,0,1,1,1,0,0,1,0,1,1,0,1,0,1,1,0,1,1,0,1,0,1,1,0)
K <- c(1,0,0,1,0,1,1,1,0,0,1,0,1,1,0,0,1,0,1,1,0,1,1,1,0,1,1,0)
yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5, 55.0, 62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)
npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P), K=factor(K), yield=yield)
npk.aov <- aov(yield ~ block + N*P*K, npk)
proj(npk.aov)

## as a test, not particularly sensible
options(contrasts=c("contr.helmert", "contr.treatment"))
npk.aovE <- aov(yield ~ N*P*K + Error(block), npk)
proj(npk.aovE)

---

**prop.test**

**Test of Equal or Given Proportions**

**Description**

`prop.test` can be used for testing the null that the proportions (probabilities of success) in several groups are the same, or that they equal certain given values.

**Usage**

```r
prop.test(x, n, p = NULL, 
alternative = c("two.sided", "less", "greater"), 
conf.level = 0.95, correct = TRUE)
```

**Arguments**

- `x` a vector of counts of successes or a matrix with 2 columns giving the counts of successes and failures, respectively.
- `n` a vector of counts of trials; ignored if `x` is a matrix.
- `p` a vector of probabilities of success. The length of `p` must be the same as the number of groups specified by `x`, and its elements must be greater than 0 and less than 1.
- `alternative` a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. Only used for testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
- `conf.level` confidence level of the returned confidence interval. Must be a single number between 0 and 1. Only used when testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
- `correct` a logical indicating whether Yates’ continuity correction should be applied.

**Details**

Only groups with finite numbers of successes and failures are used. Counts of successes and failures must be nonnegative and hence not greater than the corresponding numbers of trials which must be positive. All finite counts should be integers.

If `p` is `NULL` and there is more than one group, the null tested is that the proportions in each group are the same. If there are two groups, the alternatives are that the probability of success in the first group is less than, not equal to, or greater than the probability of success in the second
group, as specified by alternative. A confidence interval for the difference of proportions with confidence level as specified by conf.level and clipped to [-1,1] is returned. Continuity correction is used only if it does not exceed the difference of the sample proportions in absolute value. Otherwise, if there are more than 2 groups, the alternative is always "two.sided", the returned confidence interval is NULL, and continuity correction is never used.

If there is only one group, then the null tested is that the underlying probability of success is p, or .5 if p is not given. The alternative is that the probability of success if less than, not equal to, or greater than p or 0.5, respectively, as specified by alternative. A confidence interval for the underlying proportion with confidence level as specified by conf.level and clipped to [0,1] is returned. Continuity correction is used only if it does not exceed the difference between sample and null proportions in absolute value. The confidence interval is computed by inverting the score test.

Finally, if p is given and there are more than 2 groups, the null tested is that the underlying probabilities of success are those given by p. The alternative is always "two.sided", the returned confidence interval is NULL, and continuity correction is never used.

Value

A list with class "htest" containing the following components:

- statistic: the value of Pearson's chi-squared test statistic.
- parameter: the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- p.value: the p-value of the test.
- estimate: a vector with the sample proportions x/n.
- conf.int: a confidence interval for the true proportion if there is one group, or for the difference in proportions if there are 2 groups and p is not given, or NULL otherwise. In the cases where it is not NULL, the returned confidence interval has an asymptotic confidence level as specified by conf.level, and is appropriate to the specified alternative hypothesis.
- null.value: the value of p if specified by the null, or NULL otherwise.
- alternative: a character string describing the alternative.
- method: a character string indicating the method used, and whether Yates' continuity correction was applied.
- data.name: a character string giving the names of the data.

See Also

binom.test for an exact test of a binomial hypothesis.

Examples

```R
heads <- rbinom(1, size=100, pr = .5)
prop.test(heads, 100) # continuity correction TRUE by default
prop.test(heads, 100, correct = FALSE)
```

## Data from Fleiss (1981), p. 139.
## H0: The null hypothesis is that the four populations from which
## the patients were drawn have the same true proportion of smokers.
## A: The alternative is that this proportion is different in at
## least one of the populations.
prop.trend.test

Test for trend in proportions

Description

Performs chi-squared test for trend in proportions, i.e., a test asymptotically optimal for local alternatives where the log odds vary in proportion with score. By default, score is chosen as the group numbers.

Usage

prop.trend.test(x, n, score = 1:length(x))

Arguments

x Number of events
n Number of trials
score Group score

Value

An object of class "htest" with title, test statistic, p-value, etc.

Note

This really should get integrated with prop.test

Author(s)

Peter Dalgaard

See Also

prop.test

Examples

smokers <- c(83, 90, 129, 70)
patients <- c(86, 93, 136, 82)
prop.test(smokers, patients)
prop.trend.test(smokers, patients)
prop.trend.test(smokers, patients, c(0,0,1))
qqnorm

Quantile-Quantile Plots

Description

qqnorm is a generic function the default method of which produces a normal QQ plot of the values in y. qqline adds a line to a normal quantile-quantile plot which passes through the first and third quartiles.

qqplot produces a QQ plot of two datasets.

Graphical parameters may be given as arguments to qqnorm, qqplot and qqline.

Usage

qqnorm(y, ...)
## Default S3 method:
qqnorm(y, ylim, main = "Normal Q-Q Plot",
xlab = "Theoretical Quantiles",
ylab = "Sample Quantiles", plot.it = TRUE, datax = FALSE, ...
qqline(y, datax = FALSE, ...)
qqplot(x, y, plot.it = TRUE, xlab = deparse(substitute(x)),
ylab = deparse(substitute(y)), ...)

Arguments

x
The first sample for qqplot.
y
The second or only data sample.
xlab, ylab, main
plot labels.
plot.it
logical. Should the result be plotted?
datax
logical. Should data values be on the x-axis?
ylim, ...
graphical parameters.

Value

For qqnorm and qqplot, a list with components

x
The x coordinates of the points that were/would be plotted
y
The original y vector, i.e., the corresponding y coordinates including NAs.

References


See Also

ppoints.
Examples

```r
y <- rt(200, df = 5)
qqnorm(y); qqline(y, col = 2)
qqplot(y, rt(300, df = 5))

qqnorm(precip, ylab = "Precipitation [in/yr] for 70 US cities")
```

**quade.test**

**Quade Test**

**Description**

Performs a Quade test with unreplicated blocked data.

**Usage**

```r
quade.test(y, ...)  
## Default S3 method:  
quade.test(y, groups, blocks, ...)  
## S3 method for class 'formula':  
quade.test(formula, data, subset, na.action, ...)
```

**Arguments**

- `y` either a numeric vector of data values, or a data matrix.
- `groups` a vector giving the group for the corresponding elements of `y` if this is a vector; ignored if `y` is a matrix. If not a factor object, it is coerced to one.
- `blocks` a vector giving the block for the corresponding elements of `y` if this is a vector; ignored if `y` is a matrix. If not a factor object, it is coerced to one.
- `formula` a formula of the form `a ~ b | c`, where `a`, `b` and `c` give the data values and corresponding groups and blocks, respectively.
- `data` an optional data frame containing the variables in the model formula.
- `subset` an optional vector specifying a subset of observations to be used.
- `na.action` a function which indicates what should happen when the data contain NAs. Defaults to `getOption("na.action")`.
- `...` further arguments to be passed to or from methods.

**Details**

`quade.test` can be used for analyzing unreplicated complete block designs (i.e., there is exactly one observation in `y` for each combination of levels of `groups` and `blocks`) where the normality assumption may be violated.

The null hypothesis is that apart from an effect of `blocks`, the location parameter of `y` is the same in each of the `groups`.

If `y` is a matrix, `groups` and `blocks` are obtained from the column and row indices, respectively. NA's are not allowed in `groups` or `blocks`; if `y` contains NA's, corresponding blocks are removed.
Value

A list with class "htest" containing the following components:

- statistic: the value of Quade's F statistic.
- parameter: a vector with the numerator and denominator degrees of freedom of the approximate F distribution of the test statistic.
- p.value: the p-value of the test.
- method: the character string "Quade test".
- data.name: a character string giving the names of the data.

References


See Also

*friedman.test*.

Examples

```r
## Conover (1999, p. 375f):
## Numbers of five brands of a new hand lotion sold in seven stores
## during one week.

y <- matrix(c(  5,  4,  7, 10, 12,
             1,  3,  1,  0,  2,
             16, 12, 22, 22, 35,
             5,  4,  3,  5,  4,
             10,  9,  7, 13, 10,
             19, 18, 28, 37, 58,
             10,  7,  6,  8,  7),
    nr = 7, byrow = TRUE,
    dimnames =
       list(Store = as.character(1:7),
            Brand = LETTERS[1:5]))

y
quade.test(y)
```

---

**quantile**

<table>
<thead>
<tr>
<th>Sample Quantiles</th>
</tr>
</thead>
</table>

**Description**

The generic function *quantile* produces sample quantiles corresponding to the given probabilities. The smallest observation corresponds to a probability of 0 and the largest to a probability of 1.
Usage

quantile(x, ...)

# Default S3 method:
quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE, 
       names = TRUE, type = 7, ...)

Arguments

x        numeric vectors whose sample quantiles are wanted. Missing values are ignored.
probs    numeric vector of probabilities with values in [0, 1].
na.rm    logical; if true, any NA and NaN’s are removed from x before the quantiles are 
         computed.
names    logical; if true, the result has a names attribute. Set to FALSE for speedup with 
         many probs.
type     an integer between 1 and 9 selecting one of the nine quantile algorithms detailed 
         below to be used.
...      further arguments passed to or from other methods.

Details

A vector of length length(probs) is returned; if names = TRUE, it has a names attribute. 
NA and NaN values in probs are propagated to the result.

Types

quantile returns estimates of underlying distribution quantiles based on one or two order statistics from the supplied elements in x at probabilities in probs. One of the nine quantile algorithms discussed in Hyndman and Fan (1996), selected by type, is employed.

Sample quantiles of type $i$ are defined by

$$Q_i(p) = (1 - \gamma)x_j + \gamma x_{j+1}$$

where $1 \leq i \leq 9$, $\frac{j-m}{n} \leq p < \frac{j-m+1}{n}$, $x_j$ is the $j$th order statistic, $n$ is the sample size, and $m$ is a constant determined by the sample quantile type. Here $\gamma$ depends on the fractional part of $g = np + m - j$.

For the continuous sample quantile types (4 through 9), the sample quantiles can be obtained by linear interpolation between the $k$th order statistic and $p(k)$:

$$p(k) = \frac{k - \alpha}{n - \alpha - \beta + 1}$$

where $\alpha$ and $\beta$ are constants determined by the type. Further, $m = \alpha + p (1 - \alpha - \beta)$, and $\gamma = g$.

Discontinuous sample quantile types 1, 2, and 3

Type 1 Inverse of empirical distribution function.

Type 2 Similar to type 1 but with averaging at discontinuities.

Type 3 SAS definition: nearest even order statistic.

Continuous sample quantile types 4 through 9
Type 4 \( p(k) = \frac{k}{n} \). That is, linear interpolation of the empirical cdf.

Type 5 \( p(k) = \frac{k-0.5}{n} \). That is a piecewise linear function where the knots are the values midway through the steps of the empirical cdf. This is popular amongst hydrologists.

Type 6 \( p(k) = \frac{k}{n+1} \). Thus \( p(k) = \mathbb{E}[F(x_k)] \). This is used by Minitab and by SPSS.

Type 7 \( p(k) = \frac{k-1}{n} \). In this case, \( p(k) = \text{mode}[F(x_k)] \). This is used by S.

Type 8 \( p(k) = \frac{k-\frac{1}{2}}{n+\frac{1}{2}} \). Then \( p(k) \approx \text{median}[F(x_k)] \). The resulting quantile estimates are approximately median-unbiased regardless of the distribution of \( x \).

Type 9 \( p(k) = \frac{k-\frac{3}{8}}{n+\frac{1}{4}} \). The resulting quantile estimates are approximately unbiased if \( x \) is normally distributed.

Hyndman and Fan (1996) recommend type 8. The default method is type 7, as used by S and by \( R < 2.0.0 \).

Author(s)

of the version used in \( R >= 2.0.0 \), Ivan Frohne and Rob J Hyndman.

References


See Also

ecdf for empirical distributions of which quantile is the “inverse”; boxplot.stats and fivenum for computing “versions” of quartiles, etc.

Examples

```r
quantile(x <- rnorm(1001)) # Extremes & Quartiles by default
quantile(x, probs=c(.1,.5,1,2,5,10,50,NA)/100)

### Compare different types
p <- c(0.1,0.5,1,2,5,10,50)/100
res <- matrix(as.numeric(NA), 9, 7)
for(type in 1:9) res[type, ] <- y <- quantile(x, p, type=type)
dimnames(res) <- list(1:9, names(y))
round(res, 3)
```

read.ftable

**Manipulate Flat Contingency Tables**

Description

Read, write and coerce “flat” contingency tables.
Usage

read.ftable(file, sep = "", quote = "\\", row.var.names, col.vars, skip = 0)

write.ftable(x, file = "", quote = TRUE, digits = getOption("digits"))

## S3 method for class 'ftable':
as.table(x, ...)

Arguments

determines which data are to be read from or written to. "." indicates input from the console for reading and output to the console for writing.

sep

the field separator string. Values on each line of the file are separated by this string.

quote

either a character string naming a file or a connection which the data are to be read from or written to. "" indicates input from the console for reading and output to the console for writing.

a character string giving the set of quoting characters for read.ftable: to disable quoting altogether, use quote="". For write.table, a logical indicating whether strings in the data will be surrounded by double quotes.

row.var.names

a character vector with the names of the row variables, in case these cannot be determined automatically.

col.vars

a list giving the names and levels of the column variables, in case these cannot be determined automatically.

skip

the number of lines of the data file to skip before beginning to read data.

x

an object of class "ftable".

digits

an integer giving the number of significant digits to use for (the cell entries of) x.

... further arguments to be passed to or from methods.

Details

read.ftable reads in a flat-like contingency table from a file. If the file contains the written representation of a flat table (more precisely, a header with all information on names and levels of column variables, followed by a line with the names of the row variables), no further arguments are needed. Similarly, flat tables with only one column variable the name of which is the only entry in the first line are handled automatically. Other variants can be dealt with by skipping all header information using skip, and providing the names of the row variables and the names and levels of the column variable using row.var.names and col.vars, respectively. See the examples below.

Note that flat tables are characterized by their “ragged” display of row (and maybe also column) labels. If the full grid of levels of the row variables is given, one should instead use read.table to read in the data, and create the contingency table from this using xtabs.

write.ftable writes a flat table to a file, which is useful for generating “pretty” ASCII representations of contingency tables.

as.table.ftable converts a contingency table in flat matrix form to one in standard array form. This is a method for the generic function as.table.
References


See Also

`ftable` for more information on flat contingency tables.

Examples

```r
## Agresti (1990), page 157, Table 5.8.
## Not in ftable standard format, but o.k.
file <- tempfile()
cat(" Intercourse
"Race Gender Yes No
"White Male  43 134
" Female 26 149
"Black Male 29 23
" Female 22 36

file = file)
file.show(file)
ft <- read.ftable(file)
ft
unlink(file)

## Agresti (1990), page 297, Table 8.16.
## Almost o.k., but misses the name of the row variable.
file <- tempfile()
cat(" "Tonsil Size"
"Not Enl." "Enl." "Greatly Enl." 
"Noncarriers 497 560 269
"Carriers 19 29 24

file = file)
file.show(file)
ft <- read.ftable(file, skip = 2,
row.var.names = "Status",
col.vars = list("Tonsil Size" =
c("Not Enl.", "Enl.", "Greatly Enl.")))
ft
unlink(file)
```

rect.hclust  

**Draw Rectangles Around Hierarchical Clusters**

Description

Draws rectangles around the branches of a dendrogram highlighting the corresponding clusters. First the dendrogram is cut at a certain level, then a rectangle is drawn around selected branches.

Usage

```r
rect.hclust(tree, k = NULL, which = NULL, x = NULL, h = NULL,
border = 2, cluster = NULL)
```
Arguments

- tree: an object of the type produced by hclust.
- k, h: Scalar. Cut the dendrogram such that either exactly k clusters are produced or by cutting at height h.
- which, x: A vector selecting the clusters around which a rectangle should be drawn. which selects clusters by number (from left to right in the tree). x selects clusters containing the respective horizontal coordinates. Default is which = 1:k.
- border: Vector with border colors for the rectangles.
- cluster: Optional vector with cluster memberships as returned by cutree(hclust.obj, k = k), can be specified for efficiency if already computed.

Value

(Invisibly) returns a list where each element contains a vector of data points contained in the respective cluster.

See Also

hclust, identify.hclust.

Examples

```r
hca <- hclust(dist(USArrests))
plot(hca)
rect.hclust(hca, k=3, border="red")
x <- rect.hclust(hca, h=50, which=c(2,7), border=3:4)
x
```

---

### relevel

**Reorder Levels of Factor**

**Description**

The levels of a factor are re-ordered so that the level specified by ref is first and the others are moved down. This is useful for contr.treatment contrasts which take the first level as the reference.

**Usage**

```r
relevel(x, ref, ...)
```

**Arguments**

- x: An unordered factor.
- ref: The reference level.
- ...: Additional arguments for future methods.
reorder

Value

A factor of the same length as x.

See Also

factor, contr.treatment

Examples

warpbreaks$tension <- relevel(warpbreaks$tension, ref="M")
summary(lm(breaks ~ wool + tension, data=warpbreaks))

---

reorder
Reorder a dendrogram

Description

There are many different orderings of a dendrogram that are consistent with the structure imposed. This function takes a dendrogram and a vector of values and reorders the dendrogram in the order of the supplied vector, maintaining the constraints on the dendrogram.

Usage

reorder(x, ...)

## S3 method for class 'dendrogram':
reorder(x, wts, agglo.FUN = sum, ...)

Arguments

x
the (dendrogram) object to be reordered

wts
numeric weights (arbitrary values) for reordering.

agglo.FUN
a function for weights agglomeration, see below.

...
additional arguments

Details

Using the weights wts, the leaves of the dendrogram are reordered so as to be in an order as consistent as possible with the weights. At each node, the branches are ordered in increasing weights where the weight of a branch is defined as $f(w_j)$ where $f$ is agglo.FUN and $w_j$ is the weight of the $j$-th sub branch).

Value

From reorder.dendrogram, a dendrogram where each node has a further attribute value with its corresponding weight.

Author(s)

R. Gentleman and M. Maechler
See Also

rev.dendrogram which simply reverses the nodes' order; heatmap, cophenetic.

Examples

```
set.seed(123)
x <- rnorm(10)
hc <- hclust(dist(x))
dd <- as.dendrogram(hc)
dd.reorder <- reorder(dd, 10:1)
plot(dd, main = "random dendrogram `dd'")

op <- par(mfcol = 1:2)
plot(dd.reorder, main = "reorder(dd, 10:1)")
plot(reorder(dd, 10:1, agglo.FUN= mean),
     main = "reorder(dd, 10:1, mean")
par(op)
```

---

**reorder.factor**

Reorder levels of a factor

### Description

Reorders the levels of a factor depending on values of a second variable, usually numeric.

### Usage

```r
## S3 method for class 'factor':
reorder(x, X, FUN = mean, ..., 
       order = is.ordered(x))
```

### Arguments

- **x**: a factor (possibly ordered) whose levels will be reordered.
- **X**: a vector of the same length as `x`, whose subset of values for each unique level of `x` determines the eventual order of that level.
- **FUN**: a function whose first argument is a vector and returns a scalar, to be applied to each subset of `X` determined by the levels of `x`.
- **...**: optional: extra arguments supplied to `FUN`.
- **order**: logical, whether return value will be an ordered factor rather than a factor.

### Value

A factor or an ordered factor (depending on the value of `order`), with the order of the levels determined by `FUN` applied to `X` grouped by `x`. The levels are ordered such that the values returned by `FUN` are in increasing order.

Additionally, the values of `FUN` applied to the subsets of `X` (in the original order of the levels of `x`) is returned as the "scores" attribute.
**Replications**

**Author(s)**

Deepayan Sarkar (deepayan@stat.wisc.edu)

**Examples**

```r
bymedian <- with(InsectSprays, reorder(spray, count, median))
boxplot(count ~ bymedian, data = InsectSprays,
xlab = "Type of spray", ylab = "Insect count",
main = "InsectSprays data", varwidth = TRUE,
col = "lightgray")
```

<table>
<thead>
<tr>
<th>replications</th>
<th>Number of Replications of Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Returns a vector or a list of the number of replicates for each term in the formula.

**Usage**

```r
replications(formula, data=NULL, na.action)
```

**Arguments**

- `formula`: a formula or a terms object or a data frame.
- `data`: a data frame used to find the objects in `formula`.
- `na.action`: function for handling missing values. Defaults to a `na.action` attribute of `data`, then a setting of the option `na.action`, or `na.fail` if that is not set.

**Details**

If `formula` is a data frame and `data` is missing, `formula` is used for `data` with the formula `~ .`

**Value**

A vector or list with one entry for each term in the formula giving the number(s) of replications for each level. If all levels are balanced (have the same number of replications) the result is a vector, otherwise it is a list with a component for each terms, as a vector, matrix or array as required.

A test for balance is `!is.list(replications(formula, data))`.

**Author(s)**

The design was inspired by the S function of the same name described in Chambers *et al.* (1992).

**References**

See Also

model.tables

Examples


N <- c(0,1,0,1,1,0,0,1,1,0,1,0,1,0,1,1,0,1,0,1,1,0,0,1,0)
P <- c(1,1,0,0,1,1,1,0,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,1)
K <- c(1,0,1,0,1,0,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,1)
yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,
55.0, 62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)

npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P),
                  K=factor(K), yield=yield)
replications(~ . - yield, npk)

reshape
Reshape Grouped Data

Description

This function reshapes a data frame between ‘wide’ format with repeated measurements in separate columns of the same record and ‘long’ format with the repeated measurements in separate records.

Usage

reshape(data, varying = NULL, v.names = NULL, timevar = "time",
        idvar = "id", ids = 1:NROW(data),
        times = seq(length = length(varying[[1]])),
        drop = NULL, direction, new.row.names = NULL,
        split = list(regexp="\." , include=FALSE))

Arguments

- **data**: a data frame
- **varying**: names of sets of variables in the wide format that correspond to single variables in long format (‘time-varying’). A list of vectors (or optionally a matrix for direction="wide"). See below for more details and options.
- **v.names**: names of variables in the long format that correspond to multiple variables in the wide format.
- **timevar**: the variable in long format that differentiates multiple records from the same group or individual.
- **idvar**: Names of one or more variables in long format that identify multiple records from the same group/indivdual. These variables may also be present in wide format.
- **ids**: the values to use for a newly created idvar variable in long format.
- **times**: the values to use for a newly created timevar variable in long format.
- **drop**: a vector of names of variables to drop before reshaping.
- **direction**: character string, either "wide" to reshape to wide format, or "long" to reshape to long format.
new.row.names

logical; if TRUE and direction="wide", create new row names in long format from the values of the id and time variables.

split

information for guessing the varying, v.names, and times arguments. See below for details.

Details

The arguments to this function are described in terms of longitudinal data, as that is the application motivating the functions. A ‘wide’ longitudinal dataset will have one record for each individual with some time-constant variables that occupy single columns and some time-varying variables that occupy a column for each time point. In ‘long’ format there will be multiple records for each individual, with some variables being constant across these records and others varying across the records. A ‘long’ format dataset also needs a ‘time’ variable identifying which time point each record comes from and an ‘id’ variable showing which records refer to the same person.

If the data frame resulted from a previous reshape then the operation can be reversed simply by reshape(a). The direction argument is optional and the other arguments are stored as attributes on the data frame.

If direction="long" and no varying or v.names arguments are supplied it is assumed that all variables except idvar and timevar are time-varying. They are all expanded into multiple variables in wide format.

If direction="wide" the varying argument can be a vector of column names or column numbers (converted to column names). The function will attempt to guess the v.names and times from these names. The default is variable names like x.1, x.2, where split=list(regexp="\"\",include=FALSE) to specifies to split at the dot and drop it from the name. To have alphabetic followed by numeric times use split=list(regexp="[A-Za-z][0-9]",include=TRUE). This splits between the alphabetic and numeric parts of the name and does not drop the regular expression.

Value

The reshaped data frame with added attributes to simplify reshaping back to the original form.

See Also

stack, aperm

Examples

summary(Indometh)

wide <- reshape(Indometh, v.names="conc", idvar="Subject",
            timevar="time", direction="wide")

reshape(wide, direction="long")
reshape(wide, idvar="Subject", varying=list(names(wide)[2:12]),
       v.names="conc", direction="long")

## times need not be numeric
df <- data.frame(id=rep(1:4,rep(2,4)), visit=I(rep(c("Before","After"),4)),
              x=rnorm(4), y=runif(4))

df <- reshape(df, timevar="visit", idvar="id", direction="wide")
## warns that y is really varying
runmed

Running Medians – Robust Scatter Plot Smoothing

Description

Compute running medians of odd span. This is the “most robust” scatter plot smoothing possible. For efficiency (and historical reason), you can use one of two different algorithms giving identical results.

Usage

runmed(x, k, endrule = c("median", "keep", "constant"),
    algorithm = NULL, print.level = 0)

Arguments

x numeric vector, the “dependent” variable to be smoothed.
k integer width of median window; must be odd. Turlach had a default of k <- 1
    + 2 * min((n-1)%/% 2, ceiling(0.1*n)). Use k = 3 for “mini-
    mal” robust smoothing eliminating isolated outliers.
endrule character string indicating how the values at the beginning and the end (of the data) should be treated.

"keep" keeps the first and last \( k_2 \) values at both ends, where \( k_2 \) is the half-bandwidth \( k_2 = k \div 2 \), i.e., \( y[j] = x[j] \) for \( j \in \{1, \ldots, k_2; n - k_2 + 1, \ldots, n\} \);

"constant" copies \( \text{median}(y[1:k_2]) \) to the first values and analogously for the last ones making the smoothed ends constant;

"median" the default, smoothes the ends by using symmetrical medians of subsequently smaller bandwidth, but for the very first and last value where Tukey's robust end-point rule is applied, see smoothEnds.

algorithm character string (partially matching "Turlach" or "Stuetzle") or the default NULL, specifying which algorithm should be applied. The default choice depends on \( n = \text{length}(x) \) and \( k \) where "Turlach" will be used for larger problems.

print.level integer, indicating verboseness of algorithm; should rarely be changed by average users.

Details

Apart from the end values, the result \( y = \text{runmed}(x, k) \) simply has \( y[j] = \text{median}(x[(j-k_2):(j+k_2)]) \) (\( k = 2k_2+1 \)), computed very efficiently.

The two algorithms are internally entirely different:

"Turlach" is the Härdle-Steiger algorithm (see Ref.) as implemented by Berwin Turlach. A tree algorithm is used, ensuring performance \( O(n \log k) \) where \( n \leftarrow \text{length}(x) \) which is asymptotically optimal.

"Stuetzle" is the (older) Stuetzle-Friedman implementation which makes use of median updating when one observation enters and one leaves the smoothing window. While this performs as \( O(n \times k) \) which is slower asymptotically, it is considerably faster for small \( k \) or \( n \).

Value

vector of smoothed values of the same length as \( x \) with an attribute \( k \) containing (the 'oddified') \( k \).

Author(s)

Martin Maechler (maechler@stat.math.ethz.ch), based on Fortran code from Werner Stuetzle and S-plus and C code from Berwin Turlach.

References


scatter.smooth

Scatter Plot with Smooth Curve Fitted by Loess

Description
Plot and add a smooth curve computed by loess to a scatter plot.

Usage
scatter.smooth(x, y = NULL, span = 2/3, degree = 1, family = c("symmetric", "gaussian"), xlab = NULL, ylab = NULL, ylim = range(y, prediction$y, na.rm = TRUE), evaluation = 50, ...)

loess.smooth(x, y, span = 2/3, degree = 1, family = c("symmetric", "gaussian"), evaluation = 50, ...)
Arguments

\texttt{x, y} \hspace{1cm} \texttt{the x and y arguments provide the x and y coordinates for the plot. Any reasonable way of defining the coordinates is acceptable. See the function \texttt{xy.coords} for details.}

\texttt{span} \hspace{1cm} \texttt{smoothness parameter for \texttt{loess}.}

\texttt{degree} \hspace{1cm} \texttt{degree of local polynomial used.}

\texttt{family} \hspace{1cm} \texttt{if "gaussian" fitting is by least-squares, and if family="symmetric" a re-descending M estimator is used.}

\texttt{xlab} \hspace{1cm} \texttt{label for x axis.}

\texttt{ylab} \hspace{1cm} \texttt{label for y axis.}

\texttt{ylim} \hspace{1cm} \texttt{the y limits of the plot.}

\texttt{evaluation} \hspace{1cm} \texttt{number of points at which to evaluate the smooth curve.}

\texttt{...} \hspace{1cm} \texttt{graphical parameters.}

Details

\texttt{loess.smooth} is an auxiliary function which evaluates the \texttt{loess} smooth at evaluation equally spaced points covering the range of \texttt{x}.

Value

For \texttt{scatter.smooth}, none.

For \texttt{loess.smooth}, a list with two components, \texttt{x} (the grid of evaluation points) and \texttt{y} (the smoothed values at the grid points).

See Also

\texttt{loess}

Examples

\begin{verbatim}
attach(cars)
scatter.smooth(speed, dist)
detach()
\end{verbatim}

\texttt{screeplot \hspace{1cm} Screeplot of PCA Results}

Description

\texttt{screeplot} plots the variances against the number of the principal component. This is also the plot method for class "princomp".

Usage

\begin{verbatim}
screeplot(x, npcs = min(10, length(x$sdev)),
         type = c("barplot", "lines"),
         main = deparse(substitute(x)), ...)
\end{verbatim}
Arguments

- `x` an object of class "princomp", as from `princomp()`.
- `npcs` the number of principal components to be plotted.
- `type` the type of plot.
- `main, ...` graphics parameters.

References


See Also

- `princomp`.

Examples

```r
## The variances of the variables in the
## USArrests data vary by orders of magnitude, so scaling is appropriate
(pc.cr <- princomp(USArrests, cor = TRUE)) # inappropriate
screeplot(pc.cr)

fit <- princomp(covmat=Harman74.cor)
screeplot(fit)
screeplot(fit, npcs=24, type="lines")
```

---

**Standard Deviation**

This function computes the standard deviation of the values in `x`. If `na.rm` is `TRUE` then missing values are removed before computation proceeds. If `x` is a matrix or a data frame, a vector of the standard deviation of the columns is returned.

Usage

```
sd(x, na.rm = FALSE)
```

Arguments

- `x` a numeric vector, matrix or data frame.
- `na.rm` logical. Should missing values be removed?

See Also

- `var` for its square, and `mad`, the most robust alternative.

Examples

```
sd(1:2) ^ 2
```
se.contrast

Standard Errors for Contrasts in Model Terms

Description

Returns the standard errors for one or more contrasts in an aov object.

Usage

se.contrast(object, ...)
## S3 method for class 'aov':
se.contrast(object, contrast.obj, 
coef = contr.helmert(ncol(contrast))[, 1],
data = NULL, ...)

Arguments

object A suitable fit, usually from aov.
contrast.obj The contrasts for which standard errors are requested. This can be specified via a list or via a matrix. A single contrast can be specified by a list of logical vectors giving the cells to be contrasted. Multiple contrasts should be specified by a matrix, each column of which is a numerical contrast vector (summing to zero).
coef used when contrast.obj is a list; it should be a vector of the same length as the list with zero sum. The default value is the first Helmert contrast, which contrasts the first and second cell means specified by the list.
data The data frame used to evaluate contrast.obj.
... further arguments passed to or from other methods.

Details

Contrasts are usually used to test if certain means are significantly different; it can be easier to use se.contrast than compute them directly from the coefficients.

In multistratum models, the contrasts can appear in more than one stratum, in which case the standard errors are computed in the lowest stratum and adjusted for efficiencies and comparisons between strata. (See the comments in the note in the help for aov about using orthogonal contrasts.) Such standard errors are often conservative.

Suitable matrices for use with coef can be found by calling contrasts and indexing the columns by a factor.

Value

A vector giving the standard errors for each contrast.

See Also

contrasts, model.tables
Examples

N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,0,1,0,0,1,0,1,1,0,1,1,0)
P <- c(1,1,0,0,0,1,1,0,0,0,1,1,0,1,0,1,0,1,1,0,1,1,0,1,1)
yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,
           55.0, 62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)
npk <- data.frame(block = gl(6,4), N = factor(N), P = factor(P),
                  K = factor(K), yield = yield)
## Set suitable contrasts.
options(contrasts=c("contr.helmert", "contr.poly"))
npk.aov1 <- aov(yield ~ block + N + K, data=npk)
se.contrast(npk.aov1, list(N == "0", N == "1"), data=npk)
## or via a matrix
cont <- matrix(c(-1,1), 2, 1, dimnames=list(NULL, "N"))
se.contrast(npk.aov1, cont[N, , drop=FALSE]/12, data=npk)

## test a multi-stratum model
npk.aov2 <- aov(yield ~ N + K + Error(block/(N + K)), data=npk)
se.contrast(npk.aov2, list(N == "0", N == "1"))

## an example looking at an interaction contrast
## Dataset from R.E. Kirk (1995)
## 'Experimental Design: procedures for the behavioral sciences'
A <- gl(2, 18, labels=c("a1", "a2"))
B <- rep(gl(3, 6, labels=c("b1", "b2", "b3")), 2)
fit <- aov(score ~ A * B)
cont <- c(1, -1)[A] * c(1, -1, 0)[B]
sum(cont) # 0
sum(cont*score) # value of the contrast
se.contrast(fit, as.matrix(cont))
(t.stat <- sum(cont*score)/se.contrast(fit, as.matrix(cont)))
summary(fit, split=list(B=1:2), expand.split = TRUE)
## t.stat^2 is the F value on the A:B: C1 line (with Helmert contrasts)
## Now look at all three interaction contrasts
cont <- c(1, -1)[A] * cbind(c(1, -1, 0), c(1, 0, -1), c(0, 1, -1))[B,]
se.contrast(fit, cont) # same, due to balance.
rm(A,B,score)

## multi-stratum example where efficiencies play a role
example(eff.aovlist)
fit <- aov(Yield ~ A + B * C + Error(Block), data = aovdat)
cont1 <- c(-1, 1)[A]/32 # Helmert contrasts
cont2 <- c(-1, 1)[B] * c(-1, 1)[C]/32
cont <- cbind(A=cont1, BC=cont2)
colSums(cont*Yield) # values of the contrasts
se.contrast(fit, as.matrix(cont))
## Not run:
# comparison with lme
library(nlme)
fit2 <- lme(Yield ~ A + B+C, random = ~1 | Block, data = aovdat)
summary(fit2)$tTable # same estimates, similar (but smaller) se's.
## End(Not run)
Construct Self-starting Nonlinear Models

Description

Construct self-starting nonlinear models.

Usage

`selfStart(model, initial, parameters, template)`

Arguments

- `model`: a function object defining a nonlinear model or a nonlinear formula object of the form `~expression`.
- `initial`: a function object, taking three arguments: `mCall`, `data`, and `LHS`, representing, respectively, a matched call to the function `model`, a data frame in which to interpret the variables in `mCall`, and the expression from the left-hand side of the model formula in the call to `nls`. This function should return initial values for the parameters in `model`.
- `parameters`: a character vector specifying the terms on the right hand side of `model` for which initial estimates should be calculated. Passed as the `namevec` argument to the `deriv` function.
- `template`: an optional prototype for the calling sequence of the returned object, passed as the `function.arg` argument to the `deriv` function. By default, a template is generated with the covariates in `model` coming first and the parameters in `model` coming last in the calling sequence.

Details

This function is generic; methods functions can be written to handle specific classes of objects.

Value

A function object of class "selfStart", for the formula method obtained by applying `deriv` to the right hand side of the `model` formula. An initial attribute (defined by the `initial` argument) is added to the function to calculate starting estimates for the parameters in the model automatically.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

`nls`
Examples

```r
## self-starting logistic model

SSlogis <- selfStart(~ Asym/(1 + exp((xmid - x)/scal)),
  function(mCall, data, LHS)
  {
    xy <- sortedXYData(mCall[["x"]], LHS, data)
    if(nrow(xy) < 4) {
      stop("Too few distinct x values to fit a logistic")
    }
    z <- xy["y"]
    if (min(z) <= 0) { z <- z + 0.05 * max(z) } # avoid zeroes
    z <- z/(1.05 * max(z)) # scale to within unit height
    xy[["z"]]<- log(z/(1 - z)) # logit transformation
    aux <- coef(lm(x ~ z, xy))
    parameters(xy) <- list(xmid = aux[1], scal = aux[2])
    pars <- as.vector(coef(nls(y ~ 1/(1 + exp((xmid - x)/scal)),
      data = xy, algorithm = "plinear"))
    value <- c(pars[3], pars[1], pars[2])
    names(value) <- mCall[c("Asym", "xmid", "scal")]
    value
  }, c("Asym", "xmid", "scal"))
```

# 'first.order.log.model' is a function object defining a first order
# compartment model
# 'first.order.log.initial' is a function object which calculates initial
# values for the parameters in 'first.order.log.model'

## Not run:

SSfol <- selfStart(first.order.log.model, first.order.log.initial)
## End(Not run)

setNames

### Set the Names in an Object

Description

This is a convenience function that sets the names on an object and returns the object. It is most useful at the end of a function definition where one is creating the object to be returned and would prefer not to store it under a name just so the names can be assigned.

Usage

```r
setNames(object, nm)
```

Arguments

- **object**: an object for which a names attribute will be meaningful
- **nm**: a character vector of names to assign to the object

Value

An object of the same sort as object with the new names assigned.
**Description**

Performs the Shapiro-Wilk test of normality.

**Usage**

`shapiro.test(x)`

**Arguments**

- **x**: a numeric vector of data values, the number of which must be between 3 and 5000. Missing values are allowed.

**Value**

A list with class "htest" containing the following components:

- **statistic**: the value of the Shapiro-Wilk statistic.
- **p.value**: the p-value for the test.
- **method**: the character string "Shapiro-Wilk normality test".
- **data.name**: a character string giving the name(s) of the data.

**References**


See Also

`qqnorm` for producing a normal quantile-quantile plot.

Examples

```r
shapiro.test(rnorm(100, mean = 5, sd = 3))
shapiro.test(runif(100, min = 2, max = 4))
```

### SignRank

#### Distribution of the Wilcoxon Signed Rank Statistic

**Description**

Density, distribution function, quantile function and random generation for the distribution of the Wilcoxon Signed Rank statistic obtained from a sample with size n.

**Usage**

```r
dsignrank(x, n, log = FALSE)
psignrank(q, n, lower.tail = TRUE, log.p = FALSE)
qsignrank(p, n, lower.tail = TRUE, log.p = FALSE)
rsignrank(nn, n)
```

**Arguments**

- `x, q` vector of quantiles.
- `p` vector of probabilities.
- `nn` number of observations. If `length(nn) > 1`, the length is taken to be the number required.
- `n` number(s) of observations in the sample(s). A positive integer, or a vector of such integers.
- `log, log.p` logical; if TRUE, probabilities p are given as log(p).
- `lower.tail` logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

**Details**

This distribution is obtained as follows. Let $x$ be a sample of size $n$ from a continuous distribution symmetric about the origin. Then the Wilcoxon signed rank statistic is the sum of the ranks of the absolute values $x[i]$ for which $x[i]$ is positive. This statistic takes values between 0 and $n(n + 1)/2$, and its mean and variance are $n(n + 1)/4$ and $n(n + 1)(2n + 1)/24$, respectively.

If either of the first two arguments is a vector, the recycling rule is used to do the calculations for all combinations of the two up to the length of the longer vector.

**Value**

dsignrank gives the density, psignrank gives the distribution function, qsignrank gives the quantile function, and rsignrank generates random deviates.
simulate

Author(s)
Kurt Hornik

See Also
wilcox.test to calculate the statistic from data, find p values and so on.
dwilcox etc, for the distribution of two-sample Wilcoxon rank sum statistic.

Examples
par(mfrow=c(2,2))
for(n in c(4:5,10,40)) {
  x <- seq(0, n*(n+1)/2, length=501)
  plot(x, dsignrank(x,n=n), type='l', main=paste("dsignrank(x,n=",n,")"))
}

simulate

Simulate Responses

Description
Simulate one or more response vectors from the theoretical distribution corresponding to a fitted
model object.

Usage
simulate(object, nsim, seed, ...)

Arguments
object an object representing a fitted model.
nsim number of response vectors to simulate. Defaults to 1.
seed an object specifying if and how the random number generator should be initialized ("seeded”).
For the "lm" method, either NULL or an integer that will be used in a call to set.seed before simulating the response vectors. If set, the value is saved as the "seed" attribute of the returned value. The default, NULL will not change the random generator state, and return .Random.seed as "seed" attribute, see below.
... additional optional arguments.

Details
This is a generic function with a method for lm objects. Consult the individual modeling functions
for details on how to use this function.
Value

Typically, a list of length \( n_{sim} \) of simulated response vectors. When appropriate the result can be a data frame (which is a special type of list).

For the "lm" method, the result is a data frame with an attribute "seed" containing the seed argument and as.list(RNGkind()) if seed was not NULL, or the value of .Random.seed before the simulation was started when seed was NULL as by default.

See Also

\texttt{fitted.values} and \texttt{residuals} for related methods; \texttt{glm}, \texttt{lm} for model fitting.

Examples

\begin{verbatim}
x <- 1:5
mod1 <- lm(c(1:3,7,6) ~ x)
S1 <- simulate(mod1, nsim = 4)
  ## repeat the simulation:
  .Random.seed <- attr(S1, "seed")
  identical(S1, simulate(mod1, nsim = 4))
S2 <- simulate(mod1, nsim = 200, seed = 101)
rowMeans(S2) # should be about fitted(mod1)
  ## repeat identically:
  (sseed <- attr(S2, "seed")) # seed; RNGkind as attribute
  stopifnot(identical(S2, simulate(mod1, nsim = 200, seed = sseed)))
  ## To be sure about the proper RNGkind, e.g., after
  RNGversion("1.0.0")
  ## first set the RNG kind, then simulate
  do.call(RNGkind, attr(sseed, "kind"))
  identical(S2, simulate(mod1, nsim = 200, seed = sseed))
\end{verbatim}

\begin{itemize}
  \item \texttt{smooth} \quad \textit{Tukey's (Running Median) Smoothing}
\end{itemize}

Description

Tukey’s smoothers, 3RS3R, 3RSS, 3R, etc.

Usage

\begin{verbatim}
smooth(x, kind = c("3RS3R", "3RSS", "3RSR", "3R", "3", "S"),
  twiceit = FALSE, endrule = "Tukey", do.ends = FALSE)
\end{verbatim}

Arguments

- \texttt{x} \quad \text{a vector or time series}
- \texttt{kind} \quad \text{a character string indicating the kind of smoother required; defaults to "3RS3R".}
twiceit  logical, indicating if the result should be “twiced”. Twicing a smoother $S(y)$ means $S(y) + S(y - S(y))$, i.e., adding smoothed residuals to the smoothed values. This decreases bias (increasing variance).

dendrule  a character string indicating the rule for smoothing at the boundary. Either "Tukey" (default) or "copy".

do.ends  logical, indicating if the 3-splitting of ties should also happen at the boundaries ("ends"). This is only used for kind = "S".

Details

3 is Tukey’s short notation for running medians of length 3, 3R stands for Repeated 3 until convergence, and $S$ for Splitting of horizontal stretches of length 2 or 3.
Hence, 3RSS3R is a concatenation of 3R, S and 3R, 3RSS similarly, whereas 3RSR means first 3R and then (S and 3) Repeated until convergence – which can be bad.

Value

An object of class "tukeysmooth" (which has print and summary methods) and is a vector or time series containing the smoothed values with additional attributes.

Note

S and S-PLUS use a different (somewhat better) Tukey smoother in smooth(*). Note that there are other smoothing methods which provide rather better results. These were designed for hand calculations and may be used mainly for didactical purposes.
Since R version 1.2, smooth does really implement Tukey’s end-point rule correctly (see argument endrule).
kind = "3RSR" has been the default till R-1.1, but it can have very bad properties, see the examples.
Note that repeated application of smooth(*) does smooth more, for the "3RS*" kinds.

References


See Also

lowess, loess, supsmu and smooth.spline.

Examples

```r
## see also demo(smooth)!
x1 <- c(4, 1, 3, 6, 6, 4, 1, 6, 2, 4, 2) # very artificial
(x3R <- smooth(x1, "3R")) # 2 iterations of "3"
smooth(x3R, kind = "S")

sm.3RS <- function(x, ...)  
  smooth(smooth(x, "3R", ...), "S", ...)

y <- c(1,1, 19:1)
plot(y, main = "misbehaviour of \"3RSR\"", col.main = 3)
```
smooth.spline

Fit a Smoothing Spline

Description
Fits a cubic smoothing spline to the supplied data.

Usage
smooth.spline(x, y = NULL, w = NULL, df, spar = NULL,
  cv = FALSE, all.knots = FALSE, nknots = NULL, keep.data = TRUE,
  df.offset = 0, penalty = 1, control.spar = list())

Arguments
x
  a vector giving the values of the predictor variable, or a list or a two-column
  matrix specifying x and y.

y
  responses. If y is missing, the responses are assumed to be specified by x.

w
  optional vector of weights of the same length as x; defaults to all 1.

df
  the desired equivalent number of degrees of freedom (trace of the smoother ma-
  trix).

spar
  smoothing parameter, typically (but not necessarily) in \([0, 1]\). The coefficient \(\lambda\)
  of the integral of the squared second derivative in the fit (penalized log likeli-
  hood) criterion is a monotone function of spar, see the details below.

cv
  ordinary (TRUE) or “generalized” cross-validation (GCV) when FALSE.

all.knots
  if TRUE, all distinct points in x are used as knots. If FALSE (default), a subset
  of x[ ] is used, specifically x[j] where the nknots indices are evenly spaced
  in 1:n, see also the next argument nknots.

nknots
  integer giving the number of knots to use when all.knots=FALSE. Per default,
  this is less than n, the number of unique x values for n > 49.
smooth.spline

keep.data logical specifying if the input data should be kept in the result. If TRUE (as per default), fitted values and residuals are available from the result.

df.offset allows the degrees of freedom to be increased by df.offset in the GCV criterion.

penalty the coefficient of the penalty for degrees of freedom in the GCV criterion.

control.spar optional list with named components controlling the root finding when the smoothing parameter spar is computed, i.e., missing or NULL, see below.

Note that this is partly experimental and may change with general spar computation improvements!

low: lower bound for spar; defaults to -1.5 (used to implicitly default to 0 in R versions earlier than 1.4).

high: upper bound for spar; defaults to +1.5.

tol: the absolute precision (tolerance) used; defaults to 1e-4 (formerly 1e-3).

eps: the relative precision used; defaults to 2e-8 (formerly 0.00244).

trace: logical indicating if iterations should be traced.

maxit: integer giving the maximal number of iterations; defaults to 500.

Note that spar is only searched for in the interval [low, high].

Details

The x vector should contain at least four distinct values. Distinct here means “distinct after rounding to 6 significant digits”, i.e., x will be transformed to unique(sort(signif(x, 6))), and y and w are pooled accordingly.

The computational λ used (as a function of s = spar) is λ = r * 256**(s-1) where r = \text{tr}(X'WX)/\text{tr}(\Sigma), \Sigma is the matrix given by \Sigma_{ij} = \int B_i''(t)B_j''(t)dt, X is given by X_{ij} = B_j(x_i), W is the diagonal matrix of weights (scaled such that its trace is n, the original number of observations) and B_k(.) is the k-th B-spline.

Note that with these definitions, f_i = f(x_i), and the B-spline basis representation f = Xc (i.e., c is the vector of spline coefficients), the penalized log likelihood is \( L = (y - f)'W(y - f) + \lambda c'\Sigma c \), and hence c is the solution of the (ridge regression) \( (X'WX + \lambda\Sigma)c = X'Wy \).

If spar is missing or NULL, the value of df is used to determine the degree of smoothing. If both are missing, leave-one-out cross-validation (ordinary or “generalized” as determined by cv) is used to determine λ. Note that from the above relation, spar is s = s0 + 0.0601 * log λ, which is intentionally different from the S-plus implementation of smooth.spline (where spar is proportional to λ). In R’s (log λ) scale, it makes more sense to vary spar linearly.

Note however that currently the results may become very unreliable for spar values smaller than about -1 or -2. The same may happen for values larger than 2 or so. Don’t think of setting spar or the controls low and high outside such a safe range, unless you know what you are doing!

The “generalized” cross-validation method will work correctly when there are duplicated points in x. However, it is ambiguous what leave-one-out cross-validation means with duplicated points, and the internal code uses an approximation that involves leaving out groups of duplicated points. cv=TRUE is best avoided in that case.

Value

An object of class "smooth.spline" with components

x the distinct x values in increasing order, see the Details above.

y the fitted values corresponding to x.
smooth.spline

\[ w \]
the weights used at the unique values of \( x \).

\[ yin \]
the \( y \) values used at the unique \( y \) values.

\[ data \]
only if \( \text{keep.data} = \text{TRUE} \): itself a \textbf{list} with components \( x \), \( y \) and \( w \) of the same length. These are the original \((x_i, y_i, w_i), i = 1, \ldots, n\), values where \( data\$x \) may have repeated values and hence be longer than the above \( x \) component; see details.

\[ lev \]
leverages, the diagonal values of the smoother matrix.

\[ cv.crit \]
cross-validation score, “generalized” or true, depending on \( cv \).

\[ pen.crit \]
penalized criterion

\[ crit \]
the criterion value minimized in the underlying \( \text{Fortran} \) routine ‘\text{sslvrg}’.

\[ df \]
equivalent degrees of freedom used. Note that (currently) this value may become quite unprecise when the true \( df \) is between 1 and 2.

\[ spar \]
the value of \( spar \) computed or given.

\[ lambda \]
the value of \( \lambda \) corresponding to \( spar \), see the details above.

\[ iparms \]
named integer(3) vector where \( ..\$ipars\"iter\" \) gives number of \( spar \) computing iterations used.

\[ fit \]
list for use by \textbf{predict.smooth.spline}, with components

\textbf{knot:} the knot sequence (including the repeated boundary knots).

\textbf{nk:} number of coefficients or number of “proper” knots plus 2.

\textbf{coef:} coefficients for the spline basis used.

\textbf{min, range:} numbers giving the corresponding quantities of \( x \).

\textbf{call}
the matched call.

\textbf{Note}

The default \( all.knots = \text{FALSE} \) and \( nknots = \text{NULL} \) entails using only \( O(n^{0.2}) \) knots instead of \( n \) for \( n > 49 \). This cuts speed and memory requirements, but not drastically anymore since \( R \) version 1.5.1 where it is only \( O(nk) + O(n) \) where \( nk \) is the number of knots. In this case where not all unique \( x \) values are used as knots, the result is not a smoothing spline in the strict sense, but very close unless a small smoothing parameter (or large \( df \)) is used.

\textbf{Author(s)}

\( R \) implementation by B. D. Ripley and Martin Maechler (\( spar/lambda \), etc).

This function is based on code in the GAMFIT Fortran program by T. Hastie and R. Tibshirani (\url{http://lib.stat.cmu.edu/general/}), which makes use of spline code by Finbarr O’Sullivan. Its design parallels the \textbf{smooth.spline} function of Chambers & Hastie (1992).

\textbf{References}


\textbf{See Also}

\textbf{predict.smooth.spline} for evaluating the spline and its derivatives.
smoothEnds

Examples

```r
attach(cars)
plot(speed, dist, main = "data(cars) & smoothing splines")
cars.spl <- smooth.spline(speed, dist)
(cars.spl)
## This example has duplicate points, so avoid cv=TRUE
lines(cars.spl, col = "blue")
lines(smooth.spline(speed, dist, df=10), lty=2, col = "red")
legend(5,120,c(paste("default [C.V.] => df =",round(cars.spl$df,1)),
  "s( * , df = 10)"), col = c("blue","red"), lty = 1:2,
  bg='bisque')
detach()

## Residual (Tukey Anscombe) plot:
plot(residuals(cars.spl) ~ fitted(cars.spl))
abline(h = 0, col="gray")

## consistency check:
stopifnot(all.equal(cars$dist,
  fitted(cars.spl) + residuals(cars.spl)))

##-- artificial example
y18 <- c(1:3,5,4,7:3,2*(2:5),rep(10,4))
xx <- seq(1,length(y18), len=201)
(s2 <- smooth.spline(y18)) # GCV
(s02 <- smooth.spline(y18, spar = 0.2))
plot(y18, main=deparse(s2$call), col.main=2)
lines(s2, col = "gray"); lines(predict(s2, xx), col = 2)
lines(predict(s02, xx), col = 3); mtext(deparse(s02$call), col = 3)

## The following shows the problematic behavior of 'spar' searching:
(s2 <- smooth.spline(y18, con=list(trace=TRUE,tol=1e-6, low= -1.5)))
(s2m <- smooth.spline(y18, cv=TRUE, con=list(trace=TRUE,tol=1e-6, low= -1.5)))
## both above do quite similarly (Df = 8.5 +- 0.2)
```

smoothEnds

End Points Smoothing (for Running Medians)

Description

Smooth end points of a vector y using subsequently smaller medians and Tukey’s end point rule at the very end. (of odd span).

Usage

```r
smoothEnds(y, k = 3)
```

Arguments

- **y**: dependent variable to be smoothed (vector).
- **k**: width of largest median window; must be odd.
Details

smoothEnds is used to only do the “end point smoothing”, i.e., change at most the observations closer to the beginning/end than half the window k. The first and last value are computed using “Tukey’s end point rule”, i.e., sm[1] = median(y[1], sm[2], 3*sm[2] - 2*sm[3]).

Value

vector of smoothed values, the same length as y.

Author(s)

Martin Maechler

References


See Also

runmed(*, end.rule = "median") which calls smoothEnds().

Examples

y <- ys <- (-20:20)^2
y [c(1,10,21,41)] <- c(100, 30, 400, 470)
s7k <- runmed(y,7, end = "keep")
s7. <- runmed(y,7, end = "const")
s7m <- runmed(y,7)
col3 <- c("midnightblue","blue","steelblue")
plot(y, main = "Running Medians -- runmed(*, k=7, end.rule = X)")
lines(ys, col = "light gray")
matlines(cbind(s7k,s7.,s7m), lwd= 1.5, lty = 1, col = col3)
legend(1,470, paste("end.rule",c("keep","constant","median"),sep=" = "),
       col = col3, lwd = 1.5, lty = 1)
stopifnot(identical(s7m, smoothEnds(s7k, 7)))
**Arguments**

- **x**
  - A numeric vector or an expression that will evaluate in `data` to a numeric vector
- **y**
  - A numeric vector or an expression that will evaluate in `data` to a numeric vector
- **data**
  - An optional data frame in which to evaluate expressions for `x` and `y`, if they are given as expressions

**Value**

A sortedXyData object. This is a data frame with exactly two numeric columns, named `x` and `y`. The rows are sorted so the `x` column is in increasing order. Duplicate `x` values are eliminated by averaging the corresponding `y` values.

