The R Reference Index

The R Core Team

Version 1.0.0 (February 29, 2000)

Copyright (©) 1999 R Development Core Team.

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under the terms of the GNU General Public License. For more information about these matters, see http://www.gnu.org/copyleft/gpl.html.

Contents

1 The base package **1**

.Alias . . . 1
abbreviate . . . 2
abline . . . 3
abs . . . 4
add1 . . . 4
aggregate . . . 6
airmiles . . . 7
airquality . . . 8
alias . . . 9
all . . . 10
all.equal . . . 11
all.names . . . 12
anova . . . 12
anscombe . . . 13
any . . . 14
aov . . . 15
aperm . . . 16
append . . . 17
apply . . . 17
approxfun . . . 19
apropos . . . 20
args . . . 21
Arithmetic . . . 22
array . . . 23
arrows . . . 24
as.char.or.expr . . . 25
as.function . . . 25
assign . . . 26
attach . . . 27
attenu . . . 28
attitude . . . 29
attr . . . 30
attributes . . . 31
autoload . . . 31
ave . . . 32
axis . . . 33
backsolve . . . 34
barplot . . . 35
BATCH . . . 37
Bessel . . . 37

Beta . . . 39
Binomial . . . 40
body . . . 41
box . . . 42
boxplot . . . 43
boxplot.stats . . . 45
browser . . . 46
bug.report . . . 46
builtins . . . 49
bxp . . . 49
by . . . 50
C . . . 51
c . . . 52
call . . . 52
cars . . . 53
case/variable.names . . . 54
cat . . . 54
Cauchy . . . 55
cbind . . . 56
char.expand . . . 57
character . . . 58
charmatch . . . 58
check.options . . . 59
chickwts . . . 60
Chisquare . . . 61
chol . . . 62
chol2inv . . . 63
chull . . . 63
class . . . 64
close.socket . . . 65
co2 . . . 66
codes . . . 66
coefficients . . . 67
col . . . 68
colors . . . 69
commandArgs . . . 69
comment . . . 70
Comparison . . . 71
COMPILE . . . 71
complete.cases . . . 72
complex . . . 72
conflicts . . . 73
Constants . . . 74
contour . . . 75
contrast . . . 77
contrasts . . . 78
contributors . . . 79
Control . . . 80
convolve . . . 81
coplot . . . 82
copyright . . . 84
cor . . . 84

count.fields . . . 85
cov.wt . . . 86
crossprod . . . 87
cumsum . . . 87
curve . . . 88
cut . . . 89
data . . . 91
data.class . . . 92
data.frame . . . 93
data.matrix . . . 94
dataentry . . . 95
date . . . 95
debug . . . 96
debugger . . . 97
Defunct . . . 98
delay . . . 99
delete.response . . . 100
demo . . . 101
density . . . 101
deparse . . . 104
Deprecated . . . 105
deriv . . . 105
detach . . . 107
dev.xxx . . . 107
dev2 . . . 108
dev2bitmap . . . 110
deviance . . . 111
Devices . . . 112
df.residual . . . 112
diag . . . 113
diff . . . 113
dim . . . 114
dimnames . . . 115
discoveries . . . 115
do.call . . . 116
dotplot . . . 117
double . . . 118
download.file . . . 118
dput . . . 119
drop . . . 120
dummy.coef . . . 120
dump . . . 122
duplicated . . . 122
dyn.load . . . 123
edit . . . 125
edit.data.frame . . . 126
eff.aovlist . . . 127
effects . . . 128
eigen . . . 129
environment . . . 130
esoph . . . 131
euro . . . 133

eurodist . . . 134
eval . . . 134
example . . . 136
exists . . . 137
expand.grid . . . 138
Exponential . . . 138
expression . . . 139
Extract . . . 140
extractAIC . . . 141
Extremes . . . 142
factor . . . 143
factor.scope . . . 145
faithful . . . 146
family . . . 147
FDist . . . 148
fft . . . 150
file . . . 151
file.choose . . . 152
file.path . . . 153
file.show . . . 153
filled.contour . . . 154
fitted.values . . . 156
fivenum . . . 156
fix . . . 157
Foreign . . . 158
Formaldehyde . . . 161
formals . . . 162
format . . . 162
format.info . . . 163
formatC . . . 164
formula . . . 166
frame . . . 167
freeny . . . 168
ftable . . . 169
ftable.formula . . . 170
function . . . 171
GammaDist . . . 172
gc . . . 173
gctorture . . . 174
Geometric . . . 174
get . . . 175
getenv . . . 176
getwd . . . 177
gl . . . 177
glm . . . 178
glm.summaries . . . 180
Gnome . . . 181
gray . . . 182
grep . . . 183
grid . . . 184
HairEyeColor . . . 185
help . . . 186

help.search . . . 188
help.start . . . 189
Hershey . . . 190
hist . . . 199
hsv . . . 201
httpclient . . . 202
Hyperbolic . . . 202
Hypergeometric . . . 203
identify . . . 204
ifelse . . . 205
image . . . 206
index.search . . . 207
infert . . . 208
influence.measures . . . 209
InsectSprays . . . 210
INSTALL . . . 212
integer . . . 213
interaction . . . 213
interactive . . . 214
Internal . . . 214
invisible . . . 215
IQR . . . 215
iris . . . 216
is.empty.model . . . 217
is.finite . . . 217
is.function . . . 220
is.language . . . 220
is.object . . . 221
is.R . . . 221
is.recursive . . . 222
is.single . . . 222
is.symbol . . . 223
islands . . . 223
Japanese . . . 224
jitter . . . 237
kappa . . . 238
kronecker . . . 239
labels . . . 240
lapply . . . 241
Last.value . . . 241
layout . . . 242
legend . . . 244
length . . . 246
levels . . . 247
levels.factor . . . 247
library . . . 248
library.dynam . . . 250
license . . . 251
LifeCycleSavings . . . 256
lines . . . 257
list . . . 257
list.files . . . 259

lm . . . 260
lm.influence . . . 261
lm.summaries . . . 263
load . . . 264
locator . . . 265
log . . . 266
Logic . . . 267
logical . . . 267
Logistic . . . 268
loglin . . . 269
Lognormal . . . 271
longley . . . 272
lower.tri . . . 272
lowess . . . 273
ls . . . 274
ls.diag . . . 275
ls.print . . . 276
lsfit . . . 276
Machine . . . 277
machine . . . 279
mad . . . 280
mahalanobis . . . 281
make.link . . . 282
make.names . . . 282
make.socket . . . 283
make.tables . . . 284
margin.table . . . 285
mat.or.vec . . . 285
match . . . 286
match.arg . . . 286
match.call . . . 287
match.fun . . . 288
matmult . . . 289
matplot . . . 290
matrix . . . 292
mean . . . 293
median . . . 293
Memory . . . 294
memory.profile . . . 295
menu . . . 296
merge . . . 296
Methods . . . 297
methods . . . 299
missing . . . 300
mode . . . 301
model.extract . . . 302
model.frame . . . 303
model.matrix . . . 304
model.tables . . . 305
mosaicplot . . . 306
mtcars . . . 308
mtext . . . 308

NA . . . 310
na.action . . . 311
na.fail . . . 311
name . . . 312
names . . . 312
nargs . . . 313
nchar . . . 313
NegBinomial . . . 314
nextn . . . 315
nhtemp . . . 316
nlevels . . . 316
nlm . . . 317
noquote . . . 318
Normal . . . 319
NotYet . . . 321
nrow . . . 321
NULL . . . 322
numeric . . . 322
offset . . . 323
on.exit . . . 323
optim . . . 324
optimize . . . 327
options . . . 328
OrchardSprays . . . 331
order . . . 332
outer . . . 333
p.adjust . . . 334
package.contents . . . 335
package.dependencies . . . 335
page . . . 336
pairs . . . 336
pairs.formula . . . 338
palette . . . 339
Palettes . . . 340
panel.smooth . . . 341
par . . . 342
Paren . . . 347
parse . . . 348
parse.dcf . . . 348
paste . . . 349
persp . . . 350
phones . . . 352
pictex . . . 353
piechart . . . 354
PlantGrowth . . . 355
Platform . . . 356
plot . . . 357
plot.default . . . 358
plot.factor . . . 360
plot.formula . . . 360
plot.lm . . . 361
plot.window . . . 363

plot.xy . . . 364
plotmath . . . 365
pmatch . . . 371
points . . . 372
Poisson . . . 373
poly . . . 374
polygon . . . 375
polyroot . . . 376
pos.to.env . . . 377
postscript . . . 377
power . . . 379
ppoints . . . 380
precip . . . 380
predict . . . 381
predict.glm . . . 382
predict.lm . . . 383
preplot . . . 384
presidents . . . 385
pressure . . . 386
pretty . . . 386
Primitive . . . 388
print . . . 388
print.coefmat . . . 389
print.default . . . 390
print.matrix . . . 391
proc.time . . . 392
prod . . . 393
profile . . . 393
proj . . . 394
prompt . . . 395
prop.table . . . 396
qqnorm . . . 397
qr . . . 398
QR.Auxiliaries . . . 399
quakes . . . 401
quantile . . . 401
quit . . . 402
R.home . . . 403
R.Version . . . 403
Random . . . 404
Random.user . . . 407
randu . . . 408
range . . . 409
rank . . . 410
read.fwf . . . 410
read.socket . . . 411
read.table . . . 412
read.table.url . . . 414
readline . . . 415
real . . . 415
Recall . . . 416
rect . . . 416

relevel . . . 417
REMOVE . . . 418
remove . . . 418
rep . . . 419
replace . . . 419
replications . . . 420
residuals . . . 421
restart . . . 421
rev . . . 422
rgb . . . 423
RHOME . . . 423
rivers . . . 423
rle . . . 424
Round . . . 425
row . . . 426
row/colnames . . . 426
rowsum . . . 427
rug . . . 428
sample . . . 429
save . . . 430
scale . . . 430
scan . . . 431
screen . . . 433
sd . . . 435
se.aov . . . 435
se.contrast . . . 436
search . . . 437
segments . . . 437
seq . . . 438
sequence . . . 439
sets . . . 440
SHLIB . . . 441
sign . . . 441
SignRank . . . 442
sink . . . 443
sleep . . . 443
solve . . . 444
sort . . . 445
source . . . 445
Special . . . 446
splinefun . . . 447
split . . . 448
stackloss . . . 449
stars . . . 450
start . . . 452
Startup . . . 452
stat.anova . . . 453
state . . . 454
stem . . . 455
step . . . 455
stop . . . 457
str . . . 458

stripplot . . . 459
strsplit . . . 460
structure . . . 461
strwidth . . . 461
subset . . . 462
substitute . . . 463
substr . . . 465
sum . . . 466
summary . . . 466
sunflowerplot . . . 467
sunspots . . . 469
svd . . . 469
sweep . . . 470
swiss . . . 471
switch . . . 472
symnum . . . 473
sys.parent . . . 475
sys.source . . . 476
system . . . 477
system.file . . . 478
system.time . . . 478
t . . . 479
table . . . 480
tabulate . . . 481
tapply . . . 481
TDist . . . 483
tempfile . . . 484
terms . . . 485
text . . . 486
time . . . 487
Titanic . . . 488
title . . . 489
ToothGrowth . . . 490
trace . . . 491
traceback . . . 491
transform . . . 492
trees . . . 493
Trig . . . 493
ts . . . 494
tsp . . . 496
Tukey . . . 496
typeof . . . 498
UCBAdmissions . . . 498
undoc . . . 499
Uniform . . . 500
unique . . . 501
uniroot . . . 501
units . . . 502
unlink . . . 503
unlist . . . 504
unname . . . 504
update . . . 505

update.formula . . . 506
update.packages . . . 507
USArrests . . . 508
USJudgeRatings . . . 509
USPersonalExpenditure . . . 510
uspop . . . 510
VADeaths . . . 511
var . . . 512
vector . . . 513
volcano . . . 514
warning . . . 514
warnings . . . 515
warpbreaks . . . 516
Weibull . . . 516
weighted.mean . . . 517
weighted.residuals . . . 518
which . . . 519
Wilcoxon . . . 520
window . . . 521
women . . . 522
write . . . 523
write.table . . . 524
x11 . . . 524
xy.coords . . . 525
zip.file.extract . . . 526

2 The `ctest` package 529

ansari.test . . . 529
bartlett.test . . . 530
binom.test . . . 531
chisq.test . . . 533
cor.test . . . 534
fisher.test . . . 536
fligner.test . . . 538
friedman.test . . . 539
kruskal.test . . . 541
ks.test . . . 542
mantelhaen.test . . . 543
mcnemar.test . . . 544
mood.test . . . 545
pairwise.prop.test . . . 547
pairwise.t.test . . . 547
pairwise.table . . . 548
pairwise.wilcox.test . . . 549
power.prop.test . . . 549
power.t.test . . . 550
print.pairwise.htest . . . 551
print.power.htest . . . 552
prop.test . . . 553
shapiro.test . . . 554
t.test . . . 555
var.test . . . 557
wilcox.test . . . 558

3 The `eda` package **561**
line . . . 561
medpolish . . . 562
smooth . . . 563

4 The `lqs` package **565**
cov.rob . . . 565
lqs . . . 567
predict.lqs . . . 569

5 The `modreg` package **571**
ksmooth . . . 571
loess . . . 572
loess.control . . . 574
modreg-internal . . . 575
plot.ppr . . . 575
ppr . . . 576
predict.loess . . . 579
predict.smooth.spline . . . 580
rock . . . 581
scatter.smooth . . . 582
smooth.spline . . . 583
supsmu . . . 584

6 The `mva` package **587**
as.hclust . . . 587
biplot . . . 588
biplot.princomp . . . 589
cancor . . . 590
cmdscale . . . 591
cutree . . . 592
dist . . . 593
hclust . . . 594
identify.hclust . . . 596
kmeans . . . 598
prcomp . . . 599
princomp . . . 600
rect.hclust . . . 602

7 The `nls` package **603**
asOneSidedFormula . . . 603
BOD . . . 604
ChickWeight . . . 604
clearNames . . . 605
CO2 . . . 606
DNase . . . 607
formula.nls . . . 608
getInitial . . . 608
Indometh . . . 609
Loblolly . . . 610
logLik . . . 611
nls . . . 611
nls.control . . . 613

nlsModel . . . 614
NLSstAsymptotic . . . 615
NLSstClosestX . . . 616
NLSstLfAsymptote . . . 617
NLSstRtAsymptote . . . 617
Orange . . . 618
plot.profile.nls . . . 619
predict.nls . . . 620
profile.nls . . . 621
profiler . . . 622
profiler.nls . . . 623
Puromycin . . . 625
selfStart . . . 626
selfStart.default . . . 627
selfStart.formula . . . 628
setNames . . . 629
sortedXyData . . . 630
SSasymp . . . 631
SSasympOff . . . 632
SSasympOrig . . . 633
SSbiexp . . . 634
SSfol . . . 635
SSfpl . . . 636
SSlogis . . . 637
SSmicmen . . . 638

8 The `splines` package 639
asVector . . . 639
backSpline . . . 640
bs . . . 641
interpSpline . . . 642
ns . . . 643
periodicSpline . . . 644
polySpline . . . 645
predict.bs . . . 646
predict.bSpline . . . 646
splineDesign . . . 648
splineKnots . . . 649
splineOrder . . . 649
xyVector . . . 650

9 The `stepfun` package 651
ecdf . . . 651
plot.stepfun . . . 652
stepfun . . . 654

10 The `ts` package 657
acf . . . 657
ar . . . 659
ar.ols . . . 662
arima0 . . . 664
austres . . . 666
beavers . . . 667

BJsales . . . 668
Box.test . . . 668
cpgram . . . 669
diff.ts . . . 670
diffinv . . . 670
embed . . . 671
EuStockMarkets . . . 672
filter . . . 672
kernapply . . . 674
kernel . . . 674
lag . . . 676
LakeHuron . . . 677
lh . . . 677
lynx . . . 677
na.omit.ts . . . 678
nottem . . . 678
PP.test . . . 679
spec.ar . . . 680
spec.pgram . . . 681
spec.taper . . . 683
spectrum . . . 684
stl . . . 686
sunspot . . . 687
toeplitz . . . 688
treering . . . 689
ts-internal . . . 689
ts.plot . . . 690
ts.union . . . 691
UKDriverDeaths . . . 692
UKLungDeaths . . . 692
USAccDeaths . . . 692

Index **693**

Chapter 1

The base package

.Alias *Create Alias (Pointer) to R Object* .Alias

Description

.Alias creates an **alias** to another (part of) an R object which is more (memory-) efficient than usual assignment.

Usage

```
new <- .Alias(expr)
```

Arguments

expr	an R expression; typically a name.
new	new name by which expr can be accessed.

Value

an identical copy of expr.

Warning

This has a **dangerous** semantic, and consequences can be unexpected (it can be used to defeat the call-by-value illusion). Know what you are doing *before* using .Alias!

See Also

<- for usual assignments.

Examples

```
mop <- options()
mop$browser <- "a browser"   # not set on all platforms
Op <- .Alias(mop)
## A change to mop is reflected in Op and vice versa
## -- ONLY if no new slots are created ...
mop$digits <- "Wow!"
```

```
Op$browser <- "another one"
mop$browser; Op$digits
all(names(mop) == names(Op) &
    sapply(seq(mop), function(i) all(Op[[i]] == mop[[i]])))
##> TRUE -- Op and mop ARE the same thing !

mop$newslot <- pi #--->> 'newslot' ==> (shallow) COPY of 'mop'
Op$newslot # R: still the old one, i.e. NULL
all(names(mop) == names(Op))# no longer TRUE
```

`abbreviate` *Abbreviate Strings* `abbreviate`

Description

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

Usage

```
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.

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.

See Also

substr.

Examples

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

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

`abline` *Add a Straight Line to a Plot* `abline`

Description

This function adds one or more straight lines through the current plot.

Usage

```
abline(a, b, ...)
abline(h=, ...)
abline(v=, ...)
abline(untf=, ...)
abline(coef=, ...)
abline(reg=, ...)
```

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.

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 alway refer to original coordinates.

The graphical parameters `col` and `lty` can be specified as arguments to `abline`; see `par` for details.

See Also

`lines` and `segments` for connected and arbitrary lines given by their *endpoints*. `par`.

Examples

```
data(cars)
z <- lm(dist ~ speed, data = cars)
plot(cars)
abline(z)
```

---

abs *Miscellaneous Mathematical Functions* abs

Description

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

Usage

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

See Also

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

Examples

```
xx <- -9:9
plot(xx, sqrt(abs(xx)),  col = "red")
lines(spline(xx, sqrt(abs(xx)), n=101), col = "pink")
```

---

add1 *Add or Drop All Possible Single Terms to a Model* add1

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, ...)
add1.default(object, scope, scale = 0, test = c("none", "Chisq"),
             k = 2, trace = FALSE, ...)
add1.lm(object, scope, scale = 0, test = c("none", "Chisq", "F"),
        x = NULL, k = 2, ...)
add1.glm(object, scope, scale = 0, x = NULL, test = c("none", "Chisq"),
         k = 2, ...)

drop1(object, scope, ...)
drop1.default(object, scope, scale = 0, test = c("none", "Chisq"),
              k = 2, trace = FALSE, ...)
```

```
drop1.lm(object, scope, scale = 0, all.cols = TRUE,
         test=c("none", "Chisq", "F"),k = 2, ...)
drop1.glm(object, scope, scale = 0, test = c("none", "Chisq"),
          k = 2, ...)
```

Arguments

`object`	a fitted models 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 `NULL`.
`test`	should the results include a test statistic relative to the original model? The F test is only appropriate for `lm` and `aov` models. The χ^2 test can be an exact test (`lm` models with known scale) or a likelihood-ratio test 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 `add1` is to be called repeatedly.
`all.cols`	(Provided for compatibility with S.) Logical to specify whether all columns of the design matrix should be used. If `FALSE` then non-estimable columns are dropped, but the result is not usually statistically meaningful.

Details

For `drop` methods, a missing `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.

The methods for `lm` and `glm` are more efficient in that they do not recompute the model matrix and call the `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`.

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. This may be a problem if there are missing values and R's default of `na.action = na.omit` is used, although it is not for methods for `"lm"` and `"glm"`.

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)

B. D. Ripley

See Also

step, aov, lm, extractAIC.

Examples

```
example(step)#-> swiss
(alm1 <- add1(lm1, ~ I(Education^2) + .^2))
```

---

aggregate *Compute Summary Statistics of Data Subsets* aggregate

---

Description

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

Usage

```
aggregate(x, ...)
aggregate.default(x, ...)
aggregate.data.frame(x, by, FUN, ...)
aggregate.ts(x, nfrequency = 1, FUN = sum, ndeltat = 1)
```

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.
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.
...	further arguments passed to the method used.

Details

aggregate is a generic functions 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. The result returned is a time series with frequency `nfrequency` holding the aggregated values.

Author(s)

Kurt Hornik

See Also

`apply`, `lapply`, `tapply`.

Examples

```
data(state)

## 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.)

data(presidents)
## Compute the average annual approval ratings for American presidents.
aggregate(presidents, nf = 1, FUN = mean)
```

`airmiles` *Commercial Airline Mileage* `airmiles`

Description

The revenue passenger miles flown by commercial airlines in the United States for each year from 1937 to 1960.

Usage

```
data(airmiles)
```

Format

A time-series of 24 observations; yearly, 1937–1960.

Source

F.A.A. Statistical Handbook of Aviation.

References

Brown, R. G. (1963) *Smoothing, Forecasting and Prediction of Discrete Time Series.* Prentice-Hall.

Examples

```
data(airmiles)
plot(airmiles, main = "airmiles data",
     xlab = "Passenger-miles flown by U.S. commercial airlines", col = 4)
```

`airquality` *New York Air Quality Measurements* `airquality`

Description

Daily air quality measurements in New York, May to September 1973.

Usage

```
data(airquality)
```

Format

A data frame with 154 observations on 6 variables.

`[,1]`	`Ozone`	numeric	Ozone (ppb)
`[,2]`	`Solar.R`	numeric	Solar R (lang)
`[,3]`	`Wind`	numeric	Wind (mph)
`[,4]`	`Temp`	numeric	Temperature (degrees F)
`[,5]`	`Month`	numeric	Month (1–12)
`[,6]`	`Day`	numeric	Day of month (1–31)

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

Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P. A. (1983) *Graphical Methods for Data Analysis.* Belmont, CA: Wadsworth.

Examples

```
data(airquality)
pairs(airquality, panel = panel.smooth, main = "airquality data")
```

`alias` *Find Aliases (Dependencies) in a Model* `alias`

Description

Find aliases (linearly dependent terms) in a linear model specified by a formula.