**Author(s)**

Jose Pinheiro and Douglas Bates

**See Also**

- `selfStart`
- `NLSstClosestX`
- `NLSstLfAsymptote`
- `NLSstRtAsymptote`

**Examples**

```r
DNase.2 <- DNase[ DNase$Run == "2", ]
sortedXyData( expression(log(conc)), expression(density), DNase.2 )
```

---

**Description**

Fits an AR model to `x` (or uses the existing fit) and computes (and by default plots) the spectral density of the fitted model.

**Usage**

```r
spec.ar(x, n.freq, order = NULL, plot = TRUE, na.action = na.fail, method = "yule-walker", ...)
```

**Arguments**

- **x**
  - A univariate (not yet:or multivariate) time series or the result of a fit by `ar`.
- **n.freq**
  - The number of points at which to plot.
- **order**
  - The order of the AR model to be fitted. If omitted, the order is chosen by AIC.
- **plot**
  - Plot the periodogram?
- **na.action**
  - NA action function.
- **method**
  - Method for `ar` fit.
- **...**
  - Graphical arguments passed to `plot.spec`. 
Value

An object of class "spec". The result is returned invisibly if plot is true.

Warning

Some authors, for example Thomson (1990), warn strongly that AR spectra can be misleading.

Note

The multivariate case is not yet implemented.

References


See Also

ar, spectrum.

Examples

spec.ar(lh)

spec.ar(ldeaths)
spec.ar(ldeaths, method="burg")

---

**spec.pgram**

*Estimate Spectral Density of a Time Series by a Smoothed Periodogram*

**Description**

`spec.pgram` calculates the periodogram using a fast Fourier transform, and optionally smooths the result with a series of modified Daniell smoothers (moving averages giving half weight to the end values).

**Usage**

```r
spec.pgram(x, spans = NULL, kernel, taper = 0.1,
          pad = 0, fast = TRUE, demean = FALSE, detrend = TRUE,
          plot = TRUE, na.action = na.fail, ...)
```
Arguments

- **x**: univariate or multivariate time series.
- **spans**: vector of odd integers giving the widths of modified Daniell smoothers to be used to smooth the periodogram.
- **kernel**: alternatively, a kernel smoother of class "tskernel".
- **taper**: proportion of data to taper. A split cosine bell taper is applied to this proportion of the data at the beginning and end of the series.
- **pad**: proportion of data to pad. Zeros are added to the end of the series to increase its length by the proportion pad.
- **fast**: logical; if TRUE, pad the series to a highly composite length.
- **demean**: logical. If TRUE, subtract the mean of the series.
- **detrend**: logical. If TRUE, remove a linear trend from the series. This will also remove the mean.
- **plot**: plot the periodogram?
- **na.action**: NA action function.
- **...**: graphical arguments passed to `plot.spec`.

Details

The raw periodogram is not a consistent estimator of the spectral density, but adjacent values are asymptotically independent. Hence a consistent estimator can be derived by smoothing the raw periodogram, assuming that the spectral density is smooth.

The series will be automatically padded with zeros until the series length is a highly composite number in order to help the Fast Fourier Transform. This is controlled by the `fast` and not the `pad` argument.

The periodogram at zero is in theory zero as the mean of the series is removed (but this may be affected by tapering): it is replaced by an interpolation of adjacent values during smoothing, and no value is returned for that frequency.

Value

A list object of class "spec" (see `spectrum`) with the following additional components:

- **kernel**: The kernel argument, or the kernel constructed from `spans`.
- **df**: The distribution of the spectral density estimate can be approximated by a chi square distribution with `df` degrees of freedom.
- **bandwidth**: The equivalent bandwidth of the kernel smoother as defined by Bloomfield (1976, page 201).
- **taper**: The value of the `taper` argument.
- **pad**: The value of the `pad` argument.
- **detrend**: The value of the `detrend` argument.
- **demean**: The value of the `demean` argument.

The result is returned invisibly if `plot` is true.

Author(s)

Originally Martyn Plummer; kernel smoothing by Adrian Trapletti, synthesis by B.D. Ripley
References


See Also

`spectrum, spec.taper, plot.spec.fft`

Examples

```r
## Examples from Venables & Ripley
spectrum(ldeaths)
spectrum(ldeaths, spans = c(3,5))
spectrum(ldeaths, spans = c(5,7))
spectrum(mdeaths, spans = c(3,3))
spectrum(fdeaths, spans = c(3,3))

## bivariate example
mfdeaths.spc <- spec.pgram(ts.union(mdeaths, fdeaths), spans = c(3,3))
# plots marginal spectra: now plot coherency and phase
plot(mfdeaths.spc, plot.type = "coherency")
plot(mfdeaths.spc, plot.type = "phase")

## now impose a lack of alignment
mfdeaths.spc <- spec.pgram(ts.intersect(mdeaths, lag(fdeaths, 4)),
    spans = c(3,3), plot = FALSE)
plot(mfdeaths.spc, plot.type = "coherency")
plot(mfdeaths.spc, plot.type = "phase")

stocks.spc <- spectrum(EuStockMarkets, kernel("daniell", c(30,50)),
    plot = FALSE)
plot(stocks.spc, plot.type = "marginal") # the default type
plot(stocks.spc, plot.type = "coherency")
plot(stocks.spc, plot.type = "phase")

sales.spc <- spectrum(ts.union(BJsales, BJsales.lead),
    kernel("modified.daniell", c(5,7)))
plot(sales.spc, plot.type = "coherency")
plot(sales.spc, plot.type = "phase")
```

---

**spec.taper**

*Taper a Time Series by a Cosine Bell*

Description

Apply a cosine-bell taper to a time series.

Usage

`spec.taper(x, p = 0.1)`
The \textit{spectrum} function estimates the spectral density of a time series.

\subsection*{Description}

The \textit{spectrum} function estimates the spectral density of a time series.

\subsection*{Usage}

\begin{verbatim}
spectrum(x, ..., method = c("pgram", "ar"))
\end{verbatim}

\subsection*{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} A univariate or multivariate time series.
  \item \texttt{method} \hspace{1cm} String specifying the method used to estimate the spectral density. Allowed methods are "pgram" (the default) and "ar".
  \item \texttt{...} \hspace{1cm} Further arguments to specific spec methods or plot.spec.
\end{itemize}

\subsection*{Details}

\texttt{spectrum} is a wrapper function which calls the methods \texttt{spec.pgram} and \texttt{spec.ar}.

The spectrum here is defined with scaling \(1/\text{frequency}(x)\), following S-PLUS. This makes the spectral density a density over the range \((-\text{frequency}(x)/2, +\text{frequency}(x)/2)\), whereas a more common scaling is \(2\pi\) and range \((-0.5, 0.5]\) (e.g., Bloomfield) or 1 and range \((-\pi, \pi]\).

If available, a confidence interval will be plotted by \texttt{plot.spec}; this is asymmetric, and the width of the centre mark indicates the equivalent bandwidth.
Value

An object of class "spec", which is a list containing at least the following components:

freq
vector of frequencies at which the spectral density is estimated. (Possibly approximate Fourier frequencies.) The units are the reciprocal of cycles per unit time (and not per observation spacing): see Details below.

spec
Vector (for univariate series) or matrix (for multivariate series) of estimates of the spectral density at frequencies corresponding to freq.

coh
NULL for univariate series. For multivariate time series, a matrix containing the squared coherency between different series. Column $i + (j - 1) \times (j - 2)/2$ of coh contains the squared coherency between columns $i$ and $j$ of x, where $i < j$.

phase
NULL for univariate series. For multivariate time series a matrix containing the cross-spectrum phase between different series. The format is the same as coh.

series
The name of the time series.

snames
For multivariate input, the names of the component series.

method
The method used to calculate the spectrum.

The result is returned invisibly if plot is true.

Note

The default plot for objects of class "spec" is quite complex, including an error bar and default title, subtitle and axis labels. The defaults can all be overridden by supplying the appropriate graphical parameters.

Author(s)

Martyn Plummer, B.D. Ripley

References


See Also

spec.ar, spec.pgram; plot.spec.

Examples

```r
## Examples from Venables & Ripley
## spec.pgram
par(mfrow=c(2,2))
spectrum(lh)
spectrum(lh, spans=3)
spectrum(lh, spans=c(3,3))
spectrum(lh, spans=c(3,5))
```
splinefun

spectrum(ldeaths)
spectrum(ldeaths, spans=c(3,3))
spectrum(ldeaths, spans=c(3,5))
spectrum(ldeaths, spans=c(5,7))
spectrum(ldeaths, spans=c(5,7), log="dB", ci=0.8)

# for multivariate examples see the help for spec.pgram

## spec.ar
spectrum(lh, method="ar")
spectrum(ldeaths, method="ar")

---

splinefun

### Interpolating Splines

**Description**

Perform cubic spline interpolation of given data points, returning either a list of points obtained by the interpolation or a function performing the interpolation.

**Usage**

```r
splinefun(x, y = NULL, method = "fmm")
```

```r
spline(x, y = NULL, n = 3*length(x), method = "fmm",
      xmin = min(x), xmax = max(x))
```

**Arguments**

- **x, y** vectors giving the coordinates of the points to be interpolated. Alternatively a single plotting structure can be specified: see `xy.coords`.
- **method** specifies the type of spline to be used. Possible values are "fmm", "natural" and "periodic".
- **n** interpolation takes place at n equally spaced points spanning the interval [xmin, xmax].
- **xmin** left-hand endpoint of the interpolation interval.
- **xmax** right-hand endpoint of the interpolation interval.

**Details**

The inputs can contain missing values which are deleted, so at least one complete (x, y) pair is required. If method = "fmm", the spline used is that of Forsythe, Malcolm and Moler (an exact cubic is fitted through the four points at each end of the data, and this is used to determine the end conditions). Natural splines are used when method = "natural", and periodic splines when method = "periodic".

These interpolation splines can also be used for extrapolation, that is prediction at points outside the range of x. Extrapolation makes little sense for method = "fmm"; for natural splines it is linear using the slope of the interpolating curve at the nearest data point.
Value

spline returns a list containing components \( x \) and \( y \) which give the ordinates where interpolation took place and the interpolated values.

splinefun returns a function which will perform cubic spline interpolation of the given data points. This is often more useful than spline.

References


See Also

approx and approxfun for constant and linear interpolation.

Package splines, especially interpSpline and periodicSpline for interpolation splines. That package also generates spline bases that can be used for regression splines.

smooth.spline for smoothing splines.

Examples

```r
op <- par(mfrow = c(2,1), mgp = c(2,.8,0), mar = .1+c(3,3,3,1))
n <- 9
x <- 1:n
y <- rnorm(n)
plot(x, y, main = paste("spline\[fun\](.) through", n, "points"))
lines(spline(x, y))
lines(spline(x, y, n = 201), col = 2)

y <- (x-6)^2
plot(x, y, main = "spline(.) -- 3 methods")
lines(spline(x, y, n = 201), col = 2)
lines(spline(x, y, n = 201, method = "natural"), col = 3)
lines(spline(x, y, n = 201, method = "periodic"), col = 4)
legend(6,25, c("fmm","natural","periodic"), col=2:4, lty=1)

f <- splinefun(x, y)
lz <- eval(expression(z), envir = environment(f))
curve(f(x), 1, 10, col = "green", lwd = 1.5)
points(lz, col = "purple", cex = 2)
par(op)
```

SSasymp

Asymptotic Regression Model

Description

This selfStart model evaluates the asymptotic regression function and its gradient. It has an initial attribute that will evaluate initial estimates of the parameters Asym, R0, and lrc for a given set of data.
Usage

SSasymp(input, Asym, R0, lrc)

Arguments

input  
a numeric vector of values at which to evaluate the model.
Asym  
a numeric parameter representing the horizontal asymptote on the right side (very large values of input).
R0  
a numeric parameter representing the response when input is zero.
lrc  
a numeric parameter representing the natural logarithm of the rate constant.

Value

a numeric vector of the same length as input. It is the value of the expression Asym+(R0-Asym)\times\exp(-\exp(lrc)\times\text{input}). If all of the arguments Asym, R0, and lrc are names of objects, the gradient matrix with respect to these names is attached as an attribute named gradient.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

nls, selfStart

Examples

Lob.329 <- Loblolly[Loblolly$Seed == "329",]
SSasymp(Lob.329$age, 100, -8.5, -3.2) # response only
Asym <- 100
resp0 <- -8.5
lrc <- -3.2
SSasymp(Lob.329$age, Asym, resp0, lrc) # response and gradient
getInitial(height ~ SSasymp(age, Asym, resp0, lrc), data = Lob.329) # Initial values are in fact the converged values
fm1 <- nls(height ~ SSasymp(age, Asym, resp0, lrc), data = Lob.329)
summary(fm1)
SSasympOrig

Asymptotic Regression Model through the Origin

Description

This selfStart model evaluates the asymptotic regression function through the origin and its gradient. It has an initial attribute that will evaluate initial estimates of the parameters Asym and lrc for a given set of data.

Usage

SSasympOrig(input, Asym, lrc)

Arguments

input a numeric vector of values at which to evaluate the model.
Asym a numeric parameter representing the horizontal asymptote.
lrc a numeric parameter representing the natural logarithm of the rate constant.
Value

A numeric vector of the same length as input. It is the value of the expression \( \text{Asym} \cdot (1 - \exp(-\exp(\text{lrc}) \cdot \text{input})) \). If all of the arguments Asym and lrc are names of objects, the gradient matrix with respect to these names is attached as an attribute named gradient.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

nls, selfStart

Examples

```r
Lob.329 <- Loblolly[ Loblolly$Seed == "329", ]
SSasymOrig( Lob.329$age, 100, -3.2 ) # response only
Asym <- 100; lrc <- -3.2
SSasymOrig( Lob.329$age, Asym, lrc ) # response and gradient
getInitial(height ~ SSasymOrig(age, Asym, lrc), data = Lob.329)
## Initial values are in fact the converged values
fm1 <- nls(height ~ SSasymOrig( age, Asym, lrc), data = Lob.329)
summary(fm1)
```

SSbiexp

Biexponential model

Description

This selfStart model evaluates the biexponential model function and its gradient. It has an initial attribute that creates initial estimates of the parameters A1, lrc1, A2, and lrc2.

Usage

SSbiexp(input, A1, lrc1, A2, lrc2)

Arguments

- **input**: a numeric vector of values at which to evaluate the model.
- **A1**: a numeric parameter representing the multiplier of the first exponential.
- **lrc1**: a numeric parameter representing the natural logarithm of the rate constant of the first exponential.
- **A2**: a numeric parameter representing the multiplier of the second exponential.
- **lrc2**: a numeric parameter representing the natural logarithm of the rate constant of the second exponential.
Value

a numeric vector of the same length as input. It is the value of the expression $A_1 \times \exp(-\exp(lrc_1) \times \text{input}) + A_2 \times \exp(-\exp(lrc_2) \times \text{input})$. If all of the arguments $A_1$, $lrc_1$, $A_2$, and $lrc_2$ are names of objects, the gradient matrix with respect to these names is attached as an attribute named gradient.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

nls, selfStart

Examples

```r
Indo.1 <- Indometh[Indometh$Subject == 1, ]
SSbiexp( Indo.1$time, 3, 1, 0.6, -1.3 ) # response only
A1 <- 3; lrc1 <- 1; A2 <- 0.6; lrc2 <- -1.3
SSbiexp( Indo.1$time, A1, lrc1, A2, lrc2 ) # response and gradient
getInitial(conc ~ SSbiexp(time, A1, lrc1, A2, lrc2), data = Indo.1)
## Initial values are in fact the converged values
fml <- nls(conc ~ SSbiexp(time, A1, lrc1, A2, lrc2), data = Indo.1)
summary(fml)
```

SSD

### SSD Matrix and Estimated Variance Matrix in Multivariate Models

Description

Functions to compute matrix of residual sums of squares and products, or the estimated variance matrix for multivariate linear models.

Usage

```r
# S3 method for class 'mlm'
SSD(object, ...)

# S3 methods for class 'SSD' and 'mlm'
estVar(object, ...)
```

Arguments

- **object**: object of class "mlm", or "SSD" in the case of estVar.
- **...**: Unused
Value

SSD() returns a list of class "SSD" containing the following components

SSD: The residual sums of squares and products matrix
df: Degrees of freedom
call: Copied from object

estVar returns a matrix with the estimated variances and covariances.

See Also

mauchley.test, anova.mlm

Examples

# Lifted from Baron+Li:
# "Notes on the use of R for psychology experiments and questionnaires"
# Maxwell and Delaney, p. 497
reacttime <- matrix(c(
  420, 420, 480, 480, 600, 780,
  420, 480, 480, 360, 480, 600,
  480, 480, 540, 660, 780, 780,
  420, 540, 540, 480, 780, 900,
  540, 660, 540, 480, 660, 720,
  360, 420, 360, 480, 540,
  480, 480, 600, 540, 720, 840,
  480, 600, 660, 20, 720, 900,
  540, 600, 540, 480, 720, 800,
  480, 420, 540, 540, 660, 780),
ncol = 6, byrow = TRUE,
dimnames=list(subj=1:10,
  cond=c("deg0NA", "deg4NA", "deg8NA",
  "deg0NP", "deg4NP", "deg8NP")))

mlmfit <- lm(reacttime~1)
SSD(mlmfit)
estVar(mlmfit)

SSfol

First-order Compartment Model

Description

This selfStart model evaluates the first-order compartment function and its gradient. It has an initial attribute that creates initial estimates of the parameters lKe, lKa, and lCl.

Usage

SSfol(Dose, input, lKe, lKa, lCl)
Arguments

Dose a numeric value representing the initial dose.
input a numeric vector at which to evaluate the model.
lKe a numeric parameter representing the natural logarithm of the elimination rate constant.
lKa a numeric parameter representing the natural logarithm of the absorption rate constant.
lCl a numeric parameter representing the natural logarithm of the clearance.

Value

a numeric vector of the same length as input, which is the value of the expression Dose \times \exp(lKe+lKa-lCl) \times (\exp(-\exp(lKe) \times \text{input})-\exp(-\exp(lKa) \times \text{input})) / (\exp(lKa)-\exp(lKe))

If all of the arguments lKe, lKa, and lCl are names of objects, the gradient matrix with respect to these names is attached as an attribute named gradient.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

nls, selfStart

Examples

Theoph.1 <- Theoph[ Theoph$Subject == 1, ]
SSfol( Theoph.1$Dose, Theoph.1$Time, -2.5, 0.5, -3 ) # response only
lKe <- -2.5; lKa <- 0.5; lCl <- -3
SSfol( Theoph.1$Dose, Theoph.1$Time, lKe, lKa, lCl ) # response and gradient
getInitial(conc ~ SSfol(Dose, Time, lKe, lKa, lCl), data = Theoph.1)
## Initial values are in fact the converged values
fml <- nls(conc ~ SSfol(Dose, Time, lKe, lKa, lCl), data = Theoph.1)
summary(fml)

SSfpl

Four-parameter Logistic Model

Description

This selfStart model evaluates the four-parameter logistic function and its gradient. It has an initial attribute that will evaluate initial estimates of the parameters A, B, xmid, and scal for a given set of data.

Usage

SSfpl(input, A, B, xmid, scal)

SSfpl

Four-parameter Logistic Model
SSgompertz

Gompertz Growth Model

Description

This `selfStart` model evaluates the Gompertz growth model and its gradient. It has an initial attribute that creates initial estimates of the parameters `Asym`, `b2`, and `b3`.

Usage

```r
SSgompertz(x, Asym, b2, b3)
```
**SSlogis**

**Logistic Model**

**Description**

This `selfStart` model evaluates the logistic function and its gradient. It has an initial attribute that creates initial estimates of the parameters `Asym`, `xmid`, and `scal`.

**Usage**

```r
SSlogis(input, Asym, xmid, scal)
```

**Arguments**

- `input`  
  a numeric vector of values at which to evaluate the model.

- `Asym`  
  a numeric parameter representing the asymptote.

- `xmid`  
  a numeric parameter representing the x value at the inflection point of the curve. The value of `SSlogis` will be `Asym/2` at `xmid`.

- `scal`  
  a numeric scale parameter on the input axis.

**Examples**

```r
DNase.1 <- subset(DNase, Run == 1)
SSlogis(log(DNase.1$conc), 4.5, 2.3, 0.7) # response only
Asym <- 4.5; b2 <- 2.3; b3 <- 0.7
SSgompertz(log(DNase.1$conc), Asym, b2, b3 ) # response and gradient
getInitial(density ~ SSgompertz(log(conc), Asym, b2, b3),
  data = DNase.1)
## Initial values are in fact the converged values
fm1 <- nls(density ~ SSgompertz(log(conc), Asym, b2, b3),
  data = DNase.1)
summary(fm1)
```
**SSmicmen**

**Value**

A numeric vector of the same length as input. It is the value of the expression Asym/(1+exp((xmid-input)/scal)). If all of the arguments Asym, xmid, and scal are names of objects the gradient matrix with respect to these names is attached as an attribute named gradient.

**Author(s)**

Jose Pinheiro and Douglas Bates

**See Also**

`nls`, `selfStart`

**Examples**

```r
Chick1 <- ChickWeight[ChickWeight$Chick == 1, ]
SSlogis( Chick1$Time, 368, 14, 6 ) # response only
Asym <- 368; xmid <- 14; scal <- 6
SSlogis( Chick1$Time, Asym, xmid, scal ) # response and gradient
getInitial(weight ~ SSlogis(Time, Asym, xmid, scal), data = Chick1)
### Initial values are in fact the converged values
fml <- nls(weight ~ SSlogis(Time, Asym, xmid, scal), data = Chick1)
summary(fml)
```

---

**SSmicmen**

*Michaelis-Menten Model*

**Description**

This `selfStart` model evaluates the Michaelis-Menten model and its gradient. It has an initial attribute that will evaluate initial estimates of the parameters Vm and K.

**Usage**

```r
SSmicmen(input, Vm, K)
```

**Arguments**

- **input**: A numeric vector of values at which to evaluate the model.
- **Vm**: A numeric parameter representing the maximum value of the response.
- **K**: A numeric parameter representing the input value at which half the maximum response is attained. In the field of enzyme kinetics this is called the Michaelis parameter.

**Value**

A numeric vector of the same length as input. It is the value of the expression Vm*input/(K+input). If both the arguments Vm and K are names of objects, the gradient matrix with respect to these names is attached as an attribute named gradient.
SSweibull

Description

This \texttt{selfStart} model evaluates the Weibull model for growth curve data and its gradient. It has an \texttt{initial} attribute that will evaluate initial estimates of the parameters \texttt{Asym}, \texttt{Drop}, \texttt{lrc}, and \texttt{pwr} for a given set of data.

Usage

\begin{verbatim}
SSweibull(x, Asym, Drop, lrc, pwr)
\end{verbatim}

Arguments

- \texttt{x} a numeric vector of values at which to evaluate the model.
- \texttt{Asym} a numeric parameter representing the horizontal asymptote on the right side (very small values of \texttt{x}).
- \texttt{Drop} a numeric parameter representing the change from \texttt{Asym} to the \texttt{y} intercept.
- \texttt{lrc} a numeric parameter representing the natural logarithm of the rate constant.
- \texttt{pwr} a numeric parameter representing the power to which \texttt{x} is raised.

Details

This model is a generalization of the \texttt{SSasymp} model in that it reduces to \texttt{SSasymp} when \texttt{pwr} is unity.
Value

A numeric vector of the same length as \( x \). It is the value of the expression \( \text{Asym} \cdot \text{Drop} \cdot \exp(-\exp(\text{lrc}) \cdot x^{\text{pwr}}) \). If all of the arguments Asym, Drop, lrc, and pwr are names of objects, the gradient matrix with respect to these names is attached as an attribute named gradient.

Author(s)

Douglas Bates

References


See Also

nls, selfStart, SSasymp

Examples

```r
Chick.6 <- subset(ChickWeight, (Chick == 6) & (Time > 0))
SSweibull(Chick.6$Time, 160, 115, -5.5, 2.5) # response only
Asym <- 160; Drop <- 115; lrc <- -5.5; pwr <- 2.5
SSweibull(Chick.6$Time, Asym, Drop, lrc, pwr) # response and gradient
getInitial(weight ~ SSweibull(Time, Asym, Drop, lrc, pwr), data = Chick.6)
# Initial values are in fact the converged values
fm1 <- nls(weight ~ SSweibull(Time, Asym, Drop, lrc, pwr), data = Chick.6)
summary(fm1)
```

Description

Extract and encode the times the first and last observations were taken. Provided only for compatibility with S version 2.

Usage

```r
start(x, ...)  
end(x, ...)
```

Arguments

- `x` a univariate or multivariate time-series, or a vector or matrix.
- `...` extra arguments for future methods.

Details

These are generic functions, which will use the \texttt{tsp} attribute of \( x \) if it exists. Their default methods decode the start time from the original time units, so that for a monthly series 1995.5 is represented as \( c(1995, 7) \). For a series of frequency \( f \), time \( n+i/f \) is presented as \( c(n, i+1) \) (even for \( i = 0 \) and \( f = 1 \)).
Warning

The representation used by start and end has no meaning unless the frequency is supplied.

See Also

ts.time,tsp.

stat.anova

GLM Anova Statistics

Description

This is a utility function, used in lm and glm methods for anova(..., test != NULL) and should not be used by the average user.

Usage

stat.anova(table, test = c("Chisq", "F", "Cp"), scale, df.scale, n)

Arguments

table numeric matrix as results from anova.glm(..., test=NULL).
test a character string, matching one of "Chisq", "F" or "Cp".
scale a weighted residual sum of squares.
df.scale degrees of freedom corresponding to scale.
n number of observations.

Value

A matrix which is the original table, augmented by a column of test statistics, depending on the test argument.

References


See Also

anova.lm, anova.glm.

Examples

##-- Continued from '?glm':
print(ag <- anova(glm.D93))
stat.anova(ag$table, test = "Cp",
            scale = sum(resid(glm.D93, "pearson")^2)/4, df = 4, n = 9)
The R Stats Package

**Description**

R statistical functions

**Details**

This package contains functions for statistical calculations and random number generation. For a complete list of functions, use `library(help="stats")`.

**Author(s)**

R Development Core Team and contributors worldwide

Maintainer: R Core Team (R-core@r-project.org)

---

Choose a model by AIC in a Stepwise Algorithm

**Description**

Select a formula-based model by AIC.

**Usage**

```r
step(object, scope, scale = 0, direction = c("both", "backward", "forward"), trace = 1, keep = NULL, steps = 1000, k = 2, ...)
```

**Arguments**

- `object` an object representing a model of an appropriate class (mainly "lm" and "glm"). This is used as the initial model in the stepwise search.
- `scope` defines the range of models examined in the stepwise search. This should be either a single formula, or a list containing components `upper` and `lower`, both formulae. See the details for how to specify the formulae and how they are used.
- `scale` used in the definition of the AIC statistic for selecting the models, currently only for `lm`, `aov` and `glm` models.
- `direction` the mode of stepwise search, can be one of "both", "backward", or "forward", with a default of "both". If the `scope` argument is missing the default for `direction` is "backward".
- `trace` if positive, information is printed during the running of `step`. Larger values may give more detailed information.
keep is a filter function whose input is a fitted model object and the associated AIC statistic, and whose output is arbitrary. Typically keep will select a subset of the components of the object and return them. The default is not to keep anything.

steps is the maximum number of steps to be considered. The default is 1000 (essentially as many as required). It is typically used to stop the process early.

k is the multiple of the number of degrees of freedom used for the penalty. Only \( k = 2 \) gives the genuine AIC: \( k = \log(n) \) is sometimes referred to as BIC or SBC.

... any additional arguments to extractAIC.

Details

step uses add1 and drop1 repeatedly; it will work for any method for which they work, and that is determined by having a valid method for extractAIC. When the additive constant can be chosen so that AIC is equal to Mallows’ \( C_p \), this is done and the tables are labelled appropriately.

The set of models searched is determined by the scope argument. The right-hand-side of its lower component is always included in the model, and right-hand-side of the model is included in the upper component. If scope is a single formula, it specifies the upper component, and the lower model is empty. If scope is missing, the initial model is used as the upper model.

Models specified by scope can be templates to update object as used by update.formula. So using . in a scope formula means ‘what is already there’, with .^2 indicating all interactions of existing terms.

There is a potential problem in using glm fits with a variable scale, as in that case the deviance is not simply related to the maximized log-likelihood. The "glm" method for function extractAIC makes the appropriate adjustment for a gaussian family, but may need to be amended for other cases. (The binomial and poisson families have fixed scale by default and do not correspond to a particular maximum-likelihood problem for variable scale.)

Value

the stepwise-selected model is returned, with up to two additional components. There is an "anova" component corresponding to the steps taken in the search, as well as a "keep" component if the keep= argument was supplied in the call. The "Resid. Dev" column of the analysis of deviance table refers to a constant minus twice the maximized log likelihood: it will be a deviance only in cases where a saturated model is well-defined (thus excluding lm, aov and survreg fits, for example).

Warning

The model fitting must apply the models to the same dataset. This may be a problem if there are missing values and R’s default of na.action = na.omit is used. We suggest you remove the missing values first.

Note

This function differs considerably from the function in S, which uses a number of approximations and does not compute the correct AIC.

This is a minimal implementation. Use stepAIC in package MASS for a wider range of object classes.
Author(s)

B. D. Ripley: step is a slightly simplified version of stepAIC in package MASS (Venables & Ripley, 2002 and earlier editions).

The idea of a step function follows that described in Hastie & Pregibon (1992); but the implementation in R is more general.

References


See Also

stepAIC in MASS, add1, drop1

Examples

example(lm)
step(lm.D9)

summary(lm1 <- lm(Fertility ~ ., data = swiss))
slm1 <- step(lm1)
summary(slm1)
slm1$anova

stepfun

Step Function Class

Description

Given the vectors \((x_1, \ldots, x_n)\) and \((y_0, y_1, \ldots, y_n)\) (one value more!), \texttt{stepfun(x, y, \ldots)} returns an interpolating “step” function, say \(fn\). I.e., \(fn(t) = c_i\) (constant) for \(t \in (x_i, x_{i+1})\) and at the abscissa values, if (by default) \texttt{right = FALSE}, \(fn(x_i) = y_i\) and for \texttt{right = TRUE}, \(fn(x_i) = y_{i-1}\), for \(i = 1, \ldots, n\).

The value of the constant \(c_i\) above depends on the “continuity” parameter \(f\). For the default, \texttt{right = FALSE}, \(f = 0\), \(fn\) is a “cadlag” function, i.e., continuous at right, limit (“the point”) at left. In general, \(c_i\) is interpolated in between the neighbouring \(y\) values, \(c_i = (1 - f)y_i + f \cdot y_{i+1}\). Therefore, for non-0 values of \(f\), \(fn\) may no longer be a proper step function, since it can be discontinuous from both sides, unless \texttt{right = TRUE}, \(f = 1\) which is right-continuous.

Usage

\texttt{stepfun(x, y, f = as.numeric(right), ties = "ordered", right = FALSE)}

\texttt{is.stepfun(x)}
\texttt{knots(Fn, \ldots)}
\texttt{as.stepfun(x, \ldots)}

## S3 method for class 'stepfun':
\begin{verbatim}
print(x, digits = getOption("digits") - 2, ...) 

## S3 method for class 'stepfun'
summary(object, ...)
\end{verbatim}

**Arguments**

- \texttt{x} \hfill numeric vector giving the knots or jump locations of the step function for \texttt{stepfun()}. For the other functions, \texttt{x} is as \texttt{object} below.
- \texttt{y} \hfill numeric vector one longer than \texttt{x}, giving the heights of the function values \emph{between} the \texttt{x} values.
- \texttt{f} \hfill a number between 0 and 1, indicating how interpolation outside the given \texttt{x} values should happen. See \texttt{approxfun}.
- \texttt{ties} \hfill Handling of tied \texttt{x} values. Either a function or the string "ordered". See \texttt{approxfun}.
- \texttt{right} \hfill logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa.
- \texttt{Fn, object} \hfill an \texttt{R} object inheriting from "stepfun".
- \texttt{digits} \hfill number of significant digits to use, see \texttt{print}.
- ... \hfill potentially further arguments (required by the generic).

**Value**

A function of class "stepfun", say \texttt{fn}. There are methods available for summarizing ("summary(.)"), representing ("print(.)") and plotting ("plot(.)", see \texttt{plot.stepfun}) "stepfun" objects.

The \texttt{environment} of \texttt{fn} contains all the information needed;

- "x", "y" \hfill the original arguments
- "n" \hfill number of knots (\texttt{x} values)
- "f" \hfill continuity parameter
- "yleft", "yright" \hfill the function values \emph{outside} the knots
- "method" \hfill (always == "constant", from \texttt{approxfun}(.)).

The knots are also available via \texttt{knots(fn)}.

**Author(s)**

Martin Maechler, (maechler@stat.math.ethz.ch) with some basic code from Thomas Lumley.

**See Also**

ecdf for empirical distribution functions as special step functions and \texttt{plot.stepfun} for plotting step functions.

\texttt{approxfun} and \texttt{splinefun}.
Examples

```r
y0 <- c(1, 2, 4, 3)
sfun0 <- stepfun(1:3, y0, f = 0)
sfun.2 <- stepfun(1:3, y0, f = .2)
sfun1 <- stepfun(1:3, y0, f = 1)
sfun1c <- stepfun(1:3, y0, right = TRUE) # hence f = 1
sfun0
summary(sfun0)
sfun.2

## look at the internal structure:
unclass(sfun0)
ls(envir = environment(sfun0))

x0 <- seq(0.5, 3.5, by = 0.25)
rbind(x = x0, f.f0 = sfun0(x0), f.f02 = sfun.2(x0),
      f.f1 = sfun1(x0), f.f1c = sfun1c(x0))

## Identities:
stopifnot(identical(y0[-1], sfun0(1:3)), # right = FALSE
           identical(y0[-4], sfun1c(1:3))) # right = TRUE
```

---

### stl

#### Seasonal Decomposition of Time Series by Loess

**Description**

Decompose a time series into seasonal, trend and irregular components using loess, acronym STL.

**Usage**

```r
stl(x, s.window, s.degree = 0,
    t.window = NULL, t.degree = 1,
    l.window = nextodd(period), l.degree = t.degree,
    s.jump = ceiling(s.window/10),
    t.jump = ceiling(t.window/10),
    l.jump = ceiling(l.window/10),
    robust = FALSE,
    inner = if(robust) 1 else 2,
    outer = if(robust) 15 else 0,
    na.action = na.fail)
```

**Arguments**

- **x**: univariate time series to be decomposed. This should be an object of class "ts" with a frequency greater than one.
- **s.window**: either the character string "periodic" or the span (in lags) of the loess window for seasonal extraction, which should be odd. This has no default.
- **s.degree**: degree of locally-fitted polynomial in seasonal extraction. Should be zero or one.
t.window the span (in lags) of the loess window for trend extraction, which should be odd. If NULL, the default, \text{nextodd}(\text{ceiling}((1.5 \times \text{period}) / (1- (1.5/s.window)))) is taken.

t.degree degree of locally-fitted polynomial in trend extraction. Should be zero or one.

l.window the span (in lags) of the loess window of the low-pass filter used for each subseries. Defaults to the smallest odd integer greater than or equal to \text{frequency}(x) which is recommended since it prevents competition between the trend and seasonal components. If not an odd integer its given value is increased to the next odd one.

l.degree degree of locally-fitted polynomial for the subseries low-pass filter. Must be 0 or 1.

s.jump, t.jump, l.jump integers at least one to increase speed of the respective smoother. Linear interpolation happens between every \text{.jump}th value.

robust logical indicating if robust fitting be used in the \text{loess} procedure.

inner integer; the number of ‘inner’ (backfitting) iterations; usually very few (2) iterations suffice.

outer integer; the number of ‘outer’ robustness iterations.

na.action action on missing values.

Details

The seasonal component is found by \text{loess} smoothing the seasonal sub-series (the series of all January values,...); if s.window = "periodic" smoothing is effectively replaced by taking the mean. The seasonal values are removed, and the remainder smoothed to find the trend. The overall level is removed from the seasonal component and added to the trend component. This process is iterated a few times. The remainder component is the residuals from the seasonal plus trend fit.

Several methods for the resulting class "\text{stl}" objects, see \text{plot.stl}.

Value

\text{stl} returns an object of class "\text{stl}" with components

time.series a multiple time series with columns seasonal, trend and remainder.
weights the final robust weights (all one if fitting is not done robustly).
call the matched call.
win integer (length 3 vector) with the spans used for the "s", "t", and "l" smoothers.
deg integer (length 3) vector with the polynomial degrees for these smoothers.
jump integer (length 3) vector with the “jumps” (skips) used for these smoothers.
ni number of inner iterations
no number of outer robustness iterations

Note

This is similar to but not identical to the \text{stl} function in S-PLUS. The remainder component given by S-PLUS is the sum of the trend and remainder series from this function.
stlmethods

Author(s)

B.D. Ripley; Fortran code by Cleveland et al. (1990) from ‘netlib’.

References


See Also

plot.stl for stl methods; loess in package stats (which is not actually used in stl).

Examples

plot(stl(nottem, "per"))
plot(stl(nottem, s.win = 4, t.win = 50, t.jump = 1))

plot(stllc <- stl(log(co2), s.window=21))
summary(stllc)
## linear trend, strict period.
plot(stl(log(co2), s.window="per", t.window=1000))

## Two STL plotted side by side:
  stmd <- stl(mdeaths, s.w = "per") # un-robust
summary(stmR <- stl(mdeaths, s.w = "per", robust = TRUE))
op <- par(mar = c(0, 4, 0, 3), oma = c(6, 0, 4, 0), tck = -0.01, mfrow = c(nplot, 1),
  main = "stl(mdeaths, s.w = \"per\", robust = FALSE / TRUE )")
plot(stmR, main=pars=NULL)

# mark the 'outliers':
(iO <- which(stmR$weights < 1e-8)) # 10 were considered outliers
sts <- stmR$time.series
points(time(sts)[iO], .8* sts[,"remainder"][iO], pch = 4, col = "red")
par(op)# reset

stlmethods

Methods for STL Objects

Description

Methods for objects of class stl, typically the result of stl. The plot method does a multiple figure plot with some flexibility.

There are also (non-visible) print and summary methods.

Usage

## S3 method for class 'stl':
plot(x, labels = colnames(X),
  set.pars = list(mar = c(0, 6, 0, 6), oma = c(6, 0, 4, 0),
  tck = -0.01, mfrow = c(nplot, 1)),
  main = NULL, range.bars = TRUE, ...,
  col.range = "light gray")
Arguments

- `x` : `stl` object.
- `labels` : character of length 4 giving the names of the component time-series.
- `set.pars` : settings for `par(.)` when setting up the plot.
- `main` : plot main title.
- `range.bars` : logical indicating if each plot should have a bar at its right side which are of equal heights in user coordinates.
- `col.range` : colour to be used for the range bars, if plotted. Note this appears after `...` and so cannot be abbreviated.

See Also

`plot.ts` and `stl`, particularly for examples.

---

**StructTS**

**Fit Structural Time Series**

Description

Fit a structural model for a time series by maximum likelihood.

Usage

```r
StructTS(x, type = c("level", "trend", "BSM"), init = NULL,
          fixed = NULL, optim.control = NULL)
```

Arguments

- `x` : a univariate numeric time series. Missing values are allowed.
- `type` : the class of structural model. If omitted, a BSM is used for a time series with `frequency(x) > 1`, and a local trend model otherwise.
- `init` : initial values of the variance parameters.
- `fixed` : optional numeric vector of the same length as the total number of parameters. If supplied, only NA entries in `fixed` will be varied. Probably most useful for setting variances to zero.
- `optim.control` : List of control parameters for `optim`. Method "L-BFGS-B" is used.

Details

**Structural time series** models are (linear Gaussian) state-space models for (univariate) time series based on a decomposition of the series into a number of components. They are specified by a set of error variances, some of which may be zero.

The simplest model is the **local level** model specified by `type = "level"`. This has an underlying level \( \mu_t \) which evolves by

\[
\mu_{t+1} = \mu_t + \xi_t, \quad \xi_t \sim N(0, \sigma_\xi^2)
\]
The observations are
\[ x_t = \mu_t + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2_\epsilon) \]

There are two parameters, \( \sigma^2_\xi \) and \( \sigma^2_\epsilon \). It is an ARIMA(0,1,1) model, but with restrictions on the parameter set.

The local linear trend model, `type = "trend"`, has the same measurement equation, but with a time-varying slope in the dynamics for \( \mu_t \), given by
\[
\begin{align*}
\mu_{t+1} &= \mu_t + \nu_t + \xi_t, \quad \xi_t \sim N(0, \sigma^2_\xi) \\
\nu_{t+1} &= \nu_t + \zeta_t, \quad \zeta_t \sim N(0, \sigma^2_\zeta)
\end{align*}
\]

with three variance parameters. It is not uncommon to find \( \sigma^2_\xi = 0 \) (which reduces to the local level model) or \( \sigma^2_\zeta = 0 \), which ensures a smooth trend. This is a restricted ARIMA(0,2,2) model.

The basic structural model, `type = "BSM"`, is a local trend model with an additional seasonal component. Thus the measurement equation is
\[ x_t = \mu_t + \gamma_t + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2_\epsilon) \]

where \( \gamma_t \) is a seasonal component with dynamics
\[
\begin{align*}
\gamma_{t+1} &= -\gamma_t + \cdots + \gamma_{t-s+2} + \omega_t, \quad \omega_t \sim N(0, \sigma^2_\omega)
\end{align*}
\]

The boundary case \( \sigma^2_\omega = 0 \) corresponds to a deterministic (but arbitrary) seasonal pattern. (This is sometimes known as the ‘dummy variable’ version of the BSM.)

**Value**

A list of class "StructTS" with components:

- `coef` the estimated variances of the components.
- `loglik` the maximized log-likelihood. Note that as all these models are non-stationary this includes a diffuse prior for some observations and hence is not comparable with `arima` nor different types of structural models.
- `data` the time series \( x \).
- `residuals` the standardized residuals.
- `fitted` a multiple time series with one component for the level, slope and seasonal components, estimated contemporaneously (that is at time \( t \) and not at the end of the series).
- `call` the matched call.
- `series` the name of the series \( x \).
- `code` the convergence code returned by `optim`.
- `model, model0` Lists representing the Kalman Filter used in the fitting. See `KalmanLike`. `model0` is the initial state of the filter, `model` its final state.
- `xtsp` the tsp attributes of \( x \).

**Note**

Optimization of structural models is a lot harder than many of the references admit. For example, the AirPassengers data are considered in Brockwell & Davis (1996): their solution appears to be a local maximum, but nowhere near as good a fit as that produced by StructTS. It is quite common to find fits with one or more variances zero, and this can include \( \sigma^2_\epsilon \).
References


See Also

KalmanLike, tsSmooth.

Examples

```r
## see also JohnsonJohnson, Nile and AirPassengers

trees <- window(treering, start=0)
(fit <- StructTS(trees, type = "level"))
plot(trees)
lines(fitted(fit), col = "green")
tsdiag(fit)

(fit <- StructTS(log10(UKgas), type = "BSM"))
par(mfrow = c(4, 1))
plot(log10(UKgas))
plot(cbind(fitted(fit), resids=resid(fit)), main = "UK gas consumption")

## keep some parameters fixed; trace optimizer:
StructTS(log10(UKgas), type = "BSM", fixed = c(0.1, 0.001,NA,NA),
          optim.control = list(trace=TRUE))
```

summary.aov

*Summarize an Analysis of Variance Model*

Description

Summarize an analysis of variance model.

Usage

```r
## S3 method for class 'aov':
summary(object, intercept = FALSE, split,
         expand.split = TRUE, keep.zero.df = TRUE, ...)

## S3 method for class 'aovlist':
summary(object, ...)```
Arguments

object
intercept
split
expand.split
keep.zero.df
...

Value

An object of class c("summary.aov", "listof") or "summary.aovlist" respectively.

For a fits with a single stratum the result will be a list of ANOVA tables, one for each response (even if there is only one response): the tables are of class "anova" inheriting from class "data.frame". They have columns "Df", "Sum Sq", "Mean Sq", as well as "F value" and "Pr(>F)" if there are non-zero residual degrees of freedom. There is a row for each term in the model, plus one for "Residuals" if there are any.

For multistratum fits the return value is a list of such summaries, one for each stratum.

Note

The use of expand.split = TRUE is little tested: it is always possible to set it to FALSE and specify exactly all the splits required.

See Also

summary, model.tables, TukeyHSD

Examples

```r
N <- c(0,1,0,1,1,1,0,0,1,1,0,1,0,1,1,0,0,1,0,1,1,0,0,1,0,1,1,0,1,0,1,0,1,0,0,1,1,1,1,0)
P <- c(1,1,0,0,0,1,0,1,1,1,0,0,0,1,1,0,0,1,0,1,1,0,1,0,1,1,0,0,1,1,1,0,1,0,1,0,1,1,0)
K <- c(1,0,1,0,1,1,0,1,0,1,0,1,1,0,0,1,0,1,1,0,0,1,1,0,1,1,0,0,0,1,1,0,1,0,1,0,1,1,0)
yield <- c(49.5, 62.8, 46.8, 57.0, 59.8, 58.5, 55.5, 56.0, 62.8, 55.8, 69.5, 55.0, 62.0, 48.8, 45.5, 44.2, 52.0, 51.5, 49.8, 48.8, 57.2, 59.0, 53.2, 56.0)
npk <- data.frame(block=gl(6,4), N=factor(N), P=factor(P), K=factor(K), yield=yield)
(npk.aov <- aov(yield ~ block + N*P*K, npk) )
summary(npk.aov)
coefficients(npk.aov)

# Cochran and Cox (1957, p.164)
# 3x3 factorial with ordered factors, each is average of 12.
CC <- data.frame(  
  y = c(449, 413, 326, 409, 358, 291, 341, 278, 312)/12,  
P = ordered(gl(3, 3)), N = ordered(gl(3, 1, 9))  
)
CC.aov <- aov(y ~ N * P, data = CC, weights = rep(12, 9))
```
summary(CC.aov)

# Split both main effects into linear and quadratic parts.
summary(CC.aov, split = list(N = list(L = 1, Q = 2),
P = list(L = 1, Q = 2)))

# Split only the interaction
summary(CC.aov, split = list("N:P" = list(L.L = 1, Q = 2:4)))

# split on just one var
summary(CC.aov, split = list(P = list(lin = 1, quad = 2)))
summary(CC.aov, split = list(P = list(lin = 1, quad = 2)),
        expand.split=FALSE)

summary.glm  

**Summarizing Generalized Linear Model Fits**

**Description**

These functions are all *methods* for class *glm* or *summary.glm* objects.

**Usage**

```r
## S3 method for class 'glm':
summary(object, dispersion = NULL, correlation = FALSE,
symbolic.cor = FALSE, ...)

## S3 method for class 'summary.glm':
print(x, digits = max(3, getOption("digits") - 3),
symbolic.cor = x$symbolic.cor,
signif.stars = getOption("show.signif.stars"), ...)
```

**Arguments**

- `object` an object of class "glm", usually, a result of a call to `glm`.
- `x` an object of class "summary.glm", usually, a result of a call to `summary.glm`.
- `dispersion` the dispersion parameter for the fitting family. By default it is obtained from `object`.
- `correlation` logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
- `digits` the number of significant digits to use when printing.
- `symbolic.cor` logical. If TRUE, print the correlations in a symbolic form (see `symnum`) rather than as numbers.
- `signif.stars` logical. If TRUE, “significance stars” are printed for each coefficient.
- `...` further arguments passed to or from other methods.
Details

print.summary.glm tries to be smart about formatting the coefficients, standard errors, etc. and additionally gives "significance stars" if signif.stars is TRUE.

Aliased coefficients are omitted in the returned object but (as from R 1.8.0) restored by the print method.

Correlations are printed to two decimal places (or symbolically): to see the actual correlations print summary(object)$correlation directly.

Value

summary.glm returns an object of class "summary.glm", a list with components

call the component from object.
family the component from object.
deviance the component from object.
contrasts the component from object.
df.residual the component from object.
null.deviance the component from object.
df.null the component from object.
deviance.resid the deviance residuals: see residuals.glm.
coefficients the matrix of coefficients, standard errors, z-values and p-values. Aliased coefficients are omitted.
aliased named logical vector showing if the original coefficients are aliased.
dispersion either the supplied argument or the estimated dispersion if the latter in NULL
df a 3-vector of the rank of the model and the number of residual degrees of freedom, plus number of non-aliased coefficients.
cov.unscaled the unscaled (dispersion = 1) estimated covariance matrix of the estimated coefficients.
cov.scaled ditto, scaled by dispersion.
correlation (only if correlation is true.) The estimated correlations of the estimated coefficients.
symbolic.cor (only if correlation is true.) The value of the argument symbolic.cor.

See Also

glm, summary.

Examples

### --- Continuing the Example from '?glm':

summary(glm.D93)
summary.lm

Summarizing Linear Model Fits

Description

summary method for class "lm".

Usage

## S3 method for class 'lm':
summary(object, correlation = FALSE, symbolic.cor = FALSE, ...)

## S3 method for class 'summary.lm':
print(x, digits = max(3, getOption("digits") - 3),
symbolic.cor = x$symbolic.cor,
signif.stars = getOption("show.signif.stars"), ...)

Arguments

object an object of class "lm", usually, a result of a call to lm.
x an object of class "summary.lm", usually, a result of a call to summary.lm.
correlation logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
digits the number of significant digits to use when printing.
symbolic.cor logical. If TRUE, print the correlations in a symbolic form (see symnum) rather than as numbers.
signif.stars logical. If TRUE, "significance stars" are printed for each coefficient.
... further arguments passed to or from other methods.

Details

print.summary.lm tries to be smart about formatting the coefficients, standard errors, etc. and additionally gives "significance stars" if signif.stars is TRUE.

Correlations are printed to two decimal places (or symbolically): to see the actual correlations print summary(object)$correlation directly.

Value

The function summary.lm computes and returns a list of summary statistics of the fitted linear model given in object, using the components (list elements) "call" and "terms" from its argument, plus

residuals the weighted residuals, the usual residuals rescaled by the square root of the weights specified in the call to lm.

coefficients a p × 4 matrix with columns for the estimated coefficient, its standard error, t-statistic and corresponding (two-sided) p-value. Aliased coefficients are omitted.

aliased named logical vector showing if the original coefficients are aliased.
sigma the square root of the estimated variance of the random error

\[ \hat{\sigma}^2 = \frac{1}{n - p} \sum_i R_i^2, \]

where \( R_i \) is the \( i \)-th residual, residuals[i].

df degrees of freedom, a 3-vector \((p, n - p, p^*)\), the last being the number of non-aliased coefficients.

fstatistic (for models including non-intercept terms) a 3-vector with the value of the F-statistic with its numerator and denominator degrees of freedom.

r.squared \( R^2 \), the "fraction of variance explained by the model",

\[ R^2 = 1 - \frac{\sum_i R_i^2}{\sum_i (y_i - y^*)^2}, \]

where \( y^* \) is the mean of \( y_i \) if there is an intercept and zero otherwise.

adj.r.squared the above \( R^2 \) statistic "adjusted", penalizing for higher \( p \).

cov.unscaled a \( p \times p \) matrix of (unscaled) covariances of the \( \hat{\beta}_j \), \( j = 1, \ldots, p \).

correlation the correlation matrix corresponding to the above cov.unscaled, if correlation = TRUE is specified.

symbolic.cor (only if correlation is true.) The value of the argument symbolic.cor.

See Also

The model fitting function lm, summary.

Function coef will extract the matrix of coefficients with standard errors, t-statistics and p-values.

Examples

```r
##-- Continuing the lm(.) example:
coef(lm.D90)# the bare coefficients
sld90 <- summary(lm.D90 <- lm(weight ~ group -1))# omitting intercept
sld90
c coef(sld90)# much more
```

summary.manova  Summary Method for Multivariate Analysis of Variance

Description

A summary method for class "manova".

Usage

```r
## S3 method for class 'manova':
summary(object,
test = c("Pillai", "Wilks", "Hotelling-Lawley", "Roy"),
intercept = FALSE, ...)
```
Arguments

object: An object of class "manova" or an aov object with multiple responses.
test: The name of the test statistic to be used. Partial matching is used so the name can be abbreviated.
intercept: logical. If TRUE, the intercept term is included in the table.
... further arguments passed to or from other methods.

Details

The `summary.manova` method uses a multivariate test statistic for the summary table. Wilks’ statistic is most popular in the literature, but the default Pillai-Bartlett statistic is recommended by Hand and Taylor (1987).

Value

A list with components

- **SS**: A named list of sums of squares and product matrices.
- **Eigenvalues**: A matrix of eigenvalues.
- **stats**: A matrix of the statistics, approximate F value, degrees of freedom and P value.

References


See Also

`manova`, `aov`

Examples

```r
## Example on producing plastic film from Krzanowski (1998, p. 381)
tear <- c(6.5, 6.2, 5.8, 6.5, 6.5, 6.9, 7.2, 6.9, 6.1, 6.3,
         6.7, 6.6, 7.2, 7.1, 6.8, 7.1, 7.0, 7.2, 7.5, 7.6)
gloss <- c(9.5, 9.9, 9.6, 9.6, 9.2, 9.1, 10.0, 9.9, 9.5, 9.4,
           9.1, 9.3, 8.3, 8.4, 8.5, 9.2, 8.8, 9.7, 10.1, 9.2)
opacity <- c(4.4, 6.4, 3.0, 4.1, 0.8, 5.7, 2.0, 3.9, 1.9, 5.7,
            2.8, 4.1, 3.8, 1.6, 3.4, 8.4, 5.2, 6.9, 2.7, 1.9)
Y <- cbind(tear, gloss, opacity)
rate <- factor(gl(2,10), labels=c("Low", "High"))
additive <- factor(gl(2, 5, len=20), labels=c("Low", "High"))

fit <- manova(Y ~ rate * additive)
summary.aov(fit) # univariate ANOVA tables
summary(fit, test="Wilks") # ANOVA table of Wilks' lambda
```
Summary method for Principal Components Analysis

Description

The `summary` method for class "princomp".

Usage

```r
## S3 method for class 'princomp':
summary(object, loadings = FALSE, cutoff = 0.1, ...)

## S3 method for class 'summary.princomp':
print(x, digits = 3, loadings = x$print.loadings,
cutoff = x$cutoff, ...)
```

Arguments

- `object` an object of class "princomp", as from `princomp()`.
- `loadings` logical. Should loadings be included?
- `cutoff` numeric. Loadings below this cutoff in absolute value are shown as blank in the output.
- `x` an object of class "summary.princomp".
- `digits` the number of significant digits to be used in listing loadings.
- `...` arguments to be passed to or from other methods.

Value

`object` with additional components `cutoff` and `print.loadings`.

See Also

`princomp`

Examples

```r
summary(pc.cr <- princomp(USArrests, cor=TRUE))
print(summary(princomp(USArrests, cor=TRUE),
loadings = TRUE, cutoff = 0.2), digits = 2)
```
**Friedman’s SuperSmoother**

**Description**
Smooth the (x, y) values by Friedman’s “super smoother”.

**Usage**
```
supsmu(x, y, wt, span = "cv", periodic = FALSE, bass = 0)
```

**Arguments**
- `x`: x values for smoothing
- `y`: y values for smoothing
- `wt`: case weights, by default all equal
- `span`: the fraction of the observations in the span of the running lines smoother, or "cv" to choose this by leave-one-out cross-validation.
- `periodic`: if TRUE, the x values are assumed to be in [0, 1] and of period 1.
- `bass`: controls the smoothness of the fitted curve. Values of up to 10 indicate increasing smoothness.

**Details**
`supsmu` is a running lines smoother which choose between three spans for the lines. The running lines smoothers are symmetric, with k/2 data points each side of the predicted point, and values of k as 0.5 * n, 0.2 * n and 0.05 * n, where n is the number of data points. If `span` is specified, a single smoother with span `span * n` is used.

The best of the three smoothers is chosen by cross-validation for each prediction. The best spans are then smoothed by a running lines smoother and the final prediction chosen by linear interpolation.

The FORTRAN code says: “For small samples (n < 40) or if there are substantial serial correlations between observations close in x-value, then a prespecified fixed span smoother (span > 0) should be used. Reasonable span values are 0.2 to 0.4.”

**Value**
A list with components
- `x`: the input values in increasing order with duplicates removed.
- `y`: the corresponding y values on the fitted curve.

**References**

symnum

See Also

ppr

Examples

with(cars, {
  plot(speed, dist)
  lines(supsmu(speed, dist))
  lines(supsmu(speed, dist, bass = 7), lty = 2)
})

symnum Symbolic Number Coding

Description

Symbolically encode a given numeric or logical vector or array.

Usage

symnum(x, cutpoints=c(0.3, 0.6, 0.8, 0.9, 0.95),
  symbols=c(" ", ".", ",", "+", "+", "B"),
  legend = length(symbols) >= 3,
  na = "?", eps = 1e-5, corr = missing(cutpoints),
  show.max = if(corr) "1", show.min = NULL,
  abbr.colnames = has.colnames,
  lower.triangular = corr && is.numeric(x) && is.matrix(x),
  diag.lower.tri = corr && !is.null(show.max))

Arguments

  x numeric or logical vector or array.
  cutpoints numeric vector whose values cutpoints[j] = c_j (after augmentation, see corr below) are used for intervals.
  symbols character vector, one shorter than (the augmented, see corr below) cutpoints. symbols[j]= s_j are used as "code" for the (half open) interval (c_j, c_j+1].
  For logical argument x, the default is c(" .", ":") (graphical 0 / 1 s).
  legend logical indicating if a "legend" attribute is desired.
  na character or logical. How NAs are coded. If na == FALSE, NAs are coded invisibly, including the "legend" attribute below, which otherwise mentions NA coding.
  eps absolute precision to be used at left and right boundary.
  corr logical. If TRUE, x contains correlations. The cutpoints are augmented by 0 and 1 and abs(x) is coded.
  show.max if TRUE, or of mode character, the maximal cutpoint is coded especially.
  show.min if TRUE, or of mode character, the minimal cutpoint is coded especially.
symnum

abbr.colnames
logical, integer or NULL indicating how column names should be abbreviated (if they are); if NULL (or FALSE and x has no column names), the column names will all be empty, i.e., ""; otherwise if abbr.colnames is false, they are left unchanged. If TRUE or integer, existing column names will be abbreviated to abbreviate(*, minlength = abbr.colnames).

lower.triangular
logical. If TRUE and x is a matrix, only the lower triangular part of the matrix is coded as non-blank.

diag.lower.tri
logical. If lower.triangular and this are TRUE, the diagonal part of the matrix is shown.

Value
An atomic character object of class noquote and the same dimensions as x.
If legend (TRUE by default when there more than 2 classes), it has an attribute "legend" containing a legend of the returned character codes, in the form
\[ c_1 s_1 c_2 s_2 \ldots s_n c_{n+1} \]
where \( c_j = \text{cutpoints}[j] \) and \( s_j = \text{symbols}[j] \).

Author(s)
Martin Maechler (maechler@stat.math.ethz.ch)

See Also
as.character

Examples

```r
ii <- 0:8; names(ii) <- ii
symnum(ii, cut= 2*(0:4), sym = c(".", "-", "+", ":")

symnum(ii, cut= 2*(0:4), sym = c(".", "-", "+", ":"), show.max=TRUE)

symnum(1:12 %% 3 == 0)# use for logical

###-- Symbolic correlation matrices:
symnum(cor(attitude), diag = FALSE)
symnum(cor(attitude), abbr.= NULL)
symnum(cor(attitude), abbr.= FALSE)
symnum(cor(attitude), abbr.= 2)

symnum(cor(rbind(1, rnorm(25), rnorm(25)^2)))
symnum(cor(matrix(rexp(30, 1), 5, 18))) # <<< PATTERN ! --
symnum(cm1 <- cor(matrix(rnorm(90), 5, 18))) # < White Noise SMALL n
symnum(cm1, diag=FALSE)
symnum(cm2 <- cor(matrix(rnorm(900), 50, 18))) # < White Noise "BIG" n
symnum(cm2, lower=FALSE)

## NA's:
Cm <- cor(matrix(rnorm(60), 10, 6)); Cm[3,6] <- NA
symnum(Cm, show.max=NULL)
```

## Graphical P-values (aka "significance stars"):
pval <- rev(sort(c(outer(1:6, 10^-((1:3))))))
symp <- symnum(pval, corr=FALSE,
    cutpoints = c(0, .001, .01, .05, .1, 1),
    symbols = c(" *** ", " ** ", " * ", ".", " " ))
noquote(cbind(P.val = format(pval), Signif= symp))

t.test

### Description
Performs one and two sample t-tests on vectors of data.

### Usage
t.test(x, ...)

#### Default S3 method:
t.test(x, y = NULL,
    alternative = c("two.sided", "less", "greater"),
    mu = 0, paired = FALSE, var.equal = FALSE,
    conf.level = 0.95, ...)

#### S3 method for class 'formula':
t.test(formula, data, subset, na.action, ...)

### Arguments

- **x**: a numeric vector of data values.
- **y**: an optional numeric vector data values.
- **alternative**: a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
- **mu**: a number indicating the true value of the mean (or difference in means if you are performing a two sample test).
- **paired**: a logical indicating whether you want a paired t-test.
- **var.equal**: a logical variable indicating whether to treat the two variances as being equal. If TRUE then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used.
- **conf.level**: confidence level of the interval.
- **formula**: a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups.
- **data**: an optional data frame containing the variables in the model formula.
- **subset**: an optional vector specifying a subset of observations to be used.
- **na.action**: a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
- **...**: further arguments to be passed to or from methods.
Details

The formula interface is only applicable for the 2-sample tests.

If `paired` is `TRUE` then both `x` and `y` must be specified and they must be the same length. Missing values are removed (in pairs if `paired` is `TRUE`). If `var.equal` is `TRUE` then the pooled estimate of the variance is used. By default, if `var.equal` is `FALSE` then the variance is estimated separately for both groups and the Welch modification to the degrees of freedom is used.

If the input data are effectively constant (compared to the larger of the two means) an error is generated.

Value

A list with class "htest" containing the following components:

- `statistic` the value of the t-statistic.
- `parameter` the degrees of freedom for the t-statistic.
- `p.value` the p-value for the test.
- `conf.int` a confidence interval for the mean appropriate to the specified alternative hypothesis.
- `estimate` the estimated mean or difference in means depending on whether it was a one-sample test or a two-sample test.
- `null.value` the specified hypothesized value of the mean or mean difference depending on whether it was a one-sample test or a two-sample test.
- `alternative` a character string describing the alternative hypothesis.
- `method` a character string indicating what type of t-test was performed.
- `data.name` a character string giving the name(s) of the data.

See Also

`prop.test`

Examples

t.test(1:10,y=c(7:20))    # P = .00001855
t.test(1:10,y=c(7:20, 200)) # P = .1245 -- NOT significant anymore

## Classical example: Student’s sleep data
plot(extra ~ group, data = sleep)
## Traditional interface
with(sleep, t.test(extra[group == 1], extra[group == 2]))
## Formula interface
t.test(extra ~ group, data = sleep)
The Student t Distribution

Description

Density, distribution function, quantile function and random generation for the t distribution with \(df\) degrees of freedom (and optional noncentrality parameter \(ncp\)).

Usage

- \(dt(x, df, ncp=0, \text{log } = \text{FALSE})\)
- \(pt(q, df, ncp=0, \text{lower.tail } = \text{TRUE}, \text{log.p } = \text{FALSE})\)
- \(qt(p, df, \text{lower.tail } = \text{TRUE}, \text{log.p } = \text{FALSE})\)
- \(rt(n, df)\)

Arguments

- \(x, q\) vector of quantiles.
- \(p\) vector of probabilities.
- \(n\) number of observations. If \(\text{length}(n) > 1\), the length is taken to be the number required.
- \(df\) degrees of freedom (> 0, maybe non-integer). \(df = \text{Inf}\) is allowed. For \(qt\) only values of at least one are currently supported.
- \(ncp\) non-centrality parameter \(\delta\); currently for \(pt()\) and \(dt()\), only for \(ncp \leq 37.62\).
- \(\text{log, log.p}\) logical; if \(\text{TRUE}\), probabilities \(p\) are given as \(\log(p)\).
- \(\text{lower.tail}\) logical; if \(\text{TRUE}\) (default), probabilities are \(P[X \leq x]\), otherwise, \(P[X > x]\).

Details

The \(t\) distribution with \(df = \nu\) degrees of freedom has density

\[
f(x) = \frac{\Gamma((\nu + 1)/2)}{\sqrt{\pi \nu} \Gamma(\nu/2)} (1 + x^2/\nu)^{-(\nu+1)/2}
\]

for all real \(x\). It has mean 0 (for \(\nu > 1\)) and variance \(\frac{\nu}{\nu-2}\) (for \(\nu > 2\)).

The general non-central \(t\) with parameters \((\nu, \delta) = (df, ncp)\) is defined as the distribution of \(T_{\nu}(\delta) := \frac{U + \delta}{\sqrt{\nu/\nu}}\) where \(U\) and \(\chi_\nu\) are independent random variables, \(U \sim \mathcal{N}(0,1)\), and \(\chi_\nu^2\) is chi-squared, see \texttt{pchisq}.

The most used applications are power calculations for \(t\)-tests:
Let \(T = \frac{\bar{X} - \mu_0}{S/\sqrt{n}}\) where \(\bar{X}\) is the mean and \(S\) the sample standard deviation (\(sd\)) of \(X_1, X_2, \ldots, X_n\) which are i.i.d. \(N(\mu, \sigma^2)\). Then \(T\) is distributed as non-centrally \(t\) with \(df = n - 1\) degrees of freedom and non-centrality parameter \(ncp = (\mu - \mu_0)\sqrt{n}/\sigma\).

Value

- \(dt\) gives the density, \(pt\) gives the distribution function, \(qt\) gives the quantile function, and \(rt\) generates random deviates.
References


See Also
df for the F distribution.

Examples

1 - pt(1:5, df = 1)
qt(.975, df = c(1:10,20,50,100,1000))

`tt <- seq(0,10, len=21)`
`ncp <- seq(0,6, len=31)`
`ptn <- outer(tt,ncp, function(t,d) pt(t, df = 3, ncp=d))`
image(tt,ncp,ptn, zlim=c(0,1),main=t.tit <- "Non-central t - Probabilities")
persp(tt,ncp,ptn, zlim=0:1, r=2, phi=20, theta=200, main=t.tit,
    xlab = "t", ylab = "noncentrality parameter", zlab = "Pr(T <= t")

`op <- par(yaxs="i")`
plot(function(x) dt(x, df = 3, ncp = 2), -3, 11, ylim = c(0, 0.32),
    main="Non-central t - Density")
par(op)

---

**termplot**  
*Plot regression terms*

Description

Plots regression terms against their predictors, optionally with standard errors and partial residuals added.

Usage

```r
termplot(model, data=NULL, envir=environment(formula(model)),
partial.resid=FALSE, rug=FALSE,
terms=NULL, se=FALSE, xlabs=NULL, ylabs=NULL, main = NULL,
col.term = 2, lwd.term = 1.5,
col.se = "orange", lty.se = 2, lwd.se = 1,
col.res = "gray", cex = 1, pch = par("pch"),
col.smth = "darkred", lty.smth=2, span.smth=2/3,
ask = interactive() && nb.fig < n.tms && .Device !="postscript",
use.factor.levels=TRUE, smooth=NULL,
...)
```
Arguments

model     fitted model object

data      data frame in which variables in model can be found

envir     environment in which variables in model can be found

partial.resid     logical; should partial residuals be plotted?

rug       add rug plots (jittered 1-d histograms) to the axes?

terms     which terms to plot (default NULL means all terms)

se        plot pointwise standard errors?

xlabs     vector of labels for the x axes

ylabs     vector of labels for the y axes

main      logical, or vector of main titles; if TRUE, the model’s call is taken as main title, NULL or FALSE mean no titles.

col.term, lwd.term     color and line width for the “term curve”, see lines.

col.se, lty.se, lwd.se     color, line type and line width for the “twice-standard-error curve” when se = TRUE.

col.res, cex, pch     color, plotting character expansion and type for partial residuals, when partial.resid = TRUE, see points.

ask       logical; if TRUE, the user is asked before each plot, see par(ask=.).

use.factor.levels     Should x-axis ticks use factor levels or numbers for factor terms?

smooth    NULL or a function with the same arguments as panel.smooth to draw a smooth through the partial residuals for non-factor terms

lty.smth, col.smth, span.smth     Passed to smooth

...     other graphical parameters

Details

The model object must have a predict method that accepts type=terms, eg glm in the base package, coxph and survreg in the survival package.

For the partial.resid=TRUE option it must have a residuals method that accepts type="partial", which lm and glm do.

The data argument should rarely be needed, but in some cases termplot may be unable to reconstruct the original data frame. Using na.action=na.exclude makes these problems less likely.

Nothing sensible happens for interaction terms.

See Also

For (generalized) linear models, plot.lm and predict.glm.
Examples

```r
had.splines <- "package:splines" %in% search()
if(!had.splines) rs <- require(splines)
x <- 1:100
z <- factor(rep(LETTERS[1:4],25))
y <- rnorm(100,sin(x/10)+as.numeric(z))
model <- glm(y ~ ns(x,6) + z)
par(mfrow=c(2,2)) ## 2 x 2 plots for same model :
termplot(model, main = paste("termplot( ", deparse(model$call)," ...")
termplot(model, rug=TRUE)
termplot(model, partial=TRUE, se = TRUE, main = TRUE)
termplot(model, partial=TRUE, smooth=panel.smooth,span.smth=1/4)
if(!had.splines && rs) detach("package:splines")
```

### Description

The function `terms` is a generic function which can be used to extract `terms` objects from various kinds of R data objects.

### Usage

```r
terms(x, ...)
```

### Arguments

- `x` object used to select a method to dispatch.
- `...` further arguments passed to or from other methods.

### Details

There are methods for classes "aovlist", and "terms" "formula" (see `terms.formula`): the default method just extracts the terms component of the object (if any).

There are `print` and `labels` methods for class "terms": the latter prints the term labels (see `terms.object`).

### Value

An object of class `c("terms", "formula")` which contains the `terms` representation of a symbolic model. See `terms.object` for its structure.

### References


### See Also

`terms.object`, `terms.formula`, `lm`, `glm`, `formula`. 
Construct a terms Object from a Formula

Description
This function takes a formula and some optional arguments and constructs a terms object. The terms object can then be used to construct a `model.matrix`.

Usage
```r
## S3 method for class 'formula':
terms(x, specials = NULL, abb = NULL, data = NULL, neg.out = TRUE,
      keep.order = FALSE, simplify = FALSE, ..., allowDotAsName=FALSE)
```

Arguments
- `x`: a formula.
- `specials`: which functions in the formula should be marked as special in the terms object.
- `abb`: Not implemented in R.
- `data`: a data frame from which the meaning of the special symbol \( . \) can be inferred. It is unused if there is no \( . \) in the formula.
- `neg.out`: Not implemented in R.
- `keep.order`: a logical value indicating whether the terms should keep their positions. If false the terms are reordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on. Effects of a given order are kept in the order specified.
- `simplify`: should the formula be expanded and simplified, the pre-1.7.0 behaviour?
- `...`: further arguments passed to or from other methods.
- `allowDotAsName`: normally \( . \) in a formula refers to the remaining variables contained in data. Exceptionally, \( . \) can be treated as a name for non-standard uses of formulae.

Details
Not all of the options work in the same way that they do in S and not all are implemented.

Value
A `terms.object` object is returned. The object itself is the re-ordered (unless `keep.order = TRUE`) formula. In all cases variables within an interaction term in the formula are re-ordered by the ordering of the "variables" attribute, which is the order in which the variables occur in the formula.

See Also
- `terms`, `terms.object`
Description

An object of class `terms` holds information about a model. Usually the model was specified in terms of a `formula` and that formula was used to determine the terms object.

Value

The object itself is simply the formula supplied to the call of `terms.formula`. The object has a number of attributes and they are used to construct the model frame:

- **factors**: A matrix of variables by terms showing which variables appear in which terms. The entries are 0 if the variable does not occur in the term, 1 if it does occur and should be coded by contrasts, and 2 if it occurs and should be coded via dummy variables for all levels (as when an intercept or lower-order term is missing). If there are no terms other than an intercept and offsets, this is `numeric(0)`.
- **term.labels**: A character vector containing the labels for each of the terms in the model, except for offsets. Non-syntactic names will be quoted by backticks.
- **variables**: A call to `list` of the variables in the model.
- **intercept**: Either 0, indicating no intercept is to be fit, or 1 indicating that an intercept is to be fit.
- **order**: A vector of the same length as `term.labels` indicating the order of interaction for each term.
- **response**: The index of the variable (in `variables`) of the response (the left hand side of the formula). Zero, if there is no response.
- **offset**: If the model contains `offset` terms there is an `offset` attribute indicating which variable(s) are offsets.
- **specials**: If the `specials` argument was given to `terms.formula` there is a `specials` attribute, a list of vectors indicating the terms that contain these special functions.
- **dataClasses**: Optional. A named character vector giving the classes (as given by `.MFclass`) of the variables used in a fit.

The object has class `c("terms", "formula")`.

Note

These objects are different from those found in S. In particular there is no `formula` attribute, instead the object is itself a formula. Thus, the mode of a terms object is different as well.

Examples of the `specials` argument can be seen in the `aov` and `coxph` functions.

See Also

`terms`, `formula`.
**Sampling Times of Time Series**

**Description**

time creates the vector of times at which a time series was sampled.
cycle gives the positions in the cycle of each observation.
frequency returns the number of samples per unit time and deltat the time interval between observations (see ts).

**Usage**

time(x, ...)
## Default S3 method:
time(x, offset=0, ...)
cycle(x, ...)
frequency(x, ...)
deltat(x, ...)

**Arguments**

x
a univariate or multivariate time-series, or a vector or matrix.
offset
can be used to indicate when sampling took place in the time unit. 0 (the default) indicates the start of the unit, 0.5 the middle and 1 the end of the interval.
...
extra arguments for future methods.

**Details**

These are all generic functions, which will use the tsp attribute of x if it exists. time and cycle have methods for class ts that coerce the result to that class.

**References**


**See Also**

ts, start, tsp, window.
date for clock time, system.time for CPU usage.

**Examples**

cycle(presidents)
# a simple series plot: c() makes the x and y arguments into vectors
plot(c(time(presidents)), c(presidents), type="l")
toeplitz  

Form Symmetric Toeplitz Matrix

Description

Forms a symmetric Toeplitz matrix given its first row.

Usage

toeplitz(x)

Arguments

x  

the first row to form the Toeplitz matrix.

Value

The Toeplitz matrix.

Author(s)

A. Trapletti

Examples

x <- 1:5
toeplitz (x)

---

ts  

Time-Series Objects

Description

The function ts is used to create time-series objects.

as.ts and is.ts coerce an object to a time-series and test whether an object is a time series.

Usage

    ts(data = NA, start = 1, end = numeric(0), frequency = 1, 
        deltat = 1, ts.eps = getOption("ts.eps"), class = , names = )
    as.ts(x, ...)
    is.ts(x)
Arguments

- **data**: a numeric vector or matrix of the observed time-series values. A data frame will be coerced to a numeric matrix via `data.matrix`.
- **start**: the time of the first observation. Either a single number or a vector of two integers, which specify a natural time unit and a (1-based) number of samples into the time unit. See the examples for the use of the second form.
- **end**: the time of the last observation, specified in the same way as `start`.
- **frequency**: the number of observations per unit of time.
- **deltat**: the fraction of the sampling period between successive observations; e.g., 1/12 for monthly data. Only one of `frequency` or `deltat` should be provided.
- **ts.eps**: time series comparison tolerance. Frequencies are considered equal if their absolute difference is less than `ts.eps`.
- **class**: class to be given to the result, or none if `NULL` or "none". The default is "ts" for a single series, c("mts", "ts") for multiple series.
- **names**: a character vector of names for the series in a multiple series: defaults to the colnames of `data`, or Series 1, Series 2,....
- **x**: an arbitrary R object.
- **...**: arguments passed to methods (unused for the default method).

Details

The function `ts` is used to create time-series objects. These are vector or matrices with class of "ts" (and additional attributes) which represent data which has been sampled at equispaced points in time. In the matrix case, each column of the matrix `data` is assumed to contain a single (univariate) time series. Time series must have an least one observation, and although they need not be numeric there is very limited support for non-numeric series.

Class "ts" has a number of methods. In particular arithmetic will attempt to align time axes, and subsetting to extract subsets of series can be used (e.g., `EuStockMarkets[, "DAX"]`). However, subsetting the first (or only) dimension will return a matrix or vector, as will matrix subsetting. There is a method for `t` that transposes the series as a matrix (a one-column matrix if a vector) and hence returns a result that does not inherit from class "ts".

The value of argument `frequency` is used when the series is sampled an integral number of times in each unit time interval. For example, one could use a value of 7 for `frequency` when the data are sampled daily, and the natural time period is a week, or 12 when the data are sampled monthly and the natural time period is a year. Values of 4 and 12 are assumed in (e.g.) `print` methods to imply a quarterly and monthly series respectively.

`as.ts` is generic. Its default method will use the `tsp` attribute of the object if it has one to set the start and end times and frequency.

`is.ts` tests if an object is a time series. It is generic: you can write methods to handle specific classes of objects, see `InternalMethods`.

References


See Also

`tsp`, `frequency`, `start`, `end`, `time`, `window`; `print.ts`, the print method for time series objects; `plot.ts`, the plot method for time series objects.
Examples

```r
ts(1:10, frequency = 4, start = c(1959, 2)) # 2nd Quarter of 1959	nprint(ts(1:10, freq = 7, start = c(12, 2)), calendar = TRUE) # print.ts(.)
## Using July 1954 as start date:
gnp <- ts(cumsum(1 + round(rnorm(100), 2)),
          start = c(1954, 7), frequency = 12)
plot(gnp) # using 'plot.ts' for time-series plot

## Multivariate
z <- ts(matrix(rnorm(300), 100, 3), start=c(1961, 1), frequency=12)
class(z)
plot(z)
plot(z, plot.type="single", lty=1:3)

## A phase plot:
plot(nhtemp, c(nhtemp[-1], NA), cex = .8, col="blue",
     main = "Lag plot of New Haven temperatures")
## a clearer way to do this would be
## Not run:
plot(nhtemp, lag(nhtemp, 1), cex = .8, col="blue",
     main = "Lag plot of New Haven temperatures")
## End(Not run)
```

### ts-methods

Methods for Time Series Objects

**Description**

Methods for objects of class "ts", typically the result of `ts`.

**Usage**

```r
## S3 method for class 'ts':
diff(x, lag = 1, differences = 1, ...)

## S3 method for class 'ts':
na.omit(object, ...)
```

**Arguments**

- `x`: an object of class "ts" containing the values to be differenced.
- `lag`: an integer indicating which lag to use.
- `differences`: an integer indicating the order of the difference.
- `object`: a univariate or multivariate time series.
- `...`: further arguments to be passed to or from methods.

**Details**

The `na.omit` method omits initial and final segments with missing values in one or more of the series. 'Internal' missing values will lead to failure.
Value
For the \texttt{na.omit} method, a time series without missing values. The class of \texttt{object} will be preserved.

See Also
\texttt{diff}; \texttt{na.omit}, \texttt{na.fail}, \texttt{na.contiguous}.

---

**ts.plot**  
*Plot Multiple Time Series*

Description
Plot several time series on a common plot. Unlike \texttt{plot.ts} the series can have a different time bases, but they should have the same frequency.

Usage
\begin{verbatim}
  ts.plot(..., gpars = list())
\end{verbatim}

Arguments
\begin{itemize}
  \item \ldots\quad one or more univariate or multivariate time series.
  \item \texttt{gpars} \quad list of named graphics parameters to be passed to the plotting functions. Those commonly used can be supplied directly in \ldots.
\end{itemize}

Value
None.

Note
Although this can be used for a single time series, \texttt{plot} is easier to use and is preferred.

See Also
\texttt{plot.ts}

Examples
\begin{verbatim}
  ts.plot(ldeaths, mdeaths, fdeaths,
          gpars=list(xlab="year", ylab="deaths", lty=c(1:3)))
\end{verbatim}
**ts.union**

*Bind Two or More Time Series*

**Description**

Bind time series which have a common frequency. `ts.union` pads with NAs to the total time coverage, `ts.intersect` restricts to the time covered by all the series.

**Usage**

```r
ts.intersect(..., dframe = FALSE)
ts.union(..., dframe = FALSE)
```

**Arguments**

- `...`: two or more univariate or multivariate time series, or objects which can coerced to time series.
- `dframe`: logical; if TRUE return the result as a data frame.

**Details**

As a special case, `...` can contain vectors or matrices of the same length as the combined time series of the time series present, as well as those of a single row.

**Value**

A time series object if `dframe` is FALSE, otherwise a data frame.

**See Also**

- `cbind`

**Examples**

```r
ts.union(mdeaths, fdeaths)
cbind(mdeaths, fdeaths)  # same as the previous line
ts.intersect(window(mdeaths, 1976), window(fdeaths, 1974, 1978))
sales1 <- ts.union(BJsales, lead = BJsales.lead)
ts.intersect(sales1, lead3 = lag(BJsales.lead, -3))
```
tsdiag

Diagnostic Plots for Time-Series Fits

Description

A generic function to plot time-series diagnostics.

Usage

tsdiag(object, gof.lag, ...)

Arguments

object a fitted time-series model
gof.lag the maximum number of lags for a Portmanteau goodness-of-fit test
...

Details

This is a generic function. It will generally plot the residuals, often standadized, the autocorrelation function of the residuals, and the p-values of a Portmanteau test for all lags up to gof.lag.

The methods for arima and StructTS objects plots residuals scaled by the estimate of their (individual) variance, and use the Ljung–Box version of the portmanteau test.

Value

None. Diagnostics are plotted.

See Also

arima,StructTS,Box.test

Examples

## Not run: fit <- arima(lh, c(1,0,0))
tsdia(fit)

## see also examples(arima)

(fit <- StructTS(log10(JohnsonJohnson), type="BSM"))
tsdia(fit)

## End(Not run)
**tsp**

*P Attribute of Time-Series-like Objects*

**Description**

tsp returns the tsp attribute (or NULL). It is included for compatibility with S version 2. tsp<- sets the tsp attribute. hasTsp ensures x has a tsp attribute, by adding one if needed.

**Usage**

tsp(x)
tsp(x) <- value
hasTsp(x)

**Arguments**

- **x**: a vector or matrix or univariate or multivariate time-series.
- **value**: a numeric vector of length 3 or NULL.

**Details**

The tsp attribute was previously described here as c(start(x), end(x), frequency(x)), but this is incorrect. It gives the start time in time units, the end time and the frequency.

Assignments are checked for consistency.

Assigning NULL which removes the tsp attribute and any "ts" class of x.

**References**


**See Also**

ts, time, start.

---

**tsSmooth**

*Use Fixed-Interval Smoothing on Time Series*

**Description**

Performs fixed-interval smoothing on a univariate time series via a state-space model. Fixed-interval smoothing gives the best estimate of the state at each time point based on the whole observed series.

**Usage**

tsSmooth(object, ...)
Tukey

Arguments

object  a time-series fit. Currently only class "StructTS" is supported
...  possible arguments for future methods.

Value

A time series, with as many dimensions as the state space and results at each time point of the original series. (For seasonal models, only the current seasonal component is returned.)

Author(s)

B. D. Ripley

References


See Also

KalmanSmooth, StructTS.

For examples consult AirPassengers, JohnsonJohnson and Nile.

Tukey  The Studentized Range Distribution

Description

Functions on the distribution of the studentized range, R/s, where R is the range of a standard normal sample of size n and s^2 is independently distributed as chi-squared with df degrees of freedom, see pchisq.

Usage

ptukey(q, nmeans, df, nranges = 1, lower.tail = TRUE, log.p = FALSE)
qtukey(p, nmeans, df, nranges = 1, lower.tail = TRUE, log.p = FALSE)

Arguments

q  vector of quantiles.
p  vector of probabilities.
nmeans  sample size for range (same for each group).
df  degrees of freedom for s (see below).
nranges  number of groups whose maximum range is considered.
log.p  logical; if TRUE, probabilities p are given as log(p).
lower.tail  logical; if TRUE (default), probabilities are P[X ≤ x], otherwise, P[X > x].
Details

If \( n_g = \text{nranges} \) is greater than one, \( R \) is the maximum of \( n_g \) groups of \( n \text{means} \) observations each.

Value

\( \text{ptukey} \) gives the distribution function and \( \text{qtukey} \) its inverse, the quantile function.

Note

A Legendre 16-point formula is used for the integral of \( \text{ptukey} \). The computations are relatively expensive, especially for \( \text{qtukey} \) which uses a simple secant method for finding the inverse of \( \text{ptukey} \). \( \text{qtukey} \) will be accurate to the 4th decimal place.

References


See Also

\( \text{pnorm} \) and \( \text{qnorm} \) for the corresponding functions for the normal distribution.

Examples

```r
if(interactive())
  curve(ptukey(x, nm=6, df=5), from=-1, to=8, n=101)
(ptt <- ptukey(0:10, 2, df= 5))
(qtt <- qtukey(.95, 2, df= 2:11))
## The precision may be not much more than about 8 digits:
summary(abs(.95 - ptukey(qtt,2, df = 2:11)))
```

TukeyHSD

**Compute Tukey Honest Significant Differences**

Description

Create a set of confidence intervals on the differences between the means of the levels of a factor with the specified family-wise probability of coverage. The intervals are based on the Studentized range statistic, Tukey’s ‘Honest Significant Difference’ method. There is a `plot` method.

Usage

```r
TukeyHSD(x, which, ordered = FALSE, conf.level = 0.95, ...)
```
TukeyHSD

Arguments

x | A fitted model object, usually an aov fit.
which | A list of terms in the fitted model for which the intervals should be calculated. Defaults to all the terms.
ordered | A logical value indicating if the levels of the factor should be ordered according to increasing average in the sample before taking differences. If ordered is true then the calculated differences in the means will all be positive. The significant differences will be those for which the lwr end point is positive.
conf.level | A numeric value between zero and one giving the family-wise confidence level to use.
... | Optional additional arguments. None are used at present.

Details

When comparing the means for the levels of a factor in an analysis of variance, a simple comparison using t-tests will inflate the probability of declaring a significant difference when it is not in fact present. This because the intervals are calculated with a given coverage probability for each interval but the interpretation of the coverage is usually with respect to the entire family of intervals.

John Tukey introduced intervals based on the range of the sample means rather than the individual differences. The intervals returned by this function are based on this Studentized range statistics.

Technically the intervals constructed in this way would only apply to balanced designs where there are the same number of observations made at each level of the factor. This function incorporates an adjustment for sample size that produces sensible intervals for mildly unbalanced designs.

If which specifies non-factor terms these will be dropped with a warning: if no terms are left this is a an error.

Value

A list with one component for each term requested in which. Each component is a matrix with columns diff giving the difference in the observed means, lwr giving the lower end point of the interval, upr giving the upper end point and p adj giving the p-value after adjustment for the multiple comparisons.

Author(s)

Douglas Bates

References


See Also

aov, qtukey, model.tables, simint

Examples

summary(fm1 <- aov(breaks ~ wool + tension, data = warpbreaks))
TukeyHSD(fm1, "tension", ordered = TRUE)
plot(TukeyHSD(fm1, "tension"))
The Uniform Distribution

Description

These functions provide information about the uniform distribution on the interval from \( \text{min} \) to \( \text{max} \). \( \text{dunif} \) gives the density, \( \text{punif} \) gives the distribution function \( \text{qunif} \) gives the quantile function and \( \text{runif} \) generates random deviates.

Usage

\[
\begin{align*}
\text{dunif} & \quad (x, \text{min}=0, \text{max}=1, \text{log} = \text{FALSE}) \\
\text{punif} & \quad (q, \text{min}=0, \text{max}=1, \text{lower.tail} = \text{TRUE}, \text{log.p} = \text{FALSE}) \\
\text{qunif} & \quad (p, \text{min}=0, \text{max}=1, \text{lower.tail} = \text{TRUE}, \text{log.p} = \text{FALSE}) \\
\text{runif} & \quad (n, \text{min}=0, \text{max}=1)
\end{align*}
\]

Arguments

- \( x, q \) vector of quantiles.
- \( p \) vector of probabilities.
- \( n \) number of observations. If \( \text{length}(n) > 1 \), the length is taken to be the number required.
- \( \text{min}, \text{max} \) lower and upper limits of the distribution.
- \( \text{log}, \text{log.p} \) logical; if TRUE, probabilities \( p \) are given as \( \log(p) \).
- \( \text{lower.tail} \) logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

If \( \text{min} \) or \( \text{max} \) are not specified they assume the default values of 0 and 1 respectively.

The uniform distribution has density

\[
f(x) = \frac{1}{\text{max} - \text{min}}
\]

for \( \text{min} \leq x \leq \text{max} \).

For the case of \( u := \text{min} = \text{max} \), the limit case of \( X \equiv u \) is assumed.

References


See Also

- \( \text{.Random.seed} \) about random number generation, \( \text{rnorm} \), etc for other distributions.
Examples

u <- runif(20)

## The following relations always hold:
punif(u) == u
dunif(u) == 1

var(runif(10000)) # ~ = 1/12 = .08333

uniroot

One Dimensional Root (Zero) Finding

Description

The function uniroot searches the interval from lower to upper for a root (i.e., zero) of the function f with respect to its first argument.

Usage

uniroot(f, interval, lower = min(interval), upper = max(interval),
       tol = .Machine$double.eps^0.25, maxiter = 1000, ...)

Arguments

  f          the function for which the root is sought.
  interval   a vector containing the end-points of the interval to be searched for the root.
  lower      the lower end point of the interval to be searched.
  upper      the upper end point of the interval to be searched.
  tol        the desired accuracy (convergence tolerance).
  maxiter    the maximum number of iterations.
  ...        additional arguments to f.

Details

Either interval or both lower and upper must be specified. The function uses Fortran subroutine "zeroin" (from Netlib) based on algorithms given in the reference below.

If the algorithm does not converge in maxiter steps, a warning is printed and the current approximation is returned.

Value

A list with four components: root and f.root give the location of the root and the value of the function evaluated at that point. iter and estim.prec give the number of iterations used and an approximate estimated precision for root.

References

See Also
polyroot for all complex roots of a polynomial; optimize, nlm.

Examples

```r
f <- function (x, a) x - a
str(xmin <- uniroot(f, c(0, 1), tol = 0.0001, a = 1/3))
str(uniroot(function(x) x*(x^2-1) + .5, low = -2, up = 2, tol = 0.0001),
dig = 10)
str(uniroot(function(x) x*(x^2-1) + .5, low = -2, up = 2, tol = 1e-10 ),
dig = 10)

## Find the smallest value x for which exp(x) > 0 (numerically):
1e80 * exp(x) -1e-300
r <- uniroot(function(x) 1e80*exp(x) -1e-300,-1000,0, tol=1e-20)
str(r, digits= 15)##> around -745.1332191
exp(r$r) # = 0, but not for r$r * 0.999...
minexp <- r$r * (1 - .Machine$double.eps)
exp(minexp) # typically denormalized
```

Description

update will update and (by default) re-fit a model. It does this by extracting the call stored in the object, updating the call and (by default) evaluating that call. Sometimes it is useful to call update with only one argument, for example if the data frame has been corrected.

Usage

```r
update(object, ...)
```

Arguments

- **object**: An existing fit from a model function such as lm, glm and many others.
- **formula.**: Changes to the formula -- see update.formula for details.
- **...**: Additional arguments to the call, or arguments with changed values. Use name=NULL to remove the argument name.
- **evaluate**: If true evaluate the new call else return the call.

Value

If evaluate = TRUE the fitted object, otherwise the updated call.

References

update.formula

See Also
update.formula

Examples

oldcon <- options(contrasts = c("contr.treatment", "contr.poly"))
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- q1(2, 10, 20, labels = c("Ctl", "Trt"))
lm.D9 <- lm(weight ~ group)
summary(lm.D9 <- update(lm.D9, . ~ . - 1))
options(contrasts = c("contr.helmert", "contr.poly"))
update(lm.D9)
options(oldcon)

Description

update.formula is used to update model formulae. This typically involves adding or dropping terms, but updates can be more general.

Usage

## S3 method for class 'formula':
update(old, new, ...)

Arguments

old a model formula to be updated.
new a formula giving a template which specifies how to update.
... further arguments passed to or from other methods.

Details

The function works by first identifying the left-hand side and right-hand side of the old formula. It then examines the new formula and substitutes the lhs of the old formula for any occurrence of "." on the left of new, and substitutes the rhs of the old formula for any occurrence of "." on the right of new.

Value

The updated formula is returned.

See Also
terms, model.matrix.
Examples

\begin{verbatim}
update(y ~ x, ~ . + x2) #> y ~ x + x2
update(y ~ x, log(.) ~ .) #> log(y) ~ x
\end{verbatim}

\section*{Description}

Performs an F test to compare the variances of two samples from normal populations.

\section*{Usage}

\begin{verbatim}
var.test(x, ...)

## Default S3 method:
var.test(x, y, ratio = 1,
            alternative = c("two.sided", "less", "greater"),
            conf.level = 0.95, ...)

## S3 method for class 'formula':
var.test(formula, data, subset, na.action, ...)
\end{verbatim}

\section*{Arguments}

- \textbf{x, y} numeric vectors of data values, or fitted linear model objects (inheriting from class "lm").
- \textbf{ratio} the hypothesized ratio of the population variances of \( x \) and \( y \).
- \textbf{alternative} a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
- \textbf{conf.level} confidence level for the returned confidence interval.
- \textbf{formula} a formula of the form \( \text{lhs} \sim \text{rhs} \) where \( \text{lhs} \) is a numeric variable giving the data values and \( \text{rhs} \) a factor with two levels giving the corresponding groups.
- \textbf{data} an optional data frame containing the variables in the model formula.
- \textbf{subset} an optional vector specifying a subset of observations to be used.
- \textbf{na.action} a function which indicates what should happen when the data contain NAs. Defaults to \texttt{getOption("na.action")}.
- \textbf{...} further arguments to be passed to or from methods.

\section*{Details}

The null hypothesis is that the ratio of the variances of the populations from which \( x \) and \( y \) were drawn, or in the data to which the linear models \( x \) and \( y \) were fitted, is equal to \texttt{ratio}. 

varimax

Value

A list with class "htest" containing the following components:

- statistic: the value of the F test statistic.
- parameter: the degrees of freedom of the F distribution of the test statistic.
- p.value: the p-value of the test.
- conf.int: a confidence interval for the ratio of the population variances.
- estimate: the ratio of the sample variances of x and y.
- null.value: the ratio of population variances under the null.
- alternative: a character string describing the alternative hypothesis.
- method: the character string "F test to compare two variances".
- data.name: a character string giving the names of the data.

See Also

bartlett.test for testing homogeneity of variances in more than two samples from normal distributions; ansari.test and mood.test for two rank based (nonparametric) two-sample tests for difference in scale.

Examples

```r
x <- rnorm(50, mean = 0, sd = 2)
y <- rnorm(30, mean = 1, sd = 1)
var.test(x, y) # Do x and y have the same variance?
var.test(lm(x ~ 1), lm(y ~ 1)) # The same.
```

---

varimax

Rotation Methods for Factor Analysis

Description

These functions 'rotate' loading matrices in factor analysis.

Usage

```r
varimax(x, normalize = TRUE, eps = 1e-5)
promax(x, m = 4)
```

Arguments

- **x**: A loadings matrix, with \( p \) rows and \( k < p \) columns
- **m**: The power used the target for promax. Values of 2 to 4 are recommended.
- **normalize**: logical. Should Kaiser normalization be performed? If so the rows of \( x \) are re-scaled to unit length before rotation, and scaled back afterwards.
- **eps**: The tolerance for stopping: the relative change in the sum of singular values.
Details

These seek a ‘rotation’ of the factors $x \times \% \% T$ that aims to clarify the structure of the loadings matrix. The matrix $T$ is a rotation (possibly with reflection) for varimax, but a general linear transformation for promax, with the variance of the factors being preserved.

Value

A list with components

- `loadings`: The ‘rotated’ loadings matrix, $x \ % \ % \ \text{rotmat}$, of class "loadings".
- `rotmat`: The ‘rotation’ matrix.

References


See Also

- `factanal`, `Harman74.cor`.

Examples

```r
## varimax with normalize = TRUE is the default
fa <- factanal(~., 2, data = swiss)
varimax(loadings(fa), normalize = FALSE)
promax(loadings(fa))
```

---

### vcov

*Calculate Variance-Covariance Matrix for a Fitted Model Object*

**Description**

Returns the variance-covariance matrix of the main parameters of a fitted model object.

**Usage**

`vcov(object, ...)`

**Arguments**

- `object`: a fitted model object.
- `...`: additional arguments for method functions. For the `glm` method this can be used to pass a dispersion parameter.
Weibull

Details

This is a generic function. Functions with names beginning in vcov. will be methods for this function. Classes with methods for this function include: lm, mlm, glm, nls, lme, gls, coxph and survreg (the last two in package survival).

Value

A matrix of the estimated covariances between the parameter estimates in the linear or non-linear predictor of the model.

---

Weibull

The Weibull Distribution

Description

Density, distribution function, quantile function and random generation for the Weibull distribution with parameters shape and scale.

Usage

dweibull(x, shape, scale = 1, log = FALSE)
pweibull(q, shape, scale = 1, lower.tail = TRUE, log.p = FALSE)
qweibull(p, shape, scale = 1, lower.tail = TRUE, log.p = FALSE)
rweibull(n, shape, scale = 1)

Arguments

x, q vector of quantiles.
p vector of probabilities.
n number of observations. If length(n) > 1, the length is taken to be the number required.
shape, scale shape and scale parameters, the latter defaulting to 1.
log, log.p logical; if TRUE, probabilities p are given as log(p).
lower.tail logical; if TRUE (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

The Weibull distribution with shape parameter \( a \) and scale parameter \( \sigma \) has density given by

\[
f(x) = (a/\sigma)(x/\sigma)^{a-1}\exp(-(x/\sigma)^a)
\]

for \( x > 0 \). The cumulative distribution function is \( F(x) = 1 - \exp(-(x/\sigma)^a) \), the mean is \( E(X) = \sigma \Gamma(1 + 1/a) \), and the \( Var(X) = \sigma^2(\Gamma(1 + 2/a) - (\Gamma(1 + 1/a))^2) \).

Value

dweibull gives the density, pweibull gives the distribution function, qweibull gives the quantile function, and rweibull generates random deviates.
Weighted Arithmetic Mean

Description

Compute a weighted mean of a numeric vector.

Usage

weighted.mean(x, w, na.rm = FALSE)

Arguments

x

a numeric vector containing the values whose mean is to be computed.

w

a vector of weights the same length as x giving the weights to use for each element of x.

na.rm

a logical value indicating whether NA values in x should be stripped before the computation proceeds.

Details

If w is missing then all elements of x are given the same weight.

Missing values in w are not handled.

See Also

mean

Examples

## GPA from Siegel 1994
wt <- c(5, 5, 4, 1)/15
x <- c(3.7, 3.3, 3.5, 2.8)
xm <- weighted.mean(x, wt)

Note

The cumulative hazard \( H(t) = -\log(1 - F(t)) \) is \(-\text{pweibull}(t, a, b, \text{lower} = \text{FALSE}, \text{log} = \text{TRUE})\) which is just \( H(t) = (t/b)^a \).

See Also

dexp for the Exponential which is a special case of a Weibull distribution.

Examples

x <- c(0, rlnorm(50))
all.equal(dweibull(x, shape = 1), dexp(x))
all.equal(pweibull(x, shape = 1, scale = pi), pexp(x, rate = 1/pi))
## Cumulative hazard H():
all.equal(pweibull(x, 2.5, pi, lower=FALSE, log=TRUE), -(x/phi)^2.5, tol=1e-15)
all.equal(qweibull(x/11, shape = 1, scale = pi), qexp(x/11, rate = 1/pi))

weighted.mean Weighted Arithmetic Mean

Description

Compute a weighted mean of a numeric vector.

Usage

weighted.mean(x, w, na.rm = FALSE)

Arguments

x

a numeric vector containing the values whose mean is to be computed.

w

a vector of weights the same length as x giving the weights to use for each element of x.

na.rm

a logical value indicating whether NA values in x should be stripped before the computation proceeds.

Details

If w is missing then all elements of x are given the same weight.

Missing values in w are not handled.

See Also

mean

Examples

## GPA from Siegel 1994
wt <- c(5, 5, 4, 1)/15
x <- c(3.7, 3.3, 3.5, 2.8)
xm <- weighted.mean(x, wt)
weighted.residuals  Compute Weighted Residuals

Description

Computed weighted residuals from a linear model fit.

Usage

weighted.residuals(obj, drop0 = TRUE)

Arguments

obj  
R object, typically of class lm or glm.

drop0  
logical. If TRUE, drop all cases with weights == 0.

Details

Weighted residuals are based on the deviance residuals, which for a lm fit are the raw residuals \( R_i \) multiplied by \( \sqrt{w_i} \), where \( w_i \) are the weights as specified in lm’s call.

Dropping cases with weights zero is compatible with influence and related functions.

Value

Numeric vector of length \( n' \), where \( n' \) is the number of of non-0 weights (drop0 = TRUE) or the number of observations, otherwise.

See Also

residuals, lm.influence, etc.

Examples

```r
example("lm")
all.equal(weighted.residuals(lm.D9),
          residuals(lm.D9))
x <- 1:10
w <- 0:9
y <- rnorm(x)
weighted.residuals(lmxy <- lm(y ~ x, weights = w))
weighted.residuals(lmxy, drop0 = FALSE)
```
wilcox.test

Wilcoxon Rank Sum and Signed Rank Tests

Description

Performs one and two sample Wilcoxon tests on vectors of data; the latter is also known as ‘Mann-Whitney’ test.

Usage

wilcox.test(x, ...)

## Default S3 method:
wilcox.test(x, y = NULL,
          alternative = c("two.sided", "less", "greater"),
          mu = 0, paired = FALSE, exact = NULL, correct = TRUE,
          conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'formula':
wilcox.test(formula, data, subset, na.action, ...)

Arguments

x numeric vector of data values. Non-finite (e.g. infinite or missing) values will be omitted.

y an optional numeric vector of data values.

alternative a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.

mu a number specifying an optional location parameter.

paired a logical indicating whether you want a paired test.

exact a logical indicating whether an exact p-value should be computed.

correct a logical indicating whether to apply continuity correction in the normal approximation for the p-value.

conf.int a logical indicating whether a confidence interval should be computed.

conf.level confidence level of the interval.

formula a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups.

data an optional data frame containing the variables in the model formula.

subset an optional vector specifying a subset of observations to be used.

na.action a function which indicates what should happen when the data contain NAs. Defaults togetOption("na.action").

... further arguments to be passed to or from methods.
Details

The formula interface is only applicable for the 2-sample tests.

If only \( x \) is given, or if both \( x \) and \( y \) are given and \( \text{paired} \) is \text{TRUE}, a Wilcoxon signed rank test of the null that the distribution of \( x \) (in the one sample case) or of \( x-y \) (in the paired two sample case) is symmetric about \( \mu \) is performed.

Otherwise, if both \( x \) and \( y \) are given and \( \text{paired} \) is \text{FALSE}, a Wilcoxon rank sum test (equivalent to the Mann-Whitney test: see the Note) is carried out. In this case, the null hypothesis is that the distributions of \( x \) and \( y \) differ by a location shift of \( \mu \) and the alternative is that they differ by some other location shift.

By default (if \text{exact} is not specified), an exact p-value is computed if the samples contain less than 50 finite values and there are no ties. Otherwise, a normal approximation is used.

Optionally (if argument \text{conf.int} is true), a nonparametric confidence interval and an estimator for the pseudomedian (one-sample case) or for the difference of the location parameters \( x-y \) is computed. (The pseudomedian of a distribution \( F \) is the median of the distribution of \( (u+v)/2 \), where \( u \) and \( v \) are independent, each with distribution \( F \). If \( F \) is symmetric, then the pseudomedian and median coincide. See Hollander & Wolfe (1973), page 34.) If exact p-values are available, an exact confidence interval is obtained by the algorithm described in Bauer (1972), and the Hodges-Lehmann estimator is employed. Otherwise, the returned confidence interval and point estimate are based on normal approximations.

With small samples it may not be possible to achieve very high confidence interval coverages. If this happens a warning will be given and an interval with lower coverage will be substituted.

Value

A list with class "htest" containing the following components:

- \text{statistic} \quad \text{the value of the test statistic with a name describing it.}
- \text{parameter} \quad \text{the parameter(s) for the exact distribution of the test statistic.}
- \text{p.value} \quad \text{the p-value for the test.}
- \text{null.value} \quad \text{the location parameter \( \mu \).}
- \text{alternative} \quad \text{a character string describing the alternative hypothesis.}
- \text{method} \quad \text{the type of test applied.}
- \text{data.name} \quad \text{a character string giving the names of the data.}
- \text{conf.int} \quad \text{a confidence interval for the location parameter. (Only present if argument \text{conf.int} = \text{TRUE}.)}
- \text{estimate} \quad \text{an estimate of the location parameter. (Only present if argument \text{conf.int} = \text{TRUE}.)}

Warning

This function can use large amounts of memory and stack (and even crash \( \text{R} \) if the stack limit is exceeded) if \text{exact} = \text{TRUE} and one sample is large (several thousands or more).

Note

The literature is not unanimous about the definitions of the Wilcoxon rank sum and Mann-Whitney tests. The two most common definitions correspond to the sum of the ranks of the first sample with the minimum value subtracted or not: \( \text{R} \) subtracts and \( \text{S-PLUS} \) does not, giving a value which is
larger by \( m(m + 1)/2 \) for a first sample of size \( m \). (It seems Wilcoxon’s original paper used the unadjusted sum of the ranks but subsequent tables subtracted the minimum.)

R’s value can also be computed as the number of all pairs \((x[i], y[j])\) for which \( y[j] \) is not greater than \( x[i] \), the most common definition of the Mann-Whitney test.

References


See Also

`psignrank.pwilcox`

`wilcox.exact` in `exactRankTests` covers much of the same ground, but also produces exact p-values in the presence of ties.

`kruskal.test` for testing homogeneity in location parameters in the case of two or more samples; `t.test` for an alternative under normality assumptions or large samples

Examples

```r
## One-sample test.
## Hollander & Wolfe (1973), 29f.
## Hamilton depression scale factor measurements in 9 patients with
## mixed anxiety and depression, taken at the first (x) and second
## (y) visit after initiation of a therapy (administration of a
## tranquilizer).
## x <- c(1.83, 0.50, 1.62, 2.48, 1.68, 1.88, 1.55, 3.06, 1.30)
y <- c(0.878, 0.647, 0.598, 2.05, 1.06, 1.29, 1.06, 3.14, 1.29)
wilcox.test(x, y, paired = TRUE, alternative = "greater")
wilcox.test(y - x, alternative = "less")  # The same.
wilcox.test(y - x, alternative = "less",
            exact = FALSE, correct = FALSE)  # H&W large sample
            # approximation

## Two-sample test.
## Hollander & Wolfe (1973), 69f.
## Permeability constants of the human chorioamnion (a placental
## membrane) at term (x) and between 12 to 26 weeks gestational
## age (y). The alternative of interest is greater permeability
## of the human chorioamnion for the term pregnancy.
## x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
wilcox.test(x, y, alternative = "g")    # greater
wilcox.test(x, y, alternative = "greater",
            exact = FALSE, correct = FALSE) # H&W large sample
            # approximation

wilcox.test(rnorm(10), rnorm(10, 2), conf.int = TRUE)

## Formula interface.
boxplot(Ozone ~ Month, data = airquality)
wilcox.test(Ozone ~ Month, data = airquality,
```
subset = Month %in% c(5, 8))

Wilcoxon

Distribution of the Wilcoxon Rank Sum Statistic

Description

Density, distribution function, quantile function and random generation for the distribution of the Wilcoxon rank sum statistic obtained from samples with size \( m \) and \( n \), respectively.

Usage

\[
\begin{align*}
dwilcox(x, m, n, \text{log} = \text{FALSE})
pwilcox(q, m, n, \text{lower.tail} = \text{TRUE}, \text{log.p} = \text{FALSE})
qwilcox(p, m, n, \text{lower.tail} = \text{TRUE}, \text{log.p} = \text{FALSE})
\text{rwilcox(nn, m, n)}
\end{align*}
\]

Arguments

- \( x, q \): vector of quantiles.
- \( p \): vector of probabilities.
- \( nn \): number of observations. If \( \text{length}(nn) > 1 \), the length is taken to be the number required.
- \( m, n \): numbers of observations in the first and second sample, respectively. Can be vectors of positive integers.
- \( \text{log, log.p} \): logical; if \( \text{TRUE} \), probabilities \( p \) are given as \( \log(p) \).
- \( \text{lower.tail} \): logical; if \( \text{TRUE} \) (default), probabilities are \( P[X \leq x] \), otherwise, \( P[X > x] \).

Details

This distribution is obtained as follows. Let \( x \) and \( y \) be two random, independent samples of size \( m \) and \( n \). Then the Wilcoxon rank sum statistic is the number of all pairs \((x[i], y[j])\) for which \( y[j] \) is not greater than \( x[i] \). This statistic takes values between 0 and \( mn \), and its mean and variance are \( mn/2 \) and \( mn(m+n+1)/12 \), respectively.

If any of the first three arguments are vectors, the recycling rule is used to do the calculations for all combinations of the three up to the length of the longest vector.

Value

\( \text{dwilcox} \) gives the density, \( \text{pwilcox} \) gives the distribution function, \( \text{qwilcox} \) gives the quantile function, and \( \text{rwilcox} \) generates random deviates.

Warning

These functions can use large amounts of memory and stack (and even crash \( R \) if the stack limit is exceeded) if and one sample is large (several thousands or more).

Note

S-PLUS uses a different (but equivalent) definition of the Wilcoxon statistic: see \( \text{wilcox.test} \) for details.
Author(s)
Kurt Hornik

See Also
wilcox.test to calculate the statistic from data, find p values and so on.
dsignrank etc, for the distribution of the one-sample Wilcoxon signed rank statistic.

Examples