Usage

```
alias(object, ...)
alias.formula(object, data, ...)
alias.lm(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.

Details

Although the main method is for class `"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)

B.D. Ripley

Examples

```
## From Venables and Ripley (1997) p.210.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
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(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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)

## The next line is optional (for fractions package which gives neater
## results.)
has.VR <- require(MASS, quietly = TRUE)

op <- options(contrasts=c("contr.helmert", "contr.poly"))
npk.aov <- aov(yield ~ block + N*P*K, npk)
alias(npk.aov)
if(has.VR) detach(package:MASS)
options(op)# reset
```

`all` *Are All Values True?* `all`

Description

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

Usage

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

Arguments

`...`	one or more logical vectors.
`na.rm`	logical. If true `NA` values are removed before the result is computed.

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 `x` consists of a mix of `TRUE` and `NA` values, then value is `NA`.

Examples

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

`all.equal` *Test if Two Objects are (Nearly) Equal* `all.equal`

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.

Usage

```
all.equal(target, current, ...)

all.equal.numeric(target, current,
                  tolerance= .Machine$double.eps ^ 0.5, scale=NULL)
```

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 comparison is done using averge (`mean`) *relative* error, unless `scale` is used or when the absolute difference is less than `tolerance`. For complex arguments, `Mod` of difference is used.

`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

See Also

`==`, and `all` for exact equality testing.

Examples

```
all.equal(pi, 355/113) # not precise enough (default tol) > relative error

all.equal(gamma(2:14),   cumprod(1:13)) # TRUE, but
all      (gamma(2:14) == cumprod(1:13)) # FALSE, since not exactly
all.equal(gamma(2:14),   cumprod(1:13), tol=0) # to see difference

all.equal(options(), .Options)
.Options $ myopt <- TRUE
all.equal(options(), as.list(.Options))
rm(.Options)
```

all.names *Find All Names in an Expression* all.names

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.

Examples

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

anova *Anova Tables* anova

Description

Compute analysis of variance (or deviance) tables for one or more fitted model objects.

Usage

```
anova(object, ...)
print(anova.object)
```

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 specified.

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.

See Also

`coefficients`, `effects`, `fitted.values`, `residuals`, `summary`.

`anscombe` *Anscombe's Quartet of "Identical" Simple Linear Regressions* `anscombe`

Description

Four *x-y* datasets which have the same traditional statistical properties (mean, variance, correlation, regression line, etc.), yet are quite different.

Usage

```
data(anscombe)
```

Format

A data frame with 11 observations on 8 variables.

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

Tufte, Edward R. (1989) *The Visual Display of Quantitative Information*, 13–14. Graphics Press.

References

Anscombe, Francis J. (1973) Graphs in statistical analysis. *American Statistician*, **27**, 17–21.

Examples

```
data(anscombe)
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)
```

`any` *Are Values True?* `any`

Description

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

Usage

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

Arguments

`...`	one or more logical vectors.
`na.rm`	logical. If true `NA` values are removed before the result is computed.

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 `x` consists of a mix of `FALSE` and `NA` values, the value is `NA`.

Examples

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

`aov` *Fit an Analysis of Variance Model* `aov`

Description

Fit an analysis of variance model by a call to `lm` for each stratum.

Usage

```
aov(formula, data = NULL, projections = FALSE, contrasts = NULL, ...)
se.aov(object, n, type = "means")
```

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?
`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 `lm` for fitting linear models to balanced or unbalanced experimental designs.

The main difference from `lm` is in the way `print`, `summary` and so on handle the fit: this is expressed in the traditional language of the analysis of variance rather than of linear models.

If the formula contains a single `Error` term, this is used to specify error strata, and appropriate models are fitted within each error stratum.

The formula can specify multiple responses.

Value

An object of class `c("aov", "lm")` or for multiple responses of class `c("maov", "aov", "mlm", "lm")` or for multiple error strata of class `"aovlist"`. There are `print` and `summary` methods available for these.

Author(s)

B. D. Ripley

See Also

lm, alias, proj, model.tables

Examples

```
## From Venables and Ripley (1997) p.210.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
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(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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)

( npk.aov <- aov(yield ~ block + N*P*K, npk) )
summary(npk.aov)
coefficients(npk.aov)

## as a test, not particularly sensible statistically
op <- options(contrasts=c("contr.helmert", "contr.treatment"))
npk.aovE <- aov(yield ~  N*P*K + Error(block), npk)
npk.aovE
summary(npk.aovE)
options(op)# reset to previous
```

`aperm` *Array Transposition* `aperm`

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

See Also

t.

Examples

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

`append` *Vector Merging* `append`

Description

Add elements to a vector.

Usage

```
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`.

Examples

```
all(append(1:5, 0:1, after=3) ==
    append(1:3, c(0:1, 4:5)))
```

`apply` *Apply Functions Over Array Margins* `apply`

Description

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

Usage

```
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`.

Value

If each call to `FUN` returns a vector of length `n`, then `apply` returns an array of dimension `c(n,dim(x)[MARGIN])` if `n > 1`. If `n` equals `1`, `apply` returns a vector if `MARGIN` has length 1 and an array of dimension `dim(x)[MARGIN]` otherwise.

If the calls to `FUN` return vectors of different lengths, `apply` returns a list of length `dim(x)[MARGIN]`.

See Also

`lapply`, `tapply`, and convenience functions `sweep` and `aggregate`.

Examples

```
## 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)))

all( apply(x,2, is.vector)) # TRUE [was not 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=c("x1","x2"))

ma <- matrix(c(1:4, 1, 6:8), nr = 2)
ma
apply(ma, 1, table)  #--> a list of length 2
apply(ma, 1, quantile)# 5 x n matrix with rownames

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

approxfun *Interpolation Functions* approxfun

Description

Return a list of points which linearly interpolate given data points, or a function performing the linear (or constant) interpolation.

Usage

```
approx   (x, y, xout, method="linear", n=50,
          yleft, yright, rule=1, f=0)
approxfun(x, y,       method="linear",
          yleft, yright, rule=1, f=0)
```

Arguments

x,y	vectors giving the coordinates of the points to be interpolated. Alternatively a single plotting structure can be specified.
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 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 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.
f	For method="constant" a number between 0 and 1 inclusive, indicating a compromise between left- and right-continuous step functions. If y0 and y1 are the values to the left and right of the point then the value is y0*f+y1*(1-f) so that f=0 is right-continuous and f=1 is left-continuous.

Details

The inputs can contain missing values which are deleted, so at least two complete (x, y) pairs are required.

Value

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.

See Also

spline and splinefun for spline interpolation.

Examples

```
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"))  # T
curve(fc(x), 0, 10, col = "darkblue", add = TRUE)
```

---

apropos *Find Objects by (Partial) Name* apropos

Description

apropos returns a vector of character strings 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

```
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 only word.

Details

apropos returns a vector of character strings giving the names of all objects in the search list matching what. If mode != "any" only those objects which are of mode mode are considered. If where is TRUE, their position in the search list is returned as their names attribute.

find is a different user interface to the same task as apropos. However, by default (simple.words == TRUE), only full words are searched.

Author(s)

Kurt Hornik and Martin Maechler (May 1997).

See Also

`objects` for listing objects from one place, `search` for the search path.

Examples

```
apropos("lm")
apropos(ls)
apropos("lq")

lm <- 1:pi
find(lm)        ##> ".GlobalEnv"   "package:base"
find(lm, num=T) ## .. numbers with these names
find(lm, num=T, mode="function")# only the second one
rm(lm)

apropos(".", mode="list")

# need a DOUBLE backslash '\\' (in case you don't see it anymore)
apropos("\\[")

# 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,}$")
table(nchar(apropos("^.{8,}$")))
```

`args` *Argument List of a Function* `args`

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.

See Also

`formals`, `help`.

Examples

```
args(c)            # -> NULL (c is a 'primitive' function)
args(plot.default)
```

`Arithmetic` *Arithmetic Operators* `Arithmetic`

Description

These binary operators perform arithmetic on vector objects.

Usage

```
x + y
x - y
x * y
x / y
x ^ y
x %% y
x %/% y
```

Details

`1 ^ y` and `y ^ 0` are `1`, *always*. `x ^ y` should also give the proper "limit" result when either argument is infinite (i.e., `+-` `Inf`).

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

Value

They return numeric 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 * ( x %/% y )` unless `y == 0` where the result is `NA` or `NaN` (depending on the `typeof` of the arguments).

See Also

`Math` for miscellaneous and `Special` for special mathematical functions.

Examples

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

`array` *Multi-way Arrays* `array`

Description

Creates or tests for arrays.

Usage

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

Value

`array` returns an array with the extents specified in `dim` and naming information in `dimnames`. The values in `x` are taken to be those in the array with the leftmost subscript moving fastest. If there are too few elements in `x` to fill the array, then the elements in `x` are recycled.

`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.

See Also

`aperm`, `matrix`.

Examples

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

`arrows` *Add Arrows to a Plot* `arrows`

Description

Draw arrows between pairs of points.

Usage

```
arrows(x0, y0, x1, y1, length = 0.25, angle = 30, code = 2,
       col = par("fg"), lty = NULL, xpd = FALSE)
```

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, xpd`	usual graphical parameters as in `par`.

Details

For each `i`, an arrow is drawn between the point `(x0[i], y0[i])` and the point `(x1[i],y1[i])`.

If `code=2` an arrowhead is drawn at `(x0[i],y0[i])` and if `code=1` an arrowhead is drawn at `(x1[i],y1[i])`. If `code=3` a head is drawn at both ends of the arrow.

The graphical parameters `col` and `lty` can be used to specify a color and line texture for the line segments which make up the arrows (`col` may be a vector).

See Also

`segments` to draw segments.

Examples

```
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')
```

`as.char.or.expr` *Coerce to Character or Expression* `as.char.or.expr`

Description

Coerce expression and call objects to expressions, and all other objects to character.

Usage

```
as.char.or.expr(x)
```

Arguments

`x`	an arbitrary R object.

Details

This function is used for handling the `text` arguments to the text-drawing functions. These arguments can be character strings or expressions (which are interpreted as mathematical expressions and typeset according to TeX-like rules, see `plotmath` for more details). Users will typically have no need for it.

Examples

```
as.char.or.expr("This is a string.")
as.char.or.expr(call("round", 10.5))
```

`as.function` *Convert Object to Function* `as.function`

Description

`as.function` is a generic function which is used to convert objects to functions.

`as.function.default` works on a list `x`, which should contain the concatenation of a formal argument list and an expression or an object of mode `"call"` which will become the function body. The function will be defined in a specified environment, by default that of the caller.

Usage

```
as.function(x, ...)

as.function.default(l, envir = sys.frame(sys.parent()))
```

Arguments

`x`	object to convert
`...`	additional arguments, depending on object
`l`	a list
`envir`	environment in which the function should be defined

Value

The desired function.

Author(s)

Peter Dalgaard

See Also

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

Examples

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

`assign` *Assign a Value to a Name* `assign`

Description

Assign a value to a name in an environment.

Usage

```
assign(x, value, pos = -1, envir = pos.to.env(pos),
       inherits = FALSE, immediate = TRUE)
x <- value
x <<- value
value -> x
value ->> x
```

Arguments

`x`	a variable name (given as a quoted string).
`value`	a value to be assigned to `x`.
`pos`	an index into the search list which determines which environment the assignment is to take place in. A character string may also be used. The environment can also be specified directly with `envir`.
`envir`	the `environment` in which to assign. The default is the environment where the call to `assign` takes place.
`inherits`	should the enclosing frames of the environment be inspected?
`immediate`	an ignored compatibility feature.

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 global environment.

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

The arrow forms of assignment provide shortcut ways to carry out assignment. The `<-` and `->` forms carry out assignment in the local environment frame, while the `<<-` and `->>` forms 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 the action of `<<-` and `->>` differs from that in the S language, but is useful in conjunction with the scoping rules of R.

See Also

`get`, `exists`, `environment`.

Examples

```
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
```

`attach` *Attach Set of R Objects to Search Path* `attach`

Description

A new database is attached. This means that its objects are made available to R.

—— should say more here ——

Usage

```
attach(what, pos = 2, name = deparse(substitute(what)))
```

Arguments

what	"database" to attach. Typically a data.frame or list.
pos	integer specifying position in search() where to attach.
name	alternative way to specify the database to be attached.

Value

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

See Also

library, detach, search, objects, environment.

Examples

```
data(women)
attach(women)
summary(height)   #- which belongs to 'women'
detach("women")
```

---

attenu *The Joyner-Boore Attenuation Data* attenu

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

```
data(attenu)
```

Format

A dataframe with 182 observations on 5 variables.

[,1]	event	numeric	Event Number
[,2]	mag	numeric	Moment Magnitude
[,3]	station	factor	Station Number
[,4]	dist	numeric	Station-hypocenter distance (km)
[,5]	accel	numeric	Peak acceleration (g)

Source

Joyner, W.B., D.M. Boore and R.D. Porcella (1981). Peak horizontal acceleration and velocity from strong-motion records including records from the 1979 Imperial Valley, California earthquake. USGS Open File report 81-365. Menlo Park, Ca.

References

Boore, D. M. and Joyner, W.B.(1982) The empirical prediction of ground motion, *Bull. Seism. Soc. Am.*, **72**, S269–S268.

Bolt, B. A. and Abrahamson, N. A. (1982) New attenuation relations for peak and expected accelerations of strong ground motion, *Bull. Seism. Soc. Am.*, **72**, 2307–2321.

Bolt B. A. and Abrahamson, N. A. (1983) Reply to W. B. Joyner & D. M. Boore's "Comments on: New attenuation relations for peak and expected accelerations for peak and expected accelerations of strong ground motion", *Bull. Seism. Soc. Am.*, **73**, 1481–1483.

Brillinger, D. R. and Preisler, H. K. (1984) An exploratory analysis of the Joyner-Boore attenuation data, *Bull. Seism. Soc. Am.* **74**, 1441–1449.

Brillinger, D. R. and Preisler, H. K. (1984) *Further analysis of the Joyner-Boore attenuation data*. Manuscript.

Examples

```
data(attenu)
## 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)
```

`attitude` *Attitudes Toward Supervisors* `attitude`

Description

Not available (yet).

Usage

```
data(attitude)
```

Format

A data from with 30 observations on 7 variables.

[,1]	rating	numeric	Overall rating
[,2]	complaints	numeric	Handling of employee complaints
[,3]	privileges	numeric	Gives special privileges
[,4]	learning	numeric	Opportunity to learn
[,5]	raises	numeric	Gives raises
[,6]	critical	numeric	Too critical
[,7]	advancel	numeric	Advancement

Source

Chatterjee, S. and Price, B. (1977) *Regression Analysis by Example.* New York: Wiley.

Examples

```
data(attitude)
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)
```

---

attr *Object Attributes* attr

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

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.

See Also

attributes

Examples

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

`attributes` *Object Attribute Lists* `attributes`

Description

These functions access an object's attribute list. The first form above returns the an object's attribute list. The assignment forms make the list on the right-hand side of the assignment the object's attribute list (if appropriate).

Usage

```
attributes(obj)
attributes(obj) <- list
mostattributes(obj) <- list
```

Details

The `mostattributes` assignment takes special care for the `dim`, `names` and `dimnames` attributes, and assigns them only when that is valid whereas as `attributes` assignment would give an error in that case.

See Also

`attr`.

Examples

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

## strip an objects 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* `autoload`

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.

Usage

```
autoload(name, file)
autoloader(name, file)
.AutoloadEnv
```

Arguments

name	string giving the name of an object
file	string giving the name of a package containing the object

Value

This function is invoked for its side-effect.

See Also

delay, library

Examples

```
autoload("line","eda")
search()
ls("Autoloads")
all(ls("Autoloads") == ls(envir = .AutoloadEnv))
data(cars)
plot(cars)
z<-line(cars)
abline(coef(z))
search()
detach("package:eda")
search()
z<-line(cars)
search()
```

---

ave *Group Averages Over Level Combinations of Factors* ave

Description

Subsets of x[] are averaged, where each subset consist of those observations with the same factor levels.

Usage

```
ave(x, ..., FUN = 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 length(x). If ... is g1,g2, e.g., y[i] is equal to FUN(x[j], for all j with g1[j]==g1[i] and g2[j]==g2[i]).

See Also

mean, median.

Examples

```
data(warpbreaks)
attach(warpbreaks)
ave(breaks, wool)
ave(breaks, tension)
ave(breaks, tension, FUN = function(x)mean(x, trim=.1))
plot(breaks, main =
     "ave( Warpbreaks )  for   wool  x  tension  combinations")
lines(ave(breaks, wool, tension           ), type='s', col = "blue")
lines(ave(breaks, wool, tension, FUN=median), type='s', col = "green")
legend(40,70, c("mean","median"), lty=1,col=c("blue","green"), bg="gray90")
detach()
```

`axis` *Add an Axis to a Plot* `axis`

Description

Adds an axis to the current plot, allowing the specification of the side, position, labels, and other options.

Usage

```
axis(side, at, labels = TRUE, ...)
```

Arguments

`side`	an integer specifying which side of the plot the axis is to be drawn on.
`at`	the points at which tick-marks are to be drawn. Non-finite (infinite, `NaN` or `NA`) values are omitted.
`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 to be placed at the tickpoints.
`...`	graphical parameters may also be passed as arguments to this function.

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.

Value

This function is invoked for its side effect, which is to add an axis to an already existing plot. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.

Examples

```
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), type = 's', xaxt='n', col = 'red')
axis(1, 1:7, LETTERS[1:7], col.axis = 'blue')
```

---

backsolve *Solve an Upper or Lower Triangular System* backsolve

---

Description

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

Usage

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

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

Dongarra, J. J., Bunch,J. R., Moler, C. B. and Stewart, G. W. (1978) *LINPACK Users Guide.* Philadelphia: SIAM Publications.

See Also

chol, qr, solve.

Examples

```
## 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)
( y2 <- backsolve(r, x, transpose = TRUE)) # 8 -12 -5
all(t(r) %*% y2 == x)# exactly on Linux (Pentium)
all(y  == backsolve(t(r), x, upper = FALSE, transpose = TRUE))
all(y2 == backsolve(t(r), x, upper = FALSE, transpose = FALSE))
```

`barplot` *Bar Plots* `barplot`

Description

Creates a bar plot with vertical or horizontal bars.

Usage

```
barplot(height, width = 1, space = NULL, names.arg = NULL,
        legend.text = NULL, beside = FALSE, horiz = FALSE,
        col = heat.colors(NR), border = par("fg"),
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        xlim = NULL, ylim = NULL,
        axes = TRUE, axisnames = TRUE, inside = TRUE, plot = TRUE, ...)
```

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.
`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. This is only useful when `height` is a matrix. In that case the legend labels should correspond to the rows of `height`.
`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.
`col`	a vector of colors for the bars or bar components.
`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.
`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.
`plot`	logical. If `FALSE`, nothing is plotted.
`...`	further graphical parameters (`par`) are passed to `plot.window()` and `title()`.

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 `apply(mp, 2, mean)` for the midpoints of each *group* of bars, see example.

See Also

`plot(..., type="h")`, `dotplot`, `hist`.

Examples

```
tN <- table(Ni <- rpois(100, lambda=5))
r <- barplot(tN, col='gray')
#- 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)")

data(VADeaths, package = "base")
barplot(VADeaths, plot = FALSE)
barplot(VADeaths, plot = FALSE, beside = TRUE)

mp <- barplot(VADeaths) # default
tot <- apply(VADeaths, 2, sum)
text(mp, tot + 3, format(tot), xpd = NA, 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)
segments(mp, hh, mp, hh + 2*sqrt(1000*hh/100), col = mybarcol, lwd = 1.5)
all(dim(mp) == dim(hh))# corresponding matrices
mtext(side = 1, at = apply(mp, 2, mean), line = -2,
      text = paste("Mean", formatC(apply(hh, 2, mean))), col = "red")
```

`BATCH` *Batch Execution of R* `BATCH`

Description

Run R non-interactively with input from `infile` and place output (stdout/stderr) to another file.

Usage

```
R 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.
`outfile`	the name of a file to which to write output. If not given, the name used is the one of `infile`, with a possible '.R' extension stripped, and '.Rout' appended.

Details

By default, the input commands are printed along with the output. To suppress this behavior, add `options(echo = FALSE)` at the beginning of `infile`.

Note

Unlike `Splus BATCH`, this does not run the R process in the background. In most shells `R BATCH [options] infile [outfile] &` will do so.

`Bessel` *Bessel Functions* `Bessel`

Description

Bessel Functions of integer and fractional order, of first and second kind, J_ν and Y_ν, and Modified Bessel functions (of first and third kind), I_ν and K_ν.

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

Usage

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

Arguments

`x`	numeric, ≥ 0.
`nu`	numeric; ≥ 0 unless in `besselK` which is symmetric in `nu`. The *order* of the corresponding Bessel function.
`expon.scaled`	logical; if `TRUE`, the results are exponentially scaled in order to avoid overflow (I_ν) or underflow (K_ν), respectively.

Details

The underlying C code stems from *Netlib* (http://www.netlib.org/specfun/r[ijky]besl).

If `expon.scaled = 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 `gamma`. It is here for experimental purpose mainly, and *may be defunct very soon.*

Value

Numeric vector of the same length of `x` with the (scaled, if `expon.scale=TRUE`) values of the corresponding Bessel function.

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions.* Dover, New York; Chapter 9: Bessel Functions of Integer Order.

See Also

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

Examples

```
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, leg=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, leg=paste("nu=",nus), col = nus+2, lwd=1)

x0 <- 2^(-20:10)
plot(x0,x0^-8, log='xy', ylab="",type='n',
     main = "Bessel Functions  J_nu(x)  near 0\n log - log  scale")
```

```
for(nu in sort(c(nus,nus+.5))) lines(x0,besselJ(x0,nu=nu), col = nu+2)
legend(3,1e50, leg=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 log - log  scale")
for(nu in sort(c(nus,nus+.5))) lines(x0,besselK(x0,nu=nu), col = nu+2)
legend(3,1e50, leg=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, leg=paste("nu=",nus), col = nus+2, lwd=1)

## Check the Scaling :
for(nu in nus)
   print(all(abs(1- besselK(x,nu)*exp( x) / besselK(x,nu,expo=TRUE)) < 2e-15))
for(nu in nus)
   print(all(abs(1- besselI(x,nu)*exp(-x) / besselI(x,nu,expo=TRUE)) < 1e-15))

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, leg=paste("nu=",nus), col = nus+2, lwd=1)
```

`Beta` *The Beta Distribution* `Beta`

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

```
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 to generate.
`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 `shape1` $= a$ and `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 $0 < x < 1$.

Value

`dbeta` gives the density, `pbeta` the distribution function, `qbeta` the quantile function, and `rbeta` generates random deviates.

See Also

`beta` for the Beta function, and `dgamma` for the Gamma distribution.

Examples

```
x <- seq(0, 1, length=21)
dbeta(x, 1, 1)
pbeta(x, 1, 1)
```

`Binomial` *The Binomial Distribution* `Binomial`

Description

Density, distribution function, quantile function and random generation for the binomial distribution with parameters `size` and `prob`.