```r
x <- -1:(4*6 + 1)
fx <- dwilcox(x, 4, 6)
Fx <- pwilcox(x, 4, 6)

layout(rbind(1,2),width=1,heights=c(3,2))
plot(x, fx,type='h', col="violet",
     main="Probabilities (density) of Wilcoxon-Statist.(n=6,m=4)")
plot(x, Fx,type="s", col="blue",
     main="Distribution of Wilcoxon-Statist.(n=6,m=4)")
abline(h=0:1, col="gray20",lty=2)
layout(1)# set back

N <- 200
hist(U <- rwilcox(N, m=4,n=6), breaks=0:25 - 1/2, border="red", col="pink",
     sub = paste("N =",N))

mtext("N * f(x), f() = true \"density\", side=3, col="blue")
lines(x, N*fx, type='h', col='blue', lwd=2)
points(x, N*fx, cex=2)

## Better is a Quantile-Quantile Plot

qqplot(U, qw <- qwilcox((1:N - 1/2)/N, m=4,n=6),
     main = paste("Q-Q-Plot of empirical and theoretical quantiles",
                 "Wilcoxon Statistic, (m=4, n=6)"))

n <- as.numeric(names(print(tU <- table(U))))
text(n+.2, n+.5, labels=tU, col="red")
```

window

Description

window is a generic function which extracts the subset of the object x observed between the times start and end. If a frequency is specified, the series is then re-sampled at the new frequency.

Usage

```r
window(x, ...)
## S3 method for class 'ts':
window(x, ...)
## Default S3 method:
window(x, start = NULL, end = NULL,
       frequency = NULL, deltat = NULL, extend = FALSE, ...)
```
window(x, ...) <- value
## S3 method for class 'ts':
window(x, start, end, frequency, deltat, ...) <- value

Arguments

x  a time-series (or other object if not replacing values).
start  the start time of the period of interest.
end  the end time of the period of interest.
frequency, deltat  the new frequency can be specified by either (or both if they are consistent).
extend  logical. If true, the start and end values are allowed to extend the series. If false, attempts to extend the series give a warning and are ignored.
...  further arguments passed to or from other methods.
value  replacement values.

Details

The start and end times can be specified as for ts. If there is no observation at the new start or end, the immediately following (start) or preceding (end) observation time is used.

The replacement function has a method for ts objects, and is allowed to extend the series (with a warning). There is no default method.

Value

The value depends on the method. window.default will return a vector or matrix with an appropriate tsp attribute.

window.ts differs from window.default only in ensuring the result is a ts object.

If extend = TRUE the series will be padded with NAs if needed.

References


See Also

time, ts.

Examples

window(presidents, 1960, c(1969,4)) # values in the 1960's
window(presidents, deltat=1) # All Qtr1s
window(presidents, start=c(1945,3), deltat=1) # All Qtr3s
window(presidents, 1944, c(1979,2), extend=TRUE)

pres <- window(presidents, 1945, c(1949,4)) # values in the 1940's
window(pres, 1945.25, 1945.50) <- c(60, 70)
window(pres, 1944, 1944.75) <- 0 # will generate a warning
window(pres, c(1945,4), c(1949,4), freq=1) <- 85:89
pres
Description

Create a contingency table from cross-classifying factors, usually contained in a data frame, using a formula interface.

Usage

```r
xtabs(formula = ~., data = parent.frame(), subset, na.action, 
       exclude = c(NA, NaN), drop.unused.levels = FALSE)
```

Arguments

- `formula`: a formula object with the cross-classifying variables, separated by `+`, on the right hand side. Interactions are not allowed. On the left hand side, one may optionally give a vector or a matrix of counts; in the latter case, the columns are interpreted as corresponding to the levels of a variable. This is useful if the data has already been tabulated, see the examples below.

- `data`: a data frame, list or environment containing the variables to be cross-tabulated.

- `subset`: an optional vector specifying a subset of observations to be used.

- `na.action`: a function which indicates what should happen when the data contain NAs.

- `exclude`: a vector of values to be excluded when forming the set of levels of the classifying factors.

- `drop.unused.levels`: a logical indicating whether to drop unused levels in the classifying factors. If this is `FALSE` and there are unused levels, the table will contain zero marginals, and a subsequent chi-squared test for independence of the factors will not work.

Details

There is a `summary` method for contingency table objects created by `table` or `xtabs`, which gives basic information and performs a chi-squared test for independence of factors (note that the function `chisq.test` currently only handles 2-d tables).

If a left hand side is given in `formula`, its entries are simply summed over the cells corresponding to the right hand side; this also works if the lhs does not give counts.

Value

A contingency table in array representation of class `c("xtabs", "table")`, with a "call" attribute storing the matched call.

See Also

- `table` for “traditional” cross-tabulation, and `as.data.frame.table` which is the inverse operation of `xtabs` (see the DF example below).
Examples

## 'esoph' has the frequencies of cases and controls for all levels of
## the variables 'agegp', 'alcgp', and 'tobgp'.
xtabs(cbind(ncases, ncontrols) ~ ., data = esoph)
## Output is not really helpful ... flat tables are better:
ftable(xtabs(cbind(ncases, ncontrols) ~ ., data = esoph))
## In particular if we have fewer factors ...
ftable(xtabs(cbind(ncases, ncontrols) ~ agegp, data = esoph))

## This is already a contingency table in array form.
DF <- as.data.frame(UCBAdmissions)
## Now 'DF' is a data frame with a grid of the factors and the counts
## in variable 'Freq'.
DF
## Nice for taking margins ...
xtabs(Freq ~ Gender + Admit, DF)
## And for testing independence ...
summary(xtabs(Freq ~ ., DF))

## Create a nice display for the warp break data.
warpbreaks$replicate <- rep(1:9, len = 54)
ftable(xtabs(breaks ~ wool + tension + replicate, data = warpbreaks))
Chapter 8

The tools package

---

**buildVignettes** List and Build Package Vignettes

**Description**

Run Sweave and texi2dvi on all vignettes of a package.

**Usage**

```r
buildVignettes(package, dir, lib.loc = NULL, quiet = TRUE)
pkgVignettes(package, dir, lib.loc = NULL)
```

**Arguments**

- `package`: a character string naming an installed package. If given, Sweave files are searched in subdirectory doc.
- `dir`: a character string specifying the path to a package’s root source directory. This subdirectory inst/doc is searched for Sweave files.
- `lib.loc`: a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. The specified library trees are used to to search for `package`.
- `quiet`: logical. Run Sweave and texi2dvi in quiet mode.

**Value**

`buildVignettes` is called for its side effect of creating the PDF versions of all vignettes.

`pkgVignettes` returns an object of class "pkgVignettes".
checkFF  

Check Foreign Function Calls

Description

Performs checks on calls to compiled code from R code. Currently only whether the interface functions such as `.C` and `.Fortran` are called with argument PACKAGE specified, which is highly recommended to avoid name clashes in foreign function calls.

Usage

```r
checkFF(package, dir, file, lib.loc = NULL, 
        verbose = getOption("verbose"))
```

Arguments

- `package`: a character string naming an installed package. If given, the installed R code of the package is checked.
- `dir`: a character string specifying the path to a package’s root source directory. This should contain the subdirectory R (for R code). Only used if `package` is not given.
- `file`: the name of a file containing R code to be checked. Used if neither `package` nor `dir` are given.
- `lib.loc`: a character vector of directory names of R libraries, or `NULL`. The default value of `NULL` corresponds to all libraries currently known. The specified library trees are used to to search for `package`.
- `verbose`: a logical. If `TRUE`, additional diagnostics are printed (and the result is returned invisibly).

Details

If the package has a namespace and if that contains a `useDynLib` directive, calls in top-level functions in the package are not reported as their symbols will be preferentially looked up in the DLL named in the first `useDynLib` directive.

Value

An object of class "checkFF", which currently is a list of the (parsed) foreign function calls with no PACKAGE argument.

There is a `print` method for nicely displaying the information contained in such objects.

Warning

This function is still experimental. Both name and interface might change in future versions.

See Also

`.C`, `.Fortran`; Foreign.

Examples

```r
checkFF(package = "stats", verbose = TRUE)
```
checkMD5sums

Check and Create MD5 Checksum Files

Description

checkMD5sums checks the files against a file MD5.

Usage

checkMD5sums(pkg, dir)

Arguments

pkg the name of an installed package
dir the path to the top-level directory of an installed package.

Details

The file ‘MD5’ which is created is in a format which can be checked by md5sum -c MD5 if a suitable command-line version of md5sum is available. (One is supplied in the bundle at http://www.murdoch-sutherland.com/Rtools/tools.zip.)

If dir is missing, an installed package of name pkg is searched for.

The private function tools:::.installMD5sums is used to create MD5 files in the Windows build.

Value

checkMD5sums returns a logical, NA if there is no MD5 file to be checked.

See Also

md5sum

cHECK R Packages or Code for T/F

Description

Checks the specified R package or code file for occurrences of T or F, and gathers the expression containing these. This is useful as in R T and F are just variables which are set to the logicals TRUE and FALSE by default, but are not reserved words and hence can be overwritten by the user. Hence, one should always use TRUE and FALSE for the logicals.

Usage

checkTnF(package, dir, file, lib.loc = NULL)
checkVignettes

Arguments

package a character string naming an installed package. If given, the installed R code and the examples in the documentation files of the package are checked. R code installed as an image file cannot be checked.

dir a character string specifying the path to a package’s root source directory. This must contain the subdirectory ‘R’ (for R code), and should also contain ‘man’ (for documentation). Only used if package is not given. If used, the R code files and the examples in the documentation files are checked.

file the name of a file containing R code to be checked. Used if neither package nor dir are given.

lib.loc a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. The specified library trees are used to to search for package.

Value

An object of class "checkTnF" which is a list containing, for each file where occurrences of T or F were found, a list with the expressions containing these occurrences. The names of the list are the corresponding file names.

There is a print method for nicely displaying the information contained in such objects.

Warning

This function is still experimental. Both name and interface might change in future versions.

Description

Check all Sweave files of a package by running Sweave and/or Stangle on them. All R source code files found after the tangling step are sourced to check whether all code can be executed without errors.

Usage

checkVignettes(package, dir, lib.loc = NULL, tangle = TRUE, weave = TRUE, workdir = c("tmp", "src", "cur"), keepfiles = FALSE)

Arguments

package a character string naming an installed package. If given, Sweave files are searched in subdirectory doc.

dir a character string specifying the path to a package’s root source directory. This subdirectory inst/doc is searched for Sweave files.

lib.loc a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. The specified library trees are used to to search for package.
Perform a tangle and source the extracted code?

Perform a weave?

Directory used as working directory while checking the vignettes. If "tmp" then a temporary directory is created, this is the default. If "src" then the directory containing the vignettes itself is used, if "cur" then the current working directory of R is used.

Delete file in temporary directory? This option is ignored when workdir!="tmp".

An object of class "checkVignettes" which is a list with the error messages found during the tangle and weave steps. There is a print method for nicely displaying the information contained in such objects.

**Description**

Find inconsistencies between actual and documented “structure” of R objects in a package. codoc compares names and optionally also corresponding positions and default values of the arguments of functions. codocClasses and codocData compare slot names of S4 classes and variable names of data sets, respectively.

**Usage**

```r
codoc(package, dir, lib.loc = NULL,
      use.values = NULL, verbose = getOption("verbose"))
codocClasses(package, lib.loc = NULL)
codocData(package, lib.loc = NULL)
```

**Arguments**

- **package** a character string naming an installed package.
- **dir** a character string specifying the path to a package’s root source directory. This must contain the subdirectories ‘man’ with R documentation sources (in Rd format) and ‘R’ with R code. Only used if package is not given.
- **lib.loc** a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. The specified library trees are used to to search for package.
- **use.values** if FALSE, do not use function default values when comparing code and docs. Otherwise, compare all default values if TRUE, and only the ones documented in the usage otherwise (default).
- **verbose** a logical. If TRUE, additional diagnostics are printed.
Details

The purpose of codoc is to check whether the documented usage of function objects agrees with their formal arguments as defined in the R code. This is not always straightforward, in particular as the usage information for methods to generic functions often employs the name of the generic rather than the method.

The following algorithm is used. If an installed package is used, it is loaded (unless it is the base package), after possibly detaching an already loaded version of the package. Otherwise, if the sources are used, the R code files of the package are collected and sourced in a new environment. Then, the usage sections of the Rd files are extracted and parsed “as much as possible” to give the formals documented. For interpreted functions in the code environment, the formals are compared between code and documentation according to the values of the argument use.values. Synopsis sections are used if present; their occurrence is reported if verbose is true.

If a package has a namespace both exported and unexported objects are checked, as well as registered S3 methods. (In the unlikely event of differences the order is exported objects in the package, registered S3 methods and finally objects in the namespace and only the first found is checked.)

Currently, the R documentation format has no high-level markup for the basic “structure” of classes and data sets (similar to the usage sections for function synopses). Variable names for data frames in documentation objects obtained by suitably editing “shells” created by prompt are recognized by codocData and used provided that the documentation object is for a single data frame (i.e., only has one alias). codocClasses analogously handles slot names for classes in documentation objects obtained by editing shells created by promptClass.

Help files named pkg_name-defunct.Rd for the appropriate pkg_name are checked more loosely, as they may have undocumented arguments.

Value

codoc returns an object of class "codoc". Currently, this is a list which, for each Rd object in the package where an inconsistency was found, contains an element with a list of the mismatches (which in turn are lists with elements code and docs, giving the corresponding arguments obtained from the function’s code and documented usage).

codocClasses and codocData return objects of class "codocClasses" and "codocData", respectively, with a structure similar to class "codoc".

There are print methods for nicely displaying the information contained in such objects.

Warning

Both codocClasses and codocData are still experimental. Names, interfaces and values might change in future versions.

Note

The default for use.values has been changed from FALSE to NULL, for R versions 1.9.0 and later.

See Also

undoc, QC
Delimited Pattern Matching

delimMatch

Description

Match delimited substrings in a character vector, with proper nesting.

Usage

delimMatch(x, delim = c("{", "}"), syntax = "Rd")

Arguments

x
a character vector.
delim
a character vector of length 2 giving the start and end delimiters. Future versions might allow for arbitrary regular expressions.
syntax
currently, always the string "Rd" indicating Rd syntax (i.e., ‘%’ starts a comment extending till the end of the line, and ‘\’ escapes). Future versions might know about other syntaxes, perhaps via “syntax tables” allowing to flexibly specify comment, escape, and quote characters.

Value

An integer vector of the same length as x giving the starting position (in characters) of the first match, or −1 if there is none, with attribute “match.length” giving the length (in characters) of the matched text (or −1 for no match).

See Also

regexpr for “simple” pattern matching.

Examples

x <- c("\value{foo}", "function(bar)"

delimMatch(x)
delimMatch(x, c("{", "}"))

encoded_text_to_latex

Translate non-ASCII Text to LaTeX Escapes

Description

Translate non-ASCII characters in text to LaTeX escape sequences.

Usage

enencoded_text_to_latex(x, encoding = c("latin1", "latin2", "latin9", "UTF-8", "utf8"))
**Arguments**

- `x`: a character vector.
- `encoding`: the encoding to be assumed. "latin9" is officially ISO-8859-15 or Latin-9, but known as latin9 to LaTeX’s inputenc package.

**Details**

Non-ASCII characters in `x` are replaced by an appropriate LaTeX escape sequence, or ? if there is no appropriate sequence.

Even if there is an appropriate sequence, it may not be supported by the font in use. Hyphen is mapped to -. 

**Value**

A character vector of the same length as `x`.

**See Also**

- `iconv`

**Examples**

```r
x <- "fa\xE7ile"
encoded_text_to_latex(x, "latin1")
```

```r
## Not run:
## create a tex file to show the upper half of 8-bit charsets
(x <- rawToChar(as.raw(160:255), multiple=TRUE))
(xx <- matrix(x, ncol=16, byrow=TRUE))
xx[] <- encoded_text_to_latex(x, "latin1") # or latin2 or latin9
xx <- apply(xx, 1, paste, collapse="")
con <- file("test-encoding.tex", "w")
header <- c("\documentclass{article}", "\usepackage[T1]{fontenc}", "\usepackage{Rd}", "\begin{document}", "\HeaderA{test}{}{test}", "\begin{Details}\relax", "\Tabular{cccccccccccccccc}{
(trailer <- c("","\end{document}"))
writeLines(header, con)
writeLines(paste(xx, "\", sep=""), con)
writeLines(trailer, con)
close(con)
```

```r
## and some UTF_8 chars
x <- intToUtf8(as.integer(c(160:383,0x0192,0x02C6,0x02C7,0x02CA,0x02D8,
0x02D9, 0x02DD, 0x200C, 0x2018, 0x2019, 0x201C,
0x201D, 0x2020, 0x2022, 0x2026, 0x20AC)),
multiple=TRUE)
x <- matrix(x, ncol=16, byrow=TRUE)
xx <- x
xx[] <- encoded_text_to_latex(x, "UTF-8")
```
## End(Not run)
list_files_with_exts returns the paths or names of the files in directory dir with extension matching one of the elements of exts. Note that by default, full paths are returned, and that only visible files are used.

list_files_with_type returns the paths of the files in dir of the given "type", as determined by the extensions recognized by R. When listing R code and documentation files, files in OS-specific subdirectories are included if present. Note that by default, full paths are returned, and that only visible files are used.

**See Also**

file.path, file.info, list.files

**Examples**

dir <- file.path(R.home(), "library", "stats")
file_test("-d", dir)
file_test("-nt", file.path(dir, "R"), file.path(dir, "demo"))
list_files_with_exts(file.path(dir, "demo"), "R")
list_files_with_type(file.path(dir, "demo"), "demo")  # the same
file_path_sans_ext(list.files(file.path(R.home(), "modules")))

---

**getDepList**

*Functions to Retrieve Dependency Information*

**Description**

Given a dependency matrix, will create a DependsList object for that package which will include the dependencies for that matrix, which ones are installed, which unresolved dependencies were found online, which unresolved dependencies were not found online, and any R dependencies.

**Usage**

getDepList(depMtrx, instPkgs, recursive = TRUE, local = TRUE, reduce = TRUE, lib.loc = NULL)

pkgDepends(pkg, recursive = TRUE, local = TRUE, reduce = TRUE, lib.loc = NULL)

**Arguments**

- **depMtrx**: A dependency matrix as from package.dependencies
- **pkg**: The name of the package
- **instPkgs**: A matrix specifying all packages installed on the local system, as from installed.packages
- **recursive**: Whether or not to include indirect dependencies
- **local**: Whether or not to search only locally
- **reduce**: Whether or not to collapse all sets of dependencies to a minimal value
- **lib.loc**: What libraries to use when looking for installed packages. NULL indicates all library directories in the user's.libPaths().
installFoundDepends

Details

The function pkgDepends is a convenience function which wraps getDepList and takes as input a package name. It will then query `installed.packages` and also generate a dependency matrix, calling getDepList with this information and returning the result.

These functions will retrieve information about the dependencies of the matrix, resulting in a DependsList object. This is a list with four elements:

- **Depends** A vector of the dependencies for this package.
- **Installed** A vector of the dependencies which have been satisfied by the currently installed packages.
- **Found** A list representing the dependencies which are not in Installed but were found online. This list has element names which are the URLs for the repositories in which packages were found and the elements themselves are vectors of package names which were found in the respective repositories. If `local=TRUE`, the Found element will always be empty.
- **R** Any R version dependencies.

If `recursive` is `TRUE`, any package that is specified as a dependency will in turn have its dependencies included (and so on), these are known as indirect dependencies. If `recursive` is `FALSE`, only the dependencies directly stated by the package will be used.

If `local` is `TRUE`, the system will only look at the user’s local install and not online to find unresolved dependencies.

If `reduce` is `TRUE`, the system will collapse the fields in the DependsList object such that a minimal set of dependencies are specified (for instance if there was ('foo', 'foo (>= 1.0.0)', 'foo (>= 1.3.0)'), it would only return 'foo (>= 1.3.0)')

Value

An object of class DependsList

Author(s)

Jeff Gentry

See Also

installFoundDepends

Examples

```r
pkgDepends("tools", local = FALSE)
```

installFoundDepends

A function to install unresolved dependencies

Description

This function will take the Found element of a DependsList object and attempt to install all of the listed packages from the specified repositories.
Usage

installFoundDepends(depPkgList, ...)

Arguments

depPkgList A Found element from a pkgDependsList object
...
Arguments to pass on to install.packages

Details

This function takes as input the Found list from a pkgDependsList object. This list will have element names being URLs corresponding to repositories and the elements will be vectors of package names. For each element, install.packages is called for that URL to install all packages listed in the vector.

Author(s)

Jeff Gentry

See Also

pkgDepends, install.packages

Examples

## Set up a temporary directory to install packages to
tmp <- tempfile()
dir.create(tmp)

pDL <- pkgDepends("tools", local=FALSE)
installFoundDepends(pDL$Found, destdir=tmp)

makeLazyLoading  Lazy Loading of Packages

Description

Tools for Lazy Loading of Packages from a Database.

Usage

makeLazyLoading(package, lib.loc = NULL, compress = TRUE,
keep.source = getOption("keep.source.pkgs"))

Arguments

package package name string
lib.loc library trees, as in library
keep.source logical; should sources be kept when saving from source
compress logical; whether to compress entries on the database.
Details

A tool to set up packages for lazy loading from a database. For packages other than base you can use `makeLazyLoading(package)` to convert them to use lazy loading.

Author(s)

Luke Tierney and Brian Ripley

Examples

```r
# set up package "splines" for lazy loading -- already done
## Not run:
  tools:::makeLazyLoading("splines")
## End(Not run)
```

---

**md5sum**

*Compute MD5 Checksums*

Description

Compute the 32-byte MD5 checksums of one or more files.

Usage

```r
md5sum(files)
```

Arguments

- `files` character. The paths of file(s) to be check-summed.

Value

A character vector of the same length as `files`, with names equal to `files`. The elements will be `NA` for non-existent or unreadable files, otherwise a 32-character string of hexadecimal digits.

On Windows all files are read in binary mode (as the `md5sum` utilities there do): on other OSes the files are read in the default way.

See Also

- `checkMD5sums`

Examples

```r
md5sum(dir(R.home(), pattern="^COPY", full.names=TRUE))
```
package.dependencies

*Check Package Dependencies*

**Description**

Parses and checks the dependencies of a package against the currently installed version of R [and other packages].

**Usage**

```r
package.dependencies(x, check = FALSE, depLevel = c("Depends", "Imports", "Suggests"))
```

**Arguments**

- `x` A matrix of package descriptions as returned by `CRAN.packages`.
- `check` If TRUE, return logical vector of check results. If FALSE, return parsed list of dependencies.
- `depLevel` Whether to look for Depends or Suggests level dependencies.

**Details**

Currently we only check if the package conforms with the currently running version of R. In the future we might add checks for inter-package dependencies.

**See Also**

`update.packages`

---

 QC

* QC Checks for R Code and/or Documentation *

**Description**

Functions for performing various quality checks.

**Usage**

```r
checkDocFiles(package, dir, lib.loc = NULL) checkDocStyle(package, dir, lib.loc = NULL) checkReplaceFuns(package, dir, lib.loc = NULL) checkS3methods(package, dir, lib.loc = NULL)
```
Arguments

package
a character string naming an installed package.

dir
a character string specifying the path to a package’s root source directory. This should contain the subdirectories R (for R code) and ‘man’ with R documentation sources (in Rd format). Only used if package is not given.

lib.loc
a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. The specified library trees are used to to search for package.

Details

checkDocFiles checks, for all Rd files in a package, whether all arguments shown in the usage sections of the Rd file are documented in its arguments section. It also reports duplicated entries in the arguments section, and “over-documented” arguments which are given in the arguments section but not in the usage. Note that the match is for the usage section and not a possibly existing synopsis section, as the usage is what gets displayed.

checkDocStyle investigates how (S3) methods are shown in the usages of the Rd files in a package. It reports the methods shown by their full name rather than using the Rd \method markup for indicating S3 methods. Earlier versions of R also reported about methods shown along with their generic, which typically caused problems for the documentation of the primary argument in the generic and its methods. With \method now being expanded in a way that class information is preserved, “joint” documentation is no longer necessarily a problem. (The corresponding information is still contained in the object returned by checkDocStyle.)

checkReplaceFuns checks whether replacement functions or S3/S4 replacement methods in the package R code have their final argument named value.

checkS3methods checks whether all S3 methods defined in the package R code have all arguments of the corresponding generic, with positional arguments of the generics in the same positions for the method. As an exception, the first argument of a formula method may be called formula even if this is not the name used by the generic. The rules when ... is involved are subtle: see the source code. Functions recognized as S3 generics are those with a call to UseMethod in their body, internal S3 generics (see InternalMethods), and S3 group generics (see Math). Possible dispatch under a different name is not taken into account. The generics are sought first in the given package, then in the base package and (currently) the packages graphics, stats, and utils added in R 1.9.0 by splitting the former base, and, if an installed package is tested, also in the loaded namespaces/packages listed in the package’s ‘DESCRIPTION’ Depends field.

If using an installed package, the checks needing access to all R objects of the package will load the package (unless it is the base package), after possibly detaching an already loaded version of the package.

Value

The functions return objects of class the same as the respective function names containing the information about problems detected. There is a print method for nicely displaying the information contained in such objects.

Warning

These functions are still experimental. Names, interfaces and values might change in future versions.
1224

Rdindex

Rdutils

Generate Index from Rd Files

Description
Print a 2-column index table with “names” and titles from given R documentation files to a given
output file or connection. The titles are nicely formatted between two column positions (typically
25 and 72, respectively).
Usage
Rdindex(RdFiles, outFile = "", type = NULL,
width = 0.9 * getOption("width"), indent = NULL)
Arguments
RdFiles

a character vector specifying the Rd files to be used for creating the index, either
by giving the paths to the files, or the path to a single directory with the sources
of a package.

outFile

a connection, or a character string naming the output file to print to. "" (the
default) indicates output is to the console.

type

a character string giving the documentation type of the Rd files to be included
in the index, or NULL (the default). The type of an Rd file is typically specified
via the \docType tag; if type is "data", Rd files whose only keyword is
datasets are included as well.

width

a positive integer giving the target column for wrapping lines in the output.

indent

a positive integer specifying the indentation of the second column. Must not be
greater than width/2, and defaults to width/3.

Details
If a name is not a valid alias, the first alias (or the empty string if there is none) is used instead.

Rdutils

Rd Utilities

Description
Utilities for computing on the information in Rd objects.
Usage
Rd_db(package, dir, lib.loc = NULL)
Rd_parse(file, text = NULL)


Arguments

package  a character string naming an installed package.
dir      a character string specifying the path to a package’s root source directory. This should contain the subdirectory ‘man’ with R documentation sources (in Rd format). Only used if package is not given.
lib.loc  a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. The specified library trees are used to to search for package.
file     a connection, or a character string giving the name of a file or a URL to read documentation in Rd format from.
text     character vector with documentation in Rd format. Elements are treated as if they were lines of a file.

Details

Rd_db builds a simple “data base” of all Rd sources in a package, as a list of character vectors with the lines of the Rd files in the package. This is particularly useful for working on installed packages, where the individual Rd files in the sources are no longer available.

Rd_parse is a simple top-level Rd parser/analyzer. It returns a list with components

- **meta** a list containing the Rd meta data (aliases, concepts, keywords, and documentation type);
- **data** a data frame with the names (tags) and corresponding text (vals) of the top-level sections in the R documentation object;
- **rest** top-level text not accounted for (currently, silently discarded by Rdconv, and hence usually the indication of a problem).

Note that at least for the time being, only the top-level structure is analyzed.

Warning

These functions are still experimental. Names, interfaces and values might change in future versions.

Examples

```r
## Build the Rd db for the (installed) base package.
db <- Rd_db("base")
## Run Rd_parse on all entries in the Rd db.
db <- lapply(db, function(txt) Rd_parse(text = txt))
## Extract the metadata.
meta <- lapply(db, ":[[", "meta"])
## Keyword metadata per Rd file.
keywords <- lapply(meta, ":[[", "keywords"])
## Tabulate the keyword entries.
kw_table <- sort(table(unlist(keywords)))
## The 5 most frequent ones:
rev(kw_table)[1 : 5]
## The "most informative" ones:
kw_table[kw_table == 1]
## Concept metadata per Rd file.
concepts <- lapply(meta, ":[[", "concepts"])
```
## How many files already have \concept metadata?
sum(sapply(concepts, length) > 0)

## How many concept entries altogether?
length(unlist(concepts))

---

### read.00Index
#### Read 00Index-style Files

**Description**

Read item/description information from 00Index-style files. Such files are description lists rendered in tabular form, and currently used for the `INDEX` and `demo/00Index` files of add-on packages.

**Usage**

```
read.00Index(file)
```

**Arguments**

- `file`  
  the name of a file to read data values from. If the specified file is `""`, then input is taken from the keyboard (in this case input can be terminated by a blank line). Alternatively, `file` can be a `connection`, which will be opened if necessary, and if so closed at the end of the function call.

**Value**

A character matrix with 2 columns named "Item" and "Description" which hold the items and descriptions.

**See Also**

- `formatDL` for the inverse operation of creating a 00Index-style file from items and their descriptions.

---

### texi2dvi
#### Compile LaTeX Files

**Description**

Run `latex` and `bibtex` until all cross-references are resolved and create either a dvi or PDF file.

**Usage**

```
texi2dvi(file, pdf = FALSE, clean = FALSE, quiet = TRUE, texi2dvi = getOption("texi2dvi"))
```
Arguments

- **file**: character. Name of TeX source file.
- **pdf**: logical. If `TRUE`, a PDF file is produced instead of the default dvi file (text2dvi command line option `--pdf`).
- **clean**: logical. If `TRUE`, all auxiliary files are removed (text2dvi command line option `--clean`). Does not work on some platforms.
- **quiet**: logical. No output unless an error occurs.
- **text2dvi**: character (or NULL). Script or program used to compile a TeX file to dvi or PDF, respectively. If set to NULL, the `text2dvi` script in R's `bin` directory is used (if it exists), otherwise it is assumed that text2dvi is in the search path.

Details

Some TeX installations do not have `text2dvi.exe`. If `texify.exe` is present, then it can be used instead: set `options(text2dvi="texify.exe")` or to the full path of the program.

Author(s)

Achim Zeileis

Description

The functions or variables listed here are provided for compatibility with older versions of R only, and may be defunct as soon as of the next release.

Usage

See Also

* Deprecated, Defunct

Description

Tools for package development, administration and documentation

Details

This package contains tools for manipulating R packages and their documentation.

For a complete list of functions, use `library(help="tools").`
undoc

Find Undocumented Objects

Description

Finds the objects in a package which are undocumented, in the sense that they are visible to the user (or data objects or S4 classes provided by the package), but no documentation entry exists.

Usage

undoc(package, dir, lib.loc = NULL)

Arguments

package    a character string naming an installed package.
dir        a character string specifying the path to a package’s root source directory. This must contain the subdirectory ‘man’ with R documentation sources (in Rd format), and at least one of the ‘R’ or ‘data’ subdirectories with R code or data objects, respectively.
lib.loc    a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. The specified library trees are used to search for package.

Details

This function is useful for package maintainers mostly. In principle, all user level R objects should be documented; note however that the precise rules for documenting methods of generic functions are still under discussion.

Value

An object of class "undoc" which is a list of character vectors containing the names of the undocumented objects split according to documentation type. This representation is still experimental, and might change in future versions.

There is a print method for nicely displaying the information contained in such objects.

See Also

codoc, QC

Examples

undoc("tools") # Undocumented objects in 'tools'
vignetteDepends

Retrieve Dependency Information for a Vignette

Description
Given a vignette name, will create a DependsList object that reports information about the packages the vignette depends on.

Usage
vignetteDepends(vignette, recursive = TRUE, reduce = TRUE, local = TRUE, lib.loc = NULL)

Arguments
vignette  The path to the vignette source
recursive  Whether or not to include indirect dependencies
reduce    Whether or not to collapse all sets of dependencies to a minimal value
local     Whether or not to search only locally
lib.loc   What libraries to search in locally

Details
If recursive is TRUE, any package that is specified as a dependency will in turn have its dependencies included (and so on), these are known as indirect dependencies. If recursive is FALSE, only the dependencies directly named by the vignette will be used.
If local is TRUE, the system will only look at the user's local machine and not online to find dependencies.
If reduce is TRUE, the system will collapse the fields in the DependsList to the minimal set of dependencies (for instance if the dependencies were ('foo', 'foo (>= 1.0.0)', 'foo (>= 1.3.0)'), the return value would be 'foo (>= 1.3.0)')

Value
An object of class DependsList

Author(s)
Jeff Gentry

See Also
pkgDepends

Examples
gridEx <- system.file("doc", "grid.Snw", package = "grid")
vignetteDepends(gridEx)
write_PACKAGES  Generate PACKAGES files

Description

Generate ‘PACKAGES’ and ‘PACKAGES.gz’ files for a repository of source or Mac/Windows binary packages.

Usage

write_PACKAGES(dir, fields = NULL,
          type = c("source", "mac.binary", "win.binary"),
          verbose = FALSE)

Arguments

dir  Character vector describing the location of the repository (directory including source or binary packages) to generate the ‘PACKAGES’ and ‘PACKAGES.gz’ files from and write them to.

fields a character vector giving the fields to be used in the ‘PACKAGES’ and ‘PACKAGES.gz’ files in addition to the default ones, or NULL (default). The default corresponds to the fields needed by available.packages: “Package”, “Bundle”, “Priority”, “Version”, “Depends”, “Suggests”, “Imports” and “Contains”.

type Type of packages: currently source ‘.tar.gz’ archives, and Mac or Windows binary (‘.tgz’ or ‘.zip’, respectively) packages are supported. Defaults to "win.binary" on Windows and to "source" otherwise.

verbose logical. Should packages be listed as they are processed?

Details

type = "win.binary" uses unz connections to read all ‘DESCRIPTION’ files contained in the (zipped) binary packages for Windows in the given directory dir, and builds ‘PACKAGES’ and ‘PACKAGES.gz’ files from these information.

Value

Invisibly returns the number of packages described in the resulting ‘PACKAGES’ and ‘PACKAGES.gz’ files. If 0, no packages were found and no files were written.

Note

Processing ‘.tar.gz’ archives to extract the ‘DESCRIPTION’ files is quite slow.

This function can be useful on other OSes to prepare a repository to be accessed by Windows machines, so type = "win.binary" should work on all OSes.

Author(s)

Uwe Ligges and R-core.
See Also

See `read.dcf` and `write.dcf` for reading ‘DESCRIPTION’ files and writing the ‘PACKAGES’ and ‘PACKAGES.gz’ files.

Examples

```r
## Not run:
write_PACKAGES("c:/myFolder/myRepository")  # on Windows
write_PACKAGES("/pub/RWin/bin/windows/contrib/2.1",
    type="win.binary")  # on Linux
## End(Not run)
```

---

**xgettext**

Extract Translatable Messages from R Files in a Package

**Description**

For each file in the ‘R’ directory (including system-specific subdirectories) of a package, extract the unique arguments passed to `stop`, `warning`, `message`, `gettext` and `gettextf`, or to `ngettext`.

**Usage**

```
xgettext(dir, verbose = FALSE, asCall = TRUE)
xgettext(dir, verbose = FALSE)
xgettext2pot(dir, potFile)
```

**Arguments**

- `dir` : the directory of a source package.
- `verbose` : logical: should each file be listed as it is processed?
- `asCall` : logical: if TRUE each argument is returned whole, otherwise the strings within each argument are extracted.
- `potFile` : name of po template file to be produced. Defaults to "R-pkg.pot" where pkg is the basename of dir.

**Details**

Leading and trailing white space (space, tab and linefeed) is removed for calls to `gettext`, `gettextf`, `stop`, `warning`, and `message`, as it is by the internal code that passes strings for translation.

We look to see if these functions were called with `domain = NA` and if so omit the call if `asCall = TRUE`: note that the call might contain a call to `gettext` which would be visible if `asCall = FALSE`.

`xgettext2pot` calls `xgettext` and then `xngettext`, and writes a PO template file for use with the GNU Gettext tools. This ensures that the strings for simple translation are unique in the file (as GNU Gettext requires), but does not do so for `ngettext` calls (and the rules are not stated in the Gettext manual).

If applied to the `base` package, this also looks in the .R files in ‘R_HOME/share/R’. 
Value

For `xgettext`, a list of objects of class "xgettext" (which has a print method), one per source file that potentially contains translatable strings.

For `nxgettext`, a list of objects of class "xngettext", which are themselves lists of length-2 character strings.

Examples

```r
## Not run:
## in a source-directory build of R:
xgettext(file.path(R.home(), "src", "library", "splines"))
## End(Not run)
```
Chapter 9

The **utils** package

<table>
<thead>
<tr>
<th>alarm</th>
<th>Alert the user</th>
</tr>
</thead>
</table>

**Description**

Gives an audible or visual signal to the user.

**Usage**

```r
alarm()
```

**Details**

`alarm()` works by sending a `\a` character to the console. On most platforms this will ring a bell, beep, or give some other signal to the user (unless standard output has been redirected).

**Value**

No useful value is returned.

**Examples**

```r
alarm()
```

<table>
<thead>
<tr>
<th>apropos</th>
<th>Find Objects by (Partial) Name</th>
</tr>
</thead>
</table>

**Description**

`apropos` returns a character vector giving the names of all objects in the search list matching `what`.

`find` is a different user interface to the same task as `apropos`. 
Usage

```r
apropos(what, where = FALSE, mode = "any")
find(what, mode = "any", numeric. = FALSE, simple.words = TRUE)
```

Arguments

- `what`: name of an object, or regular expression to match against.
- `where`, `numeric.`: a logical indicating whether positions in the search list should also be returned.
- `mode`: character; if not "any", only objects who's `mode` equals `mode` are searched.
- `simple.words`: logical; if TRUE, the `what` argument is only searched as whole word.

Details

If `mode` != "any" only those objects which are of mode `mode` are considered. If `where` is TRUE, the positions in the search list are returned as the names attribute.

`find` is a different user interface for the same task as `apropos`. However, by default (simple.words == TRUE), only full words are searched with `grep(fixed = TRUE).

Author(s)

Kurt Hornik and Martin Maechler (May 1997).

See Also

- `objects` for listing objects from one place, `help.search` for searching the help system, `search` for the search path.

Examples

```r
## Not run: apropos("lm")
apropos(ls)
apropos("lq")
cor <- 1:pi
find(cor) #> "GlobalEnv" "package:stats"
find(cor, num=TRUE) # numbers with these names
find(cor, num=TRUE, mode="function") # only the second one
rm(cor)
## Not run: apropos(".*", mode="list") # a long list
# need a DOUBLE backslash '\\' (in case you don't see it anymore)
apropos("\."")
## Not run: # everything
length(apropos(".*"))
# those starting with 'pr'
apropos("^pr")
# the 1-letter things
apropos("^\$")
```
# the 1-2-letter things
apropos("^..?\$")
# the 2-to-4 letter things
apropos("^.{2,4}\$")

# the 8-and-more letter things
apropos("^.{8}\$")
table(nchar(apropos("^.{8}\$")))
## End(Not run)

---

**BATCH**

**Batch Execution of R**

### Description

Run R non-interactively with input from infile and send output (stdout/stderr) to another file.

### Usage

```
R CMD BATCH [options] infile [outfile]
```

### Arguments

- **infile**
  - the name of a file with R code to be executed.
- **options**
  - a list of R command line options, e.g., for setting the amount of memory available and controlling the load/save process. If infile starts with a '-', use '--' as the final option. The default options are '--restore --save'.
- **outfile**
  - the name of a file to which to write output. If not given, the name used is that of infile, with a possible '.R' extension stripped, and '.Rout' appended.

### Details

Use `R CMD BATCH --help` to be reminded of the usage.

By default, the input commands are printed along with the output. To suppress this behavior, add `options(echo = FALSE)` at the beginning of infile.

Files with an incomplete last line (no end of line mark) are now accepted.

---

**browseEnv**

**Browse Objects in Environment**

### Description

The `browseEnv` function opens a browser with list of objects currently in `sys.frame()` environment.
Usage

browseEnv(envir = .GlobalEnv, pattern,
            excludepatt = "^last\.\.warning",
            html = .Platform$OS.type != "mac",
            expanded = TRUE, properties = NULL,
            main = NULL, debugMe = FALSE)

Arguments

envir    an environment the objects of which are to be browsed.
pattern  a regular expression for object subselection is passed to the internal ls() call.
excludepatt a regular expression for dropping objects with matching names.
html     is used on non Macintosh machines to display the workspace on a HTML page in your favorite browser.
expanded whether to show one level of recursion. It can be useful to switch it to FALSE if your workspace is large. This option is ignored if html is set to FALSE.
properties a named list of global properties (of the objects chosen) to be showed in the browser; when NULL (as per default), user, date, and machine information is used.
main     a title string to be used in the browser; when NULL (as per default) a title is constructed.
debugMe  logical switch; if true, some diagnostic output is produced.

Details

Very experimental code. Only allows one level of recursion into object structures. The HTML version is not dynamic.
It can be generalized. See sources (‘....library/base/R/databrowser.R’) for details.
wsbrowser() is currently just an internally used function; its argument list will certainly change.
Most probably, this should rather work through using the ‘tkWidget’ package (from www.Bioconductor.org).

See Also

str, ls.

Examples

if(interactive()) {
    ## create some interesting objects :
ofa <- ordered(4:1)
ex1 <- expression(1+ 0:9)
ex3 <- expression(u,v, 1+ 0:9)
example(factor, echo = FALSE)
example(table, echo = FALSE)
example(ftable, echo = FALSE)
example(lm, echo = FALSE)
example(str, echo = FALSE)

    ## and browse them:
browseEnv()
```r
## a (simple) function's environment:
af12 <- approxfun(1:2, 1:2, method = "const")  
browseEnv(envir = environment(af12))
```

---

**browseURL**  
*Load URL into a WWW Browser*

**Description**  
Load a given URL into a WWW browser.

**Usage**  
```r
browseURL(url, browser = getOption("browser"))
```

**Arguments**
- **url**: a non-empty character string giving the URL to be loaded.
- **browser**: a non-empty character string giving the name of the program to be used as hypertext browser. It should be in the PATH, or a full path specified. Under Windows NULL is also allowed (and is the default), and implies that the file association mechanism will be used.

**Examples**
```r
## Not run:
browseURL("http://www.r-project.org")
  browser="C:/Program Files/Netscape/Netscape/netscp.exe")
## End(Not run)
```

---

**bug.report**  
*Send a Bug Report*

**Description**  
Invokes an editor to write a bug report and optionally mail it to the automated r-bugs repository at (r-bugs@r-project.org). Some standard information on the current version and configuration of R are included automatically.

**Usage**  
```r
bug.report(subject = "",  
  ccaddress = Sys.getenv("USER"),  
  method = getOption("mailer"),  
  address = "r-bugs@r-project.org",  
  file = "R.bug.report")
```
bug.report

Arguments

subject Subject of the email. Please do not use single quotes (') in the subject! File separate bug reports for multiple bugs.
ccaddress Optional email address for copies (default is current user). Use ccaddress = FALSE for no copies.
method Submission method, one of "mailx", "gnudoit", "none", or "ess".
address Recipient’s email address.
file File to use for setting up the email (or storing it when method is "none" or sending mail fails).

details

Currently direct submission of bug reports works only on Unix systems. If the submission method is "mailx", then the default editor is used to write the bug report. Which editor is used can be controlled using options, type getOption("editor") to see what editor is currently defined. Please use the help pages of the respective editor for details of usage. After saving the bug report (in the temporary file opened) and exiting the editor the report is mailed using a Unix command line mail utility such as mailx. A copy of the mail is sent to the current user.

If method is "gnudoit", then an emacs mail buffer is opened and used for sending the email.
If method is "none" or NULL (and in every case on Windows systems), then only an editor is opened to help writing the bug report. The report can then be copied to your favorite email program and be sent to the r-bugs list.
If method is "ess" the body of the mail is simply sent to stdout.

value

Nothing useful.

when is there a bug?

If R executes an illegal instruction, or dies with an operating system error message that indicates a problem in the program (as opposed to something like “disk full”), then it is certainly a bug.
Taking forever to complete a command can be a bug, but you must make certain that it was really R’s fault. Some commands simply take a long time. If the input was such that you KNOW it should have been processed quickly, report a bug. If you don’t know whether the command should take a long time, find out by looking in the manual or by asking for assistance.
If a command you are familiar with causes an R error message in a case where its usual definition ought to be reasonable, it is probably a bug. If a command does the wrong thing, that is a bug. But be sure you know for certain what it ought to have done. If you aren’t familiar with the command, or don’t know for certain how the command is supposed to work, then it might actually be working right. Rather than jumping to conclusions, show the problem to someone who knows for certain.
Finally, a command’s intended definition may not be best for statistical analysis. This is a very important sort of problem, but it is also a matter of judgment. Also, it is easy to come to such a conclusion out of ignorance of some of the existing features. It is probably best not to complain about such a problem until you have checked the documentation in the usual ways, feel confident that you understand it, and know for certain that what you want is not available. The mailing list r-devel@r-project.org is a better place for discussions of this sort than the bug list.
If you are not sure what the command is supposed to do after a careful reading of the manual this indicates a bug in the manual. The manual’s job is to make everything clear. It is just as important to report documentation bugs as program bugs.
If the online argument list of a function disagrees with the manual, one of them must be wrong, so report the bug.

How to report a bug

When you decide that there is a bug, it is important to report it and to report it in a way which is useful. What is most useful is an exact description of what commands you type, from when you start \texttt{R} until the problem happens. Always include the version of \texttt{R}, machine, and operating system that you are using; type \texttt{version} in \texttt{R} to print this. To help us keep track of which bugs have been fixed and which are still open please send a separate report for each bug.

The most important principle in reporting a bug is to report FACTS, not hypotheses or categorizations. It is always easier to report the facts, but people seem to prefer to strain to posit explanations and report them instead. If the explanations are based on guesses about how \texttt{R} is implemented, they will be useless; we will have to try to figure out what the facts must have been to lead to such speculations. Sometimes this is impossible. But in any case, it is unnecessary work for us.

For example, suppose that on a data set which you know to be quite large the command \texttt{data.frame(x, y, z, monday, tuesday)} never returns. Do not report that \texttt{data.frame()} fails for large data sets. Perhaps it fails when a variable name is a day of the week. If this is so then when we get your report we would try out the \texttt{data.frame()} command on a large data set, probably with no day of the week variable name, and not see any problem. There is no way in the world that we could guess that we should try a day of the week variable name.

Or perhaps the command fails because the last command you used was a \lbrack method that had a bug causing \texttt{R}'s internal data structures to be corrupted and making the \texttt{data.frame()} command fail from then on. This is why we need to know what other commands you have typed (or read from your startup file).

It is very useful to try and find simple examples that produce apparently the same bug, and somewhat useful to find simple examples that might be expected to produce the bug but actually do not. If you want to debug the problem and find exactly what caused it, that is wonderful. You should still report the facts as well as any explanations or solutions.

Invoking \texttt{R} with the '--vanilla' option may help in isolating a bug. This ensures that the site profile and saved data files are not read.

A bug report can be generated using the \texttt{bug.report()} function. This automatically includes the version information and sends the bug to the correct address. Alternatively the bug report can be emailed to \{r-bugs@r-project.org\} or submitted to the Web page at \url{http://bugs.r-project.org}.

Bug reports on \textbf{contributed packages} should be sent to the package maintainer rather than to r-bugs.

\textbf{Author(s)}

This help page is adapted from the Emacs manual and the \texttt{R FAQ}

\textbf{See Also}

\texttt{R FAQ}
Send output to a character string or file

Description

Evaluates its arguments with the output being returned as a character string or sent to a file. Related to `sink` in the same way that `with` is related to `attach`.

Usage

capture.output(..., file = NULL, append = FALSE)

Arguments

... Expressions to be evaluated
file A file name or a connection, or NULL to return the output as a string. If the connection is not open it will be opened and then closed on exit.
append Append or overwrite the file?

Value

A character string, or NULL if a `file` argument was supplied.

See Also

`sink`, `textConnection`

Examples

```
require(stats)
glmout <- capture.output(example(glm))
glmout[1:5]
capture.output(1+1, 2+2)
capture.output({1+1; 2+2})
## Not run:
## on Unix with enscript available
ps <- pipe("enscript -o tempout.ps","w")
capture.output(example(glm), file=ps)
close(ps)
## End(Not run)
```
choose.files  Choose a List of Files Interactively

Description
Use a Windows file dialog to choose a list of zero or more files interactively.

Usage
choose.files(default = "", caption = "Select files",
    multi = TRUE, filters = Filters, index = nrow(Filters))

Details
Unlike file.choose, this function will always attempt to return a character vector giving a list of files. If the user cancels the dialog, then zero files are returned, whereas file.choose would signal an error.

Windows file dialog boxes include a list of “filters”, which allow the file selection to be limited to files of specific types. The filters argument to choose.files allows the list of filters to be set. It should be an n by 2 character matrix. The first column gives, for each filter, the description the user will see, while the second column gives the mask(s) to select those files. If more than one mask is used, separate them by semicolons, with no spaces. The index argument chooses which filter will be used initially.

Filters is a matrix giving the descriptions and masks for the file types that R knows about. Print it to see typical formats for filter specifications. The examples below show how particular filters may be selected.

If you would like to display files in a particular directory, give a fully qualified file mask (e.g., 'c:\\*.*') in the default argument. If a directory is not given, the dialog will start in the current directory the first time, and “remember” the last directory on subsequent invocations.

Value
A character vector giving zero or more file paths.

See Also
file.choose
Examples

if (interactive())
  choose.files(filters = Filters[c("zip", "All"),])

chooseCRANmirror  Select a CRAN Mirror

Description

Interact with the user to choose a CRAN mirror.

Usage

chooseCRANmirror(graphics = TRUE)

Arguments

  graphics Logical. If true use a list box, otherwise use menu.

Details

The list of mirrors is stored in file ‘R_HOME/doc/CRAN_mirrors.csv’.

This function was originally written to support a Windows GUI menu item, but is also called by contrib.url if it finds the initial dummy value of options("repos").

Value

None. This function is invoked for its side effect of updating options("repos")

See Also

setRepositories, contrib.url.

citation  Citing R and R Packages in Publications

Description

How to cite R and R packages in publications.

Usage

citation(package = "base", lib.loc = NULL)
## S3 method for class 'citation':
toBibtex(object, ...)
## S3 method for class 'citationList':
toBibtex(object, ...)
**citation**

Arguments

- **package**: A character string with the name of a single package. An error occurs if more than one package name is given.
- **lib.loc**: A character vector with path names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.
- **object**: Return object of citation.
- **...**: Currently not used.

Details

The R core development team and the very active community of package authors have invested a lot of time and effort in creating R as it is today. Please give credit where credit is due and cite R and R packages when you use them for data analysis.

Execute function `citation()` for information on how to cite the base R system in publications. If the name of a non-base package is given, the function either returns the information contained in the CITATION file of the package or auto-generates citation information. In the latter case the package 'DESCRIPTION' file is parsed, the resulting citation object may be arbitrarily bad, but is quite useful (at least as a starting point) in most cases.

If only one reference is given, the print method shows both a text version and a BibTeX entry for it, if a package has more than one reference then only the text versions are shown. The BibTeX versions can be obtained using function `toBibtex` (see the examples below).

Value

An object of class "citationList".

See Also

citEntry

Examples

```r
## the basic R reference
citation()

## references for a package -- might not have these installed
if(nchar(system.file(package="lattice"))) citation("lattice")
if(nchar(system.file(package="foreign"))) citation("foreign")

## extract the bibtex entry from the return value
x <- citation()
toBibtex(x)
```
citEntry

Writing Package CITATION Files

Description

The `CITATION` file of R packages contains an annotated list of references that should be used for citing the packages.

Usage

citEntry(entry, textVersion, header = NULL, footer = NULL, ...)
citHeader(...)
citFooter(...)
readCitationFile(file)

Arguments

- entry: a character string with a BibTeX entry type
- textVersion: a character string with a text representation of the reference
- header: a character string with optional header text
- footer: a character string with optional footer text
- file: a file name
- ...: see details below

Details

The `CITATION` file of an R package should be placed in the `inst` subdirectory of the package source. The file is an R source file and may contain arbitrary R commands including conditionals and computations. The file is source()ed by the R parser in a temporary environment and all resulting objects of class "citation" (the return value of citEntry) are collected.

Typically the file will contain zero or more calls to citHeader, then one or more calls to citEntry, and finally zero or more calls to citFooter. citHeader and citFooter are simply wrappers to paste, and their ... argument is passed on to paste as is.

Value

citEntry returns an object of class "citation", readCitationFile returns an object of class "citationList".

Entry Types

citEntry creates "citation" objects, which are modeled after BibTeX entries. The entry should be a valid BibTeX entry type, e.g.,

- article: An article from a journal or magazine.
- inbook: A part of a book, which may be a chapter (or section or whatever) and/or a range of pages.
- inproceedings: An article in a conference proceedings.
**citEntry**

**manual**: Technical documentation like a software manual.

**mastersthesis**: A Master's thesis.

**misc**: Use this type when nothing else fits.

**phdthesis**: A PhD thesis.

**proceedings**: The proceedings of a conference.

**techreport**: A report published by a school or other institution, usually numbered within a series.

**unpublished**: A document having an author and title, but not formally published.

**Entry Fields**

The ... argument of citEntry can be any number of BibTeX fields, including

- **address**: The address of the publisher or other type of institution.
- **author**: The name(s) of the author(s), either as a character string in the format described in the LaTeX book, or a personList object.
- **booktitle**: Title of a book, part of which is being cited.
- **chapter**: A chapter (or section or whatever) number.
- **editor**: Name(s) of editor(s), same format as author.
- **institution**: The publishing institution of a technical report.
- **journal**: A journal name.
- **note**: Any additional information that can help the reader. The first word should be capitalized.
- **number**: The number of a journal, magazine, technical report, or of a work in a series.
- **pages**: One or more page numbers or range of numbers.
- **publisher**: The publisher’s name.
- **school**: The name of the school where a thesis was written.
- **series**: The name of a series or set of books.
- **title**: The work’s title.
- **volume**: The volume of a journal or multi-volume book.
- **year**: The year of publication.

**Examples**

```r
basecit <- system.file("CITATION", package="base")
source(basecit, echo=TRUE)
readCitationFile(basecit)
```
clipboard  

Read/Write Text to/from the Windows Clipboard

Description
Transfer text between a character vector and the Windows clipboard.

Usage
readClipboard()
writeClipboard(str)

Arguments
str   a character vector.

Value
For readClipboard, a character vector. For writeClipboard a invisible logical indicating success or failure.

See Also
file which can be used to set up a connection to a clipboard.

close.socket  

Close a Socket

Description
Closes the socket and frees the space in the file descriptor table. The port may not be freed immediately.

Usage
close.socket(socket, ...)

Arguments
socket   A socket object
...     further arguments passed to or from other methods.

Value
logical indicating success or failure

Author(s)
Thomas Lumley

See Also
make.socket, read.socket
```r
compareVersion

Compare Two Package Version Numbers

Description

Compare two package version numbers to see which is later.

Usage

compareVersion(a, b)

Arguments

a, b

Character strings representing package version numbers.

Details

R package version numbers are of the form x.y–z for integers x, y and z, with components after x optionally missing (in which case the version number is older than those with the components present).

Value

0 if the numbers are equal, −1 if b is later and 1 if a is later (analogous to the C function strcmp).

See Also

package_version, library, packageStatus.

Examples

compareVersion("1.0", "1.0-1")
compareVersion("7.2-0","7.1-12")
```

```r
data

Data Sets

Description

Loads specified data sets, or list the available data sets.

Usage

data(..., list = character(0), package = NULL, lib.loc = NULL,
    verbose =getOption("verbose"), envir = .GlobalEnv)
```
Arguments

... a sequence of names or literal character strings.

list a character vector.

package a character vector giving the package(s) to look in for data sets, or NULL.

By default, all packages in the search path are used, then the ‘data’ subdirectory (if present) of the current working directory.

lib.loc a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known.

verbose a logical. If TRUE, additional diagnostics are printed.

envir the environment where the data should be loaded.

Details

Currently, four formats of data files are supported:

1. files ending `.R` or `.r` are `source()` d in, with the R working directory changed temporarily to the directory containing the respective file.
2. files ending `.RData` or `.rda` are `load()` ed.
3. files ending `.tab`, `.txt` or `.TXT` are read using `read.table(..., header = TRUE)`, and hence result in a data frame.
4. files ending `.csv` or `.CSV` are read using `read.table(..., header = TRUE, sep = "","), and also result in a data frame.

If more than one matching file name is found, the first on this list is used.

The data sets to be loaded can be specified as a sequence of names or character strings, or as the character vector list, or as both.

For each given data set, the first two types (`.R` or `.r`, and `.RData` or `.rda` files) can create several variables in the load environment, which might all be named differently from the data set. The second two (`.tab`, `.txt`, or `.TXT`, and `.csv` or `.CSV` files) will always result in the creation of a single variable with the same name as the data set.

If no data sets are specified, `data` lists the available data sets. It looks for a new-style data index in the ‘Meta’ or, if this is not found, an old-style ‘00Index’ file in the ‘data’ directory of each specified package, and uses these files to prepare a listing. If there is a ‘data’ area but no index, available data files for loading are computed and included in the listing, and a warning is given: such packages are incomplete. The information about available data sets is returned in an object of class "packageIQR". The structure of this class is experimental. Where the datasets have a different name from the argument that should be used to retrieve them the index will have an entry like beaver1 (beavers) which tells us that dataset beaver1 can be retrieved by the call `data(beaver)`.

If `lib.loc` and `package` are both NULL (the default), the data sets are searched for in all the currently loaded packages then in the ‘data’ directory (if any) of the current working directory. If `lib.loc` = NULL but `package` is specified as a character vector, the specified package(s) are searched for first amongst loaded packages and then in the default library/ies (see `.libPaths`). If `lib.loc` is specified (and not NULL), packages are searched for in the specified library/ies, even if they are already loaded from another library.

To just look in the ‘data’ directory of the current working directory, set `package = character(0)` (and `lib.loc = NULL`, the default).
Value

A character vector of all data sets specified, or information about all available data sets in an object of class "packageIQR" if none were specified.

Note

The data files can be many small files. On some file systems it is desirable to save space, and the files in the 'data' directory of an installed package can be zipped up as a zip archive 'Rdata.zip'. You will need to provide a single-column file 'filelist' of file names in that directory.

One can take advantage of the search order and the fact that a '.R' file will change directory. If raw data are stored in 'mydata.txt' then one can set up 'mydata.R' to read 'mydata.txt' and preprocess it, e.g., using transform. For instance one can convert numeric vectors to factors with the appropriate labels. Thus, the '.R' file can effectively contain a metadata specification for the plaintext formats.

See Also

help for obtaining documentation on data sets, save for creating the second ('.rda') kind of data, typically the most efficient one.

Examples

```r
require(utils)
data() # list all available data sets
try(data(package = "rpart")) # list the data sets in the rpart package
data(USArrests, "VADeaths") # load the data sets 'USArrests' and 'VADeaths'
help(USArrests) # give information on data set 'USArrests'
```

**dataentry**  
Spreadsheet Interface for Entering Data

Description

A spreadsheet-like editor for entering or editing data.

Usage

```r
data.entry(..., Modes = NULL, Names = NULL)
dataentry(data, modes)
de(..., Modes = list(), Names = NULL)
```

Arguments

... A list of variables: currently these should be numeric or character vectors or list containing such vectors.
Modes The modes to be used for the variables.
Names The names to be used for the variables.
data A list of numeric and/or character vectors.
modes A list of length up to that of data giving the modes of (some of) the variables.
list() is allowed.
Details

The data entry editor is only available on some platforms and GUIs. Where available it provides a means to visually edit a matrix or a collection of variables (including a data frame) as described in the “Notes” section.

data.entry has side effects, any changes made in the spreadsheet are reflected in the variables. The functions de, de.ncols, de.setup and de.restore are designed to help achieve these side effects. If the user passes in a matrix, X say, then the matrix is broken into columns before dataentry is called. Then on return the columns are collected and glued back together and the result assigned to the variable X. If you don’t want this behaviour use dataentry directly.

The primitive function is dataentry. It takes a list of vectors of possibly different lengths and modes (the second argument) and opens a spreadsheet with these variables being the columns. The columns of the dataentry window are returned as vectors in a list when the spreadsheet is closed.

de.ncols counts the number of columns which are supplied as arguments to data.entry. It attempts to count columns in lists, matrices and vectors. de.setup sets things up so that on return the columns can be regrouped and reassigned to the correct name. This is handled by de.restore.

Value

de and dataentry return the edited value of their arguments. data.entry invisibly returns a vector of variable names but its main value is its side effect of assigning new version of those variables in the user’s workspace.

Note

The details of interface to the data grid may differ by platform and GUI. The following description applies to the GraphApp-based implementation under Windows.

You can navigate around the grid using the cursor keys or by clicking with the (left) mouse button on any cell. The active cell is highlighted by thickening the surrounding rectangle. Moving to the right or down will scroll the grid as needed: there is no constraint to the rows or columns currently in use.

There are alternative ways to navigate using the keys. Return and (keypad) Enter and LineFeed all move down. Tab moves right and Shift-Tab move left. Home moves to the top left.

PageDown or Control-F moves down a page, and PageUp or Control-B up by a page. End will show the last used column and the last few rows used (in any column).

Using any other key starts an editing process on the currently selected cell: moving away from that cell enters the edited value whereas Esc cancels the edit and restores the previous value. When the editing process starts the cell is cleared. The cursor changes to an I-beam to indicate that the cell is in enter mode. In numerical columns (the default) only letters making up a valid number (including -.eE) are accepted, and entering an invalid edited value (such as blank) enters NA in that cell. The last entered value can be deleted using the BackSpace or Del(ete) key. Only a limited number of characters (currently 29) can be entered in a cell, and if necessary only the start or end of the string will be displayed, with the omissions indicated by > or <. (The start is shown except when editing.) Double-clicking on a cell selects the cell and makes it into an editable field (a cursor will appear at the end of the text and it will change to the text highlight colour). The edited text is entered by selecting another cell, for example by hitting Return. There is no way to cancel the edits. The field will be expanded to the right if necessary to accommodate existing long strings, so it is preferable not to edit in the right-most displayed column. (The editable field is itself scrollable.)

Entering a value in a cell further down a column than the last used cell extends the variable and fills the gap (if any) by NAs (not shown on screen).
The column names can only be selected by clicking in them. This gives a popup menu to select the column type (currently Real (numeric) or Character) or to change the name. Changing the type converts the current contents of the column (and converting from Character to Real may generate NAs.) Enter the changes made in the popup window by clicking on its close box.

New columns are created by entering values in them (and not by just assigning a new name). The mode of the column is auto-detected from the first value entered: if this is a valid number it gives a numeric column. Unused columns are ignored, so adding data in `var5` to a three-column grid adds one extra variable, not two.

There is a popup-menu accessed by right-clicking anywhere in the window that refers to the currently selected cell. This can copy the value to or paste from the clipboard, or paste in common values in that column. Copying and pasting can also be accessed by the usual keyboard shortcuts Control-C and Control-V.

Columns can be resized by selecting and dragging a line (the cursor will change) within limits: columns must be between 4 and 50 chars wide. The Autosize item on the popup menu will resize the currently selected column.

Control-L will refresh the display, recalculating field widths to fit the current entries.

In the default mode the column widths are chosen to fit the contents of each column, with a default of 10 characters for empty columns. you can specify fixed column widths by setting option `de.cellwidth` to the required fixed width (in characters). (set it to zero to return to variable widths). The displayed width of any field is limited to 50 characters (and by the window width).

**See Also**

`vi.edit`: edit uses dataentry to edit data frames.

**Examples**

```r
# call data entry with variables x and y
## Not run: data.entry(x,y)
```

---

### debugger

**Post-Mortem Debugging**

**Description**

Functions to dump the evaluation environments (frames) and to examine dumped frames.

**Usage**

```r
dump.frames(dumpto = "last.dump", to.file = FALSE)
debugger(dump = last.dump)
```

**Arguments**

- `dumpto`: a character string. The name of the object or file to dump to.
- `to.file`: logical. Should the dump be to an R object or to a file?
- `dump`: An R dump object created by `dump.frames`. 
To use post-mortem debugging, set the option `error` to be a call to `dump.frames`. By default this dumps to an R object "last.dump" in the workspace, but it can be set to dump to a file (as dump of the object produced by a call to `save`). The dumped object contain the call stack, the active environments and the last error message as returned by `geterrmessage`.

When dumping to file, `dumpto` gives the name of the dumped object and the file name has `.rda` appended.

A dump object of class "dump.frames" can be examined by calling `debugger`. This will give the error message and a list of environments from which to select repeatedly. When an environment is selected, it is copied and the `browser` called from within the copy.

If `dump.frames` is installed as the error handler, execution will continue even in non-interactive sessions. See the examples for how to dump and then quit.

**Value**

None.

**Note**

Functions such as `sys.parent` and `environment` applied to closures will not work correctly inside `debugger`.

Of course post-mortem debugging will not work if R is too damaged to produce and save the dump, for example if it has run out of workspace.

**References**


**See Also**

`options` for setting `error` options; `recover` is an interactive debugger working similarly to `debugger` but directly after the error occurs.

**Examples**

```r
## Not run:
options(error=quote(dump.frames("testdump", TRUE)))

f <- function() {
  g <- function() stop("test dump.frames")
  g()
}
f() # will generate a dump on file "testdump.rda"
options(error=NULL)

## possibly in another R session
load("testdump.rda")
debugger(testdump)
Available environments had calls:
1: f()
2: g()
3: stop("test dump.frames")
```
Enter an environment number, or 0 to exit
Selection: 1
Browsing in the environment with call:
f()
Called from: debugger.look(ind)
Browse[1]> ls()
 [1] "g"
Browse[1]> g
function() stop("test dump.frames")
<environment: 759818>
Browse[1]>
Available environments had calls:
1: f()
2: g()
3: stop("test dump.frames")
Enter an environment number, or 0 to exit
Selection: 0

## A possible setting for non-interactive sessions
options(error=quote({dump.frames(to.file=TRUE); q()}))
## End(Not run)

demo

Demonstrations of R Functionality

Description
demo is a user-friendly interface to running some demonstration R scripts. demo() gives the list of available topics.

Usage
demo(topic, package = NULL, lib.loc = NULL,
    character.only = FALSE, verbose = getOption("verbose"))

Arguments
topic the topic which should be demonstrated, given as a name or literal character string, or a character string, depending on whether character.only is FALSE (default) or TRUE. If omitted, the list of available topics is displayed.
package a character vector giving the packages to look into for demos, or NULL. By default, all packages in the search path are used.
lib.loc a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.
character.only logical; if TRUE, use topic as character string.
verbose a logical. If TRUE, additional diagnostics are printed.
Details

If no topics are given, demo lists the available demos. The corresponding information is returned in an object of class "packageIQR". The structure of this class is experimental. In earlier versions of R, an empty character vector was returned along with listing available demos.

See Also

source which is called by demo.

Examples

demo() # for attached packages

## All available demos:
demo(package = .packages(all.available = TRUE))

demo(lm.glm, package="stats")
## Not run:
ch <- "scoping"
demo(ch, character = TRUE)
## End(Not run)

Description

Return the version of the package and the version of R used to build the DLL, if available (usually only since R version 1.2.0).

Usage

DLL.version(path)

Arguments

path character vector of length one giving the complete path to the DLL.

Value

If the DLL does not exist, NULL.

A character vector of two, giving the DLL version and the version of R used to build the DLL. If the information is not available, the corresponding string is empty.

Examples

DLL.version(file.path(R.home(), "bin/R.dll"))
DLL.version(file.path(R.home(), "library/stats/libso/stats.dll"))
download.file  Download File from the Internet

Description

This function can be used to download a file from the Internet.

Usage

```r
download.file(url, destfile, method, quiet = FALSE, mode = "w", cacheOK = TRUE)
```

Arguments

- `url`: A character string naming the URL of a resource to be downloaded.
- `destfile`: A character string with the name where the downloaded file is saved. Tilde-expansion is performed.
- `method`: Method to be used for downloading files. Currently download methods "internal", "wget" and "lynx" are available, and there is a value "auto": see Details. The method can also be set through the option "download.file.method": see options().
- `quiet`: If TRUE, suppress status messages (if any).
- `mode`: character. The mode with which to write the file. Useful values are "w", "wb" (binary), "a" (append) and "ab". Only used for the "internal" method.
- `cacheOK`: logical. Is a server-side cached value acceptable? Implemented for the "internal" and "wget" methods.

Details

The function `download.file` can be used to download a single file as described by `url` from the internet and store it in `destfile`. The `url` must start with a scheme such as "http://", "ftp://" or "file://".

If `method = "auto"` is chosen (the default), the internal method is used on Windows.

`cacheOK = FALSE` is useful for "http://" URLs, and will attempt to get a copy directly from the site rather than from an intermediate cache. (Not all platforms support it.) It is used by `CRAN.packages`.

The remaining details apply to method "internal" only.

See `url` for how "file://" URLs are interpreted, especially on Windows. This function does decode encoded URLs.

The timeout for many parts of the transfer can be set by the option `timeout` which defaults to 60 seconds.

The level of detail provided during transfer can be set by the `quiet` argument and the `internet.info` option. The details depend on the platform and scheme, but setting `internet.info` to 0 gives all available details, including all server responses. Using 2 (the default) gives only serious messages, and 3 or more suppresses all messages.

A progress bar tracks the transfer. If the file length is known, the full width of the bar is the known length. Otherwise the initial width represents 100Kbytes and is doubled whenever the current width is exceeded.
There is an alternative method if you have Internet Explorer 4 or later installed. You can use the flag `--internet2`, when the ‘Internet Options’ of the system are used to choose proxies and so on; these are set in the Control Panel and are those used for Internet Explorer. This version does not support `cacheOK = FALSE`.

Method "wget" can be used with proxy firewalls which require user/password authentication if proper values are stored in the configuration file for wget.

**Value**

An (invisible) integer code, 0 for success and non-zero for failure. For the "wget" and "lynx" methods this is the status code returned by the external program. The "internal" method can return 1, but will in most cases throw an error.

**Setting Proxies**

This applies to the internal code only.

Proxies can be specified via environment variables. Setting "no_proxy" stops any proxy being tried. Otherwise the setting of "http_proxy" or "ftp_proxy" (or failing that, the all upper-case version) is consulted and if non-empty used as a proxy site. For FTP transfers, the username and password on the proxy can be specified by "ftp_proxy_user" and "ftp_proxy_password". The form of "http_proxy" should be "http://proxy.dom.com/" or "http://proxy.dom.com:8080/" where the port defaults to 80 and the trailing slash may be omitted. For "ftp_proxy" use the form "ftp://proxy.dom.com:3128/" where the default port is 21. These environment variables must be set before the download code is first used: they cannot be altered later by calling `Sys.putenv`.

Usernames and passwords can be set for HTTP proxy transfers via environment variable `http_proxy_user` in the form user:passwd. Alternatively, "http_proxy" can be of the form "http://user:pass@proxy.dom.com:8080/" for compatibility with wget. Only the HTTP/1.0 basic authentication scheme is supported. Under Windows, if "http_proxy_user" is set to "ask" then a dialog box will come up for the user to enter the username and password. **NB:** you will be given only one opportunity to enter this, but if proxy authentication is required and fails there will be one further prompt per download.

**Note**

Methods "wget" and "lynx" are for historical compatibility. They will block all other activity on the R process.

For methods "wget" and "lynx" a system call is made to the tool given by method, and the respective program must be installed on your system and be in the search path for executables.

**See Also**

- `options` to set the timeout and `internet.info` options.
- `url` for a finer-grained way to read data from URLs.
- `url.show`, `CRAN.packages`, `download.packages` for applications
Invoke a Text Editor

Description

Invoke a text editor on an R object.

Usage

```r
## Default S3 method:
edit(name = NULL, file = "", title = NULL,
    editor = getOption("editor"), ...)
```

```r
vi(name = NULL, file = "")
emacs(name = NULL, file = "")
pico(name = NULL, file = "")
xemacs(name = NULL, file = "")
xedit(name = NULL, file = "")
```

Arguments

- **name**: a named object that you want to edit. If name is missing then the file specified by file is opened for editing.
- **file**: a string naming the file to write the edited version to.
- **title**: a display name for the object being edited.
- **editor**: a string naming the text editor you want to use. On Unix the default is set from the environment variables EDITOR or VISUAL if either is set, otherwise vi is used. On Windows it defaults to notepad.
- **...**: further arguments to be passed to or from methods.

Details

edit invokes the text editor specified by editor with the object name to be edited. It is a generic function, currently with a default method and one for data frames and matrices.

data.entry can be used to edit data, and is used by edit to edit matrices and data frames on systems for which data.entry is available.

It is important to realize that edit does not change the object called name. Instead, a copy of name is made and it is that copy which is changed. Should you want the changes to apply to the object name you must assign the result of edit to name. (Try fix if you want to make permanent changes to an object.)

In the form edit(name), edit deparses name into a temporary file and invokes the editor editor on this file. Quitting from the editor causes file to be parsed and that value returned. Should an error occur in parsing, possibly due to incorrect syntax, no value is returned. Calling edit(), with no arguments, will result in the temporary file being reopened for further editing.

Currently only the internal editor in Windows makes use of the title option; it displays the given name in the window header.
Note

The functions vi, emacs, pico, xemacs, xedit rely on the corresponding editor being available and being on the path. This is system-dependent.

See Also

edit.data.frame, data.entry, fix.

Examples

```r
## Not run:
# use xedit on the function mean and assign the changes
mean <- edit(mean, editor = "xedit")
# use vi on mean and write the result to file mean.out
vi(mean, file = "mean.out")
## End(Not run)
```

Description

Use data editor on data frame or matrix contents.

Usage

```r
## S3 method for class 'data.frame':
edit(name, factor.mode = c("character", "numeric"),
     edit.row.names = any(row.names(name) != 1:nrow(name)), ...)

## S3 method for class 'matrix':
edit(name, edit.row.names = any(rownames(name) != 1:nrow(name)), ...)
```

Arguments

- `name`: A data frame or matrix.
- `factor.mode`: How to handle factors (as integers or using character levels) in a data frame.
- `edit.row.names`: logical. Show the row names be displayed as a separate editable column?
- `...`: further arguments passed to or from other methods.

Details

At present, this only works on simple data frames containing numeric, logical or character vectors and factors. Factors are represented in the spreadsheet as either numeric vectors (which is more suitable for data entry) or character vectors (better for browsing). After editing, vectors are padded with `NA` to have the same length and factor attributes are restored. The set of factor levels can not be changed by editing in numeric mode; invalid levels are changed to `NA` and a warning is issued. If new factor levels are introduced in character mode, they are added at the end of the list of levels in the order in which they encountered.
It is possible to use the data-editor’s facilities to select the mode of columns to swap between numerical and factor columns in a data frame. Changing any column in a numerical matrix to character will cause the result to be coerced to a character matrix. Changing the mode of logical columns is not supported.

The columns are coerced on input to numeric unless logical, character or factor (which may well not be what you want), and character columns not protected by `i()` will be coerced to factor on return.

**Value**

The edited data frame.

**Note**

`fix(dataframe)` works for in-place editing by calling this function.

If the data editor is not available, a dump of the object is presented for editing using the default method of `edit`. At present the data editor is limited to 65535 rows.

**Author(s)**

Peter Dalgaard

**See Also**

`data.entry`, `edit`

**Examples**

```r
## Not run:
edit(InsectSprays)
edit(InsectSprays, factor.mode="numeric")
## End(Not run)
```

---

**Description**

Run all the R code from the Examples part of R’s online help topic `topic` with two possible exceptions, `dont.run` and `dontshow`, see Details below.

**Usage**

```r
example(topic, package = NULL, lib.loc = NULL, 
local = FALSE, echo = TRUE, verbose = getOption("verbose"), 
setRNG = FALSE, 
prompt.echo = paste(abbreviate(topic, 6),"> ", sep=""))
```
Example

Arguments

- **topic**
  - name or literal character string: the online `help` topic the examples of which should be run.

- **package**
  - a character vector giving the package names to look into for example code, or `NULL`. By default, all packages in the search path are used.

- **lib.loc**
  - a character vector of directory names of R libraries, or `NULL`. The default value of `NULL` corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.

- **local**
  - logical: if `TRUE` evaluate locally, if `FALSE` evaluate in the workspace.

- **echo**
  - logical; if `TRUE`, show the R input when sourcing.

- **verbose**
  - logical; if `TRUE`, show even more when running example code.

- **setRNG**
  - logical or expression; if not `FALSE`, the random number generator state is saved, then initialized to a specified state, the example is run and the (saved) state is restored. `setRNG = TRUE` sets the same state as `R CMD check` does for running a package’s examples. This is currently equivalent to `setRNG = {RNGkind("default", "default"); set.seed(1)}`.

- **prompt.echo**
  - character; gives the prompt to be used if `echo = TRUE`.

Details

If `lib.loc` is not specified, the packages are searched for amongst those already loaded, then in the specified libraries. If `lib.loc` is specified, they are searched for only in the specified libraries, even if they are already loaded from another library.

An attempt is made to load the package before running the examples, but this will not replace a package loaded from another location.

If `local=TRUE` objects are not created in the workspace and so not available for examination after `example` completes: on the other hand they cannot clobber objects of the same name in the workspace.

As detailed in the manual *Writing R Extensions*, the author of the help page can markup parts of the examples for two exception rules

- **dontrun** encloses code that should not be run.
- **dontshow** encloses code that is invisible on help pages, but will be run both by the package checking tools, and the `example()` function. This was previously `testonly`, and that form is still accepted.

If the examples file contains non-ASCII characters the encoding used will matter. If in a UTF-8 locale `example` first tries UTF-8 and then Latin-1. (This can be overridden by setting the encoding in the `.Rd` file.)

Value

The value of the last evaluated expression.

Note

The examples can be many small files. On some file systems it is desirable to save space, and the files in the ‘R-ex’ directory of an installed package can be zipped up as a zip archive ‘Rex.zip’.
Description

Edit one or more files in a text editor.

Usage

    file.edit(..., title = file, editor = getOption("editor"))

Arguments

... one or more character vectors containing the names of the files to be edited.

    title the title to use in the editor; defaults to the filename.

    editor the text editor to be used.

Details

The behaviour of this function is very system dependent. Currently files can be opened only one at a time on Unix; on Windows, the internal editor allows multiple files to be opened, but has a limit of 50 simultaneous edit windows.

The title argument is used for the window caption in Windows, and is ignored on other platforms.

See Also

    files, file.show, edit, fix,
Examples

```r
## Not run:
# open two R scripts for editing
text.edit("script1.R", "script2.R")
## End(Not run)
```

### fix

**Fix an Object**

**Description**

`fix` invokes `edit` on `x` and then assigns the new (edited) version of `x` in the user's workspace.

**Usage**

`fix(x, ...)`

**Arguments**

- `x` the name of an R object, as a name or a character string.
- `...` arguments to pass to editor: see `edit`.

**Details**

The name supplied as `x` need not exist as an R object, in which case a function with no arguments and an empty body is supplied for editing.

**See Also**

`edit`, `edit.data.frame`

### Examples

```r
## Not run:
## Assume 'my.fun' is a user defined function:
fix(my.fun)
## now my.fun is changed
## Also,
fix(my.data.frame) # calls up data editor
fix(my.data.frame, factor.mode="char") # use of ...
## End(Not run)
```

### flush.console

**Flush Output to A Console**

**Description**

This does nothing except on console-based versions of R. On the Mac OS X and Windows GUIs, it ensures that the display of output in the console is current, even if output buffering is on.

**Usage**

`flush.console()`
getAnywhere

Retrieve an R Object, Including from a Namespace

Description
This function locates all objects with name matching its argument, whether visible on the search path, registered as an S3 method or in a namespace but not exported.

Usage
getAnywhere(x)

Arguments
x a character string or name.

Details
The function looks at all loaded namespaces, whether or not they are associated with a package on the search list.
Where functions are found as an S3 method, an attempt is made to find which namespace registered them. This may not be correct, especially if a namespace is unloaded.

Value
An object of class "getAnywhere". This is a list with components
name the name searched for.
objs a list of objects found
where a character vector explaining where the object(s) were found
visible logical: is the object visible
dups logical: is the object identical to one earlier in the list.

Normally the structure will be hidden by the print method. There is a [ method to extract one or more of the objects found.

See Also
get, getFromNamespace

Examples
getAnywhere("format.dist")
getAnywhere("simpleLoess") # not exported from stats
getFromNamespace  Utility functions for Developing Namespaces

Description

Utility functions to access and replace the non-exported functions in a namespace, for use in developing packages with namespaces.

Usage

getFromNamespace(x, ns, pos = -1, envir = as.environment(pos))
assignInNamespace(x, value, ns, pos = -1, envir = as.environment(pos))
fixInNamespace(x, ns, pos = -1, envir = as.environment(pos), ...)

Arguments

x    an object name (given as a character string).
value an R object.
ns    a namespace, or character string giving the namespace.
pos   where to look for the object: see get.
envir an alternative way to specify an environment to look in.
...  arguments to pass to the editor: see edit.

Details

The namespace can be specified in several ways. Using, for example, ns = "stats" is the most direct, but a loaded package with a namespace can be specified via any of the methods used for get: ns can also be the environment printed as <namespace:foo>.
getFromNamespace is similar to (but predates) the ::: operator, but is more flexible in how the namespace is specified.
fixInNamespace invokes edit on the object named x and assigns the revised object in place of the original object. For compatibility with fix, x can be unquoted.

Value

getFromNamespace returns the object found (or gives an error).
assignInNamespace and fixInNamespace are invoked for their side effect of changing the object in the namespace.

Note

assignInNamespace and fixInNamespace change the copy in the namespace, but not any copies already exported from the namespace, in particular an object of that name in the package (if already attached) and any copies already imported into other namespaces. They are really intended to be used only for objects which are not exported from the namespace. They do attempt to alter a copy registered as an S3 method if one is found.

See Also

g, f, getS3method
getS3method

Get An S3 Method

Description

Get a method for an S3 generic, possibly from a namespace.

Usage

gets3method(f, class, optional = FALSE)

Arguments

f character: name of the generic.
class character: name of the class.
optional logical: should failure to find the generic or a method be allowed?

Details

S3 methods may be hidden in packages with namespaces, and will not then be found by get: this function can retrieve such functions, primarily for debugging purposes.

Value

The function found, or NULL if no function is found and optional = TRUE.

See Also

methods, get

Examples

require(stats)
exists("predict.ppr") # false
gets3method("predict", "ppr")
getWindowsHandle  Get a Windows Handle

Description

Get the Windows handle of a window or the R process.

Usage

getWindowsHandle(which = "Console")

Arguments

which  A string (see below), or the number of a graphics device window

Details

getWindowsHandle gets the Windows handle or process ID. Possible choices for which are:

- "Console"  The console window handle.
- "Frame"  The MDI frame window handle.
- "Process"  The process pseudo-handle.
- "ProcessId"  The process identifier.
- A device number  The window handle of a graphics device

These values are not normally useful to users, but may be used by developers making add-ons to R.

A zero is returned for the Frame handle if not running in MDI mode, for the Console handle when running Rterm, for any unrecognized string for which, or for a graphics device with no corresponding window.

Currently other windows (help browsers, etc.) are not accessible through this function, but there are plans to make them available later.

Value

A one element integer vector holding the Windows handle.

See Also

getIdentification

Examples

getWindowsHandle()
Change wildcard aka globbing (or “ls” like) pattern into the corresponding regular expression (regexp).

Usage

```r
glob2rx(pattern, trim.head = FALSE, trim.tail = TRUE)
```

Arguments

- `pattern` character vector
- `trim.head` logical specifying if leading "^.*" should be trimmed from the result.
- `trim.tail` logical specifying if trailing ".*$" should be trimmed from the result.

Value

a character vector of the same length as the input `pattern` where the “wild card” is translated to the corresponding regular expression.

Author(s)

Martin Maechler, Unix/sed based version, 1991; current: 2004

See Also

`regexp` about regular expression, `sub`, etc about substitutions using regexps.

Examples

```r
stopifnot(glob2rx("abc.*") == "^abc\..",
glob2rx("a?b.*") == "^a.b\..",
glob2rx("a?b.*", trim.tail=FALSE) == "^a.b\..$",
glob2rx("*.doc") == "^.*\..doc$",
glob2rx("*.doc", trim.head=TRUE) == "\..doc$",
glob2rx("*.t*") == "^.*\..t",
glob2rx("*.t??") == "^.*\..t$"
)
```
**Description**

Returns the first or last parts of a vector, matrix, data frame or function.

**Usage**

```r
head(x, ...)  
## Default S3 method:  
head(x, n = 6, ...)  
## S3 method for class 'data.frame':  
head(x, n = 6, ...)  
## S3 method for class 'matrix':  
head(x, n = 6, ...)  

tail(x, ...)  
## Default S3 method:  
tail(x, n = 6, ...)  
## S3 method for class 'data.frame':  
tail(x, n = 6, ...)  
## S3 method for class 'matrix':  
tail(x, n = 6, addrownums = TRUE, ...)  
```

**Arguments**

- `x`: an object
- `n`: size for the resulting object: number of elements for a vector (including lists), rows for a matrix or data frame or lines for a function.
- `addrownums`: if there are no row names, create them from the row numbers.
- `...`: arguments to be passed to or from other methods.

**Details**

For matrices and data frames, the first/last \( n \) rows are returned. For functions, the first/last \( n \) lines of the deparsed function are returned as character strings.

If a matrix has no row names, then `tail()` will add row names of the form "[n,]" to the result, so that it looks similar to the last lines of `x` when printed. Setting `addrownums = FALSE` suppresses this behaviour.

**Value**

An object (usually) like `x` but generally smaller.

**Author(s)**

Patrick Burns, improved and corrected by R-Core
Examples

head(freeny.x, n = 10)
head(freeny.y)

tail(freeny.x)
tail(freeny.y)
tail(library)

Description

These functions provide access to documentation. Documentation on a topic with name name (typically, an R object or a data set) can be printed with either help(name) or ?name.

Usage

help(topic, offline = FALSE, package = NULL,
     lib.loc = NULL, verbose = getOption("verbose"),
     try.all.packages = getOption("help.try.all.packages"),
     chmhelp = getOption("chmhelp"),
     htmlhelp = getOption("htmlhelp"),
     pager = getOption("pager"))

?topic

type?topic

Arguments

- **topic** usually, the name on which documentation is sought. The name may be quoted or unquoted (but note that if topic is the name of a variable containing a character string documentation is provided for the name, not for the character string). The topic argument may also be a function call, to ask for documentation on a corresponding method. See the section on method documentation.

- **offline** a logical indicating whether documentation should be displayed on-line to the screen (the default) or hardcopy of it should be produced.

- **package** a name or character vector giving the packages to look into for documentation, or NULL. By default, all packages in the search path are used.

- **lib.loc** a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.

- **verbose** logical; if TRUE, the file name is reported.

- **try.all.packages** logical; see Note.

- **chmhelp** logical (or NULL). Only relevant under Windows. If TRUE the Compiled HTML version of the help will be shown in a help viewer.

- **htmlhelp** logical (or NULL). If TRUE, the HTML version of the help will be shown in a browser specified by options("browser") or the file-association mechanism.
the pager to be used for `file.show`.

- **type**: the special type of documentation to use for this topic; for example, if the type is `class`, documentation is provided for the class with name `topic`. The function `topicName` returns the actual name used in this case. See the section on method documentation for the uses of `type` to get help on formal methods.

**Details**

In the case of unary and binary operators and control-flow special forms (including `if`, `for` and `function`), the topic may need to be quoted.

If `offline` is `TRUE`, hardcopy of the documentation is produced by running the LaTeX version of the help page through `latex` (note that LaTeX 2e is needed). You need to customize the file `\texttt{R\_HOME/bin/helpPRINT.bat}` which contains an example. The appearance of the output can be customized through a file `\texttt{Rhelp.cfg}` somewhere in your LaTeX search path.

If LaTeX versions of help pages were not built at the installation of the package, the `print` method will ask if conversion with `R CMD Rdconv` (which requires Perl) should be attempted.

**Method Documentation**

The authors of formal (`'S4'`) methods can provide documentation on specific methods, as well as overall documentation on the methods of a particular function. The `"?"` operator allows access to this documentation in three ways.

The expression `methods ? f` will look for the overall documentation methods for the function `f`. Currently, this means the documentation file containing the alias `f-methods`.

There are two different ways to look for documentation on a particular method. The first is to supply the `topic` argument in the form of a function call, omitting the `type` argument. The effect is to look for documentation on the method that would be used if this function call were actually evaluated. See the examples below. If the function is not a generic (no S4 methods are defined for it), the help reverts to documentation on the function name.

The `"?"` operator can also be called with `type` supplied as `"method"`; in this case also, the `topic` argument is a function call, but the arguments are now interpreted as specifying the class of the argument, not the actual expression that will appear in a real call to the function. See the examples below.

The first approach will be tedious if the actual call involves complicated expressions, and may be slow if the arguments take a long time to evaluate. The second approach avoids these difficulties, but you do have to know what the classes of the actual arguments will be when they are evaluated.

Both approaches make use of any inherited methods; the signature of the method to be looked up is found by using `selectMethod` (see the documentation for `getMethod`).

**Note**

Unless `lib.loc` is specified explicitly, the loaded packages are searched before those in the specified libraries. This ensures that if a library is loaded from a library not in the known library trees, then the help from the loaded library is used. If `lib.loc` is specified explicitly, the loaded packages are not searched.

If this search fails and argument `try.all.packages` is `TRUE` and neither `packages` nor `lib.loc` is specified, then all the packages in the known library trees are searched for help on `topic` and a list of (any) packages where help may be found is printed (but no help is shown).

**N.B.** searching all packages can be slow.
The help files can be many small files. On some file systems it is desirable to save space, and the text files in the ‘help’ directory of an installed package can be zipped up as a zip archive ‘Rhelp.zip’. Ensure that file ‘AnIndex’ remains un-zipped. Similarly, all the files in the ‘latex’ directory can be zipped to ‘Rhelp.zip’. This is done on Windows for the larger packages.

References


See Also

`help.search()` for finding help pages on a “vague” topic; `help.start()` which opens the HTML version of the R help pages; `library()` for listing available packages and the user-level objects they contain; `data()` for listing available data sets; `methods()`.

See `prompt()` to get a prototype for writing help pages of private packages.

Examples

```r
help()
help(help) # the same

help(lapply)
?lapply # the same

help("for") # or ?"for", but the quotes are needed
?"*"

help(package="splines") # get help even when package is not loaded

data() # list all available data sets
?women # information about data set "women"

topi <- "women"
## Not run: help(topi) ##---> Error: No documentation for 'topi'

try(help("bs", try.all.packages=FALSE)) # reports not found (an error)
help("bs", try.all.packages=TRUE) # reports can be found in package 'splines'

## Not run:
require(methods)
## define a S4 generic function and some methods
combo <- function(x, y) c(x, y)
setGeneric("combo")
setMethod("combo", c("numeric", "numeric"), function(x, y) x+y)

## assume we have written some documentation for combo, and its methods ....

?combo # produces the function documentation

methods?combo # looks for the overall methods documentation

method?combo("numeric", "numeric") # documentation for the method above

?combo(1:10, rnorm(10)) # ... the same method, selected according to
  # the arguments (one integer, the other numeric)
```
help.search

Search the Help System

Description

Allows for searching the help system for documentation matching a given character string in the
(file) name, alias, title, concept or keyword entries (or any combination thereof), using either fuzzy
matching or regular expression matching. Names and titles of the matched help entries are displayed
nicely.

Usage

help.search(pattern, fields = c("alias", "concept", "title"),
apropos, keyword, whatis, ignore.case = TRUE,
package = NULL, lib.loc = NULL,
help.db =getOption("help.db"),
verbose =getOption("verbose"),
rebuild = FALSE, agrep = NULL)

Arguments

pattern a character string to be matched in the specified fields. If this is given, the
arguments apropos, keyword, and whatis are ignored.
fields a character vector specifying the fields of the help data bases to be searched. The
entries must be abbreviations of "name", "title", "alias", "concept", and "keyword", corresponding to the help page's (file) name, its title, the
topics and concepts it provides documentation for, and the keywords it can be
classified to.
apropos a character string to be matched in the help page topics and title.
keyword a character string to be matched in the help page 'keywords'. 'Keywords' are really categories: the standard categories are listed in file
'RHOME/doc/KEYWORDS' (see also the example) and some package writers
have defined their own. If keyword is specified, agrep defaults to FALSE.
whatis a character string to be matched in the help page topics.
ignore.case a logical. If TRUE, case is ignored during matching; if FALSE, pattern matching
is case sensitive.
package a character vector with the names of packages to search through, or NULL in
which case all available packages in the library trees specified by lib.loc are
searched.
lib.loc a character vector describing the location of R library trees to search through, or
NULL. The default value of NULL corresponds to all libraries currently known.
help.db a character string giving the file path to a previously built and saved help data
base, or NULL.
verbose logical; if TRUE, the search process is traced.
rebuild

a logical indicating whether the help data base should be rebuilt.

agrep

if NULL (the default unless keyword is used) and the character string to be matched consists of alphanumeric characters, whitespace or a dash only, approximate (fuzzy) matching via \texttt{agrep} is used unless the string has fewer than 5 characters; otherwise, it is taken to contain a \textit{regular expression} to be matched via \texttt{grep}. If FALSE, approximate matching is not used. Otherwise, one can give a numeric or a list specifying the maximal distance for the approximate match, see argument \texttt{max.distance} in the documentation for \texttt{agrep}.

\section*{Details}

Upon installation of a package, a contents data base which contains the information on name, title, aliases and keywords and, concepts starting with \texttt{R} 1.8.0, is computed from the Rd files in the package and serialized as \texttt{`Rd.rds'} in the \texttt{`Meta'} subdirectory of the top-level package installation directory (or, prior to \texttt{R} 1.7.0, as \texttt{`CONTENTS'} in Debian Control Format with aliases and keywords collapsed to character strings in the top-level package installation directory). This, or a pre-built help.search index serialized as \texttt{`hsearch.rds'} in the \texttt{`Meta'} directory, is the data base searched by \texttt{help.search()}.

The arguments \texttt{apropos} and \texttt{whatis} play a role similar to the Unix commands with the same names.

If possible, the help data base is saved to the file \texttt{`help.db'} in the \texttt{`.R'} subdirectory of the user’s home directory or the current working directory.

Note that currently, the aliases in the matching help files are not displayed.

\section*{Value}

The results are returned in an object of class \texttt{"hsearch"}, which has a print method for nicely displaying the results of the query. This mechanism is experimental, and may change in future versions of \texttt{R}.

\section*{See Also}

\texttt{help; help.start} for starting the hypertext (currently HTML) version of \texttt{R}’s online documentation, which offers a similar search mechanism.

\texttt{RSiteSearch} to access an on-line search of \texttt{R} resources.

\texttt{apropos} uses regexps and has nice examples.

\section*{Examples}

help.search("linear models") \ # In case you forgot how to fit linear models
help.search("non-existent topic")
## Not run:
help.search("print") \ # All help pages with topics or title matching 'print'
help.search(apropos = "print") \ # The same
help.search(keyword = "hplot") \ # All help pages documenting high-level plots.
file.show(file.path(R.home(), "doc", "KEYWORDS")) \ # show all keywords

## Help pages with documented topics starting with 'try'.
help.search("\btry", fields = "alias")
Start the hypertext (currently HTML) version of R's online documentation.

Usage

```r
help.start(update = TRUE, gui = "irrelevant",
            browser = getOption("browser"))
```

Arguments

- `gui`: just for compatibility with S-PLUS.
- `browser`: the name of the program to be used as hypertext browser. It should be in the PATH, or a full path specified.
- `update`: logical: should this attempt to update the indices to reflect the currently installed packages.

Details

The Windows file association mechanism is used to send the HTML file to a browser, launching one if necessary.

Unlike Unix systems, running `help.start` does not send all future help requests to the browser: use `options(htmlhelp=TRUE)` to set that.

Alternatively, a specific browser can be started if `browser` is specified, but `help` will still use the file association mechanism.

`help.start` and the Java-based search engine have been tested under Internet Explorer 6, Netscape 7.1, Opera 7.54, Mozilla 1.6/1.7 and MozillaFirefox 0.8/0.9/1.0. Users of IE6 under Windows XP SP2 will probably need to allow 'active content' from the 'information bar' before the search engine can be used.

Note

There is a Java-based search facility available from the HTML page that `help.start` brings up. Should this not work, please consult the 'R Installation and Administration' manual which is linked from that page.

See Also

- `help()` for on- and off-line help in ASCII/Editor or PostScript format.
- `browseURL` for how the help file is displayed.
- `RSiteSearch` to access an on-line search of R resources.
### Examples

```r
## Not run:
help.start()
help.start(browser="C:\Program Files\Internet Explorer\IEXPLORE.EXE")
help.start(browser="C:\Program Files\Netscape\Netscape\netscp.exe")
help.start(browser="C:\Program Files\Opera\Opera.exe")
help.start(browser="C:\Program Files\mozilla.org\Mozilla\mozilla.exe")
help.start(browser="C:\Program Files\Mozilla Firefox\firefox.exe")
## End(Not run)
```

---

**index.search**  
**Search Indices for Help Files**

**Description**

Used to search the indices for help files, possibly under aliases.

**Usage**

```r
index.search(topic, path, file="AnIndex", type = "help")
```

**Arguments**

- `topic`  
The keyword to be searched for in the indices.
- `path`  
The path(s) to the packages to be searched.
- `file`  
The index file to be searched. Normally “AnIndex”.
- `type`  
The type of file required.

**Details**

For each package in `path`, examine the file `file` in directory `type`, and look up the matching file stem for topic `topic`, if any.

**Value**

A character vector of matching files, as if they are in directory `type` of the corresponding package. In the special cases of `type = "html", "R-ex"` and `"latex"` the file extensions ".html", ".R" and ".tex" are added.

**See Also**

`help`, `example`
Description

To install packages into the default library tree (which is rooted at `$R_HOME/library`), do `R CMD INSTALL pkgs`.

To install into the library tree `lib` instead of the default one, use `R CMD INSTALL -l lib pkgs`.

Usage

```
R CMD INSTALL [options] [-l lib] pkgs
```

Arguments

- **pkgs**: A space-separated list with the path names of the sources of the packages to be installed.
- **lib**: the path name of the R library tree to install to.
- **options**: a space-separated list of options through which in particular the process for building the help files can be Options should only be given once, and paths including spaces should be quoted. Use `R CMD INSTALL --help` for the current list of options.

Details

Windows Perl and the files for installing from source packages need to be installed.

Both `lib` and the elements of `pkgs` may be absolute or relative path names of directories. `pkgs` may also contain names of package archive files of the form `pkg_version.tar.gz` as obtained from CRAN (these are then extracted in a temporary directory `R.INSTALL` of the current directory).

If the option `--save` is used, the installation procedure creates a binary image of the package code, which is then loaded when the package is attached, rather than evaluating the package source at that time. Having a file `install.R` in the package directory makes this the default behavior for the package (option `--no-save` overrides). You may need `--save` if your package requires other packages to evaluate its own source. If the file `install.R` is non-empty, it should contain R expressions to be executed when the package is attached, after loading the saved image. Options to be passed to R when creating the save image can be specified via `--save=ARGS`.

Options `--lazy`, `--no-lazy`, `--lazy-data` and `--no-lazy-data` control where the R objects and the datasets are made available for lazy loading. (These options are overridden by any values set in the DESCRIPTION file.) The default is `--no-lazy --no-lazy-data` except that lazy-loading is used for package with more than 25kB of R code and no saved image.

Which forms of help are produced is controlled by the `--docs` option which takes values none or normal (text and HTML) or chm (the default, text, HTML, and CHTML).

If the attempt to install the package fails, leftovers are removed. If the package was already installed, the old version is restored.

Use `R CMD INSTALL --help` for more usage information.
Packages using the methods package

Packages that require the methods package, and that use functions such as setMethod or setClass, should be installed by creating a binary image.

The presence of a file named ‘install.R’ in the package’s main directory causes an image to be saved. Note that the file is not in the ‘R’ subdirectory: all the code in that subdirectory is used to construct the binary image.

Normally, the file ‘install.R’ will be empty; if it does contain R expressions these will be evaluated when the package is attached, e.g. by a call to the function library. (Specifically, the source code evaluated for a package with a saved image consists of a suitable definition of .First.lib to ensure loading of the saved image, followed by the R code in file ‘install.R’, if any.)

See Also

REMOVE and library for information on using several library trees; update.packages for automatic update of packages using the internet; the chapter on “Creating R packages” in “Writing R Extensions” (see the ‘doc/manual’ subdirectory of the R source tree).

---

**installed.packages**  
*Find Installed Packages*

**Description**

Find (or retrieve) details of all packages installed in the specified libraries.

**Usage**

```r
installed.packages(lib.loc = NULL, priority = NULL, noCache = FALSE)
```

**Arguments**

- `lib.loc`: character vector describing the location of R library trees to search through.
- `priority`: character vector or NULL (default). If non-null, used to select packages; "high" is equivalent to c("base", "recommended"). To select all packages without an assigned priority use `priority = "NA"`.
- `noCache`: Do not use cached information.

**Details**

`installed.packages` scans the ‘DESCRIPTION’ files of each package found along `lib.loc` and returns a matrix of package names, library paths and version numbers.

**Note:** this works with package names, not bundle names.

The information found is cached (by library) for the R session, and updated only if the top-level library directory has been altered, for example by installing or removing a package. If the cached information becomes confused, it can be refreshed by running `installed.packages(noCache = TRUE)`. 
Value

A matrix with one row per package, row names the package names and column names "Package", "LibPath", "Version", "Priority", "Bundle", "Contains", "Depends", "Suggests", "Imports" and "Built" (the R version the package was built under).

See Also

update.packages, INSTALL, REMOVE.

Examples

```r
str(ip <- installed.packages(priority = "high"))
ip[, c(1,3:5)]
```

Description

Functions to re-create the HTML documentation files to reflect all installed packages. make.packages.html creates the list of packages and make.search.html creates the database used by the Java search engine. link.html.help runs both, as well as fixup.libraries.URLs which attempts to fix up cross-library links.

Usage

```r
link.html.help(VERBOSE = FALSE, lib.loc = .libPaths())
make.packages.html(lib.loc = .libPaths())
make.search.html(lib.loc = .libPaths())

fixup.package.URLs(pkg, force = FALSE)
fixup.libraries.URLs(lib.loc = .libPaths())
```

Arguments

- `verbose` logical. If true, print out a message. For use to explain a delay when called from other functions.
- `lib.loc` character vector. List of libraries to be included.
- `pkg` character vector. The full path to a package.
- `force` logical. Should the links be fixed even if the stamp is present?

Details

Cross-library links do not work on this platform. fixup.package.URLs attempts to correct links in the named package to the doc directory (usually to icons) and to the base, datasets, utils, grDevices, graphics and stats packages, and then stamps a file ‘fixedHTMLlinks’ in the package directory. If that file is found, no correction is attempted unless force = TRUE or the R installation directory (recorded in the stamp) has been changed.

fixup.libraries.URLs calls fixup.package.URLs on all packages in current library trees except the standard one.

The functions will give a warning if the files are not writeable.
make.packages.html and fixup.package.URLs return a logical showing if they succeeded.
The files ‘R_HOME/doc/html/packages.html’ and ‘R_HOME/doc/html/search/index.txt’ are (re-)created by make.packages.html and make.search.html respectively.

Note
You will need write permission in the ‘R_HOME/doc/html’ directory to use any of the first three functions, and in the appropriate library trees to use the last two.

localeToCharset
Select a Suitable Encoding Name from a Locale Name

Description
This function aims to find a suitable coding for the locale named, by default the current locale, and if it is a UTF-8 locale a suitable single-byte encoding.

Usage
localeToCharset(locale = Sys.getlocale("LC_CTYPE"))

Arguments
locale character string naming a locale.

Details
The operation differs by OS. A Windows locale is specified like "English_United Kingdom.1252". The final component gives the codepage, and this defines the encoding.
In the C locale the answer will be "ASCII".

Value
A character vector naming an encoding and possibly a fallback single-encoding, NA if unknown.

Note
The encoding names are those used by libiconv, and ought also to work with glibc but maybe not with commercial Unixen.

See Also
Sys.getlocale, iconv.

Examples
localeToCharset()
**ls.str**  
*List Objects and their Structure*

**Description**

`ls.str` and `lsf.str` are “variations” of `ls` applying `str()` to each matched name, see section ‘Value’.

**Usage**

```r
ls.str(pos = 1, pattern, ..., envir = as.environment(pos), mode = "any")
lsf.str(pos = 1, ..., envir = as.environment(pos))
## S3 method for class 'ls_str':
print(x, max.level = 1, give.attr = FALSE, ...)
```

**Arguments**

- `pos` integer indicating search path position.
- `pattern` a regular expression passed to `ls`. Only names matching `pattern` are considered.
- `max.level` maximal level of nesting which is applied for displaying nested structures, e.g., a list containing sub lists. Default 0: Display all nesting levels.
- `give.attr` logical; if TRUE (default), show attributes as sub structures.
- `envir` environment to use, see `ls`.
- `mode` character specifying the mode of objects to consider. Passed to `exists` and `get`.
- `x` an object of class "ls_str".
- `...` further arguments to pass. and `lsf.str` passes them to `ls.str` which passes them on to `ls`. The (non-exported) print method `print.ls_str` passes them to `str`.

**Value**

`ls.str` and `lsf.str` return an object of class "ls_str", basically the character vector of matching names (functions only for `lsf.str`), similarly to `ls`, with a `print()` method that calls `str()` on each object.

**Author(s)**

Martin Maechler

**See Also**

`str, summary, args`
 Examples

```r
ls.str()#- how do the functions look like which I am using?
ls.str(mode = "list") #- what are the structured objects I have defined?