Usage

```
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 to generate.
`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.

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.

See Also

`dnbinom` for the negative binomial, and `dpois` for the Poisson distribution.

Examples

```
# 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
plot (0:n,     dbinom(0:n, n, pi/10, log=TRUE), type='l',
      main = "dbinom(*, log=TRUE) is better than  log(dbinom(*))")
lines(0:n, log(dbinom(0:n, n, pi/10)), col='red', lwd=2)
mtext("dbinom(k, log=TRUE)", adj=0)
mtext("extended range", adj=0, line = -1, font=4)
mtext("log(dbinom(k))", col="red", adj=1)
```

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

Description

Get or set the body of a function.

Usage

```
body(fun = sys.function(sys.parent()))
body(fun) <- list
```

Arguments

`fun`	a function object or a character string naming the function to be manipulated. If not specified, the function calling `body` is used.
`list`	a list of R expressions.

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.

See Also

`alist`, `args`, `function`.

Examples

```
body(body)
f <- function(x) x^5
body(f) <- expression(5^x)
f(3) # = 125
str(body(f))
```

`box` *Draw a Box around a Plot* `box`

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

```
box(which="plot", lty="solid", ...)
```

Arguments

`which`	character, one of `"plot"`, `"figure"`, `"inner"` and `"outer"`.
`lty`	line type of the box.
`...`	further graphical parameters, such as `bty`, `col`, or `lwd`, see `par`.

See Also

`rect` for drawing of arbitrary rectangles.

Examples

```
plot(1:7,abs(rnorm(7)), type='h', axes = F)
axis(1, labels = letters[1:7])
box(lty='137', col = 'red')
```

`boxplot` *Box Plots* `boxplot`

Description

Produce box-and-whisker plot(s) of the given (grouped) values.

Usage

```
boxplot(x, ...)
boxplot.default(x, ..., range = 1.5, width = NULL, varwidth = FALSE,
        notch = FALSE, names, data = sys.frame(sys.parent()),
        plot = TRUE, border = par("fg"), col = NULL, log = "",
        pars = NULL)
boxplot.formula(formula, data = NULL, subset, na.action, ...)
```

Arguments

`x`	an R object.
`...`	methods may have additional arguments.
`x, ...`	the data from which the boxplots are to be produced. The data can be specified as separate vectors, each corresponding to a component boxplot, or as a single list containing such vectors. Alternatively a symbolic specification of the form `x ~ g` can be given, indicating that the observations in the vector `x` are to be grouped according to the levels of the `factor` `g`. In this case the argument `data` can be used to provide values for the variables in the specification. `NA`s are allowed in the data.
`range`	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.
`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.
`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.
`names`	group labels which will be printed under each boxplot.
`data`	`data.frame`, `list`, or `environment` in which variable names are evaluated when `x` is a formula.
`plot`	if `TRUE` (the default) then a boxplot is produced. If not, the summaries which the boxplots are based on are returned.
`border`	an optional vector of colors for the outlines of the boxplots. The values in `border` are recycled if the length of `border` is less than the number of plots.
`col`	if `col` is non-null it is assumed to contain colors to be used to col the bodies of the box plots.

`log`	character indicating if x or y or both coordinates should be plotted in log scale.
`pars, ...`	graphical parameters can also be passed as arguments to `boxplot`.
`formula`	a formula, such as `y ~ x`.
`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 in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s.
`...`	further arguments to the default boxplot method and graphical parameters may also be passed as arguments, see `par`.

Details

This is a generic function. It currently has a default method (`boxplot.default`) and a formula interface (`boxplot.formula`).

Value

List with one component corresponding to each plot. The components are themselves lists with named components as follows:

`stats`	a vector 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 observations in the sample.
`conf`	the lower and upper extremes of the notch.
`out`	the values of any data points which lie beyond the extremes of the whiskers.

See Also

`boxplot.stats` which does the computation, `bxp` for the plotting, and `stripplot` for an alternative (with small data sets).

Examples

```
## boxplot on a formula:
data(InsectSprays)
boxplot(count ~ spray, data = InsectSprays, col = "lightgray")

data(OrchardSprays)
boxplot(decrease ~ treatment, data = OrchardSprays,
        log = "y", col="bisque")

## 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 = ...)")
```

`boxplot.stats` *Box Plot Statistics* `boxplot.stats`

Description

This function is typically is called by `boxplot` 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 (`NA`s and `NaN`s 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 interquartile coef from the box. A value of zero causes the whiskers to extend to the data extremes.
`do.conf,do.out`	logicals; if `FALSE`, the `conf` or `out` component respectively will be empty in the result.

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 of non-`NA` observations in the sample.
`conf`	the lower and upper extremes of the "notch".
`out`	the values of any data points which lie beyond the extremes of the whiskers.

Note that `$stats` and `$conf` are sorted in *in*creasing order, unlike S, and that `$n` and `$out` include any `+- Inf` values.

See Also

`fivenum`, `boxplot`, `bxp`.

Examples

```
x <- c(1:100, 1000)
str(boxplot.stats(x))
str(boxplot.stats(x, do.conf=FALSE, do.out=FALSE))
str(boxplot.stats(x, coef = 3, do.conf=FALSE))
str(boxplot.stats(x, coef = 0))

str(boxplot.stats(c(x, NA)))
str(boxplot.stats(c(x, -1:1/0)))
```

browser	*Environment Browser*	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 interpreter copy is 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.

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

See Also

`debug`, and `traceback` for the stack on error.

bug.report	*Send a Bug Report*	bug.report

Description

Invokes an editor to write a bug report and optionally mail it to the R-bugs list at ⟨r-bugs@biostat.ku.dk⟩. Some standard information on the current version and configuration of R are included automatically.

Usage

```
bug.report(subject = "", ccaddress = getenv("USER"),
           method = getOption("mailer"), address = "r-bugs@biostat.ku.dk",
           file = "R.bug.report")
```

Arguments

`subject`	Subject of the email. Please do not use single quotes (') in the subject!
`ccaddress`	Optional email address for copies (default is current user). Use `ccaddress = FALSE` for no copies.
`method`	Submission method.
`address`	Recipient's email address.
`method`	Submission method, one of `"mailx"`, `"gnuclient"`, `"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 `"gnuclient"`, then an emacs mail buffer is opened and used for sending the email.

If method is `"none"` or `NULL` (which is the default 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. 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. However, we know that the introductory documentation is seriously inadequate, so you don't need to report this.

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 R until the problem happens. Always include the version of R, machine, and operating system that you are using; type 'version' in R to print this.

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 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 `data.frame(x, y, z, monday, tuesday)` never returns. Do not report that `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 got your report we would try out the `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 `[` method that had a bug causing R's internal data structures to be corrupted and making the `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 R with the `--vanilla` option may help in isolating a bug. This ensures that the site profile and saved data files are not read.

On some systems a bug report can be generated using the `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@biostat.ku.dk⟩ or submitted to the Web page at http://r-bugs.biostat.ku.dk/R.

Author(s)

This help page is adapted from the Emacs manual

See Also

R FAQ

`builtins` *Returns the names of all built-in objects* `builtins`

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.

`bxp` *Box Plots from Summaries* `bxp`

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

```
bxp(z, notch = FALSE, width = NULL, varwidth = FALSE, notch.frac = 0.5,
    border = par("fg"), col = NULL, log = "", pars = 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.
`notch.frac`	numeric in (0,1). When `notch=TRUE`, the fraction of the box width that the notches should use.
`border`	character, the color of the box borders. Is recycled for multiple boxes.
`col`	character; the color within the box. Is recycled for multiple boxes
`log`	character, indicating if any axis should be drawn in logarithmic scale, as in `plot.default`.
`pars,...`	Graphical parameters can be passed as arguments to this function, either as a list (`pars`) or normally(`...`).

Value

An invisible vector with the x-coordinates of box centers, useful for adding to the plot.

Examples

```
str(bx.p <- boxplot(split(rnorm(100), gl(5,20)), plot = FALSE))
op <- par(mfrow= c(2,2))
bxp(bx.p)
bxp(bx.p, notch = TRUE)
bxp(bx.p, notch = TRUE, col= 'lightblue')
bxp(bx.p, notch = TRUE, col= 'lightblue', border='red', log = 'x')
par(op)
```

---

by *Apply a Function to a Data Frame split by Factors* 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(x)`.
`FUN`	a function to be applied to data frame subsets of `x`.
`...`	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 term.

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

```
data(warpbreaks)
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))
detach("warpbreaks")
```

C *Sets Contrasts for a Factor* C

Description

Sets the "contrasts" attribute for the factor.

Usage

```
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.

Author(s)

B.D. Ripley

See Also

contrasts, contr.sum, etc.

Examples

```
## reset contrasts to defaults
options(contrasts=c("contr.treatment", "contr.poly"))
data(warpbreaks)
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))

data(esoph) # following on from help(esoph)
```

```
model3 <- glm(cbind(ncases, ncontrols) ~ agegp + C(tobgp, , 1) +
     C(alcgp, , 1), data = esoph, family = binomial())
summary(model3)
```

---

c *Combine Values into a Vector or List* c

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

Usage

```
c(..., recursive=FALSE)
```

See Also

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

Examples

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

c(list(A=c(B=1)),recursive=T)

c(options(), recursive=T)
c(list(A=c(B=1,C=2),B=c(E=7)),recursive=T)
```

---

call *Function Calls* call

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.
`x`	an arbitrary R object.

Value

`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 not possible to coerce objects to mode call (objects either are calls or they are not calls). `as.call` returns its argument if it is a call and otherwise terminates with an error message.

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
cl <- call("round", 10.5)
is.call(cl)# TRUE
cl
# such a call can also be evaluated.
eval(cl)# [1] 10
```

`cars` *Stopping Distances of Cars* `cars`

Description

The data give the speed of cars and the distances taken to stop. Note that the data were recorded in the 1920s.

Usage

```
data(cars)
```

Format

A data frame with 50 observations on 2 variable.

[,1]	speed	numeric	Speed (mph)
[,2]	dist	numeric	Stopping distance (ft)

Source

Ezekiel, M. (1930) *Methods of Correlation Analysis.* Wiley.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* Wiley.

`case/variable.names` *Case and Variable Names of Fitted Models* `case/variable.names`

Description

Simple utilities returning (non-missing) case names, and (non-eliminated) variable names.

Usage

```
case.names(obj, ...)
case.names.lm(obj, full = FALSE)

variable.names(obj, ...)
variable.names.lm(obj, full = FALSE)
```

Arguments

`obj`	an R object, typically a fitted model.
`full`	logical; if `TRUE`, all names (including zero weights,..) are returned.

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
```

`cat` *Concatenate and Print* `cat`

Description

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

Usage

```
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`	character string naming the file to print to. If `""` (the default), `cat` prints to the standard output.
`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`, 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`	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 linefeeds 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`).

See Also

`print`, `format`

Examples

```
## print an informative message
cat("iteration = ", iter <- iter + 1, "\n")
```

`Cauchy` *The Cauchy Distribution* `Cauchy`

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 to generate.
`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.

See Also

`dt` for the t distribution which generalizes `dcauchy(*, l = 0, s = 1)`.

Examples

```
all.equal(dcauchy(-1:4), 1 / (pi*(1 + (-1:4)^2)))
```

---

cbind — *Combine Columns/Rows into a Matrix* — cbind

Description

Take a sequence of vector and/or matrix arguments and combine them as the *c*olumns or *r*ows, respectively, of a matrix.

Usage

```
cbind(...)
rbind(...)
```

Details

The functions `cbind` and `rbind` are generic, with methods for data frames.

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` when they are recycled only *fractionally*).

When the arguments consist of a mix of matrices and vectors the number of columns (rosw) 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.

Note

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

See Also

`c` to combine vectors (and lists).

Examples

```
cbind(1,1:7) # the '1' (= shorter vector) is recycled
cbind(1:7, diag(3))# vector is subset -> warning

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

cbind(0, matrix(1, nrow=0, ncol=4))#> Warning (making sense)
dim(cbind(0, matrix(1, nrow=2, ncol=0)))#-> 2 x 1
```

`char.expand` *Expand a String with Respect to a Target Table* `char.expand`

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)
```

---

character *Character Vectors* character

Description

Create or test for objects of type "character".

Usage

```
character(length = 0)
as.character(x)
is.character(x)
```

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 be of character type.

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

---

charmatch *Partial String Matching* charmatch

Description

charmatch 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.

See Also

`match`, `pmatch`.

Examples

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

`check.options` *Set Options with Consistency Checks* `check.options`

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 `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 overriden 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")
str(L2 <- check.options(list(a=0:2), name.opt = "L1"))
str(check.options(NULL, reset = TRUE, name.opt = "L1"))
```

---

chickwts — *Chicken Weights by Feed Type* — chickwts

Description

An experiment was conducted to measure and compare the effectiveness of various feed supplements on the growth rate of chickens.

Usage

```
data(chickwts)
```

Format

A data frame with 71 observations on 2 variables.

[1,]	weight	numeric	Chick weight
[2,]	feed	factor	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

Anonymous (1948) *Biometrika*, **35**, p.214.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

`Chisquare` *The (non-central) Chi-Square Distribution* `Chisquare`

Description

Density, distribution function, quantile function and random generation for the chi-square (χ^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)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations to generate.
`df`	degrees of freedom.
`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 chi-square distribution with `df`$= n$ degrees of freedom has density

$$f_n(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}$$

for $x > 0$. Mean and variance are n and $2n$, respectively.

The non-central chi-square 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 $x \geq 0$.

Value

`dchisq` gives the density, `pchisq` gives the distribution function, `qchisq` gives the quantile function, and `rchisq` generates random deviates.

See Also

`dgamma` for the Gamma distribution which generalizes the chi-square one.

Examples

```
dchisq(1, df=1:3)
pchisq(1, df= 3)
pchisq(1, df= 3, ncp = 0:4)# includes the above

x <- 1:10
## Chisquare( 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))
```

---

chol — *The Choleski Decomposition* — chol

Description

Compute the Choleski factorization of a symmetric (Hermitian), positive definite square matrix.

Usage

```
chol(x)
```

Arguments

x	a symmetric, positive definite matrix.

Value

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

Note that effectively, only the upper triangular part of x is used such that the above only holds when x *is* symmetric.

References

Dongarra, J. J., Bunch, J. R., Moler, C. B. and Stewart, G. W. (1978) *LINPACK Users Guide.* Philadelphia: SIAM Publications.

See Also

chol2inv for its *inverse*, backsolve for solving linear systems with upper triangular left sides.

qr, svd for related matrix factorizations.

Examples

```
( m <- matrix(c(5,1,1,3),2,2) )
( cm <- chol(m) )
t(cm) %*% cm  #-- = 'm'
all(abs(m  -  t(cm) %*% cm) < 100* .Machine$double.eps) # TRUE
```

chol2inv *Inverse from Choleski Decomposition* chol2inv

Description

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

Usage

```
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.

Value

The inverse of the decomposed matrix.

References

Dongarra, J. J., Bunch, J. R., Moler, C. B. and Stewart, G. W. (1978) *LINPACK Users Guide*. Philadelphia: SIAM Publications.

See Also

chol, solve.

Examples

```
cma <- chol(ma  <- cbind(1, 1:3, c(1,3,7)))
t(cma) %*% cma # = ma
all.equal(diag(3), ma %*% chol2inv(cma))
```

chull *Compute Convex Hull of a Set of Points* chull

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 components x and y

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.

Author(s)

B. D. Ripley

References

Eddy, W. F. (1977) A new convex hull algorithm for planar sets. *ACM Transactions on Mathematical Software*, **3**, 398–403.

Eddy, W. F. (1977) Algorithm 523. CONVEX, A new convex hull algorithm for planar sets[Z]. *ACM Transactions on Mathematical Software*, **3**, 411–412.

See Also

xy.coords,polygon

Examples

```
X <- matrix(rnorm(2000), ncol=2)
plot(X, cex=0.5)
hpts <- chull(X)
hpts <- c(hpts, hpts[1])
lines(X[hpts, ])
```

`class` *Object Classes* `class`

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.

Usage

```
class(x)
class(x) <- names
unclass(x)
inherits(x, name)
```

Details

An R "object" is a data object which has a `class` attribute. A class attribute is a vector of character strings giving the names of the classes which the object "inherits" from. 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.

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 information removed.

`inherits` indicates whether its first argument inherits from a class with name equal to its second argument.

See Also

`UseMethod`, `NextMethod`.

`close.socket` *Close a Socket* `close.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

Value

logical indicating success or failure

Author(s)

Thomas Lumley

See Also

`make.socket`, `read.socket`

co2 *Mauna Loa Atmospheric CO2 Concentration* co2

Description

Atmospheric concentrations of CO_2 are expressed in parts per million (ppm) and reported in the preliminary 1997 SIO manometric mole fraction scale.

Usage

```
data(co2)
```

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 92093-0220.

ftp://cdiac.esd.ornl.gov/pub/maunaloa-co2/maunaloa.co2.

References

Cleveland, W. S. (1993) *Visualizing Data.* New Jersey: Summit Press.

Examples

```
data(co2)
plot(co2, ylab = expression("Atmospheric concentration of CO"[2]),
     las = 1)
title(main = "co2 data set")
```

codes *Factor Codes* codes

Description

This (generic) function returns a numeric coding of a factor. It can also be used to assign to a factor using the coded form.

Usage

```
codes(x)
codes(x) <- value
```

Value

For an ordered factor, it returns the internal coding (1 for the lowest group, 2 for the second lowest, etc.).

For an unordered factor, an alphabetical ordering of the levels is assumed, i.e the level that is coded 1 is the one whose name is sorted first according to the prevailing collating sequence. **Warning:** the sort order may well depend on the locale, and should not be assumed to be ASCII.

Note

Normally `codes` is not the appropriate function to use with an unordered factor. Use `unclass` or `as.numeric` to extract the codes used in the internal representation of the factor, as these do not assume that the codes are sorted.

See Also

`factor`, `levels`, `nlevels`.

Examples

```
codes(rep(factor(c(20,10)),3))

x <- gl(3,5)
codes(x)[3] <- 2
x

data(esoph)
( ag <- esoph$alcgp[12:1] )
codes(ag)

codes(factor(1:10)) # BEWARE!
```

`coefficients` *Extract Model Coefficients* `coefficients`

Description

`coef` is a generic function which extracts model coefficients from objects returned by modeling functions. `coefficients` is an *alias* (`.Alias`) for it.

Usage

```
coef(x, ...)
coefficients(x, ...)
```

Arguments

`x`	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. (Note that the method is `coef` and not `coefficients`.)

Value

Coefficients extracted from the model object `x`.

See Also

`fitted.values` and `residuals` for related methods; `glm`, `lm` for model fitting.

Examples

```
x <- 1:5; coef(lm(c(1:3,7,6) ~ x))
```

`col` *Column Indexes* `col`

Description

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

Usage

```
col(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 `j`.

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
```

`colors` *Color Names* `colors`

Description

Returns the built-in color names which R knows about.

Usage

```
colors()
```

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.

Examples

```
str(colors())
```

`commandArgs` *Extract Command Line Arguments* `commandArgs`

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.

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

BATCH

Examples

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

---

comment — *Query or Set a 'Comment' Attribute* — comment

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* `Comparison`

Description

Binary operators which allow the comparison of values in vectors.

Usage

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

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.

Examples

```
x <- rnorm(20)
x < 1
x[x > 0]
```

`COMPILE` *Compile Files for Use with R* `COMPILE`

Description

The given source files are compiled so that they can subsequently be collected into a shared library using `R SHLIB` and be loaded into R using `dyn.load()`.

Usage

```
R COMPILE [options] srcfiles
```

Arguments

`srcfiles`	A list of the names of source files to be compiled. Currently, only C and FORTRAN are supported; the corresponding files should have the extensions '.c' and '.f', respectively.
`options`	A list of compile-relevant settings, such as special values for `CFLAGS` or `FFLAGS`.

See Also

`SHLIB`, `dyn.load`

`complete.cases` *Find Complete Cases* `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

```
data(airquality)
x <- airquality[, -1] # x is a regression design matrix
y <- airquality[,  1] # y is the corresponding response

all(!complete.cases(y) == is.na(y)) #--> TRUE
ok <- complete.cases(x,y)
sum(!ok) # how many are not "ok" ?
x <- x[ok,]
y <- y[ok]
```

`complex` *Complex Vectors* `complex`

Description

These are basic functions which support complex arithmetic in R. 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.

Usage

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

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

Details

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, $\texttt{Mod}(z) = \sqrt{x^2 + y^2}$, and for $\phi = Arg(z)$, $x = \cos(\phi)$ and $y = \sin(\phi)$.

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

Examples

```
( z <- 0i ^ (-3:3) )
all(Re(z) == 0 ^ (-3:3))
matrix(1i^ (-6:5), nr=4)#- all columns are the same
0 ^ 1i # a complex NaN

## create a complex normal vector
z <- complex(real = rnorm(100), imag = rnorm(100))
## or also (less efficiently):
z2 <- 1:2 + 1i*(8:9)

all(Mod ( 1 -  sin(z) / ( (exp(1i*z)-exp(-1i*z))/(2*1i) ))
    < 100*.Machine$double.eps)
## The Arg(.) is an angle:
zz <- (rep(1:4,len=9) + 1i*(9:1))/10
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")
```

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

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 global environment or a package is masking a system object of the same name. This helps discover unintentional masking.

Usage

```
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.

Author(s)

B.D. Ripley

Examples

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

`Constants` *Built-in Constants* `Constants`

Description

Constants built into R.

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.

See Also

`data`.

Examples

```
all(nchar(letters) == 1)                    # TRUE
all(month.abb == substr(month.name, 1, 3))  # TRUE

eps <- .Machine$double.eps
all.equal(pi, 4*atan(1), tol= 2*eps)

# John Machin (1705) computed 100 decimals of pi :
all.equal(pi/4, 4*atan(1/5) - atan(1/239), 4*eps)
```

`contour` *Display Contours* `contour`

Description

Create a contour plot, or add contour lines to an existing plot.

Usage

```
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"),
        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 **iff** 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.
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.
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) and the arguments to title may also be supplied.

Details

There is currently no documentation about the algorithm. The source code is in '$R_HOME/src/main/plot3d.c'.

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.

See Also

filled.contour for "color-filled" contours, image and the graphics demo which can be invoked as demo(graphics).

Examples

```
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 = "/")
## Should not be necessary:
z[!is.finite(z)] <- NA
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 = F)

data("volcano")
rx <- range(x <- 10*1:nrow(volcano))
ry <- range(y <- 10*1:ncol(volcano))
ry <- ry + c(-1,1) * (diff(rx) - diff(ry))/2
tcol <- terrain.colors(12)
par(opar); par(mfrow=c(1,1)); opar <- par(pty = "s", bg = "lightcyan")
plot(x = 0, y = 0,type = "n", xlim = rx, ylim = ry, xlab = "", ylab = "")
u <- par("usr")
rect(u[1], u[3], u[2], u[4], col = tcol[8], border = "red")
contour(x, y, volcano, col = tcol[2], lty = "solid", add = TRUE,
        vfont = c("sans serif", "plain"))
title("A Topographic Map of Maunga Whau", font = 4)
abline(h = 200*0:4, v = 200*0:4, col = "lightgray", lty = 2, lwd = 0.1)
par(opar)
```

`contrast` *Contrast Matrices* `contrast`

Description

Return a matrix of contrasts.

Usage

```
contr.helmert(n, contrasts = TRUE)
contr.poly(n, contrasts = TRUE)
contr.sum(n, contrasts = TRUE)
contr.treatment(n, base = 1, 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.
`base`	an integer specifying which group is considered the baseline group.

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` then an additional column of ones is prepended to the matrix.

Note that as from R version 0.62.2, `contr.poly` returns contrasts based on orthogonal (rather than raw) polynomials.

Value

A matrix with `n` rows and `k` columns, with `k=n-1` if `contrasts` is `TRUE` and `k=n` if `contrasts` is `FALSE`.

See Also

`contrasts`, `C`, and `aov`, `glm`, `lm`.

Examples

```
(cH <- contr.helmert(4))
apply(cH, 2,sum)# column sums are 0!
crossprod(cH)# diagonal -- columns are orthogonal

(cT <- contr.treatment(5))
all(crossprod(cT) == diag(4))# TRUE: even orthonormal

(cP <- contr.poly(3))# Linear and Quadratic
zapsmall(crossprod(cP), dig=15) # orthonormal up to fuzz
```

`contrasts` — *Get and Set Contrast Matrices* — `contrasts`

Description

Set and view the contrasts associated with a factor.

Usage

```
contrasts(x)
contrasts(x) <- ctr
```

Arguments

`x`	a factor.
`ctr`	either a 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.

See Also

C, contr.helmert, contr.poly, contr.sum, contr.treatment; glm, aov, lm.

Examples

```
example(factor)
(fff <- factor(ff))
contrasts(fff) # treatment
contrasts(C(fff, sum))
```

`contributors` *R Project Contributors* `contributors`

Description

The R Who-is-who.

The R Project

R is a project which is attempting to provide a modern piece of statistical software for the GNU suite of software.

Contributors

The current R is the result of a collaborative effort authors from all over the world.

R was initially written by Robert Gentleman and Ross Ihaka—also known as "*R & R*"—of the Statistics Department of the University of Auckland.

Since mid-1997 there has been a core group with write access to the R source, currently consisting of

- Douglas Bates
- Peter Dalgaard
- Robert Gentleman
- Kurt Hornik
- Ross Ihaka
- Friedrich Leisch
- Thomas Lumley
- Martin Maechler
- Guido Masarotto
- Paul Murrell
- Brian Ripley
- Duncan Temple Lang
- Luke Tierney

plus Heiner Schwarte up to October 1999.

In addition, a large group of individuals has contributed to R by donating code, bug reports and documentation, notably

Valerio Aimale, Thomas Baier, Ben Bolker, Goran Brostrom, Saikat DebRoy, Lyndon Drake, Paul Gilbert, Robert King, Philippe Lambert, Patrick Lindsey, Jim Lindsey, John Maindonald, Jens Oehlschaegel-Akiyoshi, Steve Oncley and Gordon Maclean, Richard Okeefe, Hubert Palme, Jose Pinheiro, Martyn Plummer, Jonathan Rougier, Bill Simpson, Adrian Trapletti, Terry Therneau, Bill Venables, Gregory R. Warnes, and Andreas Weingessel.

We have probably omitted some important names here because of incomplete record keeping. If we have overlooked you, please let us know and we'll update the list.

A special debt is owed to John Chambers who has graciously contributed advice and encouragement.

`Control` *Control Flow* `Control`

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
```

See Also

`ifelse`, `switch`.

Examples

```
for(i in 1:5) print(1:i)
```

`convolve` *Fast Convolution* `convolve`

Description

Use the Fast Fourier Transform to compute the several kinds of convolutions of two sequences.