## create a few objects
declared.by example(glm, echo = FALSE)
ll <- as.list(LETTERS)
print(ls.str(), max.level = 0)# don't show details

## which base functions have "file" in their name?
lsf.str(pos = length(search()), pattern = "file")
```

---

**make.socket**  
Create a Socket Connection

**Description**

With `server = FALSE` attempts to open a client socket to the specified port and host. With `server = TRUE` listens on the specified port for a connection and then returns a server socket. It is a good idea to use `on.exit` to ensure that a socket is closed, as you only get 64 of them.

**Usage**

```r
make.socket(host = "localhost", port, fail = TRUE, server = FALSE)
```

**Arguments**

- `host`  
  name of remote host
- `port`  
  port to connect to/listen on
- `fail`  
  failure to connect is an error?
- `server`  
  a server socket?

**Value**

An object of class "socket".

- `socket`  
  socket number. This is for internal use
- `port`  
  port number of the connection
- `host`  
  name of remote computer

**Warning**

I don’t know if the connecting host name returned when `server = TRUE` can be trusted. I suspect not.

**Author(s)**

Thomas Lumley
memory.size

References
Adapted from Luke Tierney’s code for XLISP-Stat, in turn based on code from Robbins and Robbins “Practical UNIX Programming”

See Also
close.socket, read.socket

Examples
daytime <- function(host = "localhost"){  
a <- make.socket(host, 13)  
on.exit(close.socket(a))  
   read.socket(a)
}
## Official time (UTC) from US Naval Observatory
## Not run: daytime("tick.usno.navy.mil")

memory.size

Description
memory.size reports the current or maximum memory allocation of the malloc function used in this version of R.
memory.limit reports or increases the limit in force on the total allocation.

Usage
memory.size(max = FALSE)
memory.limit(size = NA)

Arguments
  max      logical. If true the maximum amount of memory obtained from the OS is reported, otherwise the amount currently in use.
  size     numeric. If NA report the memory size, otherwise request a new limit, in Mb.

Details
Command-line flag --max-mem-size sets the maximum value of obtainable memory (including a very small amount of housekeeping overhead).
Memory limits can only be increased.

Value
Size in bytes.
1283

menu
Examples
memory.size()
memory.size(TRUE)
round(memory.limit()/1048576.0, 2)

menu

Menu Interaction Function

Description
menu presents the user with a menu of choices labelled from 1 to the number of choices. To exit
without choosing an item one can select ‘0’.
Usage
menu(choices, graphics = FALSE, title = "")
Arguments
choices

a character vector of choices

graphics

a logical indicating whether a graphics menu should be used if available.

title

a character string to be used as the title of the menu. NULL is also accepted.

Details
If graphics = TRUE and a windowing system is available (Windows, MacOS X or X11 via
Tcl/Tk) a listbox widget is used, otherwise a text menu.
Ten or fewer items will be displayed in a single column, more in multiple columns if possible within
the current display width.
No title is displayed if title is NULL or "".
Value
The number corresponding to the selected item, or 0 if no choice was made.
References
Brooks/Cole.
See Also
select.list, which is used to implement the graphical menu, and allows multiple selections.
Examples
## Not run:
switch(menu(c("List letters", "List LETTERS")) + 1,
cat("Nothing done\n"), letters, LETTERS)
## End(Not run)


List Methods for S3 Generic Functions or Classes

Description
List all available methods for an S3 generic function, or all methods for a class.

Usage
methods(generic.function, class)

Arguments
generic.function
  a generic function, or a character string naming a generic function.

class
  a symbol or character string naming a class: only used if
generic.function is not supplied.

Details
Function methods can be used to find out about the methods for a particular generic function or
class. The functions listed are those which are named like methods and may not actually be methods
(known exceptions are discarded in the code). Note that the listed methods may not be user-visible
objects, but often help will be available for them.

If class is used, we check that a matching generic can be found for each user-visible object named.
If generic.function is given, there is a warning if it appears not to be a generic function. (The
check for being generic used can be fooled.)

Value
An object of class "MethodsFunction", a character vector of function names with an "info"
attribute. There is a print method which marks with an asterisk any methods which are not
visible: such functions can be examined by getS3method or getAnywhere.

The "info" attribute is a data frame, currently with a logical column, visible and a factor
column from (indicating where the methods were found).

Note
This scheme is called S3 (S version 3). For new projects, it is recommended to use the more
flexible and robust S4 scheme provided in the methods package. Functions can have both S3 and
S4 methods, and function showMethods will list the S4 methods (possibly none).

The original methods function was written by Martin Maechler.

References
Chambers, J. M. (1992) Classes and methods: object-oriented programming in S. Appendix A of

See Also
S3Methods, class, getS3method
Examples

methods(summary)
methods(class = "aov")
methods("[[")  # uses C-internal dispatching
methods("$")
methods("$<-") # replacement function
methods("+")  # binary operator
methods("Math") # group generic
methods("axis") # looks like it has methods, but not generic

## Not run:
methods(print)  # over 100
## End(Not run)

Description

Functions helping to maintain CRAN, some of them may also be useful for administrators of other repository networks.

Usage

mirror2html(mirrors = NULL, file = "mirrors.html",
            head = "mirrors-head.html", foot = "mirrors-foot.html")
checkCRAN(method)

Arguments

mirrors | A data frame, by default the CRAN list of mirrors is used.
file    | A connection object or a character string.
head    | Name of optional header file.
foot    | Name of optional footer file.
method  | Download method, see download.file.

Details

mirror2html creates the HTML file for the CRAN list of mirrors and invisibly returns the HTML text.
checkCRAN performs a sanity checks on all CRAN mirrors.
normalizePath  Express File Paths in Canonical Form

Description
Convert file paths to canonical form for the platform, to display them in a user-understandable form.

Usage
normalizePath(path)

Arguments
path character vector of file paths.

Details
This converts relative paths to absolute paths, and converts short names to long names. It will always use backslashes as the path separator.
If the path is not a real path the result is undefined but will most likely be the corresponding input element.

Value
A character vector.

Examples
```r
cat(normalizePath(c(R.home(), tempdir())), sep = "\n")
```

object.size  Report the Space Allocated for an Object

Description
Provides an estimate of the memory that is being used to store an R object.

Usage
object.size(x)

Arguments
x An R object.
Details

Exactly which parts of the memory allocation should be attributed to which object is not clear-cut. This function merely provides a rough indication: it should be reasonably accurate for atomic vectors, but does not detect if elements of a list are shared, for example. (Sharing amongst elements of a character vector is taken into account.)

The calculation is of the size of the object, and excludes the space needed to store its name in the symbol table.

Associated space (e.g. the environment of a function and what the pointer in a EXTPTRSXP points to) is not included in the calculation.

Object sizes are larger on 64-bit platforms than 32-bit ones, but will very likely be the same on different platforms with the same word length and pointer size.

Value

An estimate of the memory allocation attributable to the object, in bytes.

See Also

Memory-limits for the design limitations on object size.

Examples

object.size(letters)
object.size(ls)
## find the 10 largest objects in the base package
z <- sapply(ls("package:base"), function(x)
  object.size(get(x, envir = baseenv()))
  as.matrix(rev(sort(z))[1:10])

package.skeleton

Create a Skeleton for a New Source Package

Description

package.skeleton automates some of the setup for a new source package. It creates directories, saves functions and data to appropriate places, and creates skeleton help files and ‘README’ files describing further steps in packaging.

Usage

package.skeleton(name = "anRpackage", list, environment = .GlobalEnv, path = ".", force = FALSE)

Arguments

name character string: the directory name for your package.
list character vector naming the R objects to put in the package.
environment if list is omitted, the contents of this environment are packaged.
path path to put the package directory in.
force If FALSE will not overwrite an existing directory.
Details

The package sources are placed in subdirectory name of path.
This tries to create filenames valid for all OSes known to run R. Invalid characters are replaced by _, invalid names are preceded by zz, and finally the converted names are made unique by make.unique(sep = "_"). This can be done for code and help files but not data files (which are looked for by name).

Value

used for its side-effects.

References

Read the Writing R Extensions manual for more details.
Once you have created a source package you need to install it: see the R Installation and Administration manual, INSTALL and install.packages.

See Also

prompt

Examples

## two functions and two "data sets" :
f <- function(x,y) x+y
g <- function(x,y) x-y
d <- data.frame(a=1, b=2)
e <- rnorm(1000)

package.skeleton(list=c("f","g","d","e"), name="mypkg")

packageDescription Package Description

Description

Parses and returns the ‘DESCRIPTION’ file of a package.

Usage

packageDescription(pkg, lib.loc = NULL, fields = NULL, drop = TRUE, encoding = "")

Arguments

pkg
  a character string with the package name.
lib.loc
  a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.
packageStatus

fields  
a character vector giving the tags of fields to return (if other fields occur in the file they are ignored).

drop  
If TRUE and the length of fields is 1, then a single character string with the value of the respective field is returned instead of an object of class "packageDescription".

encoding  
If there is an Encoding field, to what encoding should re-encoding be attempted? If NA, no re-encoding. The other values are as used by iconv, so the default "" indicates the encoding of the current locale.

Details

A package will not be ‘found’ unless it has a ‘DESCRIPTION’ file which contains a valid Version field. Different warnings are given when no package directory is found and when there is a suitable directory but no valid ‘DESCRIPTION’ file.

Value

If a ‘DESCRIPTION’ file for the given package is found and can successfully be read, packageDescription returns an object of class "packageDescription", which is a named list with the values of the (given) fields as elements and the tags as names, unless drop = TRUE.

If parsing the ‘DESCRIPTION’ file was not successful, it returns a named list of NAs with the field tags as names if fields is not null, and NA otherwise.

See Also

read.dcf

Examples

packageDescription("stats")
packageDescription("stats", fields = c("Package", "Version"))

packageDescription("stats", fields = "Version")
packageDescription("stats", fields = "Version", drop = FALSE)

packageStatus  
Package Management Tools

Description

Summarize information about installed packages and packages available at various repositories, and automatically upgrade outdated packages.

Usage

packageStatus(lib.loc = NULL, repositories = NULL, method, type = getOption("pkgType"))

## S3 method for class 'packageStatus':
summary(object, ...)
## S3 method for class 'packageStatus':
update(object, lib.loc = levels(object$inst$LibPath),
       repositories = levels(object$avail$Repository), ...)

## S3 method for class 'packageStatus':
upgrade(object, ask = TRUE, ...)

### Arguments

- **lib.loc**
  - a character vector describing the location of R library trees to search through, or
    NULL. The default value of NULL corresponds to all libraries currently known.

- **repositories**
  - a character vector of URLs describing the location of R package repositories on
    the Internet or on the local machine.

- **method**
  - Download method, see `download.file`.

- **type**
  - type of package distribution: see `install.packages`.

- **object**
  - an object of class "packageStatus" as returned by `packageStatus`.

- **ask**
  - if TRUE, the user is prompted which packages should be upgraded and which
    not.

... currently not used.

### Details

The URLs in `repositories` should be full paths to the appropriate contrib sections of the repositories. The default is `contrib.url(getOption("repos"))`. (Prior to R 2.1.0 this was hardcoded as the CRAN and Bioconductor repositories.)

There are `print` and `summary` methods for the "packageStatus" objects: the `print` method gives a brief tabular summary and the `summary` method prints the results.

The `update` method updates the "packageStatus" object. The `upgrade` method is similar to `update.packages`: it offers to install the current versions of those packages which are not currently up-to-date.

### Value

An object of class "packageStatus". This is a list with two components

- **inst**
  - a data frame with columns as the `matrix` returned by `installed.packages` plus "Status", a factor with levels c("ok", "upgrade"). Only the newest version of each package is reported, in the first repository in which it appears.

- **avail**
  - a data frame with columns as the `matrix` returned by `available.packages` plus "Status", a factor with levels c("installed", "not installed", "unavailable")..

### See Also

`installed.packages`, `available.packages`
Invoke a Pager on an R Object

Description
Displays a representation of the object named by \( x \) in a pager.

Usage
```r
page(x, method = c("dput", "print"), ...)
```

Arguments
- **x**: the name of an R object.
- **method**: The default method is to dump the object via \texttt{dput}. An alternative is to print to a file.
- **...**: additional arguments for \texttt{file.show}. Intended for setting \texttt{pager} as \texttt{title} and \texttt{delete.file} are already used.

See Also
- \texttt{file.show}, \texttt{edit}, \texttt{fix}.

To go to a new page when graphing, see \texttt{frame}.

Person Names and Contact Information

Description
A class and utility methods for holding information about persons like name and email address.
Usage

```r
person(first = "", last = "", middle = "", email = ")
personList(...) as.person(x) as.personList(x)
```

```r
## S3 method for class 'person':
as.character(x, ...)
## S3 method for class 'personList':
as.character(x, ...)
```

```r
## S3 method for class 'person':
toBibtex(object, ...)
## S3 method for class 'personList':
toBibtex(object, ...)
```

Arguments

- `first`: character string, first name
- `middle`: character string, middle name(s)
- `last`: character string, last name
- `email`: character string, email address
- `...`: for `personList` an arbitrary number of `person` objects
- `x`: a character string or an object of class `person` or `personList`
- `object`: an object of class `person` or `personList`

Examples

```r
## create a person object directly
p1 <- person("Karl", "Pearson", email = "pearson@stats.heaven")
p1
```

```r
## convert a string
p2 <- as.person("Ronald Aylmer Fisher")
p2
```

```r
## create one object holding both
p <- personList(p1, p2)
ps <- as.character(p)
ps
```

```r
## convert to BibTeX author field
toBibtex(p)
```
prompt

Description

Utilities for checking whether the sources of an \( R \) add-on package work correctly, and for building a source or binary package from them.

Usage

\[
\begin{align*}
\text{R CMD build} & \ [\text{options}] \ \text{pkgdirs} \\
\text{R CMD check} & \ [\text{options}] \ \text{pkgdirs}
\end{align*}
\]

Arguments

- **pkgdirs**: a list of names of directories with sources of \( R \) add-on packages.
- **options**: further options to control the processing, or for obtaining information about usage and version of the utility.

Details

- **R CMD check**: checks \( R \) add-on packages from their sources, performing a wide variety of diagnostic checks.
- **R CMD build**: builds \( R \) source or binary packages from their sources. It will create index files in the sources if necessary, so it is often helpful to run `build` before `check`.

Use `R CMD foo --help` to obtain usage information on utility `foo`.

Several of the options to `build --binary` are passed to `INSTALL` so consult its help for the details.

Note

These may not work correctly under Windows 95/98/ME because of problems Perl has launching programs on those limited OSes.

They make use of a temporary directory specified by the environment variable `TMPDIR` and defaulting to `c:/TEMP`. Do ensure that if set forward slashes are used.

See Also

The sections on “Checking and building packages” and “Processing Rd format” in “Writing \( R \) Extensions” (see the Manuals sub-menu of the Help menu on the console).

`INSTALL` is called by `build --binary`.

prompt

Produce Prototype of an \( R \) Documentation File

Description

Facilitate the constructing of files documenting \( R \) objects.
Usage

prompt(object, filename = NULL, name = NULL, ...)

## Default S3 method:
prompt(object, filename = NULL, name = NULL,
       force.function = FALSE, ...)

## S3 method for class 'data.frame':
prompt(object, filename = NULL, name = NULL, ...)

Arguments

object

an R object, typically a function for the default method.

filename

usually, a connection or a character string giving the name of the file to which
the documentation shell should be written. The default corresponds to a file
whose name is name followed by ".Rd". Can also be NA (see below).

name

a character string specifying the name of the object.

force.function

a logical. If TRUE, treat object as function in any case.

...

further arguments passed to or from other methods.

Details

Unless filename is NA, a documentation shell for object is written to the file specified by
filename, and a message about this is given. For function objects, this shell contains the proper
function and argument names. R documentation files thus created still need to be edited and moved
into the ‘man’ subdirectory of the package containing the object to be documented.

If filename is NA, a list-style representation of the documentation shell is created and re-
turned. Writing the shell to a file amounts to `cat(unlist(x), file = filename, sep
= "\n"), where x is the list-style representation.

When prompt is used in for loops or scripts, the explicit name specification will be useful.

Value

If filename is NA, a list-style representation of the documentation shell. Otherwise, the name of
the file written to is returned invisibly.

Warning

The default filename may not be a valid filename under limited file systems (e.g. those on Windows).
Currently, calling prompt on a non-function object assumes that the object is in fact a data set and
hence documents it as such. This may change in future versions of R. Use promptData to create
documentation skeletons for data sets.

Note

The documentation file produced by prompt.data.frame does not have the same format as
many of the data frame documentation files in the base package. We are trying to settle on a
preferred format for the documentation.
**promptData**

Generate a Shell for Documentation of Data Sets

**Description**

Generates a shell of documentation for a data set.

**Usage**

promptData(object, filename = NULL, name = NULL)

**Arguments**

- **object**: an R object to be documented as a data set.
- **filename**: usually, a connection or a character string giving the name of the file to which the documentation shell should be written. The default corresponds to a file whose name is `name` followed by ".Rd". Can also be NA (see below).
- **name**: a character string specifying the name of the object.

**Examples**

```
require(graphics)
prompt(plot.default)
prompt(interactive, force.function = TRUE)
unlink("plot.default.Rd")
unlink("interactive.Rd")
prompt(women)  # data.frame
unlink("women.Rd")
prompt(sunspots)  # non-data.frame data
unlink("sunspots.Rd")
```
promptPackage

Generate a Shell for Documentation of a Package

Description

Generates a shell of documentation for an installed or source package.

Usage

promptPackage(package, lib.loc = NULL, filename = NULL, name = NULL, final = FALSE)

Arguments

package the name of an installed or source package to be documented.
lib.loc a character vector describing the location of R library trees to search through, or NULL. The default value of NULL corresponds to all libraries currently known. For a source package this should specify the parent directory of the package’s sources.
filename usually a connection or a character string giving the name of the file to which
the documentation shell should be written. The default corresponds to a file
whose name is `name` followed by `".Rd"`. Can also be `NA` (see below).

name a character string specifying the name of the help topic, typically of the form
`<pkg>-package`.

final a logical value indicating whether to attempt to create a usable version of the
help topic, rather than just a shell.

Details

Unless `filename` is `NA`, a documentation shell for `package` is written to the file specified by
`filename`, and a message about this is given.

If `filename` is `NA`, a list-style representation of the documentation shell is created and re-
turned. Writing the shell to a file amounts to `cat(unlist(x), file = filename, sep
= "\n"), where `x` is the list-style representation.

If `final` is `TRUE`, the generated documentation will not include the place-holder slots for manual
editing, it will be usable as-is. In most cases a manually edited file is preferable (but `final =
TRUE` is certainly less work).

Value

If `filename` is `NA`, a list-style representation of the documentation shell. Otherwise, the name of
the file written to is returned invisibly.

See Also

`prompt`

Examples

```r
filename <- tempfile()
promptPackage("utils", file=filename)
file.show(filename)
unlink(filename)
```

Description

Read fixed-format data files using Fortran-style format specifications.

Usage

```r
read.fortran(file, format, ..., as.is = TRUE, colClasses = NA)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File or connection to read from</td>
</tr>
<tr>
<td>format</td>
<td>Character vector or list of vectors. See Details below.</td>
</tr>
<tr>
<td>...</td>
<td>Other arguments for read.table</td>
</tr>
<tr>
<td>as.is</td>
<td>Keep characters as characters?</td>
</tr>
<tr>
<td>colClasses</td>
<td>Variable classes to override defaults. See read.table for details.</td>
</tr>
</tbody>
</table>

Details

The format for a field is of one of the following forms: \( rF_l.d, rD_l.d, rX_l, rA_l, rI_l \), where \( l \) is the number of columns, \( d \) is the number of decimal places, and \( r \) is the number of repeats. \( F \) and \( D \) are numeric formats, \( A \) is character, \( I \) is integer, and \( X \) indicates columns to be skipped. The repeat code \( r \) and decimal place code \( d \) are always optional. The length code \( l \) is required except for \( X \) formats when \( r \) is present.

For a single-line record, `format` should be a character vector. For a multiline record it should be a list with a character vector for each line.

Skipped (X) columns are not passed to `read.table`, so `colClasses`, `col.names`, and similar arguments passed to `read.table` should not reference these columns.

Value

A data frame

See Also

`read.fwf`, `read.csv`

Examples

```r
ff <- tempfile()
cat(file=ff, "123456", "987654", sep="\n")
read.fortran(ff, c("F2.1","F2.0","I2"))
read.fortran(ff, c("2F1.0","2X","2A1"))
unlink(ff)
cat(file=ff, "123456AB", "987654CD", sep="\n")
read.fortran(ff, list(c("2F3.1","A2"), c("3I2","2X")))
unlink(ff)
```

Description

Read a “table” of fixed width formatted data into a `data.frame`.

Usage

```r
read.fwf(file, widths, header = FALSE, sep = "\t", as.is = FALSE,
         skip = 0, row.names, col.names, n = -1, buffersize = 2000,
         ...)```
Arguments

- **file**: the name of the file which the data are to be read from. Alternatively, `file` can be a `connection`, which will be opened if necessary, and if so closed at the end of the function call.
- **widths**: integer vector, giving the widths of the fixed-width fields (of one line), or list of integer vectors giving widths for multiline records
- **header**: a logical value indicating whether the file contains the names of the variables as its first line.
- **sep**: character; the separator used internally; should be a character that does not occur in the file.
- **as.is**: see `read.table`.
- **skip**: number of initial lines to skip; see `read.table`.
- **row.names**: see `read.table`.
- **col.names**: see `read.table`.
- **n**: the maximum number of records (lines) to be read, defaulting to no limit.
- **buffersize**: Maximum number of lines to read at one time
- **...**: further arguments to be passed to `read.table`. Useful further arguments include `na.strings` and `colClasses`.

Details

Multiline records are concatenated to a single line before processing. Fields that are of zero-width or are wholly beyond the end of the line in `file` are replaced by `NA`.

Negative-width fields are used to indicate columns to be skipped, eg `-5` to skip 5 columns. These fields are not seen by `read.table` and so should not be included in a `col.names` or `colClasses` argument.

Reducing the `buffersize` argument may reduce memory use when reading large files with long lines. Increasing `buffersize` may result in faster processing when enough memory is available.

Value

A `data.frame` as produced by `read.table` which is called internally.

Author(s)

Brian Ripley for R version: original Perl by Kurt Hornik.

See Also

`scan` and `read.table`.

Examples

```r
ff <- tempfile()
cat(file=ff, "123456", "987654", sep="\n")
read.fwf(ff, width=c(1,2,3))  #> 1 23 456 \ 9 87 654
read.fwf(ff, width=c(1,-2,3))  #> 1 456 \ 9 654
unlink(ff)
cat(file=ff, "123", "987654", sep="\n")
read.fwf(ff, width=c(1,0, 2,3))  #> 1 NA 23 NA \ 9 NA 87 654
```
unlink(ff)
cat(file=ff, "123456", "987654", sep="\n")
read.fwf(ff, width=list(c(1,0, 2,3), c(2,2,2))) #> 1 NA 23 456 98 76 54
unlink(ff)

---

**read.socket**  
*Read from or Write to a Socket*

**Description**

read.socket reads a string from the specified socket, write.socket writes to the specified socket. There is very little error checking done by either.

**Usage**

```r
read.socket(socket, maxlen = 256, loop = FALSE)
write.socket(socket, string)
```

**Arguments**

- `socket`: a socket object
- `maxlen`: maximum length of string to read
- `loop`: wait for ever if there is nothing to read?
- `string`: string to write to socket

**Value**

read.socket returns the string read.

**Author(s)**

Thomas Lumley

**See Also**

`close.socket`, `make.socket`

**Examples**

```r
finger <- function(user, host = "localhost", port = 79, print = TRUE) {
  if (!is.character(user))
    stop("user name must be a string")
  user <- paste(user, "\r\n")
  socket <- make.socket(host, port)
  on.exit(close.socket(socket))
  write.socket(socket, user)
  output <- character(0)
  repeat{
    ss <- read.socket(socket)
    if (ss == "") break
    output <- paste(output, ss)
  }
}
```
The `recover` function allows the user to browse directly on any of the currently active function calls, and is suitable as an error option. The expression `options(error=recover)` will make this the error option.

**Usage**

```r
recover()
```

**Details**

When called, `recover` prints the list of current calls, and prompts the user to select one of them. The standard R `browser` is then invoked from the corresponding environment; the user can type ordinary S language expressions to be evaluated in that environment.

When finished browsing in this call, type `c` to return to `recover` from the browser. Type another frame number to browse some more, or type `0` to exit `recover`.

The use of `recover` largely supersedes `dump.frames` as an error option, unless you really want to wait to look at the error. If `recover` is called in non-interactive mode, `recover` has the advantage that it does not need to copy out all the environments in order to browse in them. If you do decide to quit interactive debugging, call `dump.frames` directly while browsing in any frame (see the examples).

**WARNING:** The special `Q` command to go directly from the browser to the prompt level of the evaluator currently interacts with `recover` to effectively turn off the error option for the next error (on subsequent errors, `recover` will be called normally).

**Value**

Nothing useful is returned. However, you can invoke `recover` directly from a function, rather than through the error option shown in the examples. In this case, execution continues after you type `0` to exit `recover`.

**Compatibility Note**

The R `recover` function can be used in the same way as the S-Plus function of the same name; therefore, the error option shown is a compatible way to specify the error action. However, the actual functions are essentially unrelated and interact quite differently with the user. The navigating commands `up` and `down` do not exist in the R version; instead, exit the browser and select another frame.
References

John M. Chambers (1998). Programming with Data; Springer. See the compatibility note above, however.

See Also

browser for details about the interactive computations; options for setting the error option; dump.frames to save the current environments for later debugging.

Examples

```r
## Not run:
options(error = recover) # setting the error option

### Example of interaction

> myFit <- lm(y ~ x, data = xy, weights = w)
Error in lm.wfit(x, y, w, offset = offset, ...) :
  missing or negative weights not allowed

Enter a frame number, or 0 to exit
1:lm(y ~ x, data = xy, weights = w)
2:lm.wfit(x, y, w, offset = offset, ...) 
Selection: 2
Called from: eval(expr, envir, enclos)
Browse[1]> objects() # all the objects in this frame
[1] "method" "n" "ny" "offset" "tol" "w"
[7] "x" "y"
Browse[1]> w
[1] -0.5013844 1.3112515 0.2939348 -0.8983705 -0.1538642
[6] -0.9772989 0.7888790 -0.1919154 -0.3026882
Browse[1]> dump.frames() # save for offline debugging
Browse[1]> c # exit the browser

Enter a frame number, or 0 to exit
1:lm(y ~ x, data = xy, weights = w)
2:lm.wfit(x, y, w, offset = offset, ...) 
Selection: 0 # exit recover
>
## End(Not run)
```

---

**REMOVE**

Remove Add-on Packages

Description

Use R CMD REMOVE pkgs to remove the packages in pkgs from the default library tree (which is rooted at `$R_HOME/library`).

To remove from the library tree lib instead of the default one, use R CMD REMOVE -l lib pkgs.
remove.packages

Usage

R CMD REMOVE [-l lib] pkgs

Arguments

pkgs     a list with the names of the packages to be removed.
lib      the path name of the R library tree to remove from. May be absolute or relative.

Details

Windows Perl and the files for installing from source packages need to be installed.

See Also

INSTALL

remove.packages          Remove Installed Packages

Description

Removes installed packages/bundles and updates index information as necessary.

Usage

remove.packages(pkgs, lib, version)

Arguments

pkgs     a character vector with the names of the package(s) or bundle(s) to be removed.
lib      a character vector giving the library directories to remove the packages from. If
         missing, defaults to the first element in .libPaths().
version  A character vector specifying version(s) with versioned installs of the package(s)
         to remove. If none is provided, the system will remove an unversioned install of
         the package if one is found, otherwise the latest versioned install.

Details

If an element of pkgs matches a bundle name, all the packages in the bundle will be removed. This
takes precedence over matching a package name.

pkgs and version will be recycled if necessary to the length of the longer one.

See Also

install.packages for installing packages.
Rprof  

Enable Profiling of R's Execution

Description

Enable or disable profiling of the execution of R expressions.

Usage

Rprof(filename = "Rprof.out", append = FALSE, interval = 0.02)

Arguments

filename  The file to be used for recording the profiling results. Set to NULL or "" to disable profiling.
append  logical: should the file be over-written or appended to?
interval  real: time interval between samples.

Details

Enabling profiling automatically disables any existing profiling to another or the same file. Profiling works by writing out the call stack every interval seconds, to the file specified. Either the summaryRprof function or the Perl script R CMD Rprof can be used to process the output file to produce a summary of the usage; use R CMD Rprof --help for usage information.

Exactly what the time interval measures is subtle: it is time that the R process is running and executing an R command. It is not however just CPU time, for if readline() is waiting for input, that counts (on Windows, but not on Unix).

Note that the timing interval cannot be too small, for the time spent in each profiling step is added to the interval. What is feasible is machine-dependent, but 10ms seems as small as advisable on a 1GHz machine.

Note

Using R CMD Rprof needs Windows Perl to be installed.

See Also

The chapter on “Tidying and profiling R code” in “Writing R Extensions” (see the ‘doc/manual’ subdirectory of the R source tree).

summaryRprof

Examples

## Not run:
Rprof()
## some code to be profiled
Rprof(NULL)
## some code NOT to be profiled
Rprof(append=TRUE)
## some code to be profiled
RSiteSearch

Search for key words or phrases in the R-help mailing list archives or documentation.

Description

Search for key words or phrases in the R-help mailing list archives, or R manuals and help pages, using the search engine at http://search.r-project.org and view them in a web browser.

Usage

RSiteSearch(string, restrict = c("Rhelp02a", "functions", "docs"), format = "normal", sortby = "score", matchesPerPage = 20)

Arguments

string     word(s) or phrase to search. If the words are to be searched as one entity, enclose all words in braces (see example).
restrict   character: What areas to search in: Rhelp02a for R-help mailing list archive since 2002, Rhelp01 for mailing list archive before 2002, docs for R manuals, functions for help pages. Use c() to specify more than one.
format     normal or short (no excerpts).
sortby     How to sort the search results (score, date:late for sorting by date with latest results first, date:early for earliest first, subject for subject in alphabetical order, subject:descending for reverse alphabetical order, from or from:descending for sender (when applicable), size or size:descending for size.)
matchesPerPage     How many items to show per page.

Details

This function is designed to work with the search site at http://search.r-project.org, and depends on that site continuing to be made available (thanks to Jonathan Baron and the School of Arts and Sciences of the University of Pennsylvania).

Unique partial matches will work for all arguments. Each new browser window will stay open unless you close it.

Value

(Invisibly) the complete URL passed to the browser, including the query string.
Rtangle

R Driver for Stangle

Description

A driver for Stangle that extracts R code chunks.

Usage

Rtangle()
RtangleSetup(file, syntax, output = NULL, annotate = TRUE,
split = FALSE, prefix = TRUE, quiet = FALSE)

Arguments

- **file**
  - Name of Sweave source file.
- **syntax**
  - An object of class SweaveSyntax.
- **output**
  - Name of output file, default is to remove extension '.nw', '.Rnw' or '.Snw' and to add extension '.R'. Any directory names in file are also removed such that the output is created in the current working directory.
- **annotate**
  - By default, code chunks are separated by comment lines specifying the names and numbers of the code chunks. If FALSE, only the code chunks without any decorating comments are extracted.
- **split**
  - Split output in single files per code chunk?
- **prefix**
  - If split = TRUE, prefix the chunk labels by the basename of the input file to get output file names?
- **quiet**
  - If TRUE all progress messages are suppressed.

Author(s)

Friedrich Leisch
RweaveLatex

References
http://www.ci.tuwien.ac.at/~leisch/Sweave

See Also
Sweave, RweaveLatex

RweaveLatex

R/LaTeX Driver for Sweave

Description
A driver for Sweave that translates R code chunks in LaTeX files.

Usage
RweaveLatex()

RweaveLatexSetup(file, syntax, output = NULL, quiet = FALSE,
debug = FALSE, echo = TRUE, eval = TRUE,
split = FALSE, stylepath = TRUE,
pdf = TRUE, eps = TRUE)

Arguments

file        Name of Sweave source file.
syntax      An object of class SweaveSyntax.
output      Name of output file, default is to remove extension '.nw', '.Rnw' or '.Snw' and to add extension '.tex'. Any directory names in file are also removed such that the output is created in the current working directory.
quiet       If TRUE all progress messages are suppressed.
debug       If TRUE, input and output of all code chunks is copied to the console.
distpath    If TRUE, a hard path to the file 'Sweave.sty' installed with this package is set, if FALSE, only \usepackage{Sweave} is written. The hard path makes the TeX file less portable, but avoids the problem of installing the current version of 'Sweave.sty' to some place in your TeX input path. The argument is ignored if a \usepackage{Sweave} is already present in the Sweave source file.
echo        set default for option echo, see details below.
eval        set default for option eval, see details below.
split       set default for option split, see details below.
pdf          set default for option pdf, see details below.
eps         set default for option eps, see details below.
Supported Options

RweaveLatex supports the following options for code chunks (the values in parentheses show the default values):

**echo**: logical (TRUE). Include S code in the output file?

**eval**: logical (TRUE). If FALSE, the code chunk is not evaluated, and hence no text or graphical output produced.

**results**: character string (verbatim). If verbatim, the output of S commands is included in the verbatim-like Soutput environment. If tex, the output is taken to be already proper latex markup and included as is. If hide then all output is completely suppressed (but the code executed during the weave).

**print**: logical (FALSE) If TRUE, each expression in the code chunk is wrapped into a print() statement before evaluation, such that the values of all expressions become visible.

**term**: logical (TRUE). If TRUE, visibility of values emulates an interactive R session: values of assignments are not printed, values of single objects are printed. If FALSE, output comes only from explicit print or cat statements.

**split**: logical (FALSE). If TRUE, text output is written to separate files for each code chunk.

**strip.white**: character string (false). If true, blank lines at the beginning and end of output are removed. If all, then all blank lines are removed from the output.

**prefix**: logical (TRUE). If TRUE generated filenames of figures and output have a common prefix.

**prefix.string**: a character string, default is the name of the '.Snw' source file.

**include**: logical (TRUE), indicating whether input statements for text output and includegraphics statements for figures should be auto-generated. Use include = FALSE if the output should appear in a different place than the code chunk (by placing the input line manually).

**fig**: logical (FALSE), indicating whether the code chunk produces graphical output. Note that only one figure per code chunk can be processed this way.

**eps**: logical (TRUE), indicating whether EPS figures shall be generated. Ignored if fig = FALSE.

**pdf**: logical (TRUE), indicating whether PDF figures shall be generated. Ignored if fig = FALSE.

**width**: numeric (6), width of figures in inch.

**height**: numeric (6), height of figures in inch.

Author(s)

Friedrich Leisch

References

http://www.ci.tuwien.ac.at/~leisch/Sweave

See Also

Sweave, Rtangle
Description

The file ‘Rconsole’ configures the R GUI console in this Windows port. The file ‘Rdevga’ configures the graphics devices windows, win.graph, win.metafile and win.print, as well as the bitmap devices bmp, jpeg and png (which use windows internally).

Details

There are system copies of these files in ‘R_HOME\etc’. Users can have personal copies of the files: these are looked for in the location given by the environment variable R_USER. The system files are read only if a corresponding personal file is not found.

If the environment variable R_USER is not set, the R system sets it to HOME if that is set (stripping any trailing slash), otherwise to the Windows “persona” directory, otherwise to \{HOMEDRIVE}\{HOMEPATH\} if HOMEDRIVE and HOMEDRIVE are both set otherwise to the working directory. This is as described in the file ‘rw-FAQ’.

Value

Each of the files contains details in its comments of how to set the values.

At the time of writing ‘Rdevga’ configured the mapping of font numbers to fonts, and ‘Rconsole’ configured the appearance (single or multiple document interface, toolbar, statusbar on MDI), size, font and colours of the GUI console, and whether resizing the console sets options("width").

The file ‘Rconsole’ also configures the internal pager. This shares the font and colours of the console, but can be sized separately.

‘Rconsole’ can also set the initial positions of the console and the graphics device, as well as the size and position of the MDI workspace in MDI mode.

Chinese/Japanese/Korean

Users of these languages will need to select a suitable font for the console (perhaps FixedSys or MS Mincho) and for the graphics device (Arial can be replaced by Arial Unicode MS).

Which fonts are available depends on the version of Windows.

Note

The GUI preferences item on the Edit menu brings up an form which can be used to edit the console settings, and to save them to a file.

Author(s)

Guido Masarotto

See Also

windows
Examples

```r
ruser <- Sys.getenv("R_USER")
cat("\n\nLocation for personal configuration files is\n   R_USER = ",
   ruser, "\n", sep="")
## see if there are personal configuration files
sapply(c("Rconsole", "Rdevga"), function(x) file.exists(file.path(ruser, x)))

## show the configuration files used
showConfig <- function(file)
{
   ruser <- Sys.getenv("R_USER")
   path <- file.path(ruser, file)
   if(!file.exists(path)) path <- file.path(R.home(), "etc", file)
   file.show(path, header=path)
}
showConfig("Rconsole")
```

Description

Load or save or display the commands history.

Usage

```r
loadhistory(file = ".Rhistory")
savehistory(file = ".Rhistory")
history(max.show = 25, reverse = FALSE)
```

Arguments

- **file**: The name of the file in which to save the history, or from which to load it. The path is relative to the current working directory.
- **max.show**: The maximum number of lines to show. Inf will give all of the currently available history.
- **reverse**: logical. If true, the lines are shown in reverse order. Note: this is not useful when there are continuation lines.

Details

There are several history mechanisms available for the different R consoles, which work in similar but not identical ways. Other uses of R, in particular embedded uses, may have no history. This works in Rgui and interactive Rterm but not in batch use of Rterm nor in embedded/DCOM versions.

The history mechanism is controlled by two environment variables: `R_HISTSIZE` controls the number of lines that are saved (default 512), and `R_HISTFILE` sets the filename used for the loading/saving of history if requested at the beginning/end of a session (but not the default for these functions). There is no limit on the number of lines of history retained during a session, so setting `R_HISTSIZE` to a large value has no penalty unless a large file is actually generated.

These variables are read at the time of saving, so can be altered within a session by the use of `Sys.putenv`. 
select.list

Note

If you want to save the history (almost) every session, you can put a call to `savehistory()` in `.Last`.

Examples

```r
## Not run:
.Last <- function()
  if(interactive()) try(savehistory("~/.Rhistory"))
## End(Not run)
```

select.list 

Select Items from a List

Description

Select item(s) from a character vector.

Usage

```r
select.list(list, preselect = NULL, multiple = FALSE, title = NULL)
```

Arguments

- `list`: character. A list of items.
- `preselect`: a character vector, or `NULL`. If non-null and if the string(s) appear in the list, the item(s) are selected initially.
- `multiple`: logical: can more than one item be selected?
- `title`: optional character string for window title.

Details

This brings up a modal dialog box with a (scrollable) list of items, which can be selected by the mouse. If `multiple` is true, further items can be selected or deselected by holding the control key down whilst selecting, and shift-clicking can be used to select ranges.

Normal termination is via the `OK` button or by hitting Enter. Selection can be aborted via the `Cancel` button or pressing Escape.

Value

A character vector of selected items. If `multiple` is false and no item was selected (or `Cancel` was used), "" is returned. If `multiple` is true and no item was selected (or `Cancel` was used) then a character vector of length 0 is returned.

See Also

`menu`, `tk_select.list` for a graphical version using Tcl/Tk.

Examples

```r
## Not run:
select.list(sort(.packages(all.available = TRUE)))
## End(Not run)
```
**sessionInfo**

*Collect Information About the Current R Session*

**Description**

Print version information about R and attached packages.

**Usage**

```r
sessionInfo(package=NULL)
## S3 method for class 'sessionInfo':
print(x, ...)
## S3 method for class 'sessionInfo':
toLatex(object, ...)
```

**Arguments**

- `package`  a character vector naming installed packages. By default all attached packages are used.
- `x` an object of class "sessionInfo".
- `object` an object of class "sessionInfo".
- `...` currently not used.

**Examples**

```r
sessionInfo()
toLatex(sessionInfo())
```

---

**setRepositories**

*Select Package Repositories*

**Description**

Interact with the user to choose the package repositories to be used.

**Usage**

```r
setRepositories(graphics = TRUE)
```

**Arguments**

- `graphics` Logical. If true use a listbox, otherwise use a text list in the console.

**Details**

The default list of known repositories is stored in the file ‘R_HOME/etc/repositories’. That file can be edited for a site, or a user can have a personal copy in ‘HOME/.R/repositories’ which will take precedence.

The items that are preselected are those that are currently in `options("repos")` plus those marked as default in the list of known repositories.
**setWindowTitle**

Value

This function is invoked mainly for its side effect of updating `options("repos")`. It returns (invisibly) the previous `repos` options setting (as a list with component `repos`) or `NULL` if no changes were applied.

See Also

`chooseCRANmirror`, `install.packages`.

---

**setWindowTitle**  
*Set or get the Window Title*

Description

Set the title of the R window which will appear in the task bar.

Usage

```r
setWindowTitle(suffix, title = paste(getIdentification(), suffix))
gWindowTitle()
ggetIdentification()
```

Arguments

- `suffix` a character string to form part of the title
- `title` a character string forming the complete new title

Details

`setWindowTitle` appends `suffix` to the normal window identification (`RGui`, `R Console` or `Rterm`). Use suffix = "" to reset the title.

`gWindowTitle` gets the current title.

This sets the title of the frame in MDI mode, the title of the console for `RGui` --sdi, and the title of the window from which it was launched for `Rterm`. It has no effect in embedded uses of R.

`ggetIdentification` returns the normal window identification.

Value

All three functions return a length 1 character vector.

`setWindowTitle` returns the previous window title (invisibly).

`gWindowTitle` and `ggetIdentification` return the current window title and the normal window identification, respectively.
Examples

```r
## show the current working directory in the title, saving the old one
oldtitle <- setWindowTitle(getwd())
Sys.sleep(0.5)
## reset the title
setWindowTitle("")
Sys.sleep(0.5)
## restore the original title
setWindowTitle(title = oldtitle)
```

### SHLIB

#### Build a DLL for Dynamic Loading

**Description**

The given source files are first compiled. All specified object files are then linked into a shared library which can be loaded into R using `dyn.load` or `library.dynam`.

**Usage**

```r
R CMD SHLIB [options] [-o dllname] files
```

**Arguments**

- `files`: a list of names of (typically) source files to be compiled and included in the library. You can also include the names of object files which are automagically made from their sources.
- `dllname`: the full name of the shared library to be built, including the extension `.dll`. If not given, the name of the DLL is taken from the first source file.
- `options`: Further options to control the processing. Use `R CMD SHLIB --help` for a current list. The most useful one is `-d` to build a debug DLL.

**Details**

`R CMD SHLIB` uses the mechanism as used by `INSTALL` to compile source code in packages. To use `SHLIB` you need to have installed (from the R installer) the files for compiling source packages as well as the tools described in the ‘R Installation and Administration’ manual.

Please consult section ‘Creating shared objects’ in the manual ‘Writing R Extensions’ for how to customize it (for example to add `cpp` flags and to add libraries to the link step) and for details of some of its quirks.

**See Also**

`dyn.load`, `library.dynam`.

The ‘R Installation and Administration’ and ‘Writing R Extensions’ manuals.
Compactly Display the Structure of an Arbitrary R Object

Description

Compactly display the internal structure of an R object, a “diagnostic” function and an alternative to `summary` (and to some extent, `dput`). Ideally, only one line for each “basic” structure is displayed. It is especially well suited to compactly display the (abbreviated) contents of (possibly nested) lists. The idea is to give reasonable output for any R object. It calls `args` for (non-primitive) function objects.

Usage

```r
str(object, ...)
```

## S3 method for class 'data.frame':
```r
str(object, ...)
```

## Default S3 method:
```r
str(object, max.level = NA, vec.len = 4, digits.d = 3, nchar.max = 128, give.attr = TRUE, give.length = TRUE, wid = getOption("width"), nest.lev = 0, indent.str = paste(rep.int(" ", max(0, nest.lev + 1)), collapse = ":"), comp.str="$ ", no.list = FALSE, envir = baseenv()
```

Arguments

- `object`: any R object about which you want to have some information.
- `max.level`: maximal level of nesting which is applied for displaying nested structures, e.g., a list containing sub lists. Default NA: Display all nesting levels.
- `vec.len`: numeric (>= 0) indicating how many “first few” elements are displayed of each vector. The number is multiplied by different factors (from .5 to 3) depending on the kind of vector. Default 4.
- `digits.d`: number of digits for numerical components (as for `print`).
- `nchar.max`: maximal number of characters to show for character strings. Longer strings are truncated, see `longch` example below.
- `give.attr`: logical; if TRUE (default), show attributes as sub structures.
- `give.length`: logical; if TRUE (default), indicate length (as `[1:...]`).
- `wid`: the page width to be used. The default is the currently active `options("width")`.
- `nest.lev`: current nesting level in the recursive calls to `str`.
- `indent.str`: the indentation string to use.
- `comp.str`: string to be used for separating list components.
- `no.list`: logical; if true, no “list of ..” is nor the class is printed.
- `envir`: the environment to be used for `promise` (see `delayedAssign`) objects only.
- `...`: potential further arguments (required for Method/Generic reasons).
Value

`str` does not return anything, for efficiency reasons. The obvious side effect is output to the terminal.

Author(s)

Martin Maechler (maechler@stat.math.ethz.ch) since 1990.

See Also

`ls.str` for listing objects with their structure; `summary`, `args`.

Examples

```r
require(stats)
## The following examples show some of 'str' capabilities
str(1:12)
str(ls)
str(args) # more useful than  args(args)  !
str(freney)
str(str)
str(.Machine, digits = 20)
str( lsfit(1:9,1:9))
str( lsfit(1:9,1:9),  max = 1)
op <- options(); str(op) # save first; otherwise internal options() is used.
need.dev <- !exists(".Device") || is.null(.Device)
if(need.dev) postscript()
str(par()); if(need.dev) graphics.off()

ch <- letters[1:12]; is.na(ch) <- 3:5
str(ch) # character NA's
nchar(longch <- paste(rep(letters,100), collapse=""))
str(longch)
str(longch, nchar.max = 52)

str(quote( { A+B; list(C,D) } ))

## S4 classes :
require(stats4)
x <- 0:10; y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
ll <- function(ymax=15, xh=6) -sum(dpois(y, lambda=ymax/(1+x/xh), log=TRUE))
fit <- mle(ll)
str(fit)
```

Sweave

Automatic Generation of Reports
Description

Sweave provides a flexible framework for mixing text and S code for automatic report generation. The basic idea is to replace the S code with its output, such that the final document only contains the text and the output of the statistical analysis.

Usage

Sweave(file, driver = RweaveLatex(),
        syntax = getOption("SweaveSyntax"), ...)

Stangle(file, driver = Rtangle(),
        syntax = getOption("SweaveSyntax"), ...)

Arguments

file Name of Sweave source file.

driver The actual workhorse, see details below.

syntax An object of class SweaveSyntax or a character string with its name. The default installation provides SweaveSyntaxNoweb and SweaveSyntaxLatex.

... Further arguments passed to the driver’s setup function.

Details

Automatic generation of reports by mixing word processing markup (like latex) and S code. The S code gets replaced by its output (text or graphs) in the final markup file. This allows to re-generate a report if the input data change and documents the code to reproduce the analysis in the same file that also produces the report.

Sweave combines the documentation and code chunks together (or their output) into a single document. Stangle extracts only the code from the Sweave file creating a valid S source file (that can be run using source). Code inside \Sexpr{} statements is ignored by Stangle.

Stangle is just a frontend to Sweave using a simple driver by default, which discards the documentation and concatenates all code chunks the current S engine understands.

Hook Functions

Before each code chunk is evaluated, a number of hook functions can be executed. If getOption("SweaveHooks") is set, it is taken to be a collection of hook functions. For each logical option of a code chunk (echo, print,...) a hook can be specified, which is executed if and only if the respective option is TRUE. Hooks must be named elements of the list returned by getOption("SweaveHooks") and be functions taking no arguments. E.g., if option "SweaveHooks" is defined as list(fig = foo), and foo is a function, then it would be executed before the code in each figure chunk. This is especially useful to set defaults for the graphical parameters in a series of figure chunks.

Note that the user is free to define new Sweave options and associate arbitrary hooks with them. E.g., one could define a hook function for option clean that removes all objects in the global environment. Then all code chunks with clean=TRUE would start operating on an empty workspace.
SweaveSyntConv

Syntax Definition

Sweave allows a very flexible syntax framework for marking documentation and text chunks. The default is a noweb-style syntax, as alternative a latex-style syntax can be used. See the user manual for details.

Author(s)

Friedrich Leisch

References

http://www.ci.tuwien.ac.at/~leisch/Sweave


See Also

RweaveLatex, Rtangle

Examples

testfile <- system.file("Sweave", "Sweave-test-1.Rnw", package = "utils")

## create a LaTeX file
Sweave(testfile)

## create an S source file from the code chunks
Stangle(testfile)

## which can be simply sourced
source("Sweave-test-1.R")

SweaveSyntConv

Convert Sweave Syntax

Description

This function converts the syntax of files in Sweave format to another Sweave syntax definition.

Usage

SweaveSyntConv(file, syntax, output = NULL)
toLatex

Description

These methods convert R objects to character vectors with BibTeX or LaTeX markup.

Usage

toBibtex(object, ...)
toLatex(object, ...)
## S3 method for class 'Bibtex':
print(x, prefix="", ...)
## S3 method for class 'Latex':
print(x, prefix="", ...)
Arguments

object object of a class for which \texttt{toBibtex} or \texttt{toLatex} method exists.

x object of class \texttt{"Bibtex"} or \texttt{"Latex"}.

prefix a character string which is printed at the beginning of each line, mostly used to insert whitespace for indentation.

... currently not used in the print methods.

Details

Objects of class \texttt{"Bibtex"} or \texttt{"Latex"} are simply character vectors where each element holds one line of the corresponding BibTeX or LaTeX file.

See Also

\texttt{citEntry} and \texttt{sessionInfo} for examples

---

\textbf{Description}

These functions can be used to automatically compare the version numbers of installed packages with the newest available version on the repositories and update outdated packages on the fly.

Usage

\begin{verbatim}
update.packages(lib.loc = NULL, repos =getOption("repos"),
               contriburl = contrib.url(repos, type),
               method, instlib = NULL,
               ask = TRUE, available = NULL, destdir = NULL,
               installWithVers = FALSE, checkBuilt = FALSE,
               type = getOption("pkgType"))

available.packages(contriburl = contrib.url(getOption("repos")), method)

old.packages(lib.loc = NULL, repos =getOption("repos"),
             contriburl = contrib.url(repos),
             method, available = NULL, checkBuilt = FALSE)

new.packages(lib.loc = NULL, repos =getOption("repos"),
              contriburl = contrib.url(repos),
              method, available = NULL, ask = FALSE)

download.packages(pkgs, destdir, available = NULL, 
                  repos =getOption("repos"),
                  contriburl = contrib.url(repos, type),
                  method, type = getOption("pkgType"))

install.packages(pkgs, lib, repos =getOption("repos"),
                 contriburl = contrib.url(repos, type),
                 method, type = getOption("pkgType"))
\end{verbatim}
**update.packages**

```r
method, available = NULL, destdir = NULL,
installWithVers = FALSE, dependencies = FALSE,
type = getOption("pkgType"))
```

```r
contrib.url(repos, type = getOption("pkgType"))
```

**Arguments**

- **lib.loc**
  character vector describing the location of R library trees to search through (and update packages therein).

- **repos**
  character vector, the base URL(s) of the repositories to use, i.e., the URL of the CRAN master such as "http://cran.r-project.org" or its Statlib mirror, "http://lib.stat.cmu.edu/R/CRAN". Can be NULL to install from local `.zip` files.

- **contriburl**
  URL(s) of the contrib section of the repositories. Use this argument only if your CRAN mirror is incomplete, e.g., because you burned only the `contrib` section on a CD. Overrides argument repos. Can also be NULL to install from local `.zip` files.

- **method**
  Download method, see `download.file`.

- **pkgs**
  character vector of the short names of packages/bundles whose current versions should be downloaded from the repositories. If `repos = NULL`, a character vector of file paths of `.zip` files. If this is a zero-length character vector, a listbox of available packages (including those contained in bundles) is presented where possible.

- **destdir**
  directory where downloaded packages are stored.

- **available**
  an object listing packages available at the repositories as returned by `available.packages`.

- **lib**
  character vector giving the library directories where to install the packages. Recycled as needed.

- **ask**
  logical indicating whether to ask user before packages are actually downloaded and installed, or the character string "graphics", which brings up a widget to allow the user to (de-)select from the list of packages which could be updated. The latter only works on systems with a GUI version of `select.list`, and is otherwise equivalent to `ask = TRUE`.

- **installWithVers**
  If TRUE, will invoke the install of the package such that it can be referenced by package version.

- **checkBuilt**
  If TRUE, a package built under an earlier minor version of R is considered to be 'old'.

- **instlib**
  character string giving the library directory where to install the packages.

- **dependencies**
  logical indicating to also install uninstalled packages on which these packages depend/suggest/import (and so on recursively). Not used if `repos = NULL`. Can also be a character vector, a subset of c("Depends", "Imports", "Suggests").

- **type**
  character, indicating the type of package to download and install. Possible values are "win.binary" (the default) and "source" (for which suitable tools need to be installed: see the 'rw-FAQ'). Value "mac.binary" can be used to explore and download Mac OS X binaries.
Details

All of these functions work with the names of a package or bundle (and not the component packages of a bundle, except for `install.packages` if the repository provides the necessary information).

`available.packages` returns a matrix of details corresponding to packages/bundles currently available at one or more repositories. The current list of packages is downloaded over the internet (or copied from a local mirror). It returns only packages whose version requirements are met by the running version of \( R \).

`old.packages` compares the information from `available.packages` with that from `installed.packages` and reports installed packages/bundles that have newer versions on the repositories or, if `checkBuilt = TRUE`, that were built under an earlier minor version of \( R \) (for example built under 2.0.x when running \( R \) 2.1.1). (There is no check that the version on the repository was built under the current minor version of \( R \), but it is advertised as being suitable for this version.)

`new.packages` does the same comparison but reports uninstalled packages/bundles that are available at the repositories. It will also give warnings about incompletely installed bundles (provided the information is available) and bundles whose contents has changed. If `ask != FALSE` it asks which packages should be installed in the first element of `lib.loc`.

`download.packages` takes a list of package/bundle names and a destination directory, downloads the newest versions and saves them in `destdir`. If the list of available packages is not given as argument, it is obtained from repositories. If a repository is local, i.e., the URL starts with "file:", then the packages are not downloaded but used directly. (Both "file:" and "file:///" are allowed as prefixes to a file path, the latter for an absolute file path.)

The main function of the set is `update.packages`. First a list of all packages/bundles found in `lib.loc` is created and compared with those available at the repositories. If `ask = TRUE` (the default) packages/bundles with a newer version are reported and for each one the user can specify if it should be updated. If so, the pre-compiled packages are downloaded from the repositories and installed in the respective library path (or `instlib` if specified).

`install.packages` can be used to install new packages/bundles. It takes a vector of names and a destination library, downloads the packages from the repositories and installs them. (If the library is omitted it defaults to the first directory in \( .libPaths() \), with a warning if there is more than one.) Argument `pkgs` can also be a character vector of file names of zip files if `repos = NULL`. The zip files are then unpacked directly.

`contrib.url` adds the appropriate type-specific path within a repository to each URL in `repos`. For `install.packages` and `update.packages`, `destdir` is the directory to which packages will be downloaded. If it is `NULL` (the default) a directory `downloaded_packages` of the session temporary directory will be used (and the files will be deleted at the end of the session).

If `repos` or `contriburl` is a vector of length greater than one, the newest version of the package is fetched from the first repository on the list within which it is found.

Value

For `available.packages`, a matrix with one row per package/bundle, row names the package names and column names "Package", "Version", "Priority", "Bundle", "Depends", "Imports", "Suggests" "Contains" and "Repository".

For `old.packages`, `NULL` or a matrix with one row per package/bundle, row names the package names and column names "Package", "LibPath", "Installed" (the version), "Built" (the version built under), "ReposVer" and "Repository".

For `new.packages` a character vector of package/bundle names, after any have been installed.
For `download.packages`, a two-column matrix of names and destination file names, for those packages/bundles successfully downloaded. If packages are not available or there is a problem with the download, suitable warnings are given.

`install.packages` and `update.packages` have no return value.

**Warning**

Not enough information is recorded to know if a bundle is completely installed, so a bundle is regarded as installed if any of its component packages is.

**Note**

Some binary distributions of R have `INSTALL` in a separate bundle, e.g. an R-devel RPM. `install.packages` will give an error if called on such a system.

**See Also**

`installed.packages`.

See `download.file` for how to handle proxies and other options to monitor file transfers.

`INSTALL`, `REMOVE`, `library`, `.packages`, `read.dcf`

## url.show

**Display a text URL**

**Description**

Extension of `file.show` to display text files from a remote server.

**Usage**

```r
url.show(url, title = url, file = tempfile(),
          delete.file = TRUE, method, ...)
```

**Arguments**

- `url` The URL to read from.
- `title` Title for the browser.
- `file` File to copy to.
- `delete.file` Delete the file afterwards?
- `method` File transfer method: see `download.file`
- `...` Arguments to pass to `file.show`.

**See Also**

`url`, `file.show`, `download.file`

**Examples**

```r
## Not run: url.show("http://lib.stat.cmu.edu/datasets/csb/ch3a.txt")
```
URLencode

Encode or Decode a (partial) URL

Description

Functions to encode or decode characters in URLs.

Usage

URLencode(URL, reserved = FALSE)
URLdecode(URL)

Arguments

- **URL**: A character string.
- **reserved**: should reserved characters be encoded? See Details.

Details

Characters in a URL other than the English alphanumeric characters and $ - _ . + ! * ', ( ) , should be encoded as % plus a two-digit hexadecimal representation, and any single-byte character can be so encoded. (Multi-byte characters are encoded as byte-by-byte.)

In addition, ; / ? : @ = & are reserved characters, and should be encoded unless used in their reserved sense, which is scheme specific. The default in URLencode is to leave them alone, which is appropriate for file:// URLs, but probably not for http:// ones.

Value

A character string.

References


Examples

```r
(y <- URLencode("a url with spaces and / and @"))
URLdecode(y)
(y <- URLencode("a url with spaces and / and @", reserved=TRUE))
URLdecode(y)
URLdecode("ab%20cd")
```
utils-deprecated  Deprecated Functions in Package utils

Description

These functions are provided for compatibility with older versions of R only, and may be defunct as soon as of the next release.

Usage

CRAN.packages(CRAN = getOption("repos"), method,
               contriburl = contrib.url(CRAN))

Arguments

CRAN character, an earlier way to specify a repository.
method Download method, see download.file.
contriburl URL(s) of the contrib section of the repositories. Use this argument only if your CRAN mirror is incomplete, e.g., because you burned only the ‘contrib’ section on a CD. Overrides argument repos.

See Also

Deprecated, Defunct

utils-package  The R Utils Package

Description

R utility functions

Details

This package contains a collection of utility functions.

For a complete list, use library(help="utils").

Author(s)

R Development Core Team and contributors worldwide
Maintainer: R Core Team (R-core@r-project.org)
vignette  View or List Vignettes

Description

View a specified vignette, or list the available ones.

Usage

vignette(topic, package = NULL, lib.loc = NULL)

## S3 method for class 'vignette':
print(x, ...)
## S3 method for class 'vignette':
edit(name, ...)

Arguments

- **topic**: a character string giving the (base) name of the vignette to view. If omitted, all available vignettes are listed.
- **package**: a character vector with the names of packages to search through, or NULL in which case all available packages in the library trees specified by lib.loc are searched.
- **lib.loc**: a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known.
- **x, name**: Object of class vignette.
- **...**: Ignored by the print method, passed on to file.edit by the edit method.

Details

Function vignette returns an object of the same class, the print method opens a viewer for it. Currently, only PDF versions of vignettes can be viewed. If several vignettes have PDF versions with base name identical to topic, the first one found is used. The edit method extracts the R code from the vignette to a temporary file and opens the file in an editor.

If no topics are given, the available vignettes are listed. The corresponding information is returned in an object of class "packageIQR". The structure of this class is experimental.

Examples

```r
## List vignettes in all attached packages
vignette()

## Not run:
## Open the grid intro vignette
vignette("grid")

## The same
v1 <- vignette("grid")
print(v1)

## Now let us have a closer look at the code
```
## A package can have more than one vignette (package grid has several):
```
vignette(package="grid")
vignette("rotated")
```
## The same, but without searching for it:
```
vignette("rotated", package="grid")
```
## End (Not run)

---

### winDialog

#### Description

Put up a Windows dialog box to communicate with the user. There are various types, either for the user to select from a set of buttons or to edit a string.

#### Usage

```
winDialog(type = c("ok", "okcancel", "yesno", "yesnocancel"), message)
winDialogString(message, default)
```

#### Arguments

- **type**
  - character. The type of dialog box. It will have the buttons implied by its name.

- **message**
  - character. The information field of the dialog box. Limited to 255 chars (by Windows, checked by R).

- **default**
  - character. The default string.

#### Value

For `winDialog` a character string giving the name of the button pressed (in capitals) or `NULL` (invisibly) if the user had no choice.

For `winDialogString` a string giving the contents of the text box when `Ok` was pressed, or `NULL` if `Cancel` was pressed.

#### Note

The standard keyboard accelerators work with these dialog boxes: where appropriate `Return` accepts the default action, `Esc` cancels and the underlined initial letter (Y or N) can be used.

#### See Also

- `winMenuAdd`
- `file.choose` to select a file
- `package windlgs` in the package source distribution for ways to program dialogs in C in the GraphApp toolkit.

#### Examples

```
## Not run: winDialog("yesno", "Is it OK to delete file blah")
```
winextras  

*Auxiliary Functions for the Windows Port*

**Description**

Auxiliary functions for the Windows port

**Usage**

```r
win.version()
zip.unpack(zipname, dest)
```

**Arguments**

- `zipname` character string giving name of zip file.
- `dest` character string giving directory within which to unpack.

**Details**

`win.version` is an auxiliary function for `bug.report` which returns a character string describing the version of Windows in use.

`zip.unpack` unpacks the zip file `zipname` in directory `dest`; it is an internal version of `unzip zipfile -d dest` (but will use an external `unzip` if one is set in `options("unzip")`). The result is an error code, with an attribute "extracted" listing the full paths to the extracted files.

---

winMenus  

*User Menus under Windows*

**Description**

Enables users to add, delete and program menus under Windows.

**Usage**

```r
winMenuAdd(menuName)
winMenuAddItem(menuName, itemName, action)
winMenuDel(menuName)
winMenuDelItem(menuName, itemName)
winMenuNames()
winMenuItems(menuName)
```

**Arguments**

- `menuName` a character string naming a menu.
- `itemName` a character string naming a menu item on an existing menu.
- `action` a character string describing the action when that menu is selected, or "enable" or "disable".
Details

User menus are added to the right of existing menus, and items are added at the bottom of the menu.

By default the action character string is treated as R input, being echoed on the command line and parsed and executed as usual.

If the `menuname` parameter of `winMenuAddItem` does not already exist, it will be created automatically.

Normally new submenus and menu items are added to the main console menu. They may be added elsewhere using the following special names:

- `$ConsoleMain` The console menu (the default)
- `$ConsolePopup` The console popup menu
- `$Graph<n>Main` The menu for graphics window `<n>`
- `$Graph<n>Popup` The popup menu for graphics window `<n>`

Specifying an existing item in `winMenuAddItem` enables the action to be changed.

Submenus can be specified by separating the elements in `menuname` by slashes: as a consequence menu names may not contain slashes.

If the `action` is specified as "none" no action is taken: this can be useful to reserve items for future expansion.

The function `winMenuNames` can be used to find out what menus have been created by the user and returns a vector of the existing menu names.

The `winMenuItems` function will take the name of a menu and return the items that exist in that menu. The return value is a named vector where the names correspond to the names of the items and the values of the vector are the corresponding actions.

The `winMenuDel` function will delete a menu and all of its items and submenus. `winMenuDelItem` just deletes one menu item.

Value

NULL, invisibly. If an error occurs, an informative error message will be given.

See Also

- `winDialog`

Examples