Usage

```
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 `0`s (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, for $k = 1, \ldots, n + m - 1$

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

Brillinger, D. R. (1981) *Time Series: Data Analysis and Theory*, Second Edition. San Francisco: Holden-Day.

See Also

`fft`, `nextn`.

Examples

```
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")
```

`coplot` *Conditioning Plots* `coplot`

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"),
       xlab = paste("Given :", a.name),
       ylab = paste("Given :", b.name),
       number = 6, overlap = 0.5, ...)
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`. `x` and `y` must be numeric, but `a` and `b` may be either numeric or factors.
`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.
`xlab`	character; label to use for the 1st conditioning variable.
`ylab`	character; label to use for the 2nd conditioning variable.
`number`	integer; the number of conditioning intervals, possibly of length 2 for x and y direction.
`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.
`...`	additional arguments to the panel function.

Value

`co.intervals(., number, .)` returns a (`number` $\times$ 2) `matrix`, say `ci`, where `ci[k,]` is the `range` of `x` values for the `k`-th interval.

References

Cleveland, W. S. (1993) *Visualizing Data.* New Jersey: Summit Press.

See Also

`pairs`, `panel.smooth`, `points`.

Examples

```
## Tonga Trench Earthquakes
data(quakes)
coplot(long ~ lat | depth, data = quakes)
given.depth <- co.intervals(quakes$depth, number=4, overlap=.1)
coplot(long ~ lat | depth, data = quakes, given.values=given.depth)

## Conditioning on 2 variables:
ll.dm <- long ~ lat | depth * mag
coplot(ll.dm, data = quakes)
coplot(ll.dm, data = quakes, number=c(4,7), show.given = c(T,F))
coplot(ll.dm, data = quakes, number=c(3,7),
```

```
       overlap=c(-.5,.1)) # negative overlap DROPS values

data(warpbreaks)
## given two factors
coplot(breaks ~ 1:54 | wool * tension, data = warpbreaks, col = 'red')

## Example with empty panels:
data(state)
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, ...))
detach() # data.frame(state.x77)
```

`copyright` *Copyrights of Files Used to Build R* `copyright`

Description

R is released under the 'GNU Public License': see `?license` for details. The license describes your right to use 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.

Details

The file '$R_HOME/COPYRIGHTS' lists the copyrights in full detail.

`cor` *Correlation and Covariance Matrices* `cor`

Description

Compute the correlation or covariance matrix of the columns of `x` and the columns of `y`.

Usage

```
cor(x, y=x, use="all.obs")
cov(x, y=x, use="all.obs")
```

Arguments

`x`	a matrix or data frame.
`y`	a matrix or data frame.
`use`	a character string giving the method for handling missing observations. This must be one of the stringss `"all.obs"`, `"complete.obs"` or `"pairwise.complete.obs"` (abbreviations are acceptable).

Details

If `use` is `"all.obs"`, then the presence of missing observations will cause the computation to fail. If `use` has the value `"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.

See Also

`cov.wt` for *weighted* covariance computation.

Examples

```
## Two simple vectors
cor(1:10,2:11)# == 1

## Correlation Matrix of Multivariate sample:
data(longley)
(Cl <- cor(longley))
## Graphical Correlation Matrix:
symnum(Cl) # highly correlated

##--- Missing value treatment:
data(swiss)
C1 <- cov(swiss)
range(eigen(C1, only=T)$val) # 6.19  1921
swiss[1,2] <- swiss[7,3] <- swiss[25,5] <- NA # create 3 "missing"

 C2 <- cov(swiss) # Error: missing obs...

C2 <- cov(swiss, use = "complete")
range(eigen(C2, only=T)$val) # 6.46  1930
C3 <- cov(swiss, use = "pairwise")
range(eigen(C3, only=T)$val) # 6.19  1938
```

`count.fields` *Count the Number of Fields per Line* `count.fields`

Description

`count.fields` counts the number of fields, as separated by `sep`, in each of the lines of `file` read. It is used by `read.table`; a user will typically have no need for it.

Usage

```
count.fields(file, sep = "", quote = "", skip = 0)
```

Arguments

`file`	a character string naming an ASCII data file
`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.

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")
```

`cov.wt` *Weighted Covariance Matrices* `cov.wt`

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

```
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`.

`crossprod` *Matrix Crossproduct* `crossprod`

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=x)
```

See Also

`%*%` and outer product `%o%`.

Examples

```
crossprod(1:4)    # = sum(1 + 2^2 + 3^2 + 4^2)
drop(.Last.value) # scalarized
```

`cumsum` *Cumulative Sums, Products, and Extremes* `cumsum`

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`.

Examples

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

`curve` *Draw Function Plots* `curve`

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, ...)
plot.function(fn, from = 0, to = 1, n = 101, ...)
```

Arguments

`expr`	an expression written as a function of `x`, or alternatively a function which will be plotted.
`fn`	a 'vectorizing' numeric 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 `TRUE` add to already existing plot.
`...`	graphical parameters can also be specified as arguments. `plot.function` passes all these to `curve`.

Details

The evaluation of `expr` is at `n` points equally spaced over the range `[from, to]`. The points determined in this way are then joined with straight lines. `fn(x)` or `expr` (with `x` inside) must return a numeric of the same length as `x`.

This used to be a quick hack which seems to serve a useful purpose, but can give bad results for functions which are not smooth.

Value

`NULL`. For "expensive" `expr`essions, you should use smarter tools.

See Also

`splinefun` for spline interpolation, `lines`.

Examples

```
par(mfrow=c(2,2))
curve(x^3-3*x, -2, 2)
curve(x^2-2, add = TRUE, col = "violet")

curve(sin, -pi, 3*pi)
plot(cos)
chippy <- function(x) sin(cos(x)*exp(-x/2))
curve(chippy, -8, 7, n=2001)

for(ll in c("","x","y","xy"))
   curve(log(1+x), 1,100, log=ll, sub=paste("log=",ll))
```

`cut` *Convert Numeric to Factor* `cut`

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, ...)
cut.default(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.
`break`	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 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.

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 how many digits should be used in formatting the numbers `b1`, `b2`, ...

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.

See Also

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

Examples

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

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

##--- some values OUTSIDE the breaks :
table(cx  <- cut(x, br = 2*(0:4)))
table(cxl <- cut(x, br = 2*(0:4), right = F))
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
table(cut(y, breaks =  1*(-3:3), right = F))#- the same, since no exact INT!
```

`data` *Data Sets* `data`

Description

`data` loads a data set or lists (via `show.data`) the available data sets.

Usage

```
data(..., list = character(0), package = .packages(),
     lib.loc = .lib.loc)
show.data(package = .packages(), lib.loc = .lib.loc)
```

Arguments

`...`	a sequence of `name`s or character strings
`list`	a character vector
`package`	a name or character vector giving the packages to look into for data sets. By default, all packages in the search path are used.
`lib.loc`	a character vector of directory names of R libraries. Defaults to all libraries currently known.

Details

Currently, four formats of data files are supported:

1. files ending '.RData' or '.rda' are `load()`ed.
2. files ending '.R' or '.r' are `source()`d in, with the R working directory changed temporarily to the directory containing the respective file.
3. files ending '.tab' or '.txt' are read using `read.table(..., header = TRUE)`, and hence result in a data frame.
4. files ending '.csv' are read using `read.table(..., header = TRUE, sep = ";")`, and also result in a data frame.

The data sets to be loaded can be specified as a sequence of `name`s or character strings, or as the character vector `list`, or as both. If no data sets are specified or `show.data` is called directly, the available data sets are displayed.

If no data sets are specified, `data` calls `show.data`. `show.data` looks for a file '00Index' in a '`data`' directory of each specified package, and uses these files to prepare a listing. If there is a '`data`' area but no index a warning is given: such packages are incomplete.

If `lib.loc` is not specified, the packages are searched for amongst those already loaded, followed by the `data` directory (if any) of the currrent working directory. If `lib.loc` is specified, they are searched for in the specified libraries, even if they are already loaded from another library.

To just look in the `data` directory of the current working directory, set `package = NULL`.

Value

`data()` returns a character vector of all data sets specified, an empty character vector 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 text files in the 'help' directory of an installed package can be zipped up as a zip archive 'Rdata.zip'. You will need to proved a single-column file 'filelist' of file names.

See Also

`help` for obtaining documentation on data sets.

Examples

```
data()                        # list all available data sets
data(package = base)          # list the data sets in the base package
data(USArrests, "VADeaths")   # load the data sets 'USArrests and 'VADeaths'
help(USArrests)               # give information on data set 'USArrests'
```

`data.class` *Object Classes* `data.class`

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.)

See Also

`class`

Examples

```
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)
```

`data.frame` *Data Frames* `data.frame`

Description

These functions create or manipulate 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)

as.data.frame(x)
is.data.frame(x)

row.names(data.frame.obj)
row.names(data.frame.obj) <- names
print(data.frame.obj, ..., digits = NULL, quote = FALSE, right = TRUE)
plot (data.frame.obj, ...)
```

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`	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`	if `TRUE` then the names of the variables in the data frame are checked to ensure that they are valid variable names. If necessary they are adjusted (by `make.names`) so that they are.
`data.frame.obj`	objects of class `data.frame`.
`...`	optional arguments to `print` or `plot` methods.

Details

Non-numeric variables passed to `data.frame` are converted to factor columns unless protected by `I`. This applies to character and logical variables, in particular. It also applies to adding columns to a data frame.

If a list or data frame or matrix is passed to `data.frame` it is as if each column had been passed as a separate argument, with the exception of matrices of class `model.matrix`.

Value

For `data.frame(.)` a data frame, a matrix-like stucture whose columns may be of differing types (numeric, factor and character).

`as.data.frame` is generic function with many `methods`. It attempts to coerce its argument to be a data frame.

`is.data.frame` returns `TRUE` if its argument is a data frame and `FALSE` otherwise.

`row.names` can be used to set and retrieve the row names of a data frame, similarly to `rownames` for arrays.

`plot.data.frame`, a method of the `plot` generic, uses `stripplot` for *one* variable, `plot.default` (scatterplot) for *two* variables, and `pairs` (scatterplot matrix) otherwise.

For the `print` method (`print.data.frame`), see `print.matrix`.

`xpdrows.data.frame` is an auxiliary function which expands the rows of a data frame. It is used by the data frame methods of `[<-` and `[[<-` (which perform subscripted assignments on a data frame), and not intended to be called directly.

See Also

`read.table`, `Math.data.frame`,etc, about *Group* methods for `data.frame`s; `make.names`.

Examples

```
L3 <- LETTERS[1:3]
str(d <- data.frame(cbind(x=1, y=1:10), ch=sample(L3, 10, repl=TRUE)))
str(     data.frame(cbind(  1,   1:10),    sample(L3, 10, repl=TRUE)))
is.data.frame(d)
all(1:10 == row.names(d))# TRUE (coercion)
```

---

data.matrix *Data Frame to Numeric Matrix* data.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 codes.

Usage

```
data.matrix(frame)
```

Arguments

`frame`	a data frame whose components are either logical vectors, factors or numeric vectors.

See Also

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

`dataentry` *Spreadsheet Interface for Entering Data* `dataentry`

Description

This is a suite of functions designed to make the interface to the spreadsheet painless for users.

Usage

```
data.entry(..., Modes = NULL, Names = NULL)
dataentry(data, modes)
de(..., Modes = NULL, Names = NULL)
```

Details

`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`.

See Also

`vi`, `edit`.

Examples

```
# call data entry with variables x and y
data.entry(x,y)
```

`date` *System Date and Time* `date`

Description

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

Usage

```
date()
```

Value

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

Examples

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

`debug` *Debug a function* `debug`

Description

Set or unset the debugging flag on a function.

Usage

```
debug(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). Currently you can only debug functions that have bodies enclosed in braces. This is a bug and will be fixed soon. 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. 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. 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.

See Also

`browser`, `traceback` to see the stack after an `Error: ...` message.

`debugger` *Post-Mortem Debugging* `debugger`

Description

Functions to dump the evaluation environments (frames) and to examine dumped frames.

Usage

```
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`

Details

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.

Author(s)

B. D. Ripley

See Also

`options` for setting `error` options.

Examples

```
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()}))
```

`Defunct` *Defunct Functions* `Defunct`

Description

The functions listed here are no longer part of R as they are not needed (any more).

Usage

```
.Defunct()

category(x, levels, labels, ordered, exclude)
dnchisq(.)
```

```
pnchisq(.)
qnchisq(.)
rnchisq(.)
print.anova.lm(.)
print.anova.glm(.)
print.tabular(.)
print.plot(.)
save.plot(.)
```

Details

`category` has been an old-S function before there were factors; should be replaced by `factor` throughout!

The `*chisq()` functions now take an optional non-centrality argument, so the `*nchisq()` functions are no longer needed.

The new function `dev.print()` should now be used for saving plots to a file or printing them.

`.Defunct` is the function to which defunct functions are set.

See Also

`Deprecated`

`delay` *Delay Evaluation* `delay`

Description

`delay` creates a *promise* to evaluate the given expression in the specifies environment if its value is requested. This provides direct access to *lazy evaluation* mechanism used by R for the evaluation of (interpreted) functions.

Usage

```
delay(expr, env=.GlobalEnv)
```

Arguments

`expr`	an expression.
`env`	an evaluation environment

Details

This is an experimental feature and its addition is purely for evaluation purposes.

Value

A *promise* to evaluate the expression. The value which is returned by `delay` can be assigned without forcing its evaluation, but any further accesses will cause evaluation.

Examples

```
x <- delay({
    for(i in 1:7)
        cat("yippee!\n")
    10
})

x^2#- yippee
x^2#- simple number
```

`delete.response` *Modify Terms Objects* `delete.response`

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.

`reformulate` creates a formula from a vector of strings.

Usage

```
delete.response(termobj)
reformulate(termlabels,response=NULL)
drop.terms(termobj, dropx = NULL, keep.response = FALSE)
```

Arguments

`termobj`	A `terms` object
`termlabels`	vector of strings giving the right-hand side of a model formula
`response`	string 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

```
ff <- y ~ z + x + w
tt <- terms(ff)
tt
delete.response(tt)
drop.terms(tt, 2:3, keep.response = T)
reformulate(attr(tt, "term.labels"))
```

`demo` *Demonstrations of R functions* `demo`

Description

`demo` is a user-friendly interface to running some demonstration R scripts. `demo()` gives the list of available topics.

Usage

```
demo(topic, device = getOption("device"))
```

Arguments

`topic`	The topic which should be demonstrated. If omitted, the list of available topics is displayed.
`device`	The graphics device to be used.

See Also

`source` which is called by `demo`.

Examples

```
demo(lm.glm)
```

`density` *Kernel Density Estimation* `density`

Description

The function `density` computes kernel density estimates with the given kernel and bandwidth.

The generic functions `plot` and `print` have methods for density objects.

Usage

```
density(x, bw, adjust = 1,
        kernel=c("gaussian", "epanechnikov", "rectangular", "triangular",
                 "biweight", "cosine", "optcosine"),
        window = kernel, width,
        give.Rkern = FALSE,
        n = 512, from, to, cut = 3, na.rm = FALSE)
print(dobj)
plot(dobj, main = NULL, xlab = NULL, ylab = "Density", type = "l",
     zero.line = TRUE, ...)
```

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. 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") *unless* the quartiles coincide where `bw > 0` will be guaranteed. The specified (or default) 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.
`width`	this exists for compatibility with S; if given, and `bw` is not, will set `bw = width/4`.
`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.
`dobj`	a "density" object.
`main, xlab, ylab, type`	plotting parameters with useful defaults.
`...`	further plotting parameters.
`zero.line`	logical; if `TRUE`, add a base line at $y = 0$

Details

The algorithm used in `density` 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 `+/-Inf` and the density estimate is of the sub-density on `(-Inf, +Inf)`.

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

Silverman, B. W. (1986) *Density Estimation*. London: Chapman and Hall.

Venables, W. N. and B. D. Ripley (1994, 7, 9) *Modern Applied Statistics with S-PLUS*. New York: Springer.

Scott, D. W. (1992) *Multivariate Density Estimation. Theory, Practice and Visualization*. New York: Wiley.

Sheather, S. J. and Jones M. C. (1991) A reliable data-based bandwidth selection method for kernel density estimation. *J. Roy. Statist. Soc.* **B**, 683–690.

See Also

`hist`.

Examples

```
plot(density(c(-20,rep(0,98),20)), xlim = c(-4,4))# IQR = 0

# The Old Faithful geyser data
data(faithful)
d <- density(faithful$eruptions, bw = 0.15)
d
plot(d)

plot(d, type = "n")
polygon(d, col = "wheat")

## Missing values:
x <- xx <- faithful$eruptions
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(.01,10))
```

```
(kernels <- eval(formals(density)$kernel))

plot (density(0,bw = 1))
for(i in 2:length(kernels))
   lines(density(0,bw = 1, kern =  kernels[i]), col = i)
mtext(side = 3, "R's density() kernels with bw = 1")
legend(1.5,.4, leg = kernels, col = seq(kernels),lty = 1, cex = .8, y.int = 1)

(RKs <- cbind(sapply(kernels, function(k)density(kern = k, give.Rkern = TRUE))))
100*round(RKs["epanechnikov",]/RKs, 4) ## Efficiencies

data(precip)
plot(density(precip, n = 2^13))
for(i in 2:length(kernels))
   lines(density(precip, kern =  kernels[i], n = 2^13), col = i)
mtext(side = 3, "same scale bandwidths, 7 different kernels")

## 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, n = 2^13))
for(i in 2:length(kernels))
   lines(density(precip, adjust = h.f[i], kern =  kernels[i], n = 2^13),
         col = i)
mtext(side = 3, "equivalent bandwidths, 7 different kernels")
legend(55,.035, leg = kernels, col = seq(kernels), lty = 1)
```

`deparse` *Expression Deparsing* `deparse`

Description

Turn unevaluated expressions into character strings.

Usage

```
deparse(expr, width.cutoff = 60)
```

Arguments

`expr`	any R expression.
`width.cutoff`	integer in $[20, 500]$ determining the cutoff at which line-breaking is tried.

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 `expression`) into character strings (a kind of inverse `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 `deparse` and `substitute` to create labels for a plot which are character string versions of the actual arguments to the function `myplot`.

See Also

substitute, parse, expression.

Examples

```
deparse(args(lm))
deparse(args(lm), width = 100)
myplot <-
function(x, y)
    plot(x, y, xlab=deparse(substitute(x)),
        ylab=deparse(substitute(y)))
```

Deprecated *Deprecated Functions* Deprecated

Description

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

`.Deprecated("<new name>")` is called from deprecated functions.

Usage

```
.Deprecated(new)
```

See Also

Defunct

---

deriv *Symbolic and Algorithmic Derivatives of Simple Expressions* deriv

---

Description

Compute derivatives of simple expressions, symbolically.

Usage

```
D(expr, namevec)
deriv(expr, namevec, function.arg = NULL, tag = ".expr")
```

Arguments

`expr`	expression which should be differentiated.
`namevec`	character vector, giving the variable names with respect to which derivatives will be computed.
`function.arg`	*NOT YET IMPLEMENTED.* If specified, a function 'prototype' (with empty body) which will be used to return a function with the given argument list, instead of an expression.
`tag`	character; the prefix to be used for the locally created variables in result..

Details

`D` is modelled after its S namesake for taking simple symbolic derivatives.

`deriv` is a *generic* function with a default and a `formula` method. It returns a `call` for computing the `expr` and its (partial) derivatives, simultaneously. It uses so-called *"algorithmic derivatives"*.

Currently, `deriv.formula` just calls `deriv.default` after extracting the expression to the right of `~`.

Value

`D` returns an expression and therefore can easily be iterated for higher derivatives.

`deriv` returns a `call` object which becomes an `expression` when evaluated once. Evaluation of the latter expression returns the function values with a `".gradient"` attribute containing the gradient matrix.

Note

This help page should be fixed up by one of R&R or someone else who fluently speaks the language in '$R_HOME/src/main/deriv.c'.

It's author, MM, has only got a vague idea and thinks that a help page is better than none.

References

Griewank, A. and Corliss, G. F. (1991) *Automatic Differentiation of Algorithms: Theory, Implementation, and Application.* SIAM proceedings, Philadelphia.

See Also

`nlm` for numeric minimization which should make use of derivatives.

Examples

```
## formula argument :
dx2x <- deriv(~ x^2, "x") ; dx2x
expression({
         .value <- x^2
         .grad <- array(0, c(length(.value), 1), list(NULL, c("x")))
         .grad[, "x"] <- 2 * x
         attr(.value, "gradient") <- .grad
         .value
})
mode(dx2x)
x <- -1:2
eval(dx2x)

## Something 'tougher':
trig.exp <- expression(sin(cos(x + y^2)))
( D.sc <- D(trig.exp, c("x", "y")) )

( dxy <- deriv(trig.exp, c("x", "y")) )
y <- 1
eval(dxy)
eval(D.sc)
```

`detach` *Detach Objects from the Search Path* `detach`

Description

Detach a database, i.e., remove it from the `search()` patch of available R objects. Usually, this either a `data.frame` which has been `attach`ed or a package which was required previously.

Usage

```
detach(name, pos = 2)
```

Arguments

`name`	The object to detach. Defaults to `search()[pos]`.
`pos`	Index position in `search()` of database to detach. When `name` is `numeric`, `pos = name` is used.

Value

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

See Also

`attach`, `library`, `search`, `objects`.

Examples

```
require(eda)#package
detach(package:eda)
library(mva)
detach(2)# 'pos' used for 'name'
```

`dev.xxx` *Control Multiple Devices* `dev.xxx`

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.

A list of device names is stored in `.Devices`. The name of the active device is stored in `.Device`.

Usage

```
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.

`dev.list` returns the numbers and names of all devices.

`dev.next` returns the number and name of the next device in the list of devices.

`dev.prev` returns the number and name of the previous device in the list of 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; graphics.off for closing all devices; layout and its links for setting up plotting regions on the current device.

Examples

```
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
```

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`).

`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. This is most useful for producing a postscript copy from an on-screen device.

`dev.control` allows the user to control the recording of graphics operations in a device. If `displaylist` is "inhibit" then recording is turned off.

Usage

```
dev.copy(device, ..., which=dev.next())
dev.print(device=postscript, ...)
dev.control(displaylist)
```

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

Value

`dev.copy` returns the name and number of the device which has been copied to.

`dev.print` returns the name and number of the device which has been copied from.

Note

Every device has a display list which records all of the graphics operations that occur in the device. `dev.copy` and `dev.print` copy 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

```
x11()
plot(rnorm(10), main="Plot 1")
dev.copy(device=x11)
mtext("Copy 1", 3)
```

```
dev.print(width=6, height=6, horizontal=FALSE)
dev.off(dev.prev())
dev.off()
```

---

dev2bitmap *Copy Graphics Device to Bitmap File* dev2bitmap

Description

Copy the current graphics device to a file in a bitmap graphics format.

Usage

```
dev2bitmap(file, type = "png256", height = 6, width = 6, res = 72,
           pointsize, ...)
```

Arguments

file	The putput 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 passes to postscript.

Details

dev2bitmap works by copying the current device to a postscript device, and post-processing the output file using ghostscript. You will need a recent 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".

The types available will depend on the version of ghostscript, but are likely to include "pcxmono", "pcxgray", "pcx16", "pcx256", "pcx24b", "pcxcmyk", "pbm", "pbmraw", "pgm", "pgmraw", "pgnm", "pgnmraw", "pnm", "pnmraw", "ppm", "ppmraw", "pkm", "pkmraw", "tiffcrle", "tiffg3", "tiffg32d", "tiffg4", "tifflzw", "tiffpack", "tiff12nc", "tiff24nc", "psmono", "psgray", "psrgb", "bit", "bitrgb", "bitcmyk", "pngmono", "pnggray", "png16", "png256", "png16m", "jpeg", "jpeggray", "pdfwrite".

Value

None.

Warning

This is experimental and may be withdrawn in favour of an equivalent device in a later version.

Author(s)

B. D. Ripley

See Also

`postscript`

`deviance` *Model Deviance* `deviance`

Description

Returns the deviance of a fitted model object.

Usage

```
deviance(x, ...)
deviance.lm (x, ...)
deviance.glm(x, ...)
deviance.mlm(x, ...)
deviance.default(x, ...)
```

Arguments

`x`	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 details on how to use this function.

There is no default method for this function.

Value

The value of the deviance extracted from the object `x`.

See Also

`df.residual`, `extractAIC`, `glm`, `lm`.

Devices — List of Graphical Devices — Devices

Description

The following graphics devices are currently available:

- postscript Writes PostScript graphics commands to a file
- pictex Writes LaTeX/PicTeX graphics commands to a file
- x11 The graphics driver for the X11 Window system (Unix)

Usage

```
postscript(...)
pictex(...)
x11(...)
```

See Also

the individual help files for further information on any of the devices listed here;

dev.cur, dev.print, graphics.off, image, dev2bitmap

df.residual — Residual Degrees-of-Freedom — df.residual

Usage

```
df.residual(x, ...)
```

Arguments

x	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 details on how to use this function.

There is no default method for this function.

Value

The value of the residual degrees-of-freedom extracted from the object x.

See Also

deviance, glm, lm.

`diag` *Matrix Diagonals* `diag`

Description

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

Usage

```
diag(x, nrow, ncol)
diag(x) <- value
```

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 a 1-d array) then `diag(x)` returns a diagonal matrix whose diagonal is `x`. If `x` is an integer then `diag(x)` returns an identity matrix of order `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).

See Also

`matrix`.

Examples

```
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* `diff`

Description

Returns suitably lagged and iterated differences.

Usage

```
diff(x, ...)
diff.default(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.

Details

`NA`'s propagate.

Value

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

Examples

```
diff(1:10, 2)
diff(1:10, 2, 2)
```

`dim` *Dimensions of an Object* `dim`

Description

Retrieve or set the dimension of an object.

Usage

```
dim(x)
dim(x) <- values
```

Details

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

For an array (and hence in particular, for a matrix) they retrieve or set the `dim` attribute of the object.

`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").

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]
ncol0 <- function(x) dim(x)[2]
```

`dimnames` *Dimnames of an Object* `dimnames`

Description

Retrieve or set the dimnames of an object.

Usage

```
dimnames(x)
dimnames(x) <- nlist
```

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.

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

See Also

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

Examples

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

`discoveries` *Numbers of Important Discoveries* `discoveries`

Description

The numbers of "great" inventions and scientific discoveries in each year from 1860 to 1959.

Usage

```
data(discoveries)
```

Format

A time series of 100 values.

Source

The World Almanac and Book of Facts, 1975 Edition, pages 315–318.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* Wiley.

Examples

```
data(discoveries)
plot(discoveries, ylab = "Number of important discoveries",
     las = 1)
title(main = "discoveries data set")
```

`do.call` *Execute a Function Call* `do.call`

Description

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

Usage

```
do.call(what, args)
```

Arguments

`what`	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.

Value

The result of the (evaluated) function call.

See Also

`call` which creates an unevaluated call.

Examples

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

`dotplot` *Cleveland Dot Plots* `dotplot`

Description

Draw a Cleveland dot plot.

Usage

```
dotplot(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", ...)
```

Arguments

`x`	either a vector or matrix of numeric values (`NA`s 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.
`pch`	the plotting character or symbol to be used.
`gpch`	the plotting character or symbol to be usea for group values.
`bg`	the background color to be used.
`color`	the color to be used for points an labels.
`gcolor`	the color to be used for group labels and values.
`lcolor`	the color to be used for the horizontal lines.
`...`	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

Cleveland, W. S. (1985) *The Elements of Graphing Data.* Monterey, CA: Wadsworth.

Examples

```
data(VADeaths)
dotplot(VADeaths, main = "Death Rates in Virginia - 1940")
dotplot(t(VADeaths), main = "Death Rates in Virginia - 1940")
```

`double` *Double Precision Vectors* `double`

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)
```

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.

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

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.

See Also

`integer`.

Examples

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

`download.file` *Download File from the Internet* `download.file`

Description

This function can be used to download a file from the Internet either using a helper application such as `wget` or by making a direct socket connection.

Usage

```
download.file(url, destfile, method = "auto", quiet=FALSE)
```

Arguments

`url`	A character string with the URL of a file to be downloaded.
`destfile`	A character string with the name where the downloaded file is saved.
`method`	Tool to be used for downloading files. Currently download methods `"auto"`, `"wget"`, `"lynx"` `"cp"` and `"socket"` are available. To use `"wget"` or `"lynx"` the executable must be installed on your system and in your path. To use `"socket"` the file must be on an `HTTP` server, in plain text, and your system must allow socket connections to be opened to that server.
`quiet`	If `TRUE`, suppress status messages from the download tool (if any).

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`. It makes a system call to the tool given by `method`, the respective program must be installed on your system and be in the search path for executables. If method `"cp"` is used, then the `url` must start with `"file:"` and give the path to a local file. Method `"auto"` (the default) searches for available tools at runtime. The `url` must start with a type specification such as `"http://"`, `"ftp://"` or `"file:"`.

Note

Cacheing proxies and firewalls may well not allow method `"socket"` socket to be used.

`dput` *Write an Internal Object to a File* `dput`

Description

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

Usage

```
dput(x, file = "")
dget(file)
```

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.

Using `dget`, the object can be recreated (with the limitations mentioned above).

See Also

`deparse`, `dump`, `write`.

Examples

```
## Write an ASCII version of mean to the file "foo"
dput(mean, "foo")
## And read it back into 'bar'
bar <- dget("foo")
unlink("foo")
```

`drop` *Drop Redundant Extent Information* `drop`

Description

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

Usage

```
drop(x)
```

Value

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

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

See Also

`drop1` which is used for dropping terms in models.

Examples

```
dim(drop(array(1:12, dim=c(1,3,1,1,2,1,2))))# = 3 2 2
drop(1:3 %*% 2:4)# scalar product
```

`dummy.coef` *Extract Coefficients in Original Coding* `dummy.coef`

Description

This extracts coefficients in terms of the original levels of the coefficients rather than the coded variables.

Usage

```
dummy.coef(object, ...)
dummy.coef.lm(object, use.na = FALSE)
dummy.coef.aovlist(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.

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.

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.

Author(s)

B.D. Ripley

See Also

`aov`, `model.tables`

Examples

```
options(contrasts=c("contr.helmert", "contr.poly"))
## From Venables and Ripley (1997) p.210.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
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(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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)
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)
```

`dump` *Text Representations of R Objects* `dump`

Description

This function takes a vector of names of R objects and produces text representations of the objects in the file `fileout`. This file, can be `source`d into another R (or S) session.

Usage

```
dump(list, fileout="dumpdata")
```

Details

At present the implementation of `dump` is very incomplete and it really only works for functions and simple vectors.

The function `save` is designed to be used for transporting R data between machines.

See Also

`dput`, `dget`,`write`.

Examples

```
x <- 1; y <- 1:10
dump(ls(patt='^[xyz]'), "xyz.Rdmped")
unlink("xyz.Rdmped")
```

`duplicated` *Determine Duplicate Elements* `duplicated`

Description

Determines which elements of a vector are duplicates of elements with smaller subscripts, and returns a logical vector indicating which elements are duplicates.

Usage

```
duplicated(x)
```

Arguments

`x`	an atomic vector

See Also

`unique`.

Examples

```
x <- c(9:20, 1:5, 3:7, 0:8)
## extract unique elements
(xu <- x[!duplicated(x)])
all(xu == unique(x)) # TRUE, but unique(x) is more efficient
all(0:20  == sort(x[!duplicated(x)]))
```

`dyn.load` *Foreign Function Interface* `dyn.load`

Description

Load or unload shared libraries, and test whether a C function or Fortran subroutine is available.

Usage

```
dyn.load(libname, local = TRUE, now = TRUE)
dyn.unload(libname)

is.loaded(symbol)
symbol.C(name)
symbol.For(name)
```

Arguments

`libname`	a character string giving the pathname to a DLL.
`local`	a logical value controlling whether the symbols in the DLL are stored in their own local table and not shared across DLLs, or added to the global symbol table. Whether this has any effect is system-dependent.
`now`	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.
`symbol`	a character string giving a symbol name.
`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

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 one should override them only if there is good reason and you understand the implications.

The `local` argument allows one to control whether the symbols in the DLL being attached are visible to other DLLs. While maintaining the symbols in their own namespace is good practice, the ability to share symbols across related "chapters" is useful in many cases. Additionally, on certain platforms and versions of an operating system, certain libraries must have their symbols loaded globally to successfully resolve all symbols.

One should be careful of the potential side-effect of using lazy loading via the 'now' argument as `FALSE`. If a routine is called that has a missing symbol, the process will terminate immediately and unsaved session variables will be lost. The intended use is for library developers to call specify a value `TRUE` to check that all symbols are actually resolved and for regular users to all with `FALSE` so that missing symbols can be ignored and the available ones can be called.

The initial motivation for adding these was to avoid such termination in the `_init()` routines of the Java virtual machine library. However, symbols loaded locally may not be (read probably) available to other DLLs. Those added to the global table are available to all other elements of the application and so can be shared across two different DLLs.

Some systems do not provide (explicit) support for local/global and lazy/eager symbol resolution. This can be the source of subtle bugs. One can arrange to have warning messages emitted when unsupported options are used. This is done by setting either of the options `verbose` or `warn` to be non-zero via the `options` function. Currently, we know of only 2 platforms that do not provide a value for local load (RTLD_LOCAL). These are IRIX6.4 and unpatched versions of Solaris 2.5.1.

There is a short discussion of these additional arguments with some example code available at http://cm.bell-labs.com/stat/duncan/R/dynload.

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`, `.Fortran` and `.External` can then be used to execute compiled C functions or Fortran subroutines contained in the library.

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: `is.loaded` checks if the symbol name is loaded and hence available for use in `.C` or `.Fortran`.

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.

See Also

`library.dynam` to be used inside a package's `.First.lib` initialization.

`.C`, `.Fortran`, `.External`, `.Call`.

Examples

```
is.loaded(symbol.For("hcass2")) #-> probably  FALSE
library(mva)
is.loaded(symbol.For("hcass2")) #-> TRUE
```

`edit` *Invoke a Text Editor* `edit`

Description

Invoke a text editor on an R object.

Usage

```
edit(name = NULL, file = "", editor = getOption("editor"))
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.
`editor`	a string naming the text editor you want to use. On Unix the default is set from the `EDITOR` environmental variable. On Windows it defaults to `notepad`.

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.

`data.entry` can be used to edit data, and is used by `edit` to edit 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. Quiting 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.

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

```
# 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")
```

`edit.data.frame` *Edit data frames* `edit.data.frame`

Description

Use data editor on data frame contents.

Usage

```
edit.data.frame(name, factor.mode=c("numeric", "character"))
```

Arguments

`name`	A data frame.
`factor.mode`	How to handle factors (as integers or using character levels).

Details

At present, this only works on simple data frames containing numeric 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 the editing; invalid levels are changed to NA and a warning is issued.

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`.

Author(s)

Peter Dalgaard

See Also

`data.entry`, `edit`

Examples

```
data(airquality)
edit(airquality)
edit(airquality, factor.mode="character")
```

`eff.aovlist` *Compute Efficiencies of Multistratum Analysis of Variance* `eff.aovlist`

Description

Computes the efficiencies of fixed-effect terms in an analysis of variance model with multiple strata.

Usage

```
eff.aovlist(aovlist)
```

Arguments

`aovlist`	The result of a call to `aov` with a `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 is the fraction of the maximum possible precision (inverse variance) obtainable by estimating in just that stratum.

This is used to pick strata in which to estimate terms in `model.tables.aovlist` and elsewhere.

Value

A matrix giving for each non-pure-error stratum (row) the efficiencies for each fixed-effect term in the model.

Author(s)

B.D. Ripley

See Also

`aov`, `model.tables.aovlist`, `se.contrast.aovlist`

Examples

```
## for balanced designs all efficiencies are zero or one.
## so as a statistically meaningless test:
options(contrasts=c("contr.helmert", "contr.poly"))
## From Venables and Ripley (1997) p.210.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
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(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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)
npk.aovE <- aov(yield ~  N*P*K + Error(block), npk)
eff.aovlist(npk.aovE)
```

`effects` *Effects from Fitted Model* `effects`

Description

Returns (orthogonal) effects from a fitted model, usually a linear model. This is a generic function, but currently only has a method for objects inheriting from class `"lm"`.

Usage

```
effects(object, ...)
effects.lm(object, set.sign=FALSE)
```

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.

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).

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.

See Also

coef

Examples

```
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
```

`eigen` *Spectral Decomposition of a Matrix* `eigen`

Description

This function computes eigenvalues and eigenvectors by providing an interface to the EISPACK routines `RS`, `RG`, `CH` and `CG`.

Usage

```
eigen(x, symmetric, only.values=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.

Value

The spectral decomposition of `x` is returned as components of a list.

`values`	a vector containing the p eigenvalues of `x`, sorted in *decreasing* order, according to `Mod(values)` if they are complex.
`vectors`	a $p \times p$ matrix whose columns contain the eigenvectors of `x`, or `NULL` if `only.values` is `TRUE`.

Note

To compute the determinant of a matrix (do you *really* need it?), it is much more efficient to use the QR decomposition, see `qr`.

References

Smith, B. T, Boyle, J. M., Dongarra, J. J., Garbow, B. S., Ikebe,Y., Klema, V., and Moler, C. B. (1976). *Matrix Eigensystems Routines – EISPACK Guide*. Springer-Verlag Lecture Notes in Computer Science.

See Also

svd, a generalization of eigen; qr, and chol for related decompositions.

Examples

```
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

Meps <- .Alias(.Machine$double.eps)
m <- matrix(round(rnorm(25),3), 5,5)
sm <- m + t(m) #- symmetric matrix
em <- eigen(sm); V <- em$vect
print(lam <- em$values) # ordered DEcreasingly

all(abs(sm %*% V - V %*% diag(lam))          < 60*Meps)
all(abs(sm       - V %*% diag(lam) %*% t(V)) < 60*Meps)

##------- Symmetric = FALSE:  -- different to above : ---

em <- eigen(sm, symmetric = FALSE); V2 <- em$vect
print(lam2 <- em$values) # ordered decreasingly in ABSolute value !
                         # and V2 is not normalized (where V is):
print(i <- rev(order(lam2)))
all(abs(1 - lam2[i] / lam) < 60 * Meps)# [1] TRUE

zapsmall(Diag <- t(V2) %*% V2) # orthogonal, but not normalized
print(norm2V <- apply(V2 * V2, 2, sum))
all( abs(1- norm2V / diag(Diag)) < 60*Meps) #> TRUE

V2n <- sweep(V2,2, STATS= sqrt(norm2V), FUN="/")## V2n are now Normalized EV
apply(V2n * V2n, 2, sum)
##[1] 1 1 1 1 1

## Both are now TRUE:
all(abs(sm %*% V2n - V2n %*% diag(lam2))            < 60*Meps)
all(abs(sm         - V2n %*% diag(lam2) %*% t(V2n)) < 60*Meps)

## Re-ordered as with symmetric:
sV <- V2n[,i]
slam <- lam2[i]
all(abs(sm %*% sV -  sV %*% diag(slam))             < 60*Meps)
all(abs(sm        -  sV %*% diag(slam) %*% t(sV)) < 60*Meps)
## sV  *is* now equal to V  -- up to sign (+-) and rounding errors
all(abs(c(1 - abs(sV / V)))       <     1000*Meps) # TRUE (P ~ 0.95)
```

Description

Get, set, test for and create environments.

Usage

```
environment(fun = NULL)
environment(fun) <- value
is.environment(obj)
.GlobalEnv
new.env()
```

Arguments

`fun`	a `function` or `NULL`, which is the default.
`value`	
`obj`	an arbitrary R object.

Value

If `fun` is a function then `environment(fun)` returns the environment associated with that function. If `fun` is `NULL` then the global environment, `.GlobalEnv`, is returned. This variable is currently defined as `globalenv()`.

The assignment form sets the environment of the function `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.

See Also

The `envir` argument of `eval`.

Examples

```
##-- all three give the same:
environment()
environment(environment)
.GlobalEnv

ls(envir=environment(approxfun(1:2,1:2, method="const")))

is.environment(.GlobalEnv)# TRUE
```

`esoph` *Smoking, Alcohol and (O)esophageal Cancer* `esoph`

Description

Data from a case-control study of (o)esophageal cancer in Ile-et-Vilaine, France.

Usage

```
data(esoph)
```

Format

data frame with records for 88 age/alcohol/tobacco combinations.

[,1]	"agegp"	Age group	1 25–34 years
			2 35–44
			3 45–54
			4 55–64
			5 65–74
			6 75+
[,2]	"alcgp"	Alcohol consumption	1 0–39 gm/day
			2 40–79
			3 80–119
			4 120+
[,3]	"tobgp"	Tobacco consumption	1 0– 9 gm/day
			2 10–19
			3 20–29
			4 30+
[,4]	"ncases"	Number of cases	
[,5]	"ncontrols"	Number of subjects	

Author(s)

Thomas Lumley

Source

Breslow, N. E. and Day, N. E. (1980) *Statistical Methods in Cancer Research. 1: The Analysis of Case-Control Studies.* IARC Lyon / Oxford University Press.

Examples

```
data(esoph)
summary(esoph)
## effects of alcohol, tobacco and interaction, age-adjusted
model1 <- glm(cbind(ncases, ncontrols) ~ agegp + tobgp * alcgp,
              data = esoph, family = binomial())
anova(model1)
## Try a linear effect of alcohol and tobacco
model2 <- glm(cbind(ncases, ncontrols) ~ agegp + codes(tobgp)
                                       + codes(alcgp),
              data = esoph, family = binomial())
summary(model2)
## Re-arrange data for a mosaic plot
ttt <- table(esoph$agegp, esoph$alcgp, esoph$tobgp)
ttt[ttt == 1] <- esoph$ncases
tt1 <- table(esoph$agegp, esoph$alcgp, esoph$tobgp)
tt1[tt1 == 1] <- esoph$ncontrols
tt <- array(c(ttt, tt1), c(dim(ttt),2),
            c(dimnames(ttt), list(c("Cancer", "control"))))
mosaicplot(tt, main = "esoph data set", color = TRUE)
```

`euro` *Euro Conversion Rates* `euro`

Description

Conversion rates between the various Euro currencies.

Usage

```
data(euro)
```

Format

`euro` is a named vector of length 11, `euro.cross` a named matrix of size 11 by 11.

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 Portugese 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

```
data(euro)
cbind(euro)
all(euro == signif(euro,6))# TRUE: [6 digit precision in Euro's definition]

all(euro.cross == outer(1/euro, euro)) # TRUE

## 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"]

dotplot(euro, main = "euro data: 1 Euro in currency unit")
dotplot(1/euro, main = "euro data: 1 currency unit in Euros")
dotplot(log(euro, 10), main = "euro data: log10(1 Euro in currency unit)")
```

`eurodist` *Distances Between Cities in Europe* `eurodist`

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

```
data(eurodist)
```

Format

A `dist` object based on 21 objects. (You must have the mva package loaded to have the methods for this kind of object available).

Source

Crystal, D. Ed. (1990) *The Cambridge Encyclopaedia.* Cambridge: Cambridge University Press,

`eval` *Evaluate an (Unevaluated) Expression* `eval`

Description

Evaluate an R expression in a specified environment.

Usage

```
eval(expr, envir = sys.frame(sys.parent()),
     enclos = if(is.list(envir) || is.pairlist(envir))
                   sys.frame(sys.parent()))
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 a list or an integer as in `sys.call`.
`enclos`	Only relevant if `envir` is a list. 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))`.

`local` evaluates an expression in a local environment. It is equivalent to `evalq` except the 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 dataframes 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, sys.frame(sys.parent()))`.

See Also

`expression`, `quote`, `sys.frame`, `environment`.

Examples

```
eval(2 ^ 2 ^ 3)
mEx <- expression(2^2^3); mEx; 1 + eval(mEx)
eval({ xx <- pi; xx^2}) ; xx

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 <- sys.frame(sys.parent())
   ## 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
        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))))
```

`example` *Run an Examples Section from the Online Help* `example`

Description

Run all the R code from the `EXAMPLES` part of R's online help topic `topic`.

Usage

```
example(topic, package = .packages(), lib,
        echo = TRUE, verbose = getOption("verbose"),
        prompt.echo = paste(abbreviate(topic, 6),"> ", sep=""))
```

Arguments

`topic`	name or character: The online `help` topic the examples of which should be run.
`package`	a character vector with package names.
`lib`	a character vector with path names of R package libraries.
`echo`	logical; if `TRUE`, show the R input when sourcing.
`verbose`	logical; if `TRUE`, show even more when running example code.
`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. If `lib.loc` is specified, they are searched for 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 pacakge loaded from another location.

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'.

Author(s)

Martin Maechler and others

See Also

demo

Examples

```
example("smooth", package="eda", lib.loc=.lib.loc)
example(dbinom)
```

`exists` *Is an Object Defined?* `exists`

Description

Search for an R object of the given name on the search path.

Usage

```
exists(x, where = NULL, envir = sys.frame(sys.parent()),
       frame = NULL, mode = "any", inherits = TRUE)
```

Arguments

`x`	a variable name (given as a character string).
`where, envir, frame`	an environment to be searched. By default this is the environment where the call to `envir` takes place.
`mode`	the type of interest for the object.
`inherits`	should the enclosing frames of the environment be inspected.

Details

This function looks to see if the name `x` has a value bound to it. If `inherits` is `TRUE` and a value is not found for `x`, then the parent frames of `envir` are searched until the name `x` is encountered. **Warning:** This is the default behaviour for R but not for S.

If `mode` is specified then only objects of that mode are sought. The function returns `TRUE` if the variable is encountered and `FALSE` if not.

See Also

get.

Examples

```
##  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=F) # false
```

`expand.grid` *Create a Data Frame from All Combinations of Factors* `expand.grid`

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

```
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.

Author(s)

B.D. Ripley

Examples

```
expand.grid(height = seq(60, 80, 5), weight = seq(100, 300, 50),
            sex = c("Male","Female"))
```

`Exponential` *The Exponential Distribution* `Exponential`

Description

Density, distribution function, quantile function and random generation for the exponential distribution with rate `rate` (i.e., mean `1/rate`).

Usage

```
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 to generate.
`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 λ 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.

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

```
dexp(1) - exp(-1) #-> 0
r <- rexp(100)
all(abs(1 - dexp(1, r) / (r*exp(-r))) < 1e-14)
```

`expression` *Unevaluated Expressions* `expression`

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.

See Also

`call`, `eval`, `function`. Further, `text` and `legend` for plotting math expressions.

Examples

```
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* `Extract`

Description

Operators act on vectors, arrays, dataframes and lists to extract or replace subsets.

Usage

```
x[i]
x[i, j, ...]
x[i, j, ... , drop=TRUE]
x[[i]]
x[[i, j, ...]]
x$name
```

Details

If one of these expressions appears on the left side of an assignment then that part of `x` is set to the value of the right hand side of the assignment.

These operators are generic. You can write methods to handle subsetting of specific classes of data.

The `[[` operator requires all relevant subscripts be supplied. With the `[` operator a comma separated blank indicates that all entries in that dimension are selected.

When operating on a list, the `[[` operator gives the specified element of the list while the `[` operator returns a list with the specified element(s) in it.

See Also

`list`, `array`, `matrix`.

Examples

```
x <- 1:12; m <- matrix(1:6,nr=2); li <- list(pi=pi, e = exp(1))
x[10]                 # the tenth element of x
m[1,]                 # the first row of matrix m
m[1, , drop = FALSE]  # is a 1-row matrix
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
```

`extractAIC` *Extract AIC from a Fitted Model* `extractAIC`

Description

Computes the (generalized) **A**kaike **I**nformation **C**riterion for a fitted parametric model.

Usage

```
extractAIC    (fit, scale,     k = 2, ...)
extractAIC.lm (fit, scale = 0, k = 2, ...)
extractAIC.glm(fit, scale = 0, k = 2, ...)
extractAIC.aov(fit, scale = 0, k = 2, ...)
extractAIC.coxph  (fit, scale, k = 2, ...)
extractAIC.negbin (fit, scale, k = 2, ...)
extractAIC.survreg(fit, scale, k = 2, ...)
```

Arguments

`fit`	fitted model, usually the result of a fitter like `lm`.
`scale`	optional numeric specifying the scale parameter of the model, see `scale` in `step`.
`k`	numeric specifying the "weight" of the *equivalent degrees of freedom* (≡`edf`) part in the AIC formula.
`...`	further arguments (currently unused in base R).

Details

The criterion used is

$$AIC = -2 \log L + k \times \text{edf},$$

where L is the likelihood and `edf` the equivalent degrees of freedom (i.e., the number of parameters for usual parametric models) of `fit`.

For generalized linear models (i.e., for `lm`, `aov`, and `glm`), $-2 \log L$ is the *deviance*, as computed by `deviance(fit)`.

`k = 2` corresponds to the traditional AIC, using `k = log(n)` provides the BIC (Bayes IC) instead.

For further information, particularly about `scale`, see `step`.

Value

A numeric vector of length 2, giving

`edf`	the "**e**quivalent **d**egrees of **f**reedom" of the fitted model `fit`.
`AIC`	the (generalized) Akaike Information Criterion for `fit`.

Note

These functions are used in `add1`, `drop1` and `step` and that may be their main use.

Author(s)

B. D. Ripley

References

Venables, W. N. and Ripley, B. D. (1997) *Modern Applied Statistics with S-PLUS*. New York: Springer (2nd ed).

See Also

`deviance`, `add1`, `step`

Examples

```
example(glm)
extractAIC(glm.D93)#>>  5  15.129
```

`Extremes` *Maxima and Minima* `Extremes`

Description

Returns the (parallel) maxima and minima of the input values.

Usage

```
max(..., na.rm=FALSE)
min(..., na.rm=FALSE)

pmax(..., na.rm=FALSE)
pmin(..., 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 `double`. 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 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).

See Also

`range`.

Examples

```
min(5:1,pi)
pmin(5:1, pi)
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")
```

`factor` *Factors* `factor`

Description

The function `factor` is used to encode a vector as a factor (the names 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

```
factor(x, levels = sort(unique(x), na.last = TRUE), labels,
       exclude = NA, ordered = FALSE)
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 regraded 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)` 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.

Value

`factor` returns an object of class `"factor"` which has a set of numeric 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")`.

`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.

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.

See Also

`gl` for construction of "balanced" factors and `C` for factors with specified contrasts. `levels` and `nlevels` for accessing the levels, and `codes` to get integer codes.

Examples

```
ff <- factor(substring("statistics", 1:10, 1:10), levels=letters)
ff
codes(ff)
factor(ff)# drops the levels that do not occur
factor(factor(letters[7:10])[2:3]) # exercise indexing and reduction
factor(letters[1:20], label="letter")

class(ordered(4:1))# "ordered", inheriting from "factor"
```

`factor.scope` — *Compute Allowed Changes in Adding to or Dropping from a Formula* — `factor.scope`

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`	the terms or formula for the base model.
`terms2`	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.

Author(s)

B.D. Ripley

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"
```

`faithful` *Old Faithful Geyser Data* `faithful`

Description

The 'faithful' data frame has 272 rows and 2 columns; the waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

Usage

```
data(faithful)
```

Format

A data frame with 272 observations on 2 variables.

[,1]	eruptions	numeric	Eruption time in mins
[,2]	waiting	numeric	Waiting time to next eruption

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

Härdle, W. (1991) *Smoothing Techniques with Implementation in S.* New York: Springer.

Azzalini, A. and Bowman, A. W. (1990). A look at some data on the Old Faithful geyser. *Applied Statistics* **39**, 357–365.

See Also

`geyser` in package `MASS` for the Azzalini–Bowman version.

Examples

```
data(faithful)
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")
```

`family` *Family Objects for Models* `family`

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")

print.family(x, ...)
```

Arguments

`link`	a specification for the model link function. The `binomial` family admits the links `"logit"`, `"probit"`, `"log"`, and `"cloglog"` (complementary log-log); the `Gamma` family the links `"identity"`, `"inverse"`, and `"log"`; the `poisson` family the links `"identity"`, `"log"`, and `"sqrt"`; the `quasi` family 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. The other families have only one permissible link function: `"identity"` for the `gaussian` family, and `"1/mu^2"` for the `inverse.gaussian` 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`).

References

McCullagh P. and Nelder, J. A. (1989) *Generalized Linear Models.* London: Chapman and Hall.

Dobson, A. J. (1983) *An Introduction to Statistical Modelling.* London: Chapman and Hall.

Cox, D. R. and Snell, E. J. (1981). *Applied Statistics; Principles and Examples.* London: Chapman and Hall.

See Also

`glm`, `power`.

Examples

```
nf <- gaussian()# Normal family
nf
str(nf)# internal STRucture

gf <- Gamma()
gf
str(gf)
gf$linkinv
all(1:10 == gf$linkfun(gf$linkinv(1:10)))# is TRUE
gf$variance(-3:4) #- == (.)^2

## tests of quasi
x <- rnorm(100)
y <- rpois(100, exp(1+x))
glm(y ~x, family=quasi(var="mu", link="log"))
# which is the same as
glm(y ~x, family=poisson)
glm(y ~x, family=quasi(var="mu^2", link="log"))
glm(y ~x, family=quasi(var="mu^3", link="log")) # should fail
y <- rbinom(100, 1, plogis(x))
# needs to set a starting value for the next fit
glm(y ~x, family=quasi(var="mu(1-mu)", link="logit"), start=c(0,1))
```

`FDist` *The F Distribution* `FDist`

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 to generate.
`df1, df2`	degrees of freedom.
`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 F distribution with `df1` $= n_1$ and `df2` $= n_2$ degrees of freedom has density

$$f(x) = \frac{\Gamma(n_1/2 + n_2/2)}{\Gamma(n_1/2)\Gamma(n_2/2)} \left(\frac{n_1}{n_2}\right)^{n_1/2} x^{n_1/2-1} \left(1 + \frac{n_1 x}{n_2}\right)^{-(n_1+n_2)/2}$$

for $x > 0$.

Value

`df` gives the density, `pf` gives the distribution function `qf` gives the quantile function, and `rf` generates random deviates.

See Also

`dt` for Student's t distribution, the square of which is (almost) equivalent to the F distribution with `df2` $= 1$.

Examples

```
df(1,1,1) == dt(1,1)# TRUE

## 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, df1=1, df2=df), qt(p, df)^2), .90)# ~= 7e-9
```

`fft` *Fast Discrete Fourier Transform* `fft`

Description

Performs the Fast Fourier Transform of an array.

Usage

```
fft(z, inverse = FALSE)
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 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

Singleton, R. C. (1979) Mixed Radix Fast Fourier Transforms, in *Programs for Digital Signal Processing*, IEEE Digital Signal Processing Committee eds. IEEE Press.

See Also

`convolve`, `nextn`.

Examples

```
x <- 1:4
fft(x)
all(fft(fft(x), inverse = TRUE)/(x*length(x)) == 1+0i)
eps <- 1e-11 ## In general, not exactly, but still:
for(N in 1:130) {
    cat("N=",formatC(N,wid=3),": ")
    x <- rnorm(N)
    if(N %% 5 == 0) {
        m5 <- matrix(x,ncol=5)
```

```
        cat("mvfft:",all(apply(m5,2,fft) == mvfft(m5)),"")
    }
    dd <- Mod(1 - (f2 <- fft(fft(x), inverse=TRUE)/(x*length(x))))
    cat(if(all(dd < eps))paste(" all < ", formatC(eps)) else
            paste("NO: range=",paste(formatC(range(dd)),collapse=",")),"\n")
}

plot(fft(c(9:0,0:13, numeric(301))), type = "l")
periodogram <- function(x, mean.x = mean(x)) { # simple periodogram
  n <- length(x)
  x <- unclass(x) - mean.x
  Mod(fft(x))[2:(n%/%2 + 1)]^2 / (2*pi*n) # drop I(0)
}
data(sunspots)
plot(10*log10(periodogram(sunspots)), type = "b", col = "blue")
```

`file` *File Manipulation* `file`

Description

These functions provide a very basic interface to the computer's filestore.

Usage

```
file.create(...)
file.exists(...)
file.remove(...)
file.append(file1, file2)
dir.create(path)
basename(path)
dirname(path)
```

Arguments

`..., file1, file2, path`
: character vectors, containing file names.

Details

The `...` arguments are concatenated to form one character string: you can specify the files separately or as one vector.

`file.create` creates files with the given names if they do not already exist and truncates them if they do. It returns a logical vector indicating the success or failure of the operation for each file.

`file.exists` returns a logical vector indicating whether the files named by its argument exist.

`file.remove` attempts to remove the files named in its argument. It returns a logical vector indicating whether or not it succeeded in removing each file.

`file.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.

`dir.create` creates the last element of the path. It returns a logical, true for success.

`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.

In both `basename` and `dirname` trailing file separators are removed before dissecting the path, and for `dirname` any trailing file separators are removed from the result.

Author(s)

Ross Ihaka, Brian Ripley

See Also

`file.show`, `list.files`.

Examples

```
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")

file.remove("A","B")

basename(file.path("","p1","p2","p3","filename"))
dirname(file.path("","p1","p2","p3","filename"))
```

---

file.choose — *Choose a File Interactively* — file.choose

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.

`file.path` *Construct Path to File* `file.path`

Description

Construct the path to a file from components in a platform-independent way.

Usage

```
file.path(..., fsep=.Platform$file.sep)
```

Arguments

`...`	Character vectors
`fsep`	The path separator to use

Value

A character vector of length one.

`file.show` *Display One or More Files* `file.show`

Description

This function provides the core of the R help system, but it can be used for other purposes as well.

Usage

```
file.show(..., header, 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`	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 separate window is used for the display, `title` will be used as the window title.
`delete.file`	should the files be deleted after display? Used for temporary files.
`pager`	the pager to be used.

Note

How the pager is implemented is system dependent. At present the Unix version concatenates files and displays them in the pager set by the `pager` option setting. However, a GUI model might be to put up a file in a separate window and let the user leave it up while R continues running.

Author(s)

Ross Ihaka, Brian Ripley.

See Also

file, list.files, help.

Examples

```
file.show(paste(R.home(),
    "COPYRIGHTS", sep=.Platform$file.sep))
```

---

filled.contour *Level (Contour) Plots* filled.contour

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),
               nlevels = 20,  levels = pretty(zlim, nlevels),
               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,
               ...)
```

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. 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 the main plot.
`plot.axes`	statements which draw axes 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.
`...`	additional graphical parameters.

Note

This function currently uses the `layout` function and so is restricted to a full page display. In future it is likely to be replaced by a genuine `levelplot` function which will work in multipanel displays.

Author(s)

Ross Ihaka.

References

Cleveland, W. S. (1993) *Visualizing Data*. Summit, New Jersey: Hobart.

See Also

`contour`, `image`, `palette`.

Examples

```
data(volcano)
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\n(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)
```

`fitted.values` *Extract Model Fitted Values* `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`.)

Usage

```
fitted(x, ...)
fitted.values(x, ...)
```

Arguments

`x`	an object for which the extraction of model fitted values is meaningful.
`...`	other arguments.

Value

Fitted values extracted from the object `x`.

See Also

`coefficients`, `glm`, `lm`, `residuals`.

`fivenum` *Tukey Five-Number Summaries* `fivenum`

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 `NA`s and +/-`Inf`s.
`na.rm`	logical; if `TRUE`, all `NA` and `NaN`s are dropped, before the statistics are computed.

Value

A numeric vector of length 5 containing the summary information.

See Also

IQR, boxplot.stats, median, quantile, range.

Examples

```
fivenum(c(rnorm(100),-1:1/0))
```

`fix` *Fix an Object* `fix`

Description

`fix` invokes the editor specified in `options("editor")` on `x` and then assigns the new (edited) version of `x` in the global environment.

Usage

```
fix(x, ...)
```

Arguments

`x`	An R object
`...`	Arguments to pass to editor

See Also

edit, edit.data.frame

Examples

```
## 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 ...
```

`Foreign` *Foreign Function Interface* `Foreign`

Description

Functions to make calls to compiled code that has been loaded into R.

Usage

```
        .C(name, ..., NAOK = FALSE, DUP = TRUE, PACKAGE)
 .Fortran(name, ..., NAOK = FALSE, DUP = TRUE, PACKAGE)
.External(name, ...)
    .Call(name, ...)
```

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 that no other package can override their external symbols by using this argument. 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` can be used to call compiled code that uses R objects in the same way as internal R functions.

`.Call` 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).

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.

`.External` and `.Call` return an R object.

These calls are typically made in conjunction with `dyn.load` which links DLLs to R.

Argument types

The mapping of the types of R arguments to C or Fortran arguments in `.C` or `.Fortran` is

R	C	Fortran
integer	int *	integer
numeric	double *	double precision
– or –	float *	real
complex	Rcomplex *	double complex
logical	int *	integer
character	char **	[see below]
list	SEXP *	not allowed
other	SEXP	not allowed

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 use 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.

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 *`.

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 three important dangers with `DUP=FALSE`. The first is that garbage collection may move the object, resulting in the pointers pointing nowhere useful and causing hard-to-reproduce bugs.

The second is that if you pass a formal parameter of the calling function to `.C`/`.Fortran` with `DUP=FALSE`, it may not necessarily be copied. You may be able to change not only the local variable but the variable one level up. This will also be very hard to trace.

The third 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`.

1. If your C/Fortran routine calls back any R function including `S_alloc`/`R_alloc` then do not use `DUP=FALSE`. Do not even think about it. Calling almost any R function could trigger garbage collection.

2. If you don't trigger garbage collection it is safe and useful to set `DUP=FALSE` if you don't change any of the variables that might be affected, e.g.,

`.C("Cfunction", input=x, output=numeric(10))`.

In this case the output variable didn't exist before the call so it can't cause trouble. If the input variable is not changed in `Cfunction` you are safe.

See Also

`dyn.load`.

`Formaldehyde` *Determination of Formaldehyde* `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

```
data(Formaldehyde)
```

Format

A data frame with 6 observations on 2 variables.

[,1]	carb	numeric	Carbohydrate (ml)
[,2]	optden	numeric	Optical Density

Source

Bennett, N. A. and N. L. Franklin (1954) *Statistical Analysis in Chemistry and the Chemical Industry.* New York: Wiley.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(Formaldehyde)
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)
opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(fm1)
par(opar)
```

formals — *Access to and Manipulation of the Formal Arguments* — formals

Description

Get or set the formal arguments of a function.

Usage

```
formals(fun = sys.function(sys.parent()))
formals(fun) <- list
```

Arguments

`fun`	a function object or a character string naming the function to be manipulated. If not specified, the function calling `body` is used.
`list`	a list of R expressions.

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
```

format — *Encode in a Common Format* — format

Description

Format an R object for pretty printing: `format.pval` is intended for formatting p-values.

Usage

```
format(x, ...)
format.default(x, trim = FALSE, digits = getOption("digits"))
format.pval(x, digits = max(1, getOption("digits") - 2),
            eps = .Machine$double.eps)
```

Arguments

`x`	any R object (conceptually); typically numeric.
`trim`	logical; if `TRUE`, leading blanks are trimmed off the strings.
`digits`	how many significant digits are to be used for `numeric` `x`. This is a suggestion: enough decimal places will be used so that the smallest (in magnitude) number has this many significant digits.

Details

These functions convert their first argument to a vector (or array) of character strings which have a common format (as is done by `print`). The trimming with `trim = TRUE` is useful when the strings are to be used for plot `axis` annotation.

`format.pval` is mainly an auxiliary function for `print.summary.lm` etc., does separate formatting for fixed, floating point and very small values (those `< eps`).

The function `formatC` provides a rather more flexible formatting facility for numbers, but does *not* provide a common format for several numbers.

Note

Currently `format` loses trailing zeroes, so `format(6.001, digits=2)` gives `"6"` and `format(c(6.0, 13.1), digits=2)` gives `c(" 6", "13")`.

Character(s) `"` in input strings `x` are escaped to `\"`.

See Also

`formatC`, `paste`, `as.character`.

Examples

```
format(1:10)

p <- c(47,13,2,.1,.023,.0045, 1e-100)/1000
format.pval(p)
format.pval(p / 0.9)
format.pval(p / 0.9, dig=3)
```

`format.info` *format(.) Information* `format.info`

Description

Information is returned on how `format(x, digits = options("digits"))` would be formatted like.

Usage

```
format.info(x)
```

Arguments

`x`	(numeric) vector; potential argument of `format(x,...)`.

Value

An integer vector of length 3, say r.

r[1]	width (number of characters) used for format(x)
r[2]	number of digits after decimal point.
r[3]	in 0:2; if ≥1, *exponential* representation would be used, with exponent length of r[3]+1.

Note

The result **depends** on the value of options("digits").

See Also

format, formatC.

Examples

```
dd <- options("digits") ; options(digits = 7) #-- for the following
format.info(123) # 3 0 0
format.info(pi)  # 8 6 0
format.info(1e8) # 5 0 1 - exponential "1e+08"
format.info(1e222)#6 0 2 - exponential "1e+222"

x <- pi*10^c(-10,-2,0:2,8,20)
names(x) <- formatC(x,w=1,dig=3,format="g")
cbind(sapply(x,format))
t(sapply(x, format.info))

# Reset old options:
options(dd)
```

---

formatC *Formatting Using C-style Formats* formatC

Description

Formatting numbers individually and flexibly, using C style format specifications. format.char is a helper function for formatC.

Usage

```
formatC(x, digits = NULL, width = NULL,
        format = NULL, flag = "", mode = NULL)
format.char(x, width = NULL, flag = "-")
```

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 number of *significant* digits. Note that this can lead to quite long result strings, see examples below.
`flag`	format modifier as in Kernighan and Ritchie (1988, page 243). `"0"` pads leading zeros; `"-"` does left adjustment, others are `"+"`, `" "`, and `"#"`.
`mode`	`"double"` (or `"real"`), `"integer"` or `"character"`. Default: Determined from the storage mode of `x`.

Details

If you set `format` it over-rides 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`.

Value

A character object of same size and attributes as `x`. Unlike `format`, each number is formatted individually. Looping over each element of `x`, `sprintf(...)` is called (inside the C function `str_signif`).

`format.char(x)` and `formatC`, for character `x`, do simple (left or right) padding with white space.

Author(s)

Originally written by Bill Dunlap, later much improved by Martin Maechler, it was first adapted for R by Friedrich Leisch.

References

Kernighan, B. W. and Ritchie, D. M. (1988) *The C Programming Language.* Second edition. Prentice Hall.

See Also

format.

Examples

```
xx  <- pi * 10^(-5:4)
options(digits = 4)   # only for format
cbind(format(xx), formatC(xx))
cbind(formatC(xx, wid = 9, flag = "-"))
cbind(formatC(xx, dig = 5, wid = 8, format = "f", flag = "0"))

format.char(c("a", "Abc", "no way"), wid = -7)  # <=> flag = "-"
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
```

`formula` *Model Formulae* `formula`

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"`.

Usage

```
y ~ model
formula(object)
formula.default(anything)
formula.formula(formula.obj)
formula.terms(terms.obj)
formula.data.frame(df)
as.formula(object)
I(name)
```

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 ~ 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~x - 1` is a line through the origin. A model with no intercept can be also specified as `y~x + 0` or `0 + y~x`.

While formulae usually involve just variable and factor names, they can also involve arithmetic expressions. The formula `log(y) ~ 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 `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 ~ a + I(b+c)`, the term `b+c` is to be interpreted as the sum of `b` and `c`.

Value

All the functions above produce an object of class `formula` which contains a symbolic model formula.

See Also

`lm`, `glm`, `terms`.

Examples

```
class(fo <- y ~ x1*x2) # "formula"
fo
typeof(fo)# R internal : "language"
terms(fo)

## 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= "+"))))
```

`frame` — *Create / Start a New Plot Frame* — `frame`

Description

The function causes the completion of plotting in the current plot 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

```
plot.new(ask = NA)
frame(ask = NA)
```

Arguments

`ask`	logical, defaulting to `par("ask")`.

See Also

`plot.window`, `plot.default`.

`freeny` *Freeny's Revenue Data* `freeny`

Description

Freeny's data on quarterly revenue and explanatory variables.

Usage

```
data(freeny)
```

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

A. E. Freeny (1977) *A Portable Linear Regression Package with Test Programs.* Bell Laboratories memorandum.

Examples

```
data(freeny)
summary(freeny)
pairs(freeny, main = "freeny data")
summary(fm1 <- 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(fm1)
par(opar)
```

`ftable` *Flat Contingency Tables* `ftable`

Description

Create and manipulate "flat" contingency tables.

Usage

```
ftable(..., exclude = c(NA, NaN), row.vars = NULL, col.vars = NULL)
ftable2table(x)
```

Arguments

`...`	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.
`x`	an arbitrary R object.

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 `ftable2table`. 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.

Function `ftable.formula` provides a formula method for creating flat contingency tables.

`ftable2table` converts a contingency table in flat matrix form to one in standard array form.

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); `table` for "ordinary" cross-tabulation.

Examples

```
## Start with a contingency table.
data(Titanic)
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.
data(mtcars)
x <- ftable(mtcars[c("cyl", "vs", "am", "gear")])
x
ftable(x, row.vars = c(2, 4))
```

`ftable.formula` *Formula Notation for Flat Contingency Tables* `ftable.formula`

Description

Produce or manipulate a flat contingency table using formula notation.

Usage

```
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 `NA`s. 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

```
data(Titanic)
Titanic
x <- ftable(Survived ~ ., data = Titanic)
x
ftable(Sex ~ Class + Age, data = x)
```

`function` *Function Definition* `function`

Description

These functions provide the base mechanisms for defining new functions in the R language.

Usage

```
function( arglist ) expr
return(value)
```

Arguments

`arglist`	Empty or one or more name or name=expression terms.
`value`	An expression, or a series of expressions separated by commas.

Details

In R (unlike S) the names in an argument list cannot be quoted non-standard names.

If `value` is a series of expressions, the value returned is a list of the evaluated expressions, with names set to the expressions where these are the names of R objects.

See Also

`args` and `body` for accessing the arguments and body of a function.

`debug` for debugging; `invisible` for `return(.)`ing *invisibly*.

Examples

```
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)
```

`GammaDist` *The Gamma Distribution* `GammaDist`

Description

Density, distribution function, quantile function and random generation for the Gamma distribution with parameters `shape` and `scale`.

Usage

```
dgamma(x, shape, scale=1, log = FALSE)
pgamma(q, shape, scale=1, lower.tail = TRUE, log.p = FALSE)
qgamma(p, shape, scale=1, lower.tail = TRUE, log.p = FALSE)
rgamma(n, shape, scale=1)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations.
`shape, scale`	shape 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 `scale` is omitted, it assumes the default value of `1`.

The Gamma distribution with parameters `shape` $= \alpha$ and `scale` $= \sigma$ has density

$$f(x) = \frac{1}{\sigma^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-x/\sigma}$$

for $x > 0$, $\alpha > 0$ and $\sigma > 0$.

Value

`dgamma` gives the density, `pgamma` gives the distribution function `qgamma` gives the quantile function, and `rgamma` generates random deviates.

See Also

`gamma` for the Gamma function, `dbeta` for the Beta distribution and `dchisq` for the chi-square 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))
```

---

gc *Garbage Collection* gc

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 use statistics (`verbose=TRUE`).

Usage

```
gc(verbose = getOption("verbose"))
gcinfo(verbose)
```

Arguments

`verbose`	logical; if `TRUE`, the garbage collection prints statistics about cons cells [in Bytes] and the vector heap [in KBytes].

Details

A call of `gc` causes a garbage collection to take place. This takes place automagically without user intervention, and the primary purpose of calling `gc` is for the report on memory usage.

Value

`gc` returns a 2*3 matrix with rows `"Ncells"` (*cons cells*, usually 20 bytes each on 32-bit systems, 36 bytes each on 64-bit systems) and `"Vcells"` (*vector cells*, 8 bytes each), and columns `"free"`, `"total"` and `"(Mb)"` (rounded up to the next 0.1Mb).

`gcinfo` returns the previous value of the flag.

See Also

`Memory` on R's memory management and `gctorture` if you are an R hacker.

Examples

```
gc() #- do it now
gcinfo(TRUE) #-- in the future, show when R does it
x <- integer(0); for(i in 1:18) x <- c(x,i,x)
gcinfo(verbose = FALSE)#-- don't show it anymore

( rgc <- gc(TRUE) )
(100*rgc[,"free"])%/% rgc[,"total"] # the percentages
rgc[2,"free"]/ (1024 / 8) # free vector heap in K bytes
```

---

gctorture *Torture Garbage Collector* gctorture

Description

Provokes garbage collection on (nearly) every memory allocation. Intended to ferret out memory protection bugs. Also makes R run *very* slowly, unfortunately.

Usage

```
gctorture(on = TRUE)
```

Arguments

on	logical; turning it on/off.

Value

None

Author(s)

Peter Dalgaard

Geometric *The Geometric Distribution* Geometric

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 to generate.
`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 `pgeom` 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

```
pp <- sort(c((1:9)/10, 1 - .2^(2:8)))
print(qg <- qgeom(pp, prob = .2))
## test that qgeom is an inverse of pgeom
print(qg1 <- qgeom(pgeom(qg, prob=.2), prob =.2))
all(qg == qg1)
Ni <- rgeom(20, prob = 1/4); table(factor(Ni, 0:max(Ni)))
```

`get` *Return a Variable's Value* `get`

Description

Search for an R object with a given name and return it if found.

Usage

```
get(x, pos=-1, envir=pos.to.env(pos), mode="any", inherits=TRUE)
```

Arguments

`x`	a variable name (given as a quoted string).
`pos`	position in search list, can be numerical or a quoted string.
`envir`	the environment to be used.
`mode`	the type of object sought.
`inherits`	should the enclosing frames of the environment be inspected?

Value

This function searches the specified environment for a bound variable whose name is given by the string `x`. If the variable's value is not of the correct `mode`, it is ignored.

If `inherits` is `FALSE`, only the first frame of the specified environment is inspected. If `inherits` is `TRUE`, the search is continued up through the parent frames until a bound value of the right mode is found.

Using a `NULL` environment is equivalent to using the current environment.

See Also

`exists`.

Examples

```
get("%o%")
```

`getenv` *Get Environment Variables* `getenv`

Description

`getenv` obtains the values of the environment variables named by `x`.

Usage

```
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 some platforms `getenv()` will return a named vector giving the values of all the environment variables.

See Also

`getwd` for the working directory.

Examples

```
getenv(c("R_HOME", "R_PAPERSIZE", "R_PRINTCMD", "HOST"))
```

`getwd` *Get or Set Working Directory* `getwd`

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.

Examples

```
(WD <- getwd())
if (!is.null(WD)) setwd(WD)
```

`gl` *Generate Factor Levels* `gl`

Description

Generate factors by specifying the pattern of their levels.

Usage

```
gl(n, k, length = n*k, labels=1:n, ordered=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 `length`.

`gl` is modelled on the *GLIM* function of the same name.

See Also

the underlying `factor(.)`.

Examples

```
# First control, then treatment:
gl(2,8, label=c("Ctnrl","Treat"))
# 20 alternating 1s and 2s
gl(2, 1, 20)
# alternating pairs of 1s and 2s
gl(2, 2, 20)
```

`glm` *Fitting Generalized Linear Models* `glm`

Description

`glm` is used to fit generalized linear models.

Models for `glm` are specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage

```
glm(formula, family = gaussian, data, weights = NULL, subset = NULL,
    na.action, start = NULL, offset = NULL,
    control = glm.control(epsilon=0.0001, maxit=10, trace=FALSE),
    model = TRUE, method = "glm.fit", x = FALSE, y = TRUE,
    contrasts = NULL, ...)
glm.control(epsilon = 0.0001, maxit = 10, trace = FALSE)
glm.fit(x, y, weights = rep(1, nrow(x)),
        start = NULL, etastart  =  NULL, mustart = NULL,
        offset = rep(0, nrow(x)),
        family = gaussian(), control = glm.control(),
        intercept = TRUE)
```

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. See `family` for details.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment ffrom 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 `NA`s. 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 (and presently only) method `glm.fit` uses iteratively reweighted least squares.
`x, y`	logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.

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.

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`.

Value

`glm` returns an object of class `glm` which inherits from the class `lm`. 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`.

Note

Offsets specified by `offset` will not be included in predictions by `predict.glm`, whereas those specified by an offset term in the formula will be.

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.

Examples

```
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## 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 (1999, pp.217-8)


 ## Need the anorexia data from a 1999 version of the package MASS:
 library(MASS)
 data(anorexia)

anorex.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
            family = gaussian, data = anorexia)
summary(anorex.1)
```

`glm.summaries` *Accessing Generalized Linear Model Fits* `glm.summaries`

Description

These functions are all methods for class `glm` or `summary.glm` and `anova.glm` objects.

`anova.glmlist` is called from `anova.glm` for comparison of several `glm` models.

Usage

```
summary(object, dispersion = NULL, correlation = FALSE, ...)
anova(object, ..., test = NULL)
anova.glmlist(object, test = NULL, ...)
coefficients(x) ; coef(x)
df.residual(x)
family(x)
fitted.values(x) ; fitted(x)
residuals(x, type = c("deviance", "pearson", "working",
                      "response", "partial"), ...)
print.summary(summary.glm.obj, digits = max(3, getOption("digits") - 3),
              na.print = "", symbolic.cor = p > 4,
              signif.stars = getOption("show.signif.stars"), ...)
```

Arguments

`object, x`	an object of class `glm`, typically the result of a call to `glm`.
`dispersion`	the dispersion parameter for the fitting family. By default it is obtained from `glm.obj`.

`correlation`	logical; if `TRUE`, the correlation matrix of the estimated parameters is returned and printed.
`test`	a character string, matching one of `"Chisq"`, `"F"` or `"Cp"`. See `stat.anova`.
`type`	the type of residuals which should be returned. The alternatives are: `"deviance"` (default), `"pearson"`, `"working"`, `"response"`, and `"partial"`.

Value

The function `summary` can be used to obtain or print a summary of the results and the function `anova` and be used to produce and analysis of variance table.

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.

See Also

`glm` for computing `glm.obj`; the corresponding *generic* functions, `summary`, `anova`, `coefficients`, `deviance`, `effects`, `fitted.values`, `residuals`.

Examples

```
## --- Continuing the Example from  ‘‘?glm’’:

summary(glm.D93)
anova(glm.D93)
anova(glm.D93, test = "Cp")
```

`Gnome` *GNOME Desktop Graphics Device* `Gnome`

Description

`gnome` starts a `GNOME` compatible device driver. GNOME is an acronym for **G**NU **N**etwork **O**bject **M**odel **E**nvironment.

Usage

```
gnome(display="", width=7, height=7, pointsize=12)
GNOME(display="", width=7, height=7, pointsize=12)
```

Arguments

`display`	the display on which the graphics window will appear. The default is to use the value in the user's environment variable `DISPLAY`.
`width`	the width of the plotting window in inches.
`height`	the height of the plotting window in inches.
`pointsize`	the default pointsize to be used.

Details

This is still in development state.

Note

The GNOME device is only available when explicitly desired at configure/compile time, see the toplevel 'INSTALLATION' file.

Author(s)

Lyndon Drake ⟨lyndon@stat.auckland.ac.nz⟩

References

http://www.gnome.org and http://www.gtk.org for the GTK+ (GIMP Tool Kit) libraries.

See Also

x11,Devices.

Examples

```
gnome(width=9)
```

---

gray *Gray Level Specification* gray

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

```
gray(0:8 / 8)
```

---

grep *Pattern Matching and Replacement* grep

Description

grep searches for matches to pattern (its first argument) within the vector x of character strings (second argument). regexpr does too, but returns more detail in a different format.

sub and gsub perform replacement of matches determined by regular expression matching.

Usage

```
grep(pattern, x, ignore.case=FALSE, extended=TRUE, value=FALSE)
sub(pattern, replacement, x,
        ignore.case=FALSE, extended=TRUE)
gsub(pattern, replacement, x,
        ignore.case=FALSE, extended=TRUE)
regexpr(pattern, text,  extended=TRUE)
```

Arguments

pattern	character string containing a regular expression to be matched in the vector of character string vec.
x, text	a vector of character strings where matches are sought.
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.
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.
replacement	a replacement for matched pattern in sub and gsub.

Details

The two *sub functions differ only in that sub replaces only the first occurrence of a pattern whereas gsub replaces all occurrences.

The regular expressions used are those specified by POSIX 1003.2, either extended or basic, depending on the value of the extended argument.

Value

For `gsub` 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 `regexpr` 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).

See Also

`charmatch`, `pmatch`, `match`. `apropos` uses regexps and has nice examples.

Examples

```
grep("[a-z]", letters)

txt <- c("arm","foot","lefroo", "bafoobar")
if(any(i <- grep("foo",txt)))
   cat("'foo' appears at least once in\n\t",txt,"\n")
i # 2 and 4
txt[i]

## Double all 'a' or 'b's;  "\" must be escaped, i.e. 'doubled'
gsub("([ab])", "\\1_\\1_", "abc and ABC")

txt <- c("The", "licenses", "for", "most", "software", "are",
  "designed", "to", "take", "away", "your", "freedom",
  "to", "share", "and", "change", "it.",
   "", "By", "contrast,", "the", "GNU", "General", "Public", "License",
   "is", "intended", "to", "guarantee", "your", "freedom", "to",
   "share", "and", "change", "free", "software", "--",
   "to", "make", "sure", "the", "software", "is",
   "free", "for", "all", "its", "users")
( i <- grep("[gu]", txt) ) # indices
all( txt[i] == grep("[gu]", txt, value = TRUE) )
(ot <- sub("[b-e]",".", txt))
txt[ot != gsub("[b-e]",".", txt)]#- gsub does "global" substitution

txt[gsub("g","#", txt) !=
    gsub("g","#", txt, ignore.case = TRUE)] # the "G" words

regexpr("en", txt)
```

`grid` *Add Grid to a Plot* `grid`

Description

`grid` adds an `nx` by `ny` rectangular grid to an existing plot, using lines of type `lty` and color `col`.

If more fine tuning is required, use `abline(h = ., v = .)` directly.

Usage

```
grid(nx = NULL, ny = NULL, col = "lightgray", lty = "dotted")
```

Arguments

`nx,ny`	number of cells of the grid in x and y direction. Defaults to the number of tick marks on the corresponding axis.
`col`	character or (integer) numeric; color of the grid lines.
`lty`	character or (integer) numeric; line type of the grid lines.

See Also

`plot`, `abline`, `lines`, `points`.

Examples

```
data(iris)
## maybe change the desired number of tick marks:  par(lab=c(mx,my,7))
plot(iris$Sepal.L, iris$Sepal.W, col = rep(1:3, rep(50, 3)),
     xlim = c(4, 8), ylim = c(2, 4.5), panel.first = grid())
```

`HairEyeColor` *Hair and Eye Color of Statistics Students* `HairEyeColor`

Description

Distribution of hair and eye color and sex in 592 statistics students.

Usage

```
data(HairEyeColor)
```

Format

A 3-dimensional array resulting from cross-tabulating 592 observations on 3 variables. The variables and their levels are as follows:

No	Name	Levels
1	Hair	Black, Brown, Red, Blond
2	Eye	Brown, Blue, Hazel, Green
3	Sex	Male, Female

Details

This data set is useful for illustrating various techniques for the analysis of contingency tables, such as the standard chi-square test or, more generally, log-linear modelling, and graphical methods such as mosaic plots, sieve diagrams or association plots.

References

Snee, R. D. (1974), Graphical display of two-way contingency tables. *The American Statistician*, **28**, 9–12.

Friendly, M. (1992), Graphical methods for categorical data. *SAS User Group International Conference Proceedings*, **17**, 190–200. http://hotspur.psych.yorku.ca/SCS/sugi/sugi17-paper.html

Friendly, M. (1992), Mosaic displays for loglinear models. *Proceedings of the Statistical Graphics Section*, American Statistical Association, pp. 61–68. http://hotspur.psych.yorku.ca/SCS/Papers/asa92.html

Examples

```
data(HairEyeColor)
## Full mosaic
mosaicplot(HairEyeColor)
## Aggregate over sex:
x <- apply(HairEyeColor, c(1, 2), sum)
x
mosaicplot(x, main = "Relation between hair and eye color")
```

`help` *Documentation* `help`

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 = .packages()
     lib.loc = .lib.loc, verbose = getOption("verbose"),
     htmlhelp = getOption("htmlhelp"))
?topic
```

Arguments

`topic`	a name or character string on which documentation is sought (but *not* a variable containing a character string!).
`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. By default, all packages in the search path are used.
`lib.loc`	A character vector of directory names of R libraries. Defaults to all libraries currently known.
`verbose`	logical; if `TRUE`, the file name is reported.
`htmlhelp`	logical (or `NULL`). If `TRUE` (which is the default after `help.start` has been called), the HTML version of the help will be shown in a browser.

Details

In the case of unary and binary operators and control-flow special forms, the name 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) and `dvips`. Depending on your `dvips` configuration, hardcopy will be sent to the printer or saved in a file. If the programs are in non-standard locations and hence were not found at compile time, you can either set the options `latexcmd` and `dvipscmd`, or the environment variables `R_LATEXCMD` and `R_DVIPSCMD` appropriately. The appearance of the output can be customized through a file 'Rhelp.cfg' somewhere in your LaTeX search path.

Note

You will only get help for R objects in your `search()` path, i.e., `help(dist)` will only work after `library(mva)`. To look in all the packages in all the current libraries, use `help(topic, package=.packages(all=T), lib.loc=.lib.loc)`

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 the files 'AnIndex' and '00Titles' remain un-zipped. Similarly, all the files in the 'latex' directory can be zipped to 'Rhelp.zip'.

See Also

`help.search()` for finding help pages on a 'vague' topic. `help.start()` which opens the HTML version of the R Manual; `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

```
help()
help(help)              # the same

help(lapply)
?lapply                 # the same

help("for")             # or ?"for", but the quotes are needed
?"+"

help(package = stepfun) # get help even when it's not present

data()                  # list all available data sets
?women                  # information about data set "women"

topi <- "women"
help(topi) ##--> Error: No documentation for 'topi'
```

help.search *Search the Help System* help.search

Description

Allows for searching the help system for documentation matching a given regular expression in its name, alias, title, or keyword entries (or any combination thereof). Names and titles of the matched help entries are nicely displayed; currently, nothing is returned.

Usage

```
help.search(topic, fields = c("name", "title"),
            apropos, keyword, whatis, ignore.case = TRUE,
            packages = NULL, lib.loc = .lib.loc,
            help.db = getOption("help.db"),
            verbose = getOption("verbose"),
            rebuild = FALSE)
```

Arguments

`what`	a character string containing a regular expression 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"`, `"alias"`, `"title"`, and `"keyword"`, corresponding to the help page's (file) name, the objects it provides documentation for, its title, and the keywords it can be classified to.
`apropos`	a character string containing a regular expression to be matched in the help page names and titles.
`keyword`	a character string containing a regular expression to be matched in the help page keywords.
`whatis`	a character string containing a regular expression to be matched in the help page names.
`ignore.case`	a logical. If `TRUE`, case is ignored during matching; if `FALSE`, pattern matching is case sensitive.
`packages`	a character vector with the names of packages to search through, or `NULL` in which case *all* available packages in the specified library trees `lib.loc` are searched.
`lib.loc`	a character vector describing the location of R library trees to search through.
`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.

Details

Upon installation of a package, the Perl script `Rd2contents` creates a 'CONTENTS' data base which contains the information on name, aliases, title and keywords (as well as the URL of the HTML version of the help file) in Debian Control Format. This is the data base searched by `help.search()`.

The arguments `apropos` and `whatis` play a role similar to the Unix commands with the same names.

If possible, the help data base is saved to the file 'help.db' in the '.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.

See Also

`help`; `help.start` for starting the hypertext (currently HTML) version of R's online documentation, which offers a similar search mechanism.

`apropos` uses regexps and has nice examples.

Examples

```
help.search("linear models")    # In case you forgot how to fit linear
                                # models

help.search("print")            # All help pages with name or title
                                # matching 'print'
help.search(apropos = "print")  # The same
help.search(keyword = "hplot")  # All help pages documenting high-level
                                # plots.
```

`help.start` *Hypertext Documentation* `help.start`

Description

Start the hypertext (currently HTML) version of R's online documentation.

Usage

```
help.start(gui = "irrelevant", browser = "netscape", remote = NULL)
```

Arguments

`gui`	just for compatibility
`browser`	the name of the program to be used as hypertext browser.
`remote`	A character giving a valid URL for the '$R_HOME' directory on a remote location.

Details

The `browser` needs to work with a `"-remote openURL(..)"` call. Then, the hypertext R help interface is opened in any already running browser or a new one if necessary.

See Also

`help()` for on- and off-line help in ASCII/Editor or PostScript format.

Examples

```
help.start()
```

`Hershey` *Hershey Vector Fonts in R* `Hershey`

Description

If the `vfont` argument to one of the text-drawing functions (`text` and `contour`) is a character vector of length 2, hershey vector fonts are used to render the 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 wierd plotting symbols and Japanese characters.

Details

The Hershey characters are organised into a set of fonts, which are specified by a typeface (e.g., `serif` or `sans serif`) and a fontindex (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 `example(Hershey)` shows the character `a` produced by each of the different fonts.

The available values for `(typeface, fontindex)` for Hershey vector fonts are:

serif	plain
serif	italic
serif	bold
serif	bold italic
serif	cyrillic
serif	oblique cyrillic
serif	EUC
sans serif	plain
sans serif	italic
sans serif	bold
sans serif	bold italic
script	plain
script	italic
script	bold
gothic english	plain
gothic german	plain
gothic italian	plain
serif symbol	plain

serif symbol	italic
serif symbol	bold
serif symbol	bold italic
sans serif symbol	plain
sans serif symbol	italic

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 a 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.

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 `example(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 (for example, if you only have a US keyboard). 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 `example(Hershey)` shows all of the ISO Latin-1 escape sequences.

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 `example(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 `example(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 idenitified 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., 0xa4 and 0xa1 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`.

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 `example(Hershey)` shows all of the available raw glyphs.

References

http://www.gnu.org/software/plotutils/plotutils.html

See Also

text, contour, Japanese

Examples

```
######
# create tables of vector font functionality
######
make.table <- function(nr, nc) {
    savepar <- par(mar=rep(0, 4), pty="s")
    plot(c(0, nc*2 + 1), c(0, -(nr + 1)),
         type="n", xlab="", ylab="", axes=F)
    savepar
}

get.r <- function(i, nr) {
    i %% nr + 1
}

get.c <- function(i, nr) {
    i %/% nr + 1
}

draw.title <- function(title, nr, nc) {
    r <- get.r(i, nr)
    c <- get.c(i, nr)
    text((nc*2 + 1)/2, 0, title, font=2)
}

draw.sample.cell <- function(typeface, fontindex, string, i, nr) {
    r <- get.r(i, nr)
    c <- get.c(i, nr)
    text((2*(c - 1) + 1), -r, paste(typeface, fontindex))
    text((2*c), -r, string, vfont=c(typeface, fontindex), cex=1.5)
    rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5), border="grey")
}

draw.vf.cell <- function(typeface, fontindex, string, i, nr, raw.string=NULL) {
    r <- get.r(i, nr)
    c <- get.c(i, nr)
    if (is.null(raw.string))
        raw.string <- paste("\\", string, sep="")
    text((2*(c - 1) + 1), -r, raw.string, col="grey")
    text((2*c), -r, string, vfont=c(typeface, fontindex))
    rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5), border="grey")
}
```

```
nr <- 23
nc <- 1
oldpar <- make.table(nr, nc)
i <- 0
draw.title("Sample 'a' for each available font", nr, nc)
draw.sample.cell("serif", "plain", "a", i, nr); i <- i + 1
draw.sample.cell("serif", "italic", "a", i, nr); i <- i + 1
draw.sample.cell("serif", "bold", "a", i, nr); i <- i + 1
draw.sample.cell("serif", "bold italic", "a", i, nr); i <- i + 1
draw.sample.cell("serif", "cyrillic", "a", i, nr); i <- i + 1
draw.sample.cell("serif", "oblique cyrillic", "a", i, nr); i <- i + 1
draw.sample.cell("serif", "EUC", "a", i, nr); i <- i + 1
draw.sample.cell("sans serif", "plain", "a", i, nr); i <- i + 1
draw.sample.cell("sans serif", "italic", "a", i, nr); i <- i + 1
draw.sample.cell("sans serif", "bold", "a", i, nr); i <- i + 1
draw.sample.cell("sans serif", "bold italic", "a", i, nr); i <- i + 1
draw.sample.cell("script", "plain", "a", i, nr); i <- i + 1
draw.sample.cell("script", "italic", "a", i, nr); i <- i + 1
draw.sample.cell("script", "bold", "a", i, nr); i <- i + 1
draw.sample.cell("gothic english", "plain", "a", i, nr); i <- i + 1
draw.sample.cell("gothic german", "plain", "a", i, nr); i <- i + 1
draw.sample.cell("gothic italian", "plain", "a", i, nr); i <- i + 1
draw.sample.cell("serif symbol", "plain", "a", i, nr); i <- i + 1
draw.sample.cell("serif symbol", "italic", "a", i, nr); i <- i + 1
draw.sample.cell("serif symbol", "bold", "a", i, nr); i <- i + 1
draw.sample.cell("serif symbol", "bold italic", "a", i, nr); i <- i + 1
draw.sample.cell("sans serif symbol", "plain", "a", i, nr); i <- i + 1
draw.sample.cell("sans serif symbol", "italic", "a", i, nr); i <- i + 1

nr <- 25
nc <- 6
tf <- "serif"
fi <- "plain"
make.table(nr, nc)
i <- 0
draw.title("Symbol Escape Sequences", nr, nc)
draw.vf.cell(tf, fi, "\\fa", i, nr); i<-i+1; { "universal"}
draw.vf.cell(tf, fi, "\\te", i, nr); i<-i+1; { "existential"}
draw.vf.cell(tf, fi, "\\st", i, nr); i<-i+1; { "suchthat"}
draw.vf.cell(tf, fi, "\\**", i, nr); i<-i+1; { "asteriskmath"}
draw.vf.cell(tf, fi, "\\=~", i, nr); i<-i+1; { "congruent"}
draw.vf.cell(tf, fi, "\\*A", i, nr); i<-i+1; { "Alpha"}
draw.vf.cell(tf, fi, "\\*B", i, nr); i<-i+1; { "Beta"}
draw.vf.cell(tf, fi, "\\*X", i, nr); i<-i+1; { "Chi"}
draw.vf.cell(tf, fi, "\\*D", i, nr); i<-i+1; { "Delta"}
draw.vf.cell(tf, fi, "\\*E", i, nr); i<-i+1; { "Epsilon"}
draw.vf.cell(tf, fi, "\\*F", i, nr); i<-i+1; { "Phi"}
draw.vf.cell(tf, fi, "\\*G", i, nr); i<-i+1; { "Gamma"}
draw.vf.cell(tf, fi, "\\*Y", i, nr); i<-i+1; { "Eta"}
draw.vf.cell(tf, fi, "\\*I", i, nr); i<-i+1; { "Iota"}
draw.vf.cell(tf, fi, "\\+h", i, nr); i<-i+1; { "theta1"}
draw.vf.cell(tf, fi, "\\*K", i, nr); i<-i+1; { "kappa"}
draw.vf.cell(tf, fi, "\\*L", i, nr); i<-i+1; { "Lambda"}
draw.vf.cell(tf, fi, "\\*M", i, nr); i<-i+1; { "Mu"}
draw.vf.cell(tf, fi, "\\*N", i, nr); i<-i+1; { "Nu"}
draw.vf.cell(tf, fi, "\\*O", i, nr); i<-i+1; { "Omicron"}
```

```
draw.vf.cell(tf, fi, "\\*P", i, nr); i<-i+1; { "Pi"}
draw.vf.cell(tf, fi, "\\*H", i, nr); i<-i+1; { "Theta"}
draw.vf.cell(tf, fi, "\\*R", i, nr); i<-i+1; { "Rho"}
draw.vf.cell(tf, fi, "\\*S", i, nr); i<-i+1; { "Sigma"}
draw.vf.cell(tf, fi, "\\*T", i, nr); i<-i+1; { "Tau"}
draw.vf.cell(tf, fi, "\\*U", i, nr); i<-i+1; { "Upsilon"}
draw.vf.cell(tf, fi, "\\ts", i, nr); i<-i+1; { "sigma1"}
draw.vf.cell(tf, fi, "\\*W", i, nr); i<-i+1; { "Omega"}
draw.vf.cell(tf, fi, "\\*C", i, nr); i<-i+1; { "Xi"}
draw.vf.cell(tf, fi, "\\*Q", i, nr); i<-i+1; { "Psi"}
draw.vf.cell(tf, fi, "\\*Z", i, nr); i<-i+1; { "Zeta"}
draw.vf.cell(tf, fi, "\\tf", i, nr); i<-i+1; { "therefore"}
draw.vf.cell(tf, fi, "\\pp", i, nr); i<-i+1; { "perpendicular"}
draw.vf.cell(tf, fi, "\\ul", i, nr); i<-i+1; { "underline"}
draw.vf.cell(tf, fi, "\\rx", i, nr); i<-i+1; { "radicalex"}
draw.vf.cell(tf, fi, "\\*a", i, nr); i<-i+1; { "alpha"}
draw.vf.cell(tf, fi, "\\*b", i, nr); i<-i+1; { "beta"}
draw.vf.cell(tf, fi, "\\*x", i, nr); i<-i+1; { "chi"}
draw.vf.cell(tf, fi, "\\*d", i, nr); i<-i+1; { "delta"}
draw.vf.cell(tf, fi, "\\*e", i, nr); i<-i+1; { "epsilon"}
draw.vf.cell(tf, fi, "\\*f", i, nr); i<-i+1; { "phi"}
draw.vf.cell(tf, fi, "\\*g", i, nr); i<-i+1; { "gamma"}
draw.vf.cell(tf, fi, "\\*y", i, nr); i<-i+1; { "eta"}
draw.vf.cell(tf, fi, "\\*i", i, nr); i<-i+1; { "iota"}
draw.vf.cell(tf, fi, "\\+f", i, nr); i<-i+1; { "phi1"}
draw.vf.cell(tf, fi, "\\*k", i, nr); i<-i+1; { "kappa"}
draw.vf.cell(tf, fi, "\\*l", i, nr); i<-i+1; { "lambda"}
draw.vf.cell(tf, fi, "\\*m", i, nr); i<-i+1; { "mu"}
draw.vf.cell(tf, fi, "\\*n", i, nr); i<-i+1; { "nu"}
draw.vf.cell(tf, fi, "\\*o", i, nr); i<-i+1; { "omicron"}
draw.vf.cell(tf, fi, "\\*p", i, nr); i<-i+1; { "pi"}
draw.vf.cell(tf, fi, "\\*h", i, nr); i<-i+1; { "theta"}
draw.vf.cell(tf, fi, "\\*r", i, nr); i<-i+1; { "rho"}
draw.vf.cell(tf, fi, "\\*s", i, nr); i<-i+1; { "sigma"}
draw.vf.cell(tf, fi, "\\*t", i, nr); i<-i+1; { "tau"}
draw.vf.cell(tf, fi, "\\*u", i, nr); i<-i+1; { "upsilon"}
draw.vf.cell(tf, fi, "\\+p", i, nr); i<-i+1; { "omega1"}
draw.vf.cell(tf, fi, "\\*w", i, nr); i<-i+1; { "omega"}
draw.vf.cell(tf, fi, "\\*c", i, nr); i<-i+1; { "xi"}
draw.vf.cell(tf, fi, "\\*q", i, nr); i<-i+1; { "psi"}
draw.vf.cell(tf, fi, "\\*z", i, nr); i<-i+1; { "zeta"}
draw.vf.cell(tf, fi, "\\ap", i, nr); i<-i+1; { "similar"}
draw.vf.cell(tf, fi, "\\+U", i, nr); i<-i+1; { "Upsilon1"}
draw.vf.cell(tf, fi, "\\fm", i, nr); i<-i+1; { "minute"}
draw.vf.cell(tf, fi, "\\<=", i, nr); i<-i+1; { "lessequal"}
draw.vf.cell(tf, fi, "\\f/", i, nr); i<-i+1; { "fraction"}
draw.vf.cell(tf, fi, "\\if", i, nr); i<-i+1; { "infinity"}
draw.vf.cell(tf, fi, "\\Fn", i, nr); i<-i+1; { "florin"}
draw.vf.cell(tf, fi, "\\CL", i, nr); i<-i+1; { "club"}
draw.vf.cell(tf, fi, "\\DI", i, nr); i<-i+1; { "diamond"}
draw.vf.cell(tf, fi, "\\HE", i, nr); i<-i+1; { "heart"}
draw.vf.cell(tf, fi, "\\SP", i, nr); i<-i+1; { "spade"}
draw.vf.cell(tf, fi, "\\<>", i, nr); i<-i+1; { "arrowboth"}
draw.vf.cell(tf, fi, "\\<-", i, nr); i<-i+1; { "arrowleft"}
draw.vf.cell(tf, fi, "\\ua", i, nr); i<-i+1; { "arrowup"}
draw.vf.cell(tf, fi, "\\->", i, nr); i<-i+1; { "arrowright"}
draw.vf.cell(tf, fi, "\\da", i, nr); i<-i+1; { "arrowdown"}
```

```
draw.vf.cell(tf, fi, "\\de", i, nr); i<-i+1; { "degree"}
draw.vf.cell(tf, fi, "\\+-", i, nr); i<-i+1; { "plusminus"}
draw.vf.cell(tf, fi, "\\sd", i, nr); i<-i+1; { "second"}
draw.vf.cell(tf, fi, "\\>=", i, nr); i<-i+1; { "greaterequal"}
draw.vf.cell(tf, fi, "\\mu", i, nr); i<-i+1; { "multiply"}
draw.vf.cell(tf, fi, "\\pt", i, nr); i<-i+1; { "proportional"}
draw.vf.cell(tf, fi, "\\pd", i, nr); i<-i+1; { "partialdiff"}
draw.vf.cell(tf, fi, "\\bu", i, nr); i<-i+1; { "bullet"}
draw.vf.cell(tf, fi, "\\di", i, nr); i<-i+1; { "divide"}
draw.vf.cell(tf, fi, "\\!=", i, nr); i<-i+1; { "notequal"}
draw.vf.cell(tf, fi, "\\==", i, nr); i<-i+1; { "equivalence"}
draw.vf.cell(tf, fi, "\\~~", i, nr); i<-i+1; { "approxequal"}
draw.vf.cell(tf, fi, "\\..", i, nr); i<-i+1; { "ellipsis"}
draw.vf.cell(tf, fi, "\\an", i, nr); i<-i+1; { "arrowhorizex"}
draw.vf.cell(tf, fi, "\\CR", i, nr); i<-i+1; { "carriagereturn"}
draw.vf.cell(tf, fi, "\\Ah", i, nr); i<-i+1; { "aleph"}
draw.vf.cell(tf, fi, "\\Im", i, nr); i<-i+1; { "Ifraktur"}
draw.vf.cell(tf, fi, "\\Re", i, nr); i<-i+1; { "Rfraktur"}
draw.vf.cell(tf, fi, "\\wp", i, nr); i<-i+1; { "weierstrass"}
draw.vf.cell(tf, fi, "\\c*", i, nr); i<-i+1; { "circlemultiply"}
draw.vf.cell(tf, fi, "\\c+", i, nr); i<-i+1; { "circleplus"}
draw.vf.cell(tf, fi, "\\es", i, nr); i<-i+1; { "emptyset"}
draw.vf.cell(tf, fi, "\\ca", i, nr); i<-i+1; { "cap"}
draw.vf.cell(tf, fi, "\\cu", i, nr); i<-i+1; { "cup"}
draw.vf.cell(tf, fi, "\\SS", i, nr); i<-i+1; { "superset"}
draw.vf.cell(tf, fi, "\\ip", i, nr); i<-i+1; { "reflexsuperset"}
draw.vf.cell(tf, fi, "\\n<", i, nr); i<-i+1; { "notsubset"}
draw.vf.cell(tf, fi, "\\SB", i, nr); i<-i+1; { "subset"}
draw.vf.cell(tf, fi, "\\ib", i, nr); i<-i+1; { "reflexsubset"}
draw.vf.cell(tf, fi, "\\mo", i, nr); i<-i+1; { "element"}
draw.vf.cell(tf, fi, "\\nm", i, nr); i<-i+1; { "notelement"}
draw.vf.cell(tf, fi, "\\/_", i, nr); i<-i+1; { "angle"}
draw.vf.cell(tf, fi, "\\gr", i, nr); i<-i+1; { "nabla"}
draw.vf.cell(tf, fi, "\\rg", i, nr); i<-i+1; { "registerserif"}
draw.vf.cell(tf, fi, "\\co", i, nr); i<-i+1; { "copyrightserif"}
draw.vf.cell(tf, fi, "\\tm", i, nr); i<-i+1; { "trademarkserif"}
draw.vf.cell(tf, fi, "\\PR", i, nr); i<-i+1; { "product"}
draw.vf.cell(tf, fi, "\\sr", i, nr); i<-i+1; { "radical"}
draw.vf.cell(tf, fi, "\\md", i, nr); i<-i+1; { "dotmath"}
draw.vf.cell(tf, fi, "\\no", i, nr); i<-i+1; { "logicalnot"}
draw.vf.cell(tf, fi, "\\AN", i, nr); i<-i+1; { "logicaland"}
draw.vf.cell(tf, fi, "\\OR", i, nr); i<-i+1; { "logicalor"}
draw.vf.cell(tf, fi, "\\hA", i, nr); i<-i+1; { "arrowdblboth"}
draw.vf.cell(tf, fi, "\\lA", i, nr); i<-i+1; { "arrowdblleft"}
draw.vf.cell(tf, fi, "\\uA", i, nr); i<-i+1; { "arrowdblup"}
draw.vf.cell(tf, fi, "\\rA", i, nr); i<-i+1; { "arrowdblright"}
draw.vf.cell(tf, fi, "\\dA", i, nr); i<-i+1; { "arrowdbldown"}
draw.vf.cell(tf, fi, "\\lz", i, nr); i<-i+1; { "lozenge"}
draw.vf.cell(tf, fi, "\\la", i, nr); i<-i+1; { "angleleft"}
draw.vf.cell(tf, fi, "\\RG", i, nr); i<-i+1; { "registersans"}
draw.vf.cell(tf, fi, "\\CO", i, nr); i<-i+1; { "copyrightsans"}
draw.vf.cell(tf, fi, "\\TM", i, nr); i<-i+1; { "trademarksans"}
draw.vf.cell(tf, fi, "\\SU", i, nr); i<-i+1; { "summation"}
draw.vf.cell(tf, fi, "\\lc", i, nr); i<-i+1; { "bracketlefttp"}
draw.vf.cell(tf, fi, "\\lf", i, nr); i<-i+1; { "bracketleftbt"}
draw.vf.cell(tf, fi, "\\ra", i, nr); i<-i+1; { "angleright"}
draw.vf.cell(tf, fi, "\\is", i, nr); i<-i+1; { "integral"}
```

```
draw.vf.cell(tf, fi, "\\rc", i, nr); i<-i+1; { "bracketrighttp"}
draw.vf.cell(tf, fi, "\\rf", i, nr); i<-i+1; { "bracketrightbt"}
draw.vf.cell(tf, fi, "\\~=", i, nr); i<-i+1; { "congruent"}
draw.vf.cell(tf, fi, "\\pr", i, nr); i<-i+1; { "minute"}
draw.vf.cell(tf, fi, "\\in", i, nr); i<-i+1; { "infinity"}
draw.vf.cell(tf, fi, "\\n=", i, nr); i<-i+1; { "notequal"}
draw.vf.cell(tf, fi, "\\dl", i, nr); i<-i+1; { "nabla"}

nr <- 25
nc <- 4
make.table(nr, nc)
i <- 0
draw.title("ISO Latin-1 Escape Sequences", nr, nc)
draw.vf.cell(tf, fi, "\\r!", i, nr); i<-i+1; { "exclamdown"}
draw.vf.cell(tf, fi, "\\ct", i, nr); i<-i+1; { "cent"}
draw.vf.cell(tf, fi, "\\Po", i, nr); i<-i+1; { "sterling"}
draw.vf.cell(tf, fi, "\\Ye", i, nr); i<-i+1; { "yen"}
draw.vf.cell(tf, fi, "\\bb", i, nr); i<-i+1; { "brokenbar"}
draw.vf.cell(tf, fi, "\\sc", i, nr); i<-i+1; { "section"}
draw.vf.cell(tf, fi, "\\ad", i, nr); i<-i+1; { "dieresis"}
draw.vf.cell(tf, fi, "\\co", i, nr); i<-i+1; { "copyright"}
draw.vf.cell(tf, fi, "\\Of", i, nr); i<-i+1; { "ordfeminine"}
draw.vf.cell(tf, fi, "\\no", i, nr); i<-i+1; { "logicalnot"}
draw.vf.cell(tf, fi, "\\hy", i, nr); i<-i+1; { "hyphen"}
draw.vf.cell(tf, fi, "\\rg", i, nr); i<-i+1; { "registered"}
draw.vf.cell(tf, fi, "\\a-", i, nr); i<-i+1; { "macron"}
draw.vf.cell(tf, fi, "\\de", i, nr); i<-i+1; { "degree"}
draw.vf.cell(tf, fi, "\\+-", i, nr); i<-i+1; { "plusminus"}
draw.vf.cell(tf, fi, "\\S2", i, nr); i<-i+1; { "twosuperior"}
draw.vf.cell(tf, fi, "\\S3", i, nr); i<-i+1; { "threesuperior"}
draw.vf.cell(tf, fi, "\\aa", i, nr); i<-i+1; { "acute"}
draw.vf.cell(tf, fi, "\\*m", i, nr); i<-i+1; { "mu"}
draw.vf.cell(tf, fi, "\\md", i, nr); i<-i+1; { "periodcentered"}
draw.vf.cell(tf, fi, "\\S1", i, nr); i<-i+1; { "onesuperior"}
draw.vf.cell(tf, fi, "\\Om", i, nr); i<-i+1; { "ordmasculine"}
draw.vf.cell(tf, fi, "\\14", i, nr); i<-i+1; { "onequarter"}
draw.vf.cell(tf, fi, "\\12", i, nr); i<-i+1; { "onehalf"}
draw.vf.cell(tf, fi, "\\34", i, nr); i<-i+1; { "threequarters"}
draw.vf.cell(tf, fi, "\\r?", i, nr); i<-i+1; { "questiondown"}
draw.vf.cell(tf, fi, "\\`A", i, nr); i<-i+1; { "Agrave"}
draw.vf.cell(tf, fi, "\\'A", i, nr); i<-i+1; { "Aacute"}
draw.vf.cell(tf, fi, "\\^A", i, nr); i<-i+1; { "Acircumflex"}
draw.vf.cell(tf, fi, "\\~A", i, nr); i<-i+1; { "Atilde"}
draw.vf.cell(tf, fi, "\\:A", i, nr); i<-i+1; { "Adieresis"}
draw.vf.cell(tf, fi, "\\oA", i, nr); i<-i+1; { "Aring"}
draw.vf.cell(tf, fi, "\\AE", i, nr); i<-i+1; { "AE"}
draw.vf.cell(tf, fi, "\\,C", i, nr); i<-i+1; { "Ccedilla"}
draw.vf.cell(tf, fi, "\\`E", i, nr); i<-i+1; { "Egrave"}
draw.vf.cell(tf, fi, "\\'E", i, nr); i<-i+1; { "Eacute"}
draw.vf.cell(tf, fi, "\\^E", i, nr); i<-i+1; { "Ecircumflex"}
draw.vf.cell(tf, fi, "\\:E", i, nr); i<-i+1; { "Edieresis"}
draw.vf.cell(tf, fi, "\\`I", i, nr); i<-i+1; { "Igrave"}
draw.vf.cell(tf, fi, "\\'I", i, nr); i<-i+1; { "Iacute"}
draw.vf.cell(tf, fi, "\\^I", i, nr); i<-i+1; { "Icircumflex"}
draw.vf.cell(tf, fi, "\\:I", i, nr); i<-i+1; { "Idieresis"}
draw.vf.cell(tf, fi, "\\~N", i, nr); i<-i+1; { "Ntilde"}
draw.vf.cell(tf, fi, "\\`O", i, nr); i<-i+1; { "Ograve"}
```

```
draw.vf.cell(tf, fi, "\\'O", i, nr); i<-i+1; { "Oacute"}
draw.vf.cell(tf, fi, "\\^O", i, nr); i<-i+1; { "Ocircumflex"}
draw.vf.cell(tf, fi, "\\~O", i, nr); i<-i+1; { "Otilde"}
draw.vf.cell(tf, fi, "\\:O", i, nr); i<-i+1; { "Odieresis"}
draw.vf.cell(tf, fi, "\\mu", i, nr); i<-i+1; { "multiply"}
draw.vf.cell(tf, fi, "\\/O", i, nr); i<-i+1; { "Oslash"}
draw.vf.cell(tf, fi, "\\`U", i, nr); i<-i+1; { "Ugrave"}
draw.vf.cell(tf, fi, "\\'U", i, nr); i<-i+1; { "Uacute"}
draw.vf.cell(tf, fi, "\\^U", i, nr); i<-i+1; { "Ucircumflex"}
draw.vf.cell(tf, fi, "\\:U", i, nr); i<-i+1; { "Udieresis"}
draw.vf.cell(tf, fi, "\\'Y", i, nr); i<-i+1; { "Yacute"}
draw.vf.cell(tf, fi, "\\ss", i, nr); i<-i+1; { "germandbls"}
draw.vf.cell(tf, fi, "\\`a", i, nr); i<-i+1; { "agrave"}
draw.vf.cell(tf, fi, "\\'a", i, nr); i<-i+1; { "aacute"}
draw.vf.cell(tf, fi, "\\^a", i, nr); i<-i+1; { "acircumflex"}
draw.vf.cell(tf, fi, "\\~a", i, nr); i<-i+1; { "atilde"}
draw.vf.cell(tf, fi, "\\:a", i, nr); i<-i+1; { "adieresis"}
draw.vf.cell(tf, fi, "\\oa", i, nr); i<-i+1; { "aring"}
draw.vf.cell(tf, fi, "\\ae", i, nr); i<-i+1; { "ae"}
draw.vf.cell(tf, fi, "\\,c", i, nr); i<-i+1; { "ccedilla"}
draw.vf.cell(tf, fi, "\\`e", i, nr); i<-i+1; { "egrave"}
draw.vf.cell(tf, fi, "\\'e", i, nr); i<-i+1; { "eacute"}
draw.vf.cell(tf, fi, "\\^e", i, nr); i<-i+1; { "ecircumflex"}
draw.vf.cell(tf, fi, "\\:e", i, nr); i<-i+1; { "edieresis"}
draw.vf.cell(tf, fi, "\\`i", i, nr); i<-i+1; { "igrave"}
draw.vf.cell(tf, fi, "\\'i", i, nr); i<-i+1; { "iacute"}
draw.vf.cell(tf, fi, "\\^i", i, nr); i<-i+1; { "icircumflex"}
draw.vf.cell(tf, fi, "\\:i", i, nr); i<-i+1; { "idieresis"}
draw.vf.cell(tf, fi, "\\~n", i, nr); i<-i+1; { "ntilde"}
draw.vf.cell(tf, fi, "\\`o", i, nr); i<-i+1; { "ograve"}
draw.vf.cell(tf, fi, "\\'o", i, nr); i<-i+1; { "oacute"}
draw.vf.cell(tf, fi, "\\^o", i, nr); i<-i+1; { "ocircumflex"}
draw.vf.cell(tf, fi, "\\~o", i, nr); i<-i+1; { "otilde"}
draw.vf.cell(tf, fi, "\\:o", i, nr); i<-i+1; { "odieresis"}
draw.vf.cell(tf, fi, "\\di", i, nr); i<-i+1; { "divide"}
draw.vf.cell(tf, fi, "\\/o", i, nr); i<-i+1; { "oslash"}
draw.vf.cell(tf, fi, "\\`u", i, nr); i<-i+1; { "ugrave"}
draw.vf.cell(tf, fi, "\\'u", i, nr); i<-i+1; { "uacute"}
draw.vf.cell(tf, fi, "\\^u", i, nr); i<-i+1; { "ucircumflex"}
draw.vf.cell(tf, fi, "\\:u", i, nr); i<-i+1; { "udieresis"}
draw.vf.cell(tf, fi, "\\'y", i, nr); i<-i+1; { "yacute"}
draw.vf.cell(tf, fi, "\\:y", i, nr); i<-i+1; { "ydieresis"}

nr <- 25
nc <- 2
make.table(nr, nc)
i <- 0
draw.title("Special Escape Sequences", nr, nc)
draw.vf.cell(tf, fi, "\\AR", i, nr); i<-i+1; { "aries"}
draw.vf.cell(tf, fi, "\\TA", i, nr); i<-i+1; { "taurus"}
draw.vf.cell(tf, fi, "\\GE", i, nr); i<-i+1; { "gemini"}
draw.vf.cell(tf, fi, "\\CA", i, nr); i<-i+1; { "cancer"}
draw.vf.cell(tf, fi, "\\LE", i, nr); i<-i+1; { "leo"}
draw.vf.cell(tf, fi, "\\VI", i, nr); i<-i+1; { "virgo"}
draw.vf.cell(tf, fi, "\\LI", i, nr); i<-i+1; { "libra"}
draw.vf.cell(tf, fi, "\\SC", i, nr); i<-i+1; { "scorpio"}
draw.vf.cell(tf, fi, "\\SG", i, nr); i<-i+1; { "sagittarius"}
```

```
draw.vf.cell(tf, fi, "\\CP", i, nr); i<-i+1; { "capricornus"}
draw.vf.cell(tf, fi, "\\AQ", i, nr); i<-i+1; { "aquarius"}
draw.vf.cell(tf, fi, "\\PI", i, nr); i<-i+1; { "pisces"}
draw.vf.cell(tf, fi, "\\~-", i, nr); i<-i+1; { "modifiedcongruent"}
draw.vf.cell(tf, fi, "\\hb", i, nr); i<-i+1; { "hbar"}
draw.vf.cell(tf, fi, "\\IB", i, nr); i<-i+1; { "interbang"}
draw.vf.cell(tf, fi, "\\Lb", i, nr); i<-i+1; { "lambdabar"}
draw.vf.cell(tf, fi, "\\UD", i, nr); i<-i+1; { "undefined"}
draw.vf.cell(tf, fi, "\\SO", i, nr); i<-i+1; { "sun"}
draw.vf.cell(tf, fi, "\\ME", i, nr); i<-i+1; { "mercury"}
draw.vf.cell(tf, fi, "\\VE", i, nr); i<-i+1; { "venus"}
draw.vf.cell(tf, fi, "\\EA", i, nr); i<-i+1; { "earth"}
draw.vf.cell(tf, fi, "\\MA", i, nr); i<-i+1; { "mars"}
draw.vf.cell(tf, fi, "\\JU", i, nr); i<-i+1; { "jupiter"}
draw.vf.cell(tf, fi, "\\SA", i, nr); i<-i+1; { "saturn"}
draw.vf.cell(tf, fi, "\\UR", i, nr); i<-i+1; { "uranus"}
draw.vf.cell(tf, fi, "\\NE", i, nr); i<-i+1; { "neptune"}
draw.vf.cell(tf, fi, "\\PL", i, nr); i<-i+1; { "pluto"}
draw.vf.cell(tf, fi, "\\LU", i, nr); i<-i+1; { "moon"}
draw.vf.cell(tf, fi, "\\CT", i, nr); i<-i+1; { "comet"}
draw.vf.cell(tf, fi, "\\ST", i, nr); i<-i+1; { "star"}
draw.vf.cell(tf, fi, "\\AS", i, nr