```r
## Not run:
winMenuAdd("Testit")
winMenuAddItem("Testit", "one", "aaaa")
winMenuAddItem("Testit", "two", "bbbb")
winMenuAdd("Testit/extras")
winMenuAddItem("Testit", "-", "")
winMenuAddItem("Testit", "two", "disable")
winMenuAddItem("Testit", "three", "cccc")
winMenuAddItem("Testit/extras", "one more", "ddd")
winMenuAddItem("Testit/extras", "and another", "eee")
winMenuAdd("$ConsolePopup/Testit")
winMenuAddItem("$ConsolePopup/Testit", "six", "fff")
winMenuNames()
winMenuItems("Testit")
## End(Not run)
```
Index

=! (Logic), 208
!= (Comparison), 57
*Topic **NA**
  complete.cases, 868
  factor, 126
  NA, 233
  na.action, 1016
  na.fail, 1017
  naprint, 1018
  naresid, 1018
*Topic **algebra**
  backsolve, 30
  chol, 49
  chol2inv, 51
  colSums, 54
  crossprod, 73
  eigen, 108
  matrix, 222
  qr, 273
  QR.Auxiliaries, 276
  solve, 341
  svd, 373
*Topic **aplot**
  abline, 559
  arrows, 560
  axis, 562
  box, 570
  bxp, 574
  contour, 578
  coplot, 581
  filled.contour, 586
  frame, 590
  grid, 591
  Hershey, 522
  image, 597
  Japanese, 526
  legend, 601
  lines, 605
  matplot, 607
  mtext, 612
  persp, 624
  plot.window, 638
  plot.xy, 639
  plotmath, 535
  points, 640
  polygon, 642
  rect, 644
  rect.hclust, 1105
  rug, 645
  screen, 646
  segments, 648
  text, 660
  title, 661
*Topic **arith**
  all.equal, 9
  approxfun, 826
  Arithmetic, 16
  cumsum, 73
  diff, 94
  Extremes, 124
  findInterval, 134
  gl, 163
  matmult, 221
  ppoints, 1068
  prod, 271
  range, 287
  Round, 312
  sign, 337
  sort, 343
  sum, 371
  tabulate, 391
*Topic **array**
  addmargins, 810
  aggregate, 812
  aperm, 12
  apply, 13
  array, 17
  backsolve, 30
  cbind, 43
  cbind2, 742
  chol, 49
  chol2inv, 51
  col, 53
  colSums, 54
  contrast, 871
  cor, 876
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>crossprod</td>
<td>73</td>
</tr>
<tr>
<td>data.matrix</td>
<td>79</td>
</tr>
<tr>
<td>det</td>
<td>90</td>
</tr>
<tr>
<td>diag</td>
<td>93</td>
</tr>
<tr>
<td>dim</td>
<td>96</td>
</tr>
<tr>
<td>dimnames</td>
<td>97</td>
</tr>
<tr>
<td>drop</td>
<td>102</td>
</tr>
<tr>
<td>eigen</td>
<td>108</td>
</tr>
<tr>
<td>expand.grid</td>
<td>116</td>
</tr>
<tr>
<td>Extract</td>
<td>118</td>
</tr>
<tr>
<td>Extract.data.frame</td>
<td>121</td>
</tr>
<tr>
<td>kronecker</td>
<td>187</td>
</tr>
<tr>
<td>lm.fit</td>
<td>976</td>
</tr>
<tr>
<td>lower.tri</td>
<td>210</td>
</tr>
<tr>
<td>margin.table</td>
<td>215</td>
</tr>
<tr>
<td>mat.or.vec</td>
<td>216</td>
</tr>
<tr>
<td>matmult</td>
<td>221</td>
</tr>
<tr>
<td>matrix</td>
<td>222</td>
</tr>
<tr>
<td>maxCol</td>
<td>223</td>
</tr>
<tr>
<td>merge</td>
<td>228</td>
</tr>
<tr>
<td>nrow</td>
<td>240</td>
</tr>
<tr>
<td>outer</td>
<td>256</td>
</tr>
<tr>
<td>prop.table</td>
<td>272</td>
</tr>
<tr>
<td>qr</td>
<td>273</td>
</tr>
<tr>
<td>QR.Auxiliaries</td>
<td>276</td>
</tr>
<tr>
<td>row</td>
<td>314</td>
</tr>
<tr>
<td>row/collnames</td>
<td>316</td>
</tr>
<tr>
<td>scale</td>
<td>321</td>
</tr>
<tr>
<td>slice.index</td>
<td>339</td>
</tr>
<tr>
<td>svd</td>
<td>373</td>
</tr>
<tr>
<td>sweep</td>
<td>374</td>
</tr>
<tr>
<td>t</td>
<td>388</td>
</tr>
</tbody>
</table>

**Topic attribute**

- attr                  | 27 |
- attributes             | 28 |
- call                   | 40 |
- comment                | 56 |
- length                 | 191|
- mode                   | 231|
- name                   | 234|
- names                  | 235|
- NULL                   | 245|
- numeric                | 245|
- structure              | 365|
- typeof                 | 410|
- which                  | 423|

**Topic character**

- abbreviate             | 5  |
- agrep                  | 7  |
- char.expand            | 45 |
- character              | 46 |
- charmatch              | 47 |
- chartr                 | 48 |
- delimMatch             | 1215|
- format                 | 140|
- format.info            | 144|
- formatC                | 146|
- gettext                | 160|
- glob2rx                | 1267|
- grep                   | 163|
- iconv                  | 171|
- make.names             | 212|
- make.unique            | 213|
- nchar                  | 237|
- paste                  | 259|
- pmatch                 | 261|
- regex                  | 303|
- sprintf                | 350|
- sQuote                 | 353|
- strsplit               | 363|
- strtrim                | 364|
- strwidth               | 656|
- strwrap                | 365|
- substr                 | 369|
- symnum                 | 1169|
- utf8Conversion         | 418|

**Topic chron**

- as.POSIX*              | 21 |
- axis.POSIXct           | 565|
- cut.POSIXct            | 76 |
- Dates                  | 81 |
- DateTimeClasses        | 81 |
- difftime               | 95 |
- format.Date            | 142|
INDEX

hist.POSIXt, 595
rep, 308
round.POSIXt, 313
seq.Date, 329
seq.POSIXt, 330
strftime, 359
Sys.time, 384
weekdays, 422

*Topic classes
as, 735
as.data.frame, 18
BasicClasses, 739
callNextMethod, 740
character, 46
class, 52
Classes, 743
classRepresentation-class, 744
data.class, 77
data.frame, 78
Documentation, 746
double, 99
environment-class, 747
fixPre1.8, 748
genericFunction-class, 749
GenericFunctions, 750
getClass, 753
getMethod, 754
integer, 175
is, 760
is.object, 182
is.recursive, 183
is.single, 184
isSealedMethod, 763
language-class, 764
LinearMethodsList-class, 765
logical, 209
makeClassRepresentation, 766
MethodDefinition-class, 767
Methods, 768
MethodsList-class, 771
MethodWithNext-class, 772
new, 773
numeric, 245
ObjectsWithPackage-class, 774
promptClass, 775
raw, 289
rawConversion, 290
real, 301
representation, 778
row.names, 315
SClassExtension-class, 779

setClass, 780
setClassUnion, 784
setMethod, 789
signature-class, 797
slot, 798
StructureClasses, 800
TraceClasses, 800
validObject, 801
vector, 418

*Topic cluster
as.hclust, 842
cophenetic, 875
cutree, 883
dist, 898
hclust, 941
identify.hclust, 950
kmeans, 965
rect.hclust, 1105

*Topic color
col2rgb, 504
colorRamp, 505
colors, 507
convertColor, 508
gray, 518
gray.colors, 519
hcl, 520
hsv, 525
make.rgb, 526
palette, 529
Palettes, 530
rgb, 547
rgb2hsv, 548

*Topic complex
complex, 58

*Topic connection
cat, 42
connections, 63
dput, 100
dump, 102
gzcon, 169
parse, 258
pushBack, 272
read.00Index, 1226
read.fortran, 1297
read.fwf, 1298
read.table, 292
readBin, 295
readChar, 298
readLines, 300
scan, 322
seek, 326
showConnections, 335
sink, 338
socketSelect, 341
source, 345
textConnection, 399
write, 426
writeLines, 429

*Topic datagen
simulate, 1123

*Topic datasets
ability.cov, 433
airmiles, 434
AirPassengers, 434
airquality, 435
anscombe, 436
attenu, 437
attitude, 438
austres, 439
beavers, 439
BJsales, 441
BOD, 441
cars, 442
ChickWeight, 443
chickwts, 444
CO2, 445
c02, 446
data, 1247
discoveries, 447
DNase, 447
esoph, 448
euro, 449
eurodist, 450
faithful, 451
Formaldehyde, 452
freeny, 453
HairEyeColor, 454
Harman23.cor, 455
Harman74.cor, 455
Indometh, 456
infert, 457
InsectSprays, 458
iris, 458
islands, 460
JohnsonJohnson, 460
LakeHuron, 461
lh, 461
LifeCycleSavings, 462
Loblolly, 463
longley, 463
lynx, 464
morley, 465
mtcars, 466
nhtemp, 466
Nile, 467
nottem, 468
Orange, 469
OrchardSprays, 470
PlantGrowth, 471
precip, 471
presidents, 472
pressure, 473
Puromycin, 473
quakes, 475
randu, 475
rivers, 476
sleep, 477
stackloss, 477
state, 478
sunspot.month, 480
sunspot.year, 481
sunspots, 481
swiss, 482
Theoph, 483
Titanic, 484
ToothGrowth, 485
treering, 486
trees, 487
UCBAAdmissions, 487
UKDriverDeaths, 488
UKgas, 490
UKLungDeaths, 490
USAccDeaths, 491
USArrests, 491
USJudgeRatings, 492
USPersonalExpenditure, 493
uspop, 493
VADeaths, 494
volcano, 495
warpbreaks, 495
women, 496
WorldPhones, 497
WWWusage, 498

*Topic data
apropos, 1233
as.environment, 19
assign, 23
assignOps, 24
attach, 26
autoload, 29
bquote, 36
delayedAssign, 86
deparse, 87
detach, 91
environment, 111
eval, 113
exists, 115
force, 135
get, 152
getAnywhere, 1263
getFromNamespace, 1264
getS3method, 1265
libPaths, 194
library, 195
library.dynam, 198
ns-load, 243
search, 325
substitute, 368
sys.parent, 379
with, 425
zpackages, 431

*Topic debugging
recover, 1301
trace, 402

*Topic design
contrast, 871
contrasts, 873
TukeyHSD, 1188

*Topic device
.Device, 1
bringToTop, 501
dev.interactive, 510
dev.xxx, 511
dev2, 512
Devices, 515
pdf, 531
pictex, 533
png, 538
postscript, 539
postscriptFonts, 544
recordGraphics, 545
screen, 646
windows, 551
windowsFonts, 554
xfig, 555

*Topic distribution
bandwidth, 845
Beta, 847
Binomial, 850
birthday, 854
Cauchy, 859
chisq.test, 860
Chisquare, 862
density, 890
Exponential, 908
FDist, 915
fivenum, 922
GammaDist, 931
Geometric, 933
hist, 592
Hypergeometric, 949
IQR, 958
Logistic, 984
Lognormal, 988
Multinomial, 1014
NegBinomial, 1019
Normal, 1034
Poisson, 1063
ppoints, 1068
qqnorm, 1099
r2dtable, 281
Random, 282
Random.user, 285
sample, 318
SignRank, 1122
stem, 654
TDist, 1173
Tukey, 1187
Uniform, 1190
Weibull, 1197
Wilcoxon, 1203

*Topic documentation
apropos, 1233
args, 15
buildVignettes, 1209
checkTnF, 1211
checkVignettes, 1212
codoc, 1213
data, 1247
Defunct, 86
demo, 1253
Deprecated, 90
Documentation, 746
documented, 1259
tools.start, 1272
help, 1269
help.search, 1272
help.start, 1274
NotYet, 240
NumericConstants, 246
prompt, 1293
promptData, 1295
promptPackage, 1296
QC, 1222
Quotes, 278
Rdpackage, 1224
Rdutils, 1224
RSiteSearch, 1305
str, 1315
Syntax, 376
tools-package, 1227
undoc, 1228
d vignette, 1326
*Topic dplot
absolute.size, 665
approxfun, 826
axTicks, 566
boxplot.stats, 499
cm, 503
col2rgb, 504
colors, 507
countourLines, 507
convertNative, 666
convolve, 874
dataViewport, 667
drawDetails, 668
ecdf, 902
editDetails, 669
expression, 117
extendrange, 516
fft, 916
gEdit, 669
getAddress, 670
gpar, 671
gPath, 673
Grid, 674
Grid Viewports, 674
ggrid.add, 678
ggrid.arrows, 679
ggrid.circle, 682
ggrid.collection, 683
ggrid.convert, 684
ggrid.copy, 686
ggrid.display.list, 686
ggrid.draw, 687
ggrid.edit, 688
ggrid.frame, 689
ggrid.get, 690
ggrid.grab, 691
ggrid.grill, 693
ggrid.grob, 693
ggrid.layout, 695
ggrid.lines, 697
ggrid.locator, 698
ggrid.move.to, 699
ggrid.newpage, 700
ggrid.pack, 701
ggrid.place, 702
ggrid.plot.and.legend, 703
ggrid.points, 704
ggridpolygon, 705
ggrid.pretty, 706
ggrid.prompt, 707
ggrid.record, 707
ggrid.rect, 708
ggrid.refresh, 710
ggrid.remove, 710
ggrid.segments, 711
ggrid.set, 712
ggrid.show.layout, 713
ggrid.show.viewport, 714
ggrid.text, 715
ggrid.xaxis, 717
ggrid.yaxis, 719
grobWidth, 720
hcl, 520
hist, 592
hist.POSIXt, 595
hsv, 525
jitter, 184
layout, 599
Palettes, 530
panel.smooth, 616
par, 617
plot.density, 1052
plotViewport, 720
pop.viewport, 721
ppoints, 1068
pretty, 263
push.viewport, 722
Querying the Viewport Tree, 722
rgb2hsv, 548
screen, 646
splinefun, 1137
stepfun, 1153
stringWidth, 724
strwidth, 656
trans3d, 551
unit, 724
unit.c, 726
unit.length, 727
unit.pmin, 727
unit.rep, 728
units, 663
validDetails, 729
vpPath, 730
widthDetails, 731
Working with Viewports, 731
xy.coords, 556
xyz.coords, 557
*Topic environment
apropos, 1233
as.environment, 19
browser, 37
commandArgs, 55
debug, 85
eapply, 107
gc, 150
gctorture, 152
interactive, 177
is.R, 182
layout, 599
ls, 211
Memory, 225
Memory-limits, 227
options, 249
par, 617
quit, 277
R.Version, 280
reg.finalizer, 302
remove, 307
Startup, 355
stop, 357
stopifnot, 358
Sys.getenv, 377
Sys.putenv, 381
taskCallback, 393
taskCallbackManager, 395
taskCallbackNames, 397

*Topic error
  bug.report, 1237
  conditions, 59
debugger, 1251
  options, 249
  stop, 357
  stopifnot, 358
  warning, 420
  warnings, 421

*Topic file
  .Platform, 3
  basename, 32
  browseURL, 1237
cat, 42
choose.files, 1241
connections, 63
count.fields, 72
dataentry, 1249
dcf, 84
dput, 100
dump, 102
file.access, 128
file.choose, 129
file.info, 130
file.path, 131
file.show, 131
files, 132
fileutils, 1217
glob2rx, 1267
gzcon, 169
list.files, 202
load, 203
package.skeleton, 1287
parse, 258
path.expand, 260
read.00Index, 1226
read.fortran, 1297
read.fwf, 1298
read.table, 292
readBin, 295
readChar, 298
readLines, 300
save, 319
scan, 322
seek, 326
sink, 338
source, 345
sys.source, 383
system, 385
system.file, 386
tempfile, 398
textConnection, 399
unlink, 412
url.show, 1323
write, 426
write.table, 427
write_PACKAGES, 1230
writeLines, 429
zip.file.extract, 430

*Topic graphs
  chull, 502

*Topic hplot
  assocplot, 561
  barplot, 567
  biplot, 851
  biplot.princomp, 853
  boxplot, 571
cdplot, 576
contour, 578
 coplot, 581
cpgram, 882
curve, 583
dendrogram, 887
dotchart, 585
ecdf, 902
filled.contour, 586
fourfoldplot, 588
heatmap, 944
hist, 592
INDEX

loess, 981
loess.control, 983

*Topic logic
  all, 8
  all.equal, 9
  any, 11
  Comparison, 57
  complete.cases, 868
  Control, 70
  duplicated, 104
  identical, 172
  ifelse, 174
  Logic, 208
  logical, 209
  match, 216
  NA, 233
  unique, 411
  which, 423

*Topic manip
  addmargins, 810
  append, 13
  c, 39
  cbind, 43
  cbind2, 742
  cut.POSIXt, 76
  deparse, 87
  dimnames, 97
  duplicated, 104
  expand.model.frame, 907
  getInitial, 934
  head, 1268
  list, 200
  mapply, 214
  match, 216
  merge, 228
  model.extract, 1005
  NA, 233
  NLSstAsymptotic, 1031
  NLSstClosestX, 1032
  NLSstLfAsymptote, 1033
  NLSstRtAsymptote, 1033
  NULL, 245
  order, 254
  order.dendrogram, 1045
  reorder, 1107
  rep, 308
  replace, 310
  reshape, 1110
  rev, 310
  rle, 311
  row/colnames, 316
  rowsum, 317
  seq, 327
  seq.Date, 329
  seq.POSIXt, 330
  sequence, 331
  slotOp, 340
  sort, 343
  sortedXyData, 1130
  stack, 354
  structure, 365
  subset, 366
  transform, 406
  type.convert, 409
  unique, 411
  unlist, 413

*Topic math
  .Machine, 1
  abs, 6
  Bessel, 33
  convolve, 874
  deriv, 893
  fft, 916
  Hyperbolic, 170
  integrate, 954
  is.finite, 179
  kappa, 186
  log, 206
  nextn, 1021
  polyroot, 262
  Special, 346
  splinefun, 1137
  Trig, 407

*Topic methods
  .BasicFunsList, 735
  addmargins, 810
  as, 735
  as.data.frame, 18
  callNextMethod, 740
  class, 52
  Classes, 743
  data.class, 77
  data.frame, 78
  Documentation, 746
  GenericFunctions, 750
  getMethod, 754
  groupGeneric, 166
  initialize-methods, 759
  InternalMethods, 178
  is, 760
  is.object, 182
  isSealedMethod, 763
  Methods, 768
  methods, 1284
SSfol, 1143
Ssfpl, 1144
SSgompertz, 1145
SSlogis, 1146
SSmicmen, 1147
SSweibull, 1148
stat.anova, 1150
step, 1151
summary.aov, 1160
summary.glm, 1162
summary.lm, 1164
summary.manova, 1165
terms, 1176
terms.formula, 1177
terms.object, 1178
tilde, 400
TukeyHSD, 1188
update, 1192
update.formula, 1193
vcov, 1196

*Topic multivariate
anova.mlm, 820
as.hclust, 842
biplot, 851
biplot.princomp, 853
cancor, 857
cmdscale, 865
cophenetic, 875
cor, 876
cov.wt, 881
cutree, 883
dendrogram, 887
dist, 898
factanal, 909
hclust, 941
kmeans, 965
loadings, 981
mahalanobis, 995
mauchley.test, 1001
prcomp, 1072
princomp, 1085
scatterplot, 1115
SSD, 1142
stars, 652
summary.princomp, 1167
varimax, 1195

*Topic nonlinear
deriv, 893
getInitial, 934
nlm, 1022
nls, 1026
nls.control, 1029
nlsModel, 1030
optim, 1038
plot.profile.nls, 1058
predict.nls, 1082
profile.nls, 1091
profiler, 1092
profiler.nls, 1093
vcov, 1196

*Topic nonparametric
sunflowerplot, 658

*Topic optimize
cstrOptim, 869
glm.control, 939
nlm, 1022
nlminb, 1024
optim, 1038
optimize, 1043
uniroot, 1191

*Topic package
base-package, 32
datasets-package, 446
graphics-package, 591
grid-package, 678
methods-package, 771
stats-package, 1151
tools-package, 1227
utils-package, 1325

*Topic print
cat, 42
dcf, 84
format, 140
format.info, 144
format.pval, 145
formatC, 146
formatD, 148
labels, 189
loadings, 981
ls.str, 1280
noquote, 239
octmode, 247
options, 249
plot.isoreg, 1054
print, 265
print.data.frame, 267
print.default, 267
printCoeffmat, 1089
prmatrix, 269
printf, 350
str, 1315
write.table, 427

*Topic programming
.BlackFunsList, 735
INDEX

 anomaly, 816
 anomaly.glm, 817
 anomaly.lm, 818
 anomaly.ml, 819
 anomaly.mml, 820
 .Machine, 1
 all.equal, 9
 all.names, 10
 as, 735
 as.function, 20
 autoload, 29
 body, 35
 bquote, 36
 browser, 37
 call, 40
 callNextMethod, 740
 check.options, 501
 checkFF, 1210
 Classes, 743
 commandArgs, 55
 conditions, 59
 Control, 70
 debug, 85
 delayedAssign, 86
 delete.response, 885
 deparse, 87
 deparseOpts, 89
 do.call, 98
 Documentation, 746
 dput, 100
 environment, 111
 eval, 113
 expression, 117
 fixPre1.8, 748
 force, 135
 Foreign, 136
 formals, 139
 format.info, 144
 function, 149
 GenericFunctions, 750
 getCallingDLL, 154
 getClass, 753
 getMethod, 754
 getNumCConverters, 158
 getPackageName, 757
 hasArg, 758
 identical, 172
 ifelse, 174
 initialize-methods, 759
 interactive, 177
 invisible, 178
 is, 760
 is.finite, 179
 is.function, 180
 is.language, 181
 is.recursive, 183
 isSealedMethod, 763
 Last.value, 191
 makeClassRepresentation, 766
 match.arg, 218
 match.call, 219
 match.fun, 220
 menu, 1283
 message, 230
 Methods, 768
 missing, 230
 model.extract, 1005
 name, 234
 nargs, 236
 new, 773
 ns-dblcolon, 241
 ns-topenv, 244
 on.exit, 248
 Paren, 258
 parse, 258
 promptClass, 775
 promptMethods, 777
 R.Version, 280
 Recall, 302
 recover, 1301
 reg.finalizer, 302
 representation, 778
 setClass, 780
 setClassUnion, 784
 setGeneric, 786
 setMethod, 789
 setOldClass, 792
 show, 794
 slot, 798
 source, 345
 stop, 357
 stopifnot, 358
 substitute, 368
 switch, 375
 sys.parent, 379
 tools-package, 1227
 trace, 402
 traceback, 405
 try, 408
 utils-package, 1325
 validObject, 801
 warning, 420
 warnings, 421
 with, 425
 *Topic regression
 anova, 816
 anova.glm, 817
 anova.lm, 818
 anova.mlm, 820
INDEX

aov, 824
anci, 858
case/variable.names, 858
coeff, 867
coefficient, 871
contrast, 871
contrasts, 873
df.residual, 896
effects, 905
expand.model.frame, 907
fitted, 921
fitted.values, 921
fitted.values, 921
glm, 935
glm.summaries, 940
influence.measures, 951
isoreg, 959
line, 973
lm, 973
lm.fit, 976
lm.influence, 978
lm.summaries, 979
ls.diag, 991
ls.print, 992
lsfit, 992
nls, 1026
nls.control, 1029
plot.lm, 1055
plot.profile.nls, 1058
ppr, 1069
predict.glm, 1076
predict.lm, 1079
predict.nls, 1082
profile.nls, 1091
profiler.nls, 1093
stat.anova, 1150
summary.aov, 1160
summary.glm, 1162
summary.lm, 1164
termplot, 1174
weighted.residuals, 1199
loess, 981
loess.control, 983
lowess, 990
predict.loess, 1080
predict.smooth.spline, 1083
runmed, 1112
scatter.smooth, 1114
smooth, 1124
smooth.spline, 1126
smoothEnds, 1129

*Topic sysdata
Machine, 1
colors, 507
commandArgs, 55
Constants, 69
NULL, 245
palette, 529
R.Version, 280
Random, 282
Random.user, 285

*Topic tree
dendrogram, 887

*Topic ts
acf, 806
acf2AR, 808
ar, 828
ar.ols, 831
arima, 833
arima.sim, 836
arima0, 837
ARMAacf, 840
ARMAtoMA, 841
Box.test, 855
cpgram, 882
decompose, 884
diffinv, 897
embed, 906
filter, 917
HoltWinters, 947
KalmanLike, 961
kernapply, 962
kernel, 963
lag, 970
lag.plot, 971
monthplot, 1011
na.contiguous, 1016
plot.acf, 1051
plot.HoltWinters, 1053
plot.spec, 1059
plot.ts, 1062

*Topic robust
fivenum, 922
IQR, 958
line, 973
mad, 994
median, 1003
medpolish, 1004
runmed, 1112
smooth, 1124
smoothEnds, 1129

*Topic smooth
bandwidth, 845
density, 890
isoreg, 959
ksmooth, 969

/* Topic sysdata */
PP.test, 1067
predict.Arima, 1075
predict.HoltWinters, 1078
print.ts, 1088
spec.ar, 1131
spec.pgram, 1132
spec.taper, 1134
spectrum, 1135
start, 1149
stl, 1155
stlmethods, 1157
StructTS, 1158
time, 1179
toeplitz, 1180
ts, 1180
ts-methods, 1182
ts.plot, 1183
ts.union, 1184
tsdag, 1185
tsp, 1186
tSSmooth, 1186
window, 1204

*Topic univar
ave, 844
cor, 876
Extremes, 124
fivenum, 922
IQR, 958
mad, 994
mean, 224
median, 1003
nclass, 528
order, 254
quantile, 1101
range, 287
rank, 288
sd, 1116
sort, 343
stem, 654
weighted.mean, 1198

*Topic utilities
.Platform, 3
.checkMFClasses, 805
alarm, 1233
all.equal, 9
as.POSIX*, 21
axis.POSIXct, 565
BATCH, 1235
bug.report, 1237
buildVignettes, 1209
builtins, 37
capabilities, 41
capture.output, 1240
check.options, 501
checkFF, 1210
checkMD5sums, 1211
checkTnF, 1211
checkVignettes, 1212
chooseCRANmirror, 1242
clipboard, 1246
compareVersion, 1247
conflicts, 63
dataentry, 1249
date, 80
Dates, 81
DateTimeClasses, 81
dep, 1251
Defunct, 86
demo, 1253
Deprecated, 90
dev2bitmap, 514
diffftime, 95
DLL.version, 1254
download.file, 1255
das, 1257
das.data.frame, 1258
encoded_text_to_latex, 1215
encodeString, 109
das.example, 1259
das.file.edit, 1261
das.findInterval, 134
das.fix, 1262
das.flush.console, 1262
das.format.Date, 142
das.gc.time, 151
das.getDepList, 1218
das.getpid, 160
dasgettext, 160
das.getwd, 162
das.getWindowsHandle, 1266
glob2rx, 1267
grep, 163
iconv, 171
index.search, 1275
INSTALL, 1276
installed.packages, 1277
install1FoundDepends, 1219
integrate, 954
is.R, 182
jitter, 184
110n_info, 188
link.html.help, 1278
localeconv, 204
locales, 205
localeToCharset, 1279
ls.str, 1280
makeLazyLoading, 1220
manglePackageName, 214
mapply, 214
maxCol, 223
md5sum, 1221
memory.profile, 227
memory.size, 1282
menu, 1283
noquote, 239
normalizePath, 1286
NotYet, 240
ns-hooks, 242
ns-load, 243
object.size, 1286
package-version, 257
package.dependencies, 1222
package.skeleton, 1287
packageDescription, 1288
packageStatus, 1289
page, 1291
PkgUtils, 1292
pos.to.env, 263
proc.time, 270
QC, 1222
R.home, 279
Rdindex, 1224
RdUtils, 291
Rdutils, 1224
readline, 299
relevel, 1106
REMOVE, 1302
remove.packages, 1303
reorder.factor, 1108
Rprof, 1304
RSiteSearch, 1305
Rtangle, 1306
RweaveLatex, 1307
Rwin configuration, 1309
savehistory, 1310
select.list, 1311
setRepositories, 1312
setWindowTitle, 1313
shell, 333
shell.exec, 334
SHLIB, 1314
shQuote, 336
str, 1315
strptime, 359
strtrim, 364
Sweave, 1316
SweaveSyntConv, 1318
symnum, 1169
Sys.getenv, 377
Sys.info, 378
Sys.putenv, 381
Sys.sleep, 382
sys.source, 383
Sys.time, 384
system, 385
system.file, 386
system.time, 387
texi2dvi, 1226
toString, 401
uname, 414
update.packages, 1320
URLencode, 1324
UserHooks, 416
utf8Conversion, 418
vignetteDepends, 1229
which.min, 424
winDialog, 1327
winextras, 1328
winMenus, 1328
write_PACKAGES, 1230
xgettext, 1231
zutils, 432
'(Quotes), 278
*(Arithmetic), 16
*.difftime (difftime), 95
+(Arithmetic), 16
+.Date (Dates), 81
+.POSIXt (DateTimeClasses), 81
-(Arithmetic), 16
-.Date (Dates), 81
-.POSIXt (DateTimeClasses), 81
-> (assignOps), 24
>> (assignOps), 24
.AutoloadEnv (autoload), 29
.Autoloaded (autoload), 29
.BaseNamespaceEnv (environment), 111
.BasicFunsList, 735
.C, 100, 106, 107, 154, 155, 157–159, 177,
1210
.C (Foreign), 136
.Call, 106, 107, 154, 155, 157, 158
.Call (Foreign), 136
.Class (groupGeneric), 166
.Defunct (Defunct), 86
.Deprecated (Deprecated), 90
.Device, 1
.Devices (.Device), 1
.External, 106, 107, 154, 155, 157, 158
.External (Foreign), 136
.First, 177, 278
.First (Startup), 355
.First.lib, 107, 199, 200, 242
.First.lib (library), 195
.Fortran, 100, 106, 107, 154, 155, 157, 158, 177, 1210
.Fortran (Foreign), 136
.Generic (groupGeneric), 166
.GlobalEnv, 325, 368, 379
.GlobalEnv (environment), 111
.Group (groupGeneric), 166
.InitTraceFunctions
 (TraceClasses), 800
.Internal, 37, 265
.Internal (Internal), 177
.Last, 355, 356, 1311
.Last (quit), 277
.Last.lib, 199, 242
.Last.lib (library), 195
.Last.value (Last.value), 191
.Library (libPaths), 194
.MFclass, 1178
.MFclass (.checkMFClasses), 805
.Machine, 1, 4, 297, 1044
.Method (groupGeneric), 166
.NotYetImplemented (NotYet), 240
.NotYetUsed (NotYet), 240
.OldClassesList (setOldClass), 792
.Options (options), 249
.Pars (par), 617
.Platform, 3, 3, 41, 280, 379
.Primitive, 177, 258
.Primitive (Primitive), 265
.Random.seed, 1035, 1123, 1124, 1190
.Random.seed (Random), 282
.Renviron (Startup), 355
.Rprofile (Startup), 355
.Script, 4
.Traceback (traceback), 405
.checkMFClasses, 805
.decode_package_version
 (package-version), 257
.deparseOpt, 88, 100, 103
.deparseOpt (deparseOpt), 89
.doTracePrint (TraceClasses), 800
.dynLibs (library.dynam), 198
.encode_package_version
 (package-version), 257
.getXlevels (.checkMFClasses), 805
.handleSimpleError (conditions), 59
.helpForCall (help), 1269
.isMethodsDispatchOn (UseMethod), 415
.leap.seconds (DateTimeClasses), 81
.libPaths, 198, 200, 431, 1248, 1303
.libPaths (libPaths), 194
.makeTracedFunction
 (TraceClasses), 800
.noGenerics (library), 195
.onAttach, 244
.onAttach (ns-hooks), 242
.onLoad, 196, 199, 244
.onLoad (ns-hooks), 242
.onUnload, 199
.onUnload (ns-hooks), 242
.packages, 198, 200, 326, 1323
.packages (zpackages), 431
.primTrace (trace), 402
.primUntrace (trace), 402
.ps.prolog (postscript), 539
.setOldIs (setOldClass), 792
.signalSimpleWarning
 (conditions), 59
.slotNames (slot), 798
.standard_regexps (zutils), 432
.tryHelp (help), 1269
.untracedFunction (TraceClasses), 800
.userHooksEnv (UserHooks), 416
/ (Arithmetic), 16
/.difftime (difftime), 95
::, 177
: (seq), 327
:: (ns-dbicolon), 241
::: (ns-dbicolon), 241
<(Comparison), 57
<=, 24
<- (assignOps), 24
<-class (language-class), 764
<= (Comparison), 57
<<-, 26
<<- (assignOps), 24
=(assignOps), 24
==, 10
==(Comparison), 57
>(Comparison), 57
>= (Comparison), 57
? (help), 1269
INDEX

[. 102, 178, 367, 963, 964
[(Extract), 118
[.AsIs (AsIs), 22
[.Date (Dates), 81
[.POSIXct (DateTimeClasses), 81
[.POSIXlt (DateTimeClasses), 81
[.SavedPlots (windows), 551
[.acf (acf), 806
[.data.frame, 79, 118, 120, 1007
[.data.frame
  (Extract.data.frame), 121
[.difftime (difftime), 95
[.factor, 118, 120, 127
[.factor (Extract.factor), 123
[.formula (formula), 924
[.getAnywhere (getAnywhere), 1263
[.noquote (noquote), 239
[.octmode (octmode), 247
[.package_version
  (package-version), 257
[.terms (delete.response), 885
[.ts (ts), 1180
[-, 178
[- (Extract), 118
[-.Date (Dates), 81
[-.POSIXct (DateTimeClasses), 81
[-.POSIXlt (DateTimeClasses), 81
[-.data.frame
  (Extract.data.frame), 121
[-.factor (Extract.factor), 123
[(, 178, 889
[[(Extract), 118
[[.Date (Dates), 81
[[(.POSIXct (DateTimeClasses), 81
[[(.data.frame
  (Extract.data.frame), 121
[[(.dendrogram (dendrogram), 887
[[(.package_version
  (package-version), 257
[(-, 178
[( (Extract), 118
[[(.data.frame
  (Extract.data.frame), 121
$. 178
$. (Extract), 118
$.package_version
  (package-version), 257
$-, 178
$ (Extract), 118
$-.data.frame
  (Extract.data.frame), 121
%-% (matmult), 221
%/% (Arithmetic), 16
%/(Arithmetic), 16
%in%, 332
%in% (match), 216
%o, 73
%o (outer), 256
%x% (kronecker), 187
% (Logic), 208
%% (Logic), 208
___ClassMetaData (Classes), 743
^ (Arithmetic), 16
~ (tilde), 400
\ (Quotes), 278
\ (Logic), 208
abbreviate, 5, 1170
ability.cov, 433, 911
abline, 559, 591, 592, 643
abs, 6, 337
absolute.size, 665, 731
acf, 806, 1052
acf2AR, 808
acos, 171
acos (Trig), 407
acosh (Hyperbolic), 170
adapt, 955
add.scope (factor.scope), 912
addi, 808, 817, 913, 1152, 1153
addGrob, 673, 679, 689, 691
addGrob (grid.add), 678
addmargins, 810
addTaskCallback, 394, 396, 397
addTaskCallback (taskCallback), 393
aggregate, 14, 317, 393, 812
agnes, 843, 875, 943
agrep, 7, 165, 1273
AIC, 814, 1027
airmiles, 434
AirPassengers, 434, 1159, 1187
airquality, 435
alarm, 1233
alias, 815, 825, 867
alist, 21, 35, 139
alist (list), 200
all, 8, 10, 11, 209, 358
all.equal, 9, 57, 173
all.equal.POSIXct
  (DateTimeClasses), 81
all.names, 10
all.vars, 925
all.vars (all.names), 10
anova-class (ar.burg), 792
anova.glm, 817, 936, 940, 1150
anova.glm-class (setOldClass), 792
anova glm.null-class (setOldClass), 792
anova glmlist (anova glm), 817
anova lm, 819, 976, 980, 1150
anova mlmlist (anova mlm), 820
ansari.test, 822, 847, 923, 1014, 1195
anscombe, 436
any, 8, 11, 209
ANY-class (BasicClasses), 739
aov, 252, 633, 809, 810, 824, 872, 873, 902, 905, 906, 913, 973, 976, 980, 997, 998, 1005, 1011, 1056, 1094, 1095, 1117, 1151, 1161, 1166, 1178, 1189
aov-class (setOldClass), 792
aperm, 12, 18, 389, 1111
append, 13
apply, 13, 54, 55, 104, 190, 220, 375, 393, 813
applyEdit (gEdit), 669
applyEdits (gEdit), 669
approx, 1135, 1138
approx (approxfun), 826
approxfun, 826, 903, 1061, 1138, 1154
apropos, 165, 211, 307, 1233, 1273
ar, 828, 832, 835, 840, 1131, 1132
ar.burg (ar), 828
ar.mle (ar), 828
ar.ols, 829, 830, 831
ar.yw, 808
ar.yw (ar), 828
Arg (complex), 58
args, 15, 35, 139, 150, 236, 1280, 1315, 1316
arima, 833, 836, 837, 839–842, 962, 1075, 1076, 1159, 1185
arima.sim, 835, 836, 918
arima0, 830, 834, 835, 837
Arith (groupGeneric), 166
Arithmetic, 6, 16, 180, 207, 221, 347, 377, 1066
ARMAacf, 807, 808, 840, 842
ARMAcoMA, 841, 841
array, 17, 98, 102, 120, 241, 392, 423
array-class (StructureClasses), 800
arrows, 560, 649
arrowsGrob (grid arrows), 679
as, 53, 419, 735, 744, 759, 780
as.array (array), 17
as.call (call), 40
as.character, 110, 141, 142, 178, 237, 260, 351, 1170
as.character (character), 46
as.character.condition (conditions), 59
as.character.Date (format Date), 142
as.character.error (conditions), 59
as.character.octmode (octmode), 247
as.character.package_version (package-version), 257
as.character.person (person), 1291
as.character.personList (person), 1291
as.character.POSIXt (strptime), 359
as.complex (complex), 58
as.data.frame, 18, 22, 78, 122, 390
as.data.frame.Date (Dates), 81
as.data.frame.package_version (package-version), 257
as.data.frame.POSIXct (DateTimeClasses), 81
as.data.frame.POSIXlt (DateTimeClasses), 81
as.data.frame.table, 19, 1206
as.data.frame.table (table), 389
as.Date (format Date), 142
dendrogram, 886, 1045
dendrogram (dendrogram), 887
difftime (difftime), 95
dist (dist), 898
double, 246, 351
double (double), 99
describe (environment), 19, 748
description (expression), 117
def (factor), 349
def (factor), 126
def (formula), 924
def (function), 20
hclust, 842, 875
integer, 71, 118, 121, 313
integer (integer), 175
INDEX

as.list, 413
as.list (list), 200
as.logical (logical), 209
as.matrix, 80, 389, 428, 898, 899
as.matrix (matrix), 222
as.matrix.dist (dist), 898
as.matrix.noquote (noquote), 239
as.matrix.POSIXt
  (DateTimeClasses), 81
as.name, 183
as.name (name), 234
as.null (NULL), 245
as.numeric (numeric), 245
as.ordered (factor), 126
as.package_version
  (package-version), 257
as.pairlist (list), 200
as.person (person), 1291
as.personList (person), 1291
as.POSIXt*, 21
as.POSIXct, 83
as.POSIXct (as.POSIXt*), 21
as.POSIXlt, 83, 205, 360, 423
as.POSIXlt (as.POSIXt*), 21
as.qr (qr), 273
as.raw (raw), 289
as.real (real), 301
as.single, 137
as.single (double), 99
as.stepfun, 960
as.stepfun (stepfun), 1153
as.symbol (name), 234
as.table (table), 389
as.table.ftable (read.ftable),
  1103
as.ts (ts), 1180
as.vector, 39, 46, 59, 71, 100, 176, 178,
  210
as.vector (vector), 418
as<-(as), 735
asin, 171
asin (Trig), 407
asinh (Hyperbolic), 170
Asis, 22, 141
asOneSidedFormula, 843
asp, 564
assign, 23, 25, 26, 153, 154, 501
assignInNamespace
  (getFromNamespace), 1264
assignOps, 24
assocplot, 561, 612
atan, 171
atan (Trig), 407
atan2 (Trig), 407
atanh (Hyperbolic), 170
attach, 23, 26, 91, 92, 198, 325, 326, 425,
  1240
attachNamespace (ns-load), 243
attenu, 437
attitude, 438
attr, 27, 28, 56, 250, 1113
attr.all.equal (all.equal), 9
attr<- (attr), 27
attributes, 9, 28, 28, 56, 98, 119, 125,
  232, 501, 886
attributes<- (attributes), 28
austres, 439
autoload, 29, 198
autoloader (autoload), 29
Autoloads (autoload), 29
available.packages, 1230, 1290
available.packages
  (update.packages), 1320
ave, 844
axis, 535, 537, 562, 565–567, 569, 575, 617,
  646
axis.Date (axis.POSIXct), 565
axis.POSIXct, 565, 596
axTicks, 264, 564, 566, 591, 621
backsolve, 30, 50, 342
backtick (Quotes), 278
bandwidth, 845
bandwidth.kernel (kernel), 963
bandwidth.nrd, 846
barplot, 567, 594, 603, 634, 635, 645
barplot.default, 519
bartlett.test, 824, 846, 923, 1014, 1195
base (base-package), 32
base-deprecated, 90
base-deprecated, 31
base-package, 32
baseenv (environment), 111
basename, 32, 133, 260
BasicClasses, 739
BATCH, 1235
bcv, 846
beaver1 (beavers), 439
beaver2 (beavers), 439
beavers, 439
Bessel, 33, 347
bessel (Bessel), 33
bessel1 (Bessel), 33
besselJ (Bessel), 33
besselK (Bessel), 33
besselY (Bessel), 33
Beta, 847
beta, 34, 848
beta (Special), 346
bindtextdomain (gettext), 160
binom.test, 849, 1097
Binomial, 850
binomial, 937
binomial (family), 913
biplot, 851, 854, 1086
biplot.default, 853
biplot.prcomp, 1074
biplot.prcomp (biplot.prcomp), 853
biplot.prcomp, 852, 853, 853, 1087
birthday, 854
bitmap, 515, 539
bitmap (dev2bitmap), 514
BJsales, 441
bmp, 515
bmp (png), 538
BOD, 441
body, 35, 139, 150
body<- (body), 35
body<-, MethodDefinition-method
(MethodsList-class), 771
box, 570, 575, 587, 638, 643, 645, 653
Box.test, 855, 1185
boxplot, 500, 571, 574, 634–636, 655
boxplot.stats, 499, 572, 573, 922, 1103
bquote, 36, 369, 537
break (Control), 70
bringToTop, 501, 554
browseEnv, 307, 1235
browser, 37, 85, 402–404, 1301, 1302
browseURL, 1237, 1274, 1306
bs, 997
bug.report, 250, 1237, 1328
build (PkgUtils), 1292
buildVignettes, 1209
builtins, 37
bw.bcv (bandwidth), 845
bw.nrd, 891, 892
bw.nrd (bandwidth), 845
bw.nrd0 (bandwidth), 845
bw.SJ (bandwidth), 845
bw.ucv (bandwidth), 845
bxp, 500, 572, 573, 574
by, 38, 229, 393
bzf ile (connections), 63

C, 127, 856, 872, 873, 1009
c, 39, 44, 83, 178, 201, 239, 413, 419
c. Date (Dates), 81
c.noquote (noquote), 239
c.package_version
(package-version), 257
c.POSIXxct (DateTimeClasses), 81
c.POSIXlt (DateTimeClasses), 81
call, 20, 40, 99, 117, 119, 181, 219, 232,
234, 302, 602, 894, 899, 960
call-class (language-class), 764
callGeneric (GenericFunctions), 750
callNextMethod, 740, 768, 772
cancor, 857
capabilities, 41, 67, 109, 172, 374
capture.output, 339, 400, 1240
cars, 442, 997
case.names, 316
case.names (case/variable.names), 858
case/variable.names, 858
casefold (chartr), 48
cat, 42, 260, 266, 421, 430, 939, 1308
Cauchy, 859
cbind, 43, 229, 742, 743, 1184
cbind.ts (ts), 1180
cbind1, 742
cbind2, ANY, ANY-method (cbind2), 742
cbind2, ANY, missing-method
(cbind2), 742
ccbind2-methods (cbind2), 742
ccf (acf), 806
cdplot, 576, 651
celling (Round), 312
cchar.expand, 45
ccharacter, 46, 127, 213, 239, 268, 280,
660, 661, 1315
ccharacter-class (BasicClasses), 739
ccharmatch, 45, 47, 165, 217, 261
ccharToRaw, 290
ccharToRaw (rawConversion), 290
cchartr, 47, 48, 165
ccheck, 1260
ccheck (PkgUtils), 1292
ccheck.options, 501, 540, 543
ccheck_tzzones (DateTimeClasses), 81
ccheckCRAN (mirrorAdmin), 1285
checkDocFiles (QC), 1222
checkDocStyle (QC), 1222
ccheckFF, 1210
ccheckMD5sums, 1211, 1221
INDEX

checkReplaceFuns (QC), 1222
checkS3methods (QC), 1222
checkTnF, 1211
checkVignettes, 1212
ChickWeight, 443
chickwts, 444
childNames (grid.grob), 693
chisq.test, 390, 454, 562, 860, 920, 1206
Chisquare, 862, 916
chol, 31, 49, 51, 109
chol2inv, 50, 51, 342
choose (Special), 346
choose.files, 129, 203, 1241
chooseCRANmirror, 1242, 1313
chron, 21, 143
chull, 502
circleGrob (grid.circle), 682
CITATION, 1243
CITATION (citEntry), 1244
citation, 1242
citEntry, 1243, 1244, 1320
citFooter (citEntry), 1244
citHeader (citEntry), 1244
class, 52, 77, 168, 182, 211, 239, 265, 371, 390, 416, 592, 636, 816, 975, 1074, 1284
class<- (class), 52
Classes, 743, 745, 754, 774, 799
classRepresentation-class, 766, 780
classRepresentation-class, 744, 785
ClassUnionRepresentation-class (setClassUnion), 784
clearNames, 864, 1121
clipboard, 1246
close (connections), 63
close.screen (screen), 646
close.socket, 1246, 1282, 1300
closeAllConnections (showConnections), 335
cm, 503
cm.colors (Palettes), 530
cmdscale, 865
cointervals (coplot), 581
CO2, 445
c02, 446
codoc, 1213, 1228
codocClasses (codoc), 1213
codocData (codoc), 1213
coeff, 834, 867, 869, 906, 940, 976, 980, 1027, 1165
coefficients, 817, 921, 936
coefficients (coef), 867
coe<ke (as), 735
coe<ke, ANY, array-method (as), 735
coe<ke, ANY, call-method (as), 735
coe<ke, ANY, character-method (as), 735
coe<ke, ANY, complex-method (as), 735
coe<ke, ANY, environment-method (as), 735
coe<ke, ANY, expression-method (as), 735
coe<ke, ANY, function-method (as), 735
coe<ke, ANY, integer-method (as), 735
coe<ke, ANY, list-method (as), 735
coe<ke, ANY, logical-method (as), 735
coe<ke, ANY, matrix-method (as), 735
coe<ke, ANY, name-method (as), 735
coe<ke, ANY, NULL-method (as), 735
coe<ke, ANY, numeric-method (as), 735
coe<ke, ANY, single-method (as), 735
coe<ke, ANY, ts-method (as), 735
coe<ke, ANY, vector-method (as), 735
coe<ke-methods (as), 735
coe<ke<- (as), 735
col, 53, 315, 329, 340
col2rgb, 504, 507, 509, 529, 531, 547, 548
colMeans (colSums), 54
colnames, 98
colnames (row/colnames), 316
colnames<- (row/colnames), 316
colorConverter, 509
colorConverter (make.rgb), 526
colorRamp, 505
colorRampPalette (colorRamp), 505
colors, 504, 507, 509, 529, 531, 622, 623, 639, 902
colorspaces (convertColor), 508
colours (colors), 507
colSums, 54
commandArgs, 55, 356
comment, 56
comment<- (comment), 56
Compare (groupGeneric), 166
compareVersion, 257, 1247
Comparison, 57, 173, 289, 343, 377
complete.cases, 234, 868
Complex, 59
Complex(groupGeneric), 166
complex, 5, 58, 247, 262
complex-class(BasicClasses), 739
computeRestarts(conditions), 59
condition(conditions), 59
conditionCall(conditions), 59
conditionMessage(conditions), 59
conditions, 59, 230
confint, 868, 976
cointf.glm, 869
cointf.nls, 869
conflicts, 26, 63, 195
Conj(complex), 58
connection, 72, 293, 322, 1226, 1299
connection(connections), 63
connections, 63, 250, 251, 273, 297, 299, 301, 327, 335, 400, 430
Constants, 69
constrOptim, 869, 1041
contour, 508, 524, 526, 578, 587, 599, 625
contourLines, 507, 580
contr.helmert, 873
contr.helmert(contrast), 871
contr.poly, 873
contr.poly(contrast), 871
contr.SAS(contrast), 871
contr.sum, 857, 873
contr.sum(contrast), 871
contr.treatment, 873, 1107
contr.treatment(contrast), 871
contrast, 871
contrasts, 124, 252, 857, 872, 873, 1009, 1117
contrasts<- (contrasts), 873
contrib.url, 1242
contrib.url(update.packages), 1320
contributors, 70, 71
Control, 70, 377
convertColor, 508, 526, 527
convertHeight(grid.convert), 684
convertNative, 666
convertUnit(grid.convert), 684
convertWidth(grid.convert), 684
convolve(grid.convert), 684
convolveY(grid.convert), 684
convolve, 874, 917, 918, 963, 1021
cooks.distance, 979, 1056
cooks.distance
(influence.measures), 951
cophenetic, 875, 1108
coplot, 581, 616, 617, 648, 925
copyright, 71
copyrights(copyright), 71
cor, 876, 1074, 1086, 1087
cor.test, 878, 879
cos, 171
cos(Trig), 407
cosh(Hyperbolic), 170
count.fields, 72, 295
cov, 882, 995, 1074, 1087
cov(corr), 876
cov.mcd, 1085
cov.mve, 1085
cov.wt, 878, 881, 909, 1085
cov2cor(corr), 876
covratio, 979
covratio(influence.measures), 951
coxph, 1175, 1178
cpgram, 882, 1135
CRAN.packages, 1222, 1255, 1256
CRAN.packages (utils-deprecated), 1325
crossprod, 73
cummax(cumsum), 73
cummin(cumsum), 73
cumprod, 271
cumprod(cumsum), 73
cumsum, 73, 271
current.transform(Querying the Viewport Tree), 722
current.viewport(Querying the Viewport Tree), 722
current.vpPath(Querying the Viewport Tree), 722
current.vpTree(Querying the Viewport Tree), 722
curve, 583
cut, 74, 76, 77, 349, 598
cut.Date, 81
cut.Date(cut.POSIXt), 76
cut.dendrogram(dendrogram), 887
cut.POSIXt, 76, 83
cutree, 883, 943
cycle(time), 1179
D(deriv), 893
daisy, 900
data, 69, 70, 198, 320, 1247, 1271
data.class, 77
data.entry, 1258, 1259
data.entry(dataentry), 1249
data.frame, 19, 22, 23, 44, 46, 78, 80, 91, 92, 97, 98, 122, 123, 167, 213, 222,
INDEX

229, 267, 295, 316, 325, 388, 407, 414, 629, 630, 1006, 1008, 1298, 1299
data.frame-class (setOldClass), 792
data.matrix, 79, 222, 614, 630
dataentry, 252, 1249
datasets (datasets-package), 446
datasets-package, 446
dataViewport, 667, 721
Date, 94, 143, 314, 330, 361, 384, 423
Date (Dates), 81
date, 21, 80, 143, 205, 384, 1179
dates, 81, 83, 565
DateTimeClasses, 22, 70, 81, 96, 130, 314, 331, 362, 384, 423, 565
dbeta, 933
dbeta (Beta), 847
dbinom, 1020, 1064
dbinom (Binomial), 850
dcauchy (Cauchy), 859
dcf, 84
dchisq, 916, 933
dchisq (Chisquare), 862
de (dataentry), 1249
debug, 37, 85, 150
debugger, 1251
deimMatch, 1215
deltat, 971
deltat (time), 1179
demo, 346, 1253, 1261
dendrapply, 886
dendrogram, 310, 875, 883, 886, 887, 943, 944, 1045
density, 577, 578, 594, 629, 637, 659, 845, 846, 890, 1052
density-class (setOldClass), 792
deparse, 47, 87, 89, 101, 103, 237, 259, 279, 368
deparseOpts, 89
Deprecation, 31, 86, 90, 240, 1227, 1325
deriv, 893, 1022, 1023
deriv3 (deriv), 893
det, 90, 109, 275
detach, 27, 91, 196–198, 244, 326, 417
determinant (det), 90
dev.control (dev2), 512
dev.copy (dev2), 512
dev.copy2eps (dev2), 512
dev.cur, 513, 516
dev.cur (dev.xxx), 511
dev.interactive, 510, 516
dev.list (dev.xxx), 511
dev.next (dev.xxx), 511
dev.off (dev.xxx), 511
dev.prev (dev.xxx), 511
dev.print, 516, 539, 550
dev.print (dev2), 512
dev.set (dev.xxx), 511
dev.xxx, 511
dev2, 512
dev2bitmap, 514, 516
deviance, 896, 897, 940, 980
device (Devices), 515
Devices, 511, 512, 515, 533, 534, 539, 543, 554, 556, 648
dexp, 1198
dexp (Exponential), 908
df, 1174
df (FDist), 915
df.kernel (kernel), 963
df.residual, 896, 896, 940, 980
dfbeta (influence.measures), 951
dfbetas, 979
dfbetas (influence.measures), 951
dffits, 979
dffits (influence.measures), 951
dgamma, 848, 864, 909
dgamma (GammaDist), 931
dgeom, 1020
dgeom (Geometric), 933
dget, 104
dget (dput), 100
dhyper (Hypergeometric), 949
diag, 93, 210, 221
diag <- (diag), 93
diana, 843
diff, 94, 897, 898, 971, 1183
diff.ts, 95
diff.ts (ts-methods), 1182
diffinv, 95, 897
difftime, 81–83, 95, 167, 330, 331
digamma (Special), 346
dim, 18, 28, 96, 125, 178, 224, 241, 392, 423
dim <-, 178
dim <- (dim), 96
dimnames, 18, 28, 97, 97, 178, 222, 269, 316, 414, 638
dimnames <-, 178
dimnames <- (dimnames), 97
dir(list.files), 202
dir.create(files), 132
dirname(basename), 32
discoveries, 447
dist, 866, 876, 898, 899, 944
DLL.version, 1254
dlnorm, 1035
dlnorm(Lognormal), 988
dlogis(Logistic), 984
dmultinom(Multinomial), 1014
DNase, 447
dnbinom, 851, 934, 1064
dnbinom(NegBinomial), 1019
dnorm, 989
dnorm(Normal), 1034
do.call, 40, 98, 302
Documentation, 746
Documentation-class
  (Documentation), 746
Documentation-methods
  (Documentation), 746
dotchart, 569, 585, 628
double, 99, 125, 175, 558
double-class(BasicClasses), 739
download.file, 41, 65, 203, 250, 251, 1255, 1290, 1321, 1323, 1325
download.packages, 1256
download.packages
  (update.packages), 1320
downViewport, 676, 730
downViewport(Working with Viewports), 731
dpois, 851, 1020
dpois(Poisson), 1063
dput, 89, 100, 104, 320, 1291, 1315
dQuote, 369
dQuote(sQuote), 353
draw.details(drawDetails), 668
drawDetails, 668
drop, 102, 221
drop.scope(factor.scope), 912
drop.terms(delete.response), 885
drop1, 102, 817, 818, 820, 913, 1152, 1153
drop1(add1), 808
dsignrank, 1204
dsignrank(SignRank), 1122
dt, 860, 916
dt(TDist), 1173
dummy.coef, 901
dump, 89, 101, 102, 320
dump.frames, 250, 1301, 1302
dump.frames(debugger), 1251
dump.frames-class(setOldClass), 792
dumpMethod(GenericFunctions), 750
dumpMethods(GenericFunctions), 750
dunif(Uniform), 1190
duplicated, 104, 411
duplicated.POSIXlt
  (DateTimeClasses), 81
dweibull, 909
dweibull(Weibull), 1197
dwilcox, 1123
dwilcoxon(Wilcoxon), 1203
dyn.load, 3, 105, 137, 138, 158, 199, 200, 253, 1314
dyn.unload(dyn.load), 105
eapply, 107
ecdf, 135, 902, 1061, 1103, 1154
dit, 252, 402, 403, 1251, 1257, 1259, 1261, 1262, 1264, 1291
dit.data.frame, 1258, 1258, 1262
dit.matrix(dit.data.frame), 1258
dit.vignette(vignette), 1326
ditDetails, 669
ditGrob, 670, 673
ditGrob(grid.edit), 688
eff.aovlist, 904
effects, 817, 905, 938, 940, 975, 976, 980
eigen, 108, 275, 374, 1074, 1086, 1087
eelse(Control), 70
emacs(edit), 1257
embed, 906
encoded_text_to_latex, 1215
encodeString, 109, 142, 238, 269
dend, 1181
dend(start), 1149
eengine.display.list
  (grid.display.list), 686
environment-class, 747
environment<-(environment), 111
erase.screen(screen), 646
Error(aov), 824
esoph, 448, 938
estVar(SSD), 1142
euro, 449
eurodist, 450
eval, 112, 113, 117, 119, 259, 346, 369, 380, 546
formals <- (formals), 139
gEdit, 669
gEditList (gEdit), 669
genericFunction-class, 749
GenericFunctions, 750, 757, 767, 797
genericFunctionWithTrace-class (TraceClasses), 800
Geometric, 933
get, 24, 111, 112, 116, 152, 220, 241, 263, 251, 782, 1263–1265, 1280
get.gpar (gpar), 671
getAllConnections (showConnections), 335
getAnywhere, 1263, 1284
callingDLL, 154
callingDLLLe (getCallingDLL), 154
cgetConverterDescriptions (getNumCConverters), 158
cgetConverterStatus (getNumCConverters), 158
cgetClass, 745, 753, 798, 799
classDef, 745
classDef (getClass), 753
cClasses (setClass), 780
cgetConnection (showConnections), 335
cDepList, 1218
cDLLRegisteredRoutines, 155, 156, 158
cgeterrmessage, 409, 1252
cgeterrmessage (stop), 357
cFromClassNamespace, 1263, 1264
cgetGeneric, 751
cgetGenerics, 775, 776
cgetGenerics (GenericFunctions), 750
cgetGraphicsEvent, 517
cgetGrob, 673, 679, 689, 691, 711
cgetGrob (grid.grob), 690
cgetHook (UserHooks), 416
cgetIdentification, 1266
cgetIdentification (setTitle), 1313
cgetInitial, 934
cgetLoadedDLLs, 106, 155, 156, 200
cgetMethod, 753, 754, 1270
cgetMethods (getMethod), 754
cgetMethodsMetaData, 756
cgetName, 670
cgetNativeSymbolInfo, 156, 157
cgetNumCConverters, 158
getOption, 140, 144, 268, 420
getSocketAddress, 249
getPackageNamespace, 757, 766
getpid, 160
getRversion, 280
getRversion(packageVersion), 257
gets3method, 416, 1264, 1265, 1284
getTaskCallbackNames, 394, 396
getTaskCallbackNames(taskCallbackNames), 397
ggettext, 160, 230, 350, 352, 357, 358, 420, 421, 1231
ggettextf, 1231
ggettextf(sprintf), 350
getValidity, 802
getwd, 162, 202, 293, 322, 378
getWindowsHandle, 1266
getWindowTitle(setWindowTitle), 1313
gl, 127, 163, 332
gList(grid.grob), 693
glm, 372, 809, 817, 818, 867, 872, 873, 896, 897, 913, 914, 921, 924, 925, 935, 939, 940, 952, 953, 959, 976, 980, 996, 1006, 1017, 1037, 1055, 1077, 1124, 1151, 1152, 1162, 1163, 1175, 1176, 1196, 1199
glm-class(setOldClass), 792
glm.control, 935, 939
glm.fit, 939
glm.null-class(setOldClass), 792
glm.summaries, 940
glob2rx, 1267
globalenv, 21, 35
globalenv(environment), 111
gpar, 671
gPath, 673, 702, 703
graphics(graphics-package), 591
graphics-package, 591
graphics.off, 516
graphics.off(dev.xxx), 511
gray, 507, 518, 519, 525, 529, 531, 547, 623
gray.colors, 519, 577, 630
gregexpr(grep), 163
grep, 8, 47, 48, 163, 211, 261, 303, 307, 363, 1273
grey, 505
grey(gray), 518
grey.colors(gray.colors), 519
Grid, 674, 676, 680, 683, 693, 696, 697, 700, 701, 704, 706, 709, 712, 713, 714, 716, 718, 720
grid, 591, 591, 1054
Grid Viewports, 674
ggrid-package, 678
ggrid.add, 678
ggrid.arrows, 679
ggrid.circle, 682
ggrid.collection, 683
ggrid.convert, 666, 684
ggrid.convertHeight(grid.convert), 684
ggrid.convertWidth(grid.convert), 684
ggrid.convertX(grid.convert), 684
ggrid.convertY(grid.convert), 684
ggrid.copy, 686
ggrid.display.list, 686
ggrid.draw, 668, 687, 695
ggrid.edit, 669, 688, 695, 702, 703, 729
ggrid.frame, 689, 702, 703
ggrid.get, 690, 695
ggrid.grab, 691
ggrid.grabExpr(grid.grab), 691
ggrid.grill, 693
ggrid.grob, 683, 686, 693, 713
ggrid.layout, 674, 676, 695, 714, 725
ggrid.line.to, 680
ggrid.line.to(grid.move.to), 699
ggrid.lines, 680, 697
ggrid.locator, 698
ggrid.move.to, 699
ggrid.newpage, 700, 707
ggrid.pack, 690, 701, 703
ggrid.place, 702, 702
ggrid.plot.and.legend, 703
ggrid.points, 704
ggrid.polygon, 705
ggrid.pretty, 706
ggrid.prompt, 707
ggrid.record, 707
ggrid.rect, 708
ggrid.refresh, 710
ggrid.remove, 710
ggrid.segments, 680, 711
ggrid.set, 712
ggrid.show.layout, 676, 696, 713
ggrid.show.viewport, 714
ggrid.text, 715
ggrid.xaxis, 717, 720
ggrid.yaxis, 718, 719
grob, 670, 673, 679, 688, 689, 691, 711
grob(grid.grob), 693
grobHeight(grobWidth), 720
hist.Date, 81
hist.Date (hist.POSIXt), 595
hist.default, 595
hist.POSIXt, 595
history (savehistory), 1310
 HoltWinters, 947, 1054, 1078
hsearch-class (setOldClass), 792
 hsv, 507, 519, 521, 525, 529–531, 547, 548, 599, 619
Hyperbolic, 170
Hypergeometric, 949
I, 19, 78, 79, 428, 925, 1259
I(AsIs), 22
iconv, 41, 66, 171, 1216, 1279, 1289
iconvlist (iconv), 171
identical, 9, 10, 57, 172, 179
identify, 596, 607, 951
identify.hclust, 943, 950, 1106
if, 174, 208, 258
if (Control), 70
if-class (language-class), 764
ifelse, 71, 174
Im (complex), 58
image, 515, 516, 580, 587, 597, 625, 639, 944–946
index.search, 1275
Indometh, 456
Inf, 16, 135, 136, 247, 922
Inf (is.finite), 179
inert, 457, 938
influence, 952–954, 980, 1056, 1199
influence (lm.influence), 978
influence.measures, 951, 978, 979
inherits (class), 52
initialize, 746, 760, 801
initialize (new), 773
initialize, ANY-method
 (initialize-methods), 759
initialize, environment-method
 (initialize-methods), 759
initialize, signature-method
 (initialize-methods), 759
initialize,traceable-method
 (initialize-methods), 759
initialize-methods, 773
initialize-methods, 759
InsectSprays, 458
INSTALL, 197, 198, 758, 1276, 1278, 1288,
1293, 1303, 1314, 1323
install.packages, 197, 198, 253, 1220,
1288, 1290, 1303, 1313
install.packages
 (update.packages), 1320
installed.packages, 196, 198, 1219,
1277, 1290, 1322, 1323
installFoundDepends, 1219, 1219
Insurance, 1037
integer, 77, 97, 100, 125, 144, 175, 192,
241, 283, 424
integer-class (BasicClasses), 739
integrate, 954
INDEX

interegrate-class (setOldClass), 792
interaction, 176, 328, 329
interaction.plot, 634, 956
interactive, 177, 250
Internal, 177
InternalMethods, 18, 40, 46, 59, 100,
112, 118, 127, 176, 178, 179,
181–184, 192, 201, 210, 222, 233,
234, 245, 246, 413, 1181, 1223
interpSpline, 1138
intersect (sets), 332
intToBits (rawConversion), 290
intToUtf8 (utf8Conversion), 418
inverse.gaussian, 986
inverse.gaussian (family), 913
inverse.rle (rle), 311
invisible, 150, 178, 265, 529, 951
invokeRestart (conditions), 59
invokeRestartInteractively (conditions), 59
IQR, 922, 958, 994
iris, 458
iris (iris), 458
is, 419, 744, 760, 780
is.array, 178
is.array (array), 17
is.atomic, 178, 324
is.atomic (is.recursive), 183
is.call, 178
is.call (call), 40
is.character, 178
is.character (character), 46
is.complex, 178
is.complex (complex), 58
is.data.frame (as.data.frame), 18
is.double, 178
is.double (double), 99
is.element, 217
is.element (sets), 332
is.empty.model, 959
is.environment, 178
is.environment (environment), 111
is.expression (expression), 117
is.factor (factor), 126
is.finite, 179
is.function, 178, 180
is.infinite (is.finite), 179
is.integer, 178
is.integer (integer), 175
is.language, 40, 178, 181, 183, 234
is.leaf (dendrogram), 887
is.list, 178, 183, 419
is.list (list), 200
is.loaded, 157, 158
is.loaded (dyn.load), 105
is.logical, 178
is.logical (logical), 209
is.matrix, 178
is.matrix (matrix), 222
is.mts (ts), 1180
is.na, 127, 178, 868
is.na (NA), 233
is.na.POSIXlt (DateTimeClasses), 81
is.na <- (NA), 233
is.na <-.factor (factor), 126
is.name (name), 234
is.nan, 178, 234
is.nan (is.finite), 179
is.null, 178
is.null (NULL), 245
is.numeric, 178, 419
is.numeric (numeric), 245
is.object, 178, 182
is.ordered (factor), 126
is.package_version (package-version), 257
is.pairlist, 178
is.pairlist (list), 200
is.primitive (is.function), 180
is.qr (qr), 273
is.R, 182
is.real (real), 301
is.recursive, 118, 178, 183
is.single, 178, 184
is.stepfun (stepfun), 1153
is.symbol, 44, 178
is.symbol (name), 234
is.table (table), 389
is.ts (ts), 1180
is.tskernel (kernel), 963
is.unsorted (sort), 343
is.vector (vector), 418
isClass, 753, 754
isClass (setClass), 780
isClassUnion (setClassUnion), 784
isGeneric (GenericFunctions), 750
isGroup (GenericFunctions), 750
isIncomplete, 399
isIncomplete (connections), 63
islands, 460
ISOdate (strptime), 359
ISOdatetime (strptime), 359
isoMDS, 866, 960
INDEX

897, 905, 906, 913, 921, 924, 925, 938, 952, 959, 967, 973, 97, 976–980, 992, 993, 1017, 1055, 1080, 1095, 1123, 1124, 1151, 1164, 1165, 1175, 1176, 1199

lm-class (setOldClass), 792
lm.fit, 275, 974, 976
lm.influence, 952–954, 976, 978, 991, 992, 1056, 1199

lm.summaries, 979
lm.wfit, 976
lm.wfit (lm.fit), 976
lme, 825
load, 203, 320, 1248
loadedNamespaces, 326
loadedNamespaces (ns-load), 243
loadhistory (savehistory), 1310
loadings, 981, 1086
loadNamespace, 242, 417
loadNamespace (ns-load), 243
Lobololly, 463
local, 302, 355
local (eval), 113
localeconv, 188, 204
locales, 57, 143, 205, 304, 362
localeToCharset, 172, 346, 1279
locator, 597, 601, 606, 698
loess, 981, 984, 990, 1081, 1115, 1125, 1157
loess.control, 982, 983, 983
loess.smooth (scatter.smooth), 1114
log, 6, 206
log10 (log), 206
log1p (log), 206
log2 (log), 206
logb (log), 206
Logic, 208, 289, 377, 424
logical, 209, 209, 358, 423
logical-class (BasicClasses), 739
Logistic, 984
logLik, 814, 985
logLik-class (setOldClass), 792
logLik.gls, 986
logLik.lme, 986
loglin, 454, 611, 612, 987
Lognormal, 988
longley, 463
lower.tri, 94, 210
lowess, 557, 617, 983, 990, 1125
lqs, 1007
ls, 112, 211, 263, 307, 308, 1236, 1280
ls.diag, 991, 992, 993
ls.print, 991, 992, 993
ls.str, 1211, 1280, 1316
lsf.str (ls.str), 1280
lsfit, 275, 276, 991, 992, 992
lynx, 464
mad, 958, 994, 1116
mahalanobis, 995
make.link, 996, 1066
make.names, 79, 79, 212, 213, 294
make.packages.html
   (link.html.help), 1278
make.rgb, 509, 526
make.search.html
   (link.html.help), 1278
make.socket, 41, 1246, 1281, 1300
make.unique, 122, 212, 213, 213, 1288
makeARIMA (KalmanLike), 961
makeClassRepresentation, 766, 783
makeLazyLoading, 1220
makepredictcall, 996
manglePackageName, 214
manova, 997, 1166
mantelhaen.test, 998
maov-class (setOldClass), 792
mapply, 214, 393
margin.table, 215, 272, 811
mat.or.vec, 216
match, 48, 165, 216, 261, 424
match.arg, 217, 218, 219, 220, 261, 899
match.call, 218, 219, 261, 756
match.fun, 218, 219, 220, 261
Math, 6, 74, 171, 207, 313, 337, 347, 407, 1223
Math (groupGeneric), 166
Math.data.frame, 79
Math.Date (Dates), 81
Math.diff (Math), 81
Math.diff (Math), 95
Math.POSIXlt (DateTimeClasses), 81
Math.POSIXt (DateTimeClasses), 81
Math2 (groupGeneric), 166
matlines (matplot), 607
matmult, 221
matplot, 459, 607
matpoints (matplot), 607
matrix, 18, 80, 94, 98, 120, 210, 221, 222, 241, 583, 609
matrix-class (StructureClasses), 800
mauchley.test, 1001, 1143
max, 287, 424, 425, 827
max (Extremes), 124
INDEX

na.omit (na.fail), 1017
na.omit.ts, 1017
na.omit.ts (ts-methods), 1182
na.pass (na.fail), 1017
name, 181, 195, 234, 1253
name-class (language-class), 764
names, 18, 28, 93, 98, 125, 190, 213, 235, 316, 414, 1102
names<- (names), 235
NaI, 16, 136, 233, 234, 247, 499, 922, 1102
NaN (is.finite), 179
napredict, 911, 921, 1017, 1086
napredict (naresid), 1018
naprint, 1018
naresid, 1017, 1018
nargs, 236
nchar, 141, 237, 260, 363, 370, 401, 657
nclass, 528
nclass.FD, 594
nclass.scott, 594
nclass.Sturges, 594
NCOL, 316
NCOL (nrow), 240
ncol, 97
ncol (nrow), 240
NegBinomial, 1019
new, 744, 745, 759, 773, 781, 793
new.env, 748
new.env (environment), 111
new.packages (update.packages), 1320
next (Control), 70
NextMethod, 53
NextMethod (UseMethod), 415
nextn, 874, 917, 1021
ngetext, 1231
ngettext (gettext), 160
nhtemp, 466
Nile, 467, 1187
nlevels, 127, 193, 238
nlm, 695, 1022, 1041, 1044, 1192
nlminb, 1024
nls, 158, 926, 934, 1023, 1026, 1029–1031, 1059, 1083, 1091, 1093, 1094, 1119, 1119–1142, 1144–1149
nls.control, 1029
nlsModel, 1027, 1030, 1093, 1094
NLSstAsymptotic, 1031
NLSstClosestX, 1032, 1033, 1034, 1131
NLSstLfAsymptote, 1032, 1033, 1131
NLSstRtAsymptote, 1032, 1033, 1033, 1034, 1131
noquote, 239, 266, 269, 1170
Normal, 1034
normalizePath, 1286
nottem, 468
NotYet, 240
NotYetImplemented (NotYet), 240
NotYetUsed (NotYet), 240
NROW, 316
NROW (nrow), 240
nrow, 97, 240
ns, 997
ns-dblcolon, 241
ns-hooks, 242
ns-load, 243
ns-topenv, 244
NULL, 241, 245, 359, 548, 944, 945, 1313
NULL-class (BasicClasses), 739
numeric, 100, 245, 287
numeric-class (BasicClasses), 739
NumericConstants, 70, 246, 377
numericDeriv, 1036
object.size, 226, 227, 1286
objects, 27, 92, 198, 308, 326, 1234
objects (is), 211
ObjectsWithPackage-class, 774
octmode, 247
offset, 974, 1006, 1036, 1178
old-style S graphics, 678
old.packages (update.packages), 1320
oldClass, 168
oldClass (class), 52
oldClass-class (setOldClass), 792
oldClass<- (class), 52
on.exit, 248, 380, 1281
oneway.test, 1037
open (connections), 63
Ops, 16, 57, 96, 208
Ops (groupGeneric), 166
Ops.Date (Dates), 81
Ops.difftime (difftime), 95
Ops.package_version (package-version), 257
Ops.POSIXt (DateTimeClasses), 81
Ops.ts (ts), 1180
optim, 833, 834, 838, 839, 870, 895, 910, 948, 1023, 1038, 1158, 1159
optimise (optimize), 1043
optimize, 1023, 1041, 1043, 1192
INDEX

plot.dendrogram (dendrogram), 887
plot.density, 892, 1052
plot.design, 632
plot.ecdf (ecdf), 902
plot.factor, 634, 636, 638
plot.function (curve), 583
plot.hclust, 888
plot.histogram, 592, 593, 636
plot.HoltWinters, 1053
plot.isoreg, 960, 1054
plot.lm, 249, 1055, 1175
plot.mlml (plot.lm), 1055
plot.new, 417, 621, 638
plot.new (frame), 590
plot.POSIXct (axis.POSIXct), 565
plot.POSIXlt (axis.POSIXct), 565
plot.ppr, 1057, 1071
plot.prcomp (prcomp), 1072
plot.princomp (princomp), 1085
plot.profile.nls, 1058, 1091
plot.spec, 252, 1059, 1131, 1134, 1136
plot.stepfun, 902, 903, 1060, 1154
plot.stl, 1156, 1157
plot.stl (stlmethods), 1157
plot.table, 637
plot.ts, 972, 1062, 1158, 1181, 1183
plot.tskernel (kernel), 963
plot.TukeyHSD (TukeyHSD), 1188
plot.window, 569, 579, 585, 587, 590, 598, 631, 632, 638
plot.xy, 608, 639, 639, 641
plotmath, 522, 534, 535, 602, 614, 661, 662
plotViewport, 667, 720
pmatch, 45, 48, 165, 217–219, 261
pmax (Extremes), 124
pmin (Extremes), 124
pnbinom (NegBinomial), 1019
png, 41, 515, 538
pnorm, 1188
pnorm (Normal), 1034
points, 557, 583, 592, 602, 606, 608, 609, 617, 618, 621, 625, 629, 631, 640, 640, 888, 889, 1054, 1055, 1175
points.default, 639
points.formula (plot.formula), 635
pointsGrob (grid.points), 704
Poisson, 1063
poisson (family), 913
poly, 997
polygon, 503, 627, 642, 645, 649, 888
polygonGrob (grid.polygon), 705
polyroot, 262, 1192
pop.viewport, 721, 722
popViewport, 676, 730
popViewport (Working with Viewports), 731
pos.to.env, 263
POSIXct, 19, 143
POSIXct (DateTimeClasses), 81
POSIXct-class (setOldClass), 792
POSIXlt, 19, 143
POSIXlt (DateTimeClasses), 81
POSIXlt-class (setOldClass), 792
POSIxt, 94, 167
POSIxt (DateTimeClasses), 81
POSIxt-class (setOldClass), 792
possibleExtends, 736
postDrawDetails (drawDetails), 668
postscript, 250, 512–515, 532–534, 539, 545, 554, 556, 623, 641
postscriptFonts (postscriptFonts), 544
postscriptFonts, 532, 543, 544
power, 914, 1066
power.prop.test, 1088
power.t.test, 1065, 1066, 1088
PP.test, 1067
ppoints, 1068, 1099
p pois (Poisson), 1063
ppr, 1007, 1058, 1069, 1169
prcomp, 853, 1072, 1086, 1087
precip, 471
predict, 116, 907, 976, 1019, 1074, 1080, 1083
predict.ar, 1075
predict.ar (ar), 828
predict.Arima, 835, 1075, 1075
predict.arima0, 1075
predict.arima0 (arima0), 837
predict.glm, 938, 1075, 1076, 1175
predict.HoltWinters, 948, 1054, 1075, 1078
predict.lm, 975, 976, 1075, 1079
predict.loess, 983, 1075, 1080
predict.mlm (predict.lm), 1079
predict.nls, 1075, 1082
predict.poly, 1075
predict.prcomp (prcomp), 1072
predict.princomp, 1075
predict.princomp (princomp), 1085
predict.smooth.spline, 1075, 1083, 1128
predict.StructTS, 1075
predict.StructTS (StructTS), 1158
preDrawDetails (drawDetails), 668
preplot, 1084
presidents, 472
pressure, 473
pretty, 263, 516, 564, 567
prettyNum (formatC), 146
Primitive, 265
princomp, 853, 854, 911, 981, 1074, 1085, 1116, 1167
print, 42, 56, 142, 239, 250, 265, 267, 269, 270, 816, 825, 889, 902, 943, 963, 1004, 1027, 1086, 1088, 1089, 1154, 1176, 1308, 1315
print.anova, 1089
print.anova (anova), 816
print.aov (aov), 824
print.aovlist (aovlist), 824
print.ar (ar), 828
print.arima0 (arima0), 837
print.AsIs (AsIs), 22
print.Bibtext (toLatex), 1319
print.by (by), 38
print.checkDocFiles (QC), 1222
print.checkDocStyle (QC), 1222
print.checkFF (checkFF), 1210
print.checkReplaceFuns (QC), 1222
print.checkS3methods (QC), 1222
print.checkTnF (checkTnF), 1211
print.checkVignettes (checkVignettes), 1212
print.codoc (codoc), 1213
print.codocClasses (codoc), 1213
print.codocData (codoc), 1213
print.condition (conditions), 59
print.connection (connections), 63
print.data.frame, 79, 267
print.Date (Dates), 81
print.default, 36, 110, 141, 250, 266, 267, 267, 270, 888
print.dendrogram (dendrogram), 887
print.density (density), 890
print.difftime (difftime), 95
print.dist (dist), 898
print.DLLInfo (getLoadedDLLs), 156
print.DLLInfoList
   (getLoadedDLLs), 156
print.DLLRegisteredRoutines
   (getDLLRegisteredRoutines), 155
print.dummy.coef (dummy.coef), 901
print.ecdf (ecdf), 902
print.factanal (loadings), 981
print.family (family), 913
print.formula (formula), 924
print.ftable (ftable), 928
print.getAnywhere (getAnywhere), 1263
print.glm (glm), 935
print.hclust (hclust), 941
print.hsearch (help.search), 1272
print.infl (influence.measures), 951
print.integrate (integrate), 954
print.kmeans (kmeans), 965
print.Latex (toLatex), 1319
print.libraryIQR (library), 195
print.lm (lm), 973
print.loadings, 911
print.loadings (loadings), 981
print.logLik (logLik), 985
print.ls_str (ls.str), 1280
print.MethodsFunction (methods), 1284
print.NativeRoutineList
   (getDLLRegisteredRoutines), 155
print.noquote (noquote), 239
print.octmode (octmode), 247
print.package_version
   (package-version), 257
print.packageDescription
   (packageDescription), 1288
print.packageInfo (library), 195
print.packageIQR (data), 1247
print.packageStatus
   (packageStatus), 1289
print.POSIXct (DateTimeClasses), 81
print.POSIXlt (DateTimeClasses), 81
print.power.htest, 1087
print.prcomp (prcomp), 1072
print.princomp (princomp), 1085
print.recordedplot (recordPlot), 546
print.restart (conditions), 59
INDEX

rainbow, 507, 519, 525, 529, 547, 599, 623
rainbow (Palettes), 530
Random, 282
Random, user, 283, 285
randu, 475
range, 125, 287, 516, 583, 922, 958
rank, 255, 288, 344
raw, 289
raw-class (BasicClasses), 739
rawConversion, 290
rawShift, 290
rawShift (rawConversion), 290
rawToBits (rawConversion), 290
rawToChar, 289
rawToChar (rawConversion), 290
rbeta (Beta), 847
rbind, 743
rbind (cbind), 43
rbind2 (cbind2), 742
rbind2, ANY, ANY-method (cbind2), 742
rbind2-methods (cbind2), 742
rbinom, 1015
rbinom (Binomial), 850
rcauchy (Cauchy), 859
rchisq (Chisquare), 862
Rconsole, 398, 552
Rconsole (Rwin configuration), 1309
Rd2dvi (RdUtils), 291
Rd2txt (RdUtils), 291
Rd_db (Rdutils), 1224
Rd_parse (Rdutils), 1224
Rdconv, 776
Rdconv (RdUtils), 291
Rdevga (Rwin configuration), 1309
Rdindex, 1224
RdUtils, 291
Rdutils, 1224
Re (complex), 58
read.00Index, 1226
read.csv, 1298
read.csv2 (read.table), 292
read.csv2 (read.table), 292
read.dcf, 1231, 1289, 1323
read.dcf (dcf), 84
read.delim (read.table), 292
read.delim2 (read.table), 292
read.fortran, 1297
read.ftable, 930, 1103
read.fwf, 295, 1298, 1298
read.socket, 1246, 1282, 1300
read.table, 72, 79, 292, 325, 410, 429, 1248, 1298, 1299
readBin, 3, 67, 295, 298, 299, 301
readChar, 296, 298
readCitationFile (citEntry), 1244
readClipboard (clipboard), 1244
readline, 299, 1295
readLines, 66, 67, 273, 297, 299, 300, 300, 324, 325, 430
real, 301
Recall, 40, 302
recordedplot-class (setOldClass), 792
recordGraphics, 545, 708
recordGrob (grid.record), 707
recordPlot, 546, 553
recover, 85, 250, 402–404, 1252, 1301
rect, 571, 637, 643, 644, 650
rect.hclust, 943, 951, 1105
rsignrank (SignRank), 1122
RSiteSearch, 1273, 1274, 1305
rstandard (influence.measures), 951
rstudent, 980
rstudent (influence.measures), 951
rt (TDist), 1173
Rtangle, 1306, 1308, 1318, 1319
RtangleSetup (Rtangle), 1306
rug, 185, 645, 1175
runif, 284, 1035
runif (Uniform), 1190
runmed, 1112, 1130
RweaveLatex, 1307, 1307, 1318, 1319
RweaveLatexSetup (RweaveLatex), 1307
rweibull (Weibull), 1197
rwilcox (Wilcoxon), 1203
Rwin configuration, 1309
S3Methods, 1284
S3Methods (UseMethod), 415
SafePrediction, 1077, 1080
SafePrediction (makepredictcall), 996
sammon, 866
sample, 318
sapply, 214, 215, 393
sapply (lapply), 189
save, 26, 103, 104, 203, 251, 319, 427, 1249, 1252
savehistory, 1310
savePlot, 550, 554
scale, 321, 375, 997, 1073
scan, 72, 259, 273, 279, 293–295, 301, 322, 346, 427, 1299
scatter, smooth, 1114
SClassExtension-class, 745, 779
screen, 646
screenplot, 1086, 1087, 1115
sd, 878, 1116, 1173
Sd2Rd (RdUtils), 291
se.contrast, 1011, 1117
se.contrast.acovlist, 904, 905
sealClass (setClass), 780
SealedMethodDefinition-class (MethodDefinition-class), 767
search, 20, 23, 26, 27, 63, 91, 92, 115, 153, 195, 197, 198, 211, 307, 325, 501, 758, 1234, 1280
searchpaths (search), 325
Seatbelts (UKDriverDeaths), 488
seek, 67, 326
seekViewport, 676, 730
seekViewport (Working with Viewports), 731
segments, 560, 561, 643, 645, 648, 888, 889, 1061
segmentsGrobal (grid.segments), 711
select.list, 1283, 1311, 1321
selectMethod, 741, 747, 796, 797
selectMethod (getMethod), 754
selfStart.default, 934
selfStart.formula, 934
seq, 309, 311, 327, 329, 330, 332
seq.Date, 77, 81, 329
seq.POSIXt, 77, 83, 329, 330, 596
sequence, 309, 329, 331
sessionInfo, 1312, 1320
set.seed (Random), 282
setAs, 737, 761
setAs (as), 735
setCConverterStatus
(getNumCConverters), 158
setChildren (grid.add), 678
setClass, 737, 744, 745, 753, 754, 761, 764, 766, 767, 771, 778, 779, 780, 794, 802, 803, 1277
setClassUnion, 747, 783, 784, 794
setdiff (sets), 332
setequal (sets), 332
setGeneric, 749, 753, 769, 771, 786, 790
setGrobal, 673
setGrobal (grid.set), 712
setGroupGeneric, 749
setGroupGeneric (setGeneric), 786
setHook, 196, 243, 590, 625, 700
setHook (UserHooks), 416
setIs, 737, 744, 745, 754, 769, 779, 780
setIs(is), 760
setMethod, 404, 735, 749, 760, 763, 764, 767, 782, 789, 793, 794, 797, 1277
setName, 865, 1120
setOldClass, 783, 790, 792
setName (setName), 757
setReplaceMethod
(GenericFunctions), 750
setRepositories, 253, 1242, 1312
sets, 332
setValidity (validObject), 801
setwd, 382

INDEX
INDEX

setwd(getwd), 162
setWindowTitle, 1313
shapiro.test, 969, 1121
shell, 333, 334, 336, 386
shell.exec, 333, 334, 334, 386
SHLIB, 107, 200, 1314
show, 250, 268, 794, 796
show, ANY-method (show), 794
show, classRepresentation-method (show), 794
show, genericFunction-method (show), 794
show, MethodDefinition-method (show), 794
show, MethodWithNext-method (show), 794
show, ObjectsWithPackage-method (show), 794
show, traceable-method (show), 794
show-methods (show), 794
showClass, 795
showConnections, 67, 335, 400
showDefault, 794
showMethods, 753, 769, 795, 796, 1284
showMlist, 795
shQuote, 279, 336, 353
sign, 337
signalCondition, 357
signalCondition (conditions), 59
signature (GenericFunctions), 750
signature-class, 797
signif, 140, 146, 268, 372
signif (Round), 312
SignRank, 1122
simint, 1189
simpleCondition (conditions), 59
simpleError (conditions), 59
simpleMessage (conditions), 59
simpleWarning (conditions), 59
simulate, 1123
sin, 6, 171
sin (Trig), 407
single, 137
single (double), 99
single-class (BasicClasses), 739
sinh (Hyperbolic), 170
sink, 42, 335, 338, 1240
sleep, 477
slice.index, 339
slot, 340, 744, 798
slot<- (slot), 798
slotNames (slot), 798
slotOp, 340
smooth, 1114, 1124
smooth.spline, 1071, 1084, 1125, 1126, 1138
smoothEnds, 1113, 1114, 1129
socket-class (setOldClass), 792
socketConnection (connections), 63
socketSelect, 341
solve, 31, 51, 274, 341, 995
solve.qr, 342
solve.qr (qr), 273
sort, 205, 255, 288, 310, 311, 343
sort.list (order), 254
sortedXyData, 1032–1034, 1130
source, 102, 103, 177, 259, 345, 383, 1212,
1213, 1248, 1254, 1317
spec (spectrum), 1135
spec.ar, 1131, 1135, 1136
spec.pgram, 1132, 1135, 1136
spec.taper, 1134, 1134
Special, 6, 17, 346
spectrum, 963, 1060, 1132–1134, 1135
spineplot, 577, 578, 634, 635, 649
spline, 827
spline (splinefun), 1137
splinefun, 584, 827, 903, 1061, 1137, 1154
split, 75, 348
split.screen, 600, 620, 623
split.screen (screen), 646
split<- (split), 348
sprintf, 142, 147, 161, 260, 350, 532, 541, 556
sqrt, 17, 207, 347
sqr (abs), 6
sQuote, 279, 337, 353, 369
SSasymp, 1032, 1138, 1148, 1149
SSasympOff, 1139
SSasympOrig, 1140
SSbiexp, 1141
SSD, 1002, 1142
SSFol, 484, 1143
SSFpl, 1144
SSgompertz, 1145
SSlogis, 1146
SSmicmen, 1147
SSweibull, 1148
stack, 354, 1111
stack.loss (stackloss), 477
stack.x (stackloss), 477
stackloss, 477
standardGeneric, 749, 751
Stangle, 1212, 1306
Stangle (Sweave), 1316
stars, 652
start, 1149, 1179, 1181, 1186
Startup, 56, 250, 355
stat.anova, 817, 1150
state, 478, 492
stats (stats-package), 1151
stats-package, 1151
stayOnTop (bringToTop), 501
stderr (showConnections), 335
stdin, 66, 273
stdout (showConnections), 335
stream, 594, 637, 654
step, 810, 1151
stepAIC, 1152, 1153
stepfun, 903, 960, 1060, 1153
stopifnot, 1157
stop, 161, 230, 250, 357, 358, 359, 421, 1231
stopifnot, 8, 358, 358
storage.mode, 410
storage.mode (mode), 231
storage.mode <- (mode), 231
str, 211, 889, 1236, 1280, 1315
str.default, 889
str.dendrogram (dendrogram), 887
str.logLik (logLik), 985
str.POSIXt (DateTimeClasses), 81
strftime, 143
strptime (strptime), 359
strheight, 553
stringHeight (strwidth), 656
stringHeight (stringWidth), 724
stripchart, 573, 630, 655
strptime, 21, 22, 83, 205, 206, 359, 565, 595
strsplit, 47, 238, 260, 303, 307, 363, 370
strtrim, 364, 370
StructTS, 962, 1012, 1158, 1185, 1187
structure, 365
structure-class
 (StructureClasses), 800
StructureClasses, 800
strwidth, 238, 553, 602, 619, 656
strwrap, 365
substr, 47, 48, 363, 1267
substring (grep), 163
Subscript (Extract), 118
subset, 123, 366, 407
substitute, 36, 87, 88, 98, 231, 368, 403,
404, 537, 602
substr, 5, 47, 238, 260, 363, 364, 369
substr <- (substr), 369
substring (substr), 369
substring <- (substr), 369
sum, 54, 271, 371
Summary, 8, 11, 125, 271, 287, 371
Summary (groupGeneric), 166
summary, 371, 817, 825, 936, 938, 979, 1027,
1161, 1163, 1165, 1167, 1280, 1315, 1316
summary.aov, 825, 1160
summary.aovlist (summary.aov), 1160
summary.connection (connections), 63
Summary.Date (Dates), 81
summary.Date (Dates), 81
Summary.difftime (difftime), 95
summary.glm, 372, 936, 938, 940, 1162
summary.infl
 (influence.measures), 951
summary.lm, 372, 976, 979, 980, 991, 1164
summary.manova, 821, 998, 1165
summary.mlm (summary.lm), 1164
Summary.package_version
 (package-version), 257
summary.packageStatus
 (packageStatus), 1289
Summary.POSIXct
 (DateTimeClasses), 81
summary.POSIXct
 (DateTimeClasses), 81
Summary.POSIXlt
 (DateTimeClasses), 81
summary.POSIXlt
 (DateTimeClasses), 81
summary.prcomp (prcomp), 1072
summary.princomp, 1087, 1167
summary.stepfun (stepfun), 1153
summary.table (table), 389
summary.table-class
 (setOldClass), 792
summaryRprof, 1304
sunflowerplot, 658
sunspot.month, 480, 481
sunspot.year, 480, 481, 481
sunspots, 480, 481
suppressMessages (message), 230
suppressWarnings (warning), 420
supsmu, 1071, 1125, 1168
INDEX

survreg, 21, 143
survdiff, 1175
svd, 50, 109, 186, 275, 373, 858, 1074, 1086
Sweave, 253, 1209, 1212, 1307, 1308, 1316, 1318
SweaveSyntaxLatex (Sweave), 1316
SweaveSyntaxNoweb (Sweave), 1316
SweaveSyntaxConv, 1318
sweep, 14, 220, 322, 374, 877
swiss, 482
switch, 71, 375
symbol.C(dyn.load), 105
symbol.For(dyn.load), 105
symnum, 1162, 1164, 1169
Syntax, 17, 58, 71, 120, 209, 247, 258, 279, 376
sys.call, 236
sys.call(sys.parent), 379
sys.calls(sys.parent), 379
Sys.Date, 81
Sys.Date(Sys.time), 384
sys.frame, 23, 113–115, 153, 211, 307
sys.frame(sys.parent), 379
sys.frames(sys.parent), 379
sys.function(sys.parent), 379
Sys.getenv, 377, 381, 382
Sys.getlocale, 188, 378, 1279
Sys.getlocale(locales), 205
Sys.getpid(getpid), 160
Sys.info, 4, 378
sys.load.image(save), 319
Sys.localeconv, 205, 206
Sys.localeconv(localeconv), 204
sys.nframe(sys.parent), 379
sys.on.exit, 248
sys.on.exit(sys.parent), 379
sys.parent, 379, 1252
sys.parents(sys.parent), 379
Sys.putenv, 378, 381, 1310
sys.save.image(save), 319
Sys.setlocale, 204
Sys.setlocale(locales), 205
Sys.sleep, 382
sys.source, 26, 244, 251, 383
sys.status(sys.parent), 379
Sys.time, 80, 83, 384
Sys.timezone(Sys.time), 384
system, 4, 183, 333, 334, 336, 385
system.file, 386
system.time, 270, 271, 387, 1179
T(logical), 209
t, 12, 388, 1181
t.test, 967, 1038, 1049, 1066, 1171, 1202
t.ts(ts), 1180
table, 75, 266, 389, 391, 392, 637, 638, 811,
929–931, 988, 1206
table-class(setOldClass), 792
tabulate, 75, 391
tail(head), 1268
tan, 171
tan(Trig), 407
tanh(Hyperbolic), 170
tapply, 14, 38, 190, 317, 392, 813
taskCallback, 393
taskCallbackManager, 393, 394, 395, 397
taskCallbackNames, 397
TDist, 1173
tempdir(tempfile), 398
tempfile, 398
termpath, 1056, 1174
terms, 633, 885, 925, 937, 975, 1007, 1009,
1176, 1177, 1178, 1193
terms.formula, 1176, 1177, 1178
terms.object, 1176, 1177, 1178
terrain.colors, 506, 529, 597–599
terrain.colors(Palettes), 530
texi2dvi, 1209, 1226
text, 117, 522, 524, 526, 535, 537, 542, 579,
580, 603, 613, 614, 617, 618, 657, 660, 662, 1062, 1063
textConnection, 67, 399, 1240
textGrob(grid.text), 715
Theoph, 483
tilde, 400
time, 388, 556, 1150, 1179, 1181, 1186, 1205
Titanic, 484
title, 537, 569, 575, 579, 582, 585, 587,
608, 614, 617, 624, 629, 633, 661, 661,
661, 941, 1054, 1055
tk_select.list, 1311
toBibtex(toLatex), 1319
toBibtex.citation(citation), 1242
toBibtex.citationList(citation), 1242
toBibtex.person(person), 1291
toBibtex.personList(person), 1291
topeplitz, 1180
toLatex, 1319
toLatex.sessionInfo
(sessionInfo), 1312
tolower, 165
tolower(chartr), 48
tools(tools-package), 1227
tools-deprecated, 1227
tools-package, 1227
ToothGrowth, 485
topenv, 251, 383

topenv (ns-topenv), 244
topicName (help), 1269
topo.colors, 505, 507, 597–599
topo.colors (Palettes), 530
toString, 141, 142, 401
toupper, 165
toupper (chartr), 48
trace, 85, 402, 759, 800, 801
traceable-class, 759
traceable-class (TraceClasses), 800
traceback, 37, 85, 358, 405
TraceClasses, 800
tracingState (trace), 402
trans3d, 551, 625
transform, 367, 406
treering, 486
trees, 487
Trig, 207, 407
trigamma (Special), 346
TRUE, 209, 358, 1063
TRUE (logical), 209
truhist, 594
trunc, 175
trunc (Round), 312
trunc.Date (round.POSIXt), 313
trunc.POSIXt, 83
trunc.POSIXt (round.POSIXt), 313
truncate (seek), 326
try, 128, 129, 196, 251, 358, 406, 408, 417
tryCatch, 406
tryCatch (conditions), 59
ts, 94, 167, 252, 1012, 1063, 1088, 1150, 1179, 1180, 1182, 1186, 1205
ts-class (StructureClasses), 800
ts-methods, 1182
ts.intersect, 975
ts.intersect (ts.union), 1184
ts.plot, 1183
ts.union, 1184
tsdia, 835, 840, 1185
tsp, 17, 209, 1149, 1150, 1179, 1181, 1186, 1205
tsp<- (tsp), 1186
tsSmooth, 961, 962, 1160, 1186
Tukey, 1187
TukeyHSD, 825, 1011, 1161, 1188
type, 100, 176, 246
type (typeof), 410
type.convert, 294, 295, 409
typeof, 16, 153, 181, 228, 232, 234, 410
UCBAAdmissions, 487
ucv, 846
UKDriverDeaths, 488
UKgas, 490
UKLungDeaths, 490
unclass, 127
unclass (class), 52
undec (debug), 85
undoc, 1214, 1228
Uniform, 1190
union (sets), 332
unique, 105, 411
unique.POSIXlt (DateTimeClasses), 81
unisroot, 262, 1023, 1044, 1066, 1191
unit, 666, 674, 676, 684, 685, 698, 720, 724, 724, 727
unit.c, 726, 726
unit.length, 726, 727
unit.pmax (unit.pmin), 727
unit.pmin, 727
unit.rep, 728
units, 663
unix (system), 385
unix.time (system.time), 387
unlink, 133, 398, 412
unlist, 39, 141, 178, 190, 413
unloadNamespace, 242
unloadNamespace (ns-load), 243
unloadNamespace (ns-load), 243
unname, 223, 414
unspli (split), 348
unstack (stack), 354
untrace, 800
untrace (trace), 402
unz, 1230
unz (connections), 63
update, 1192
update.formula, 1152, 1193, 1193
update.packages, 253, 1222, 1277, 1278, 1290, 1320
update.packageStatus
(packages), 1289
upgrade (packageStatus), 1289
upper.tri, 94
upper.tri (lower.tri), 210
upViewport, 676, 730
upViewport (Working with Viewports), 731
url, 41, 203, 1255, 1256, 1323
url(connections), 63
url.show, 1256, 1323
URIDecode, 65
URIDecode(URLEncode), 1324
URLEncode, 1324
USAccDeaths, 491
USArrests, 491
UseMethod, 44, 53, 415
UserHooks, 416
USJudgeRatings, 492
USPersonalExpenditure, 493
uspop, 493
utf8Conversion, 418
utf8ToInt(utf8Conversion), 418
utils(utils-package), 1325
utils-deprecated, 1325
utils-package, 1325
VADeaths, 494
validDetails, 729
validObject, 745, 766, 781, 801
var, 882, 994, 995, 1116
var(cor), 876
var.test, 824, 847, 923, 1014, 1194
variable.names, 316
variable.names
   (case/variable.names), 858
varimax, 911, 1195
vcov, 834, 869, 976, 1027, 1196
vector, 43, 105, 144, 201, 411, 418
vector-class(BasicClasses), 739
version(R.Version), 280
vi, 1251
vi(edit), 1257
viewport, 667, 674, 680, 683, 693,
    696–698, 700, 704, 706, 709, 712,
    714–716, 718, 720, 721, 723, 730, 732
viewport(Grid Viewports), 674
vignette, 1326
vignette Depends, 1229
vignettes, 678
VIRTUAL-class(BasicClasses), 739
volcano, 495
vplist(Grid Viewports), 674
vpPath, 730, 732
vpStack(Grid Viewports), 674
vpTree(Grid Viewports), 674
warning, 16, 43, 161, 208, 230, 249, 253,
    358, 359, 420, 421, 1231
warnings, 251, 420, 421, 421
warpbreaks, 495
weekdays, 81, 422
weekdays.POSIXt, 83
Weibull, 1197
weighted.mean, 225, 1198
weighted.residuals, 980, 1199
weights, 1199
weights(lm.summaries), 979
weights.glm(glm), 935
which, 423, 425
which.is.max, 425
which.max, 223
which.max(which.min), 424
which.min, 125, 424, 424
while(Control), 70
while-class(language-class), 764
width.SJ, 846
widthDetails, 665, 731
wilcox.exact, 1202
wilcox.test, 967, 1050, 1123, 1200, 1203,
    1204
Wilcoxon, 1203
win.graph(windows), 551
win.metafile(windows), 551
win.print(windows), 551
win.version(winextras), 1328
winDialog, 1327, 1329
winDialogString(winDialog), 1327
window, 1179, 1181, 1204
window<- (window), 1204
windows, 252, 501, 515, 550, 551, 555, 1309
windowsFont(windowsFonts), 554
windowsFonts, 554
winextras, 1328
winMenuAdd, 1327
winMenuAdd(winMenus), 1328
winMenuAddItem(winMenus), 1328
winMenuDel1(winMenus), 1328
winMenuDelItem(winMenus), 1328
winMenuItems(winMenus), 1328
winMenuNames(winMenus), 1328
winMenus, 1328
with, 27, 425, 1240
withCallingHandlers(conditions), 59
withRestarts(conditions), 59
women, 496
Working with Viewports, 731
WorldPhones, 497
write, 101, 104, 266, 325, 426, 429
write.csv(write.table), 427
write.csv2(write.table), 427
write.dcf, 1231
write.dcf (dcf), 84
write.ftable (read.ftable), 1103
write.matrix, 429
write.socket (read.socket), 1300
write.table, 85, 295, 427, 427
write.table0 (base-deprecated), 31
write_PACKAGES, 1230
writeBin, 67, 298, 430
writeBin (readBin), 295
writeChar, 430
writeChar (readChar), 298
writeClipboard (clipboard), 1246
writeLines, 67, 297, 299, 301, 429
wsbrowser (browseEnv), 1235
WWWusage, 498
X11 (windows), 551
x11, 623
x11 (windows), 551
xaxisGrob (grid.xaxis), 717
xedit (edit), 1257
xemacs (edit), 1257
xfig, 515, 555, 641
xgettext, 161, 1231
xgettext2pot (xgettext), 1231
xinch (units), 663
xngettext (xgettext), 1231
xor (Logic), 208
xtabs, 390, 496, 930, 1104, 1206
xy.coords, 502, 503, 556, 558, 596, 601,
603, 606, 631, 632, 639, 641, 643,
658, 660, 826, 959, 1115, 1137
xyinch (units), 663
xyz.coords, 557
yaxisGrob (grid.yaxis), 719
yinch (units), 663
zapsmall, 1089
zapsmall (Round), 312
zip.file.extract, 430
zip.unpack (winextras), 1328
zpackages, 431
zutils, 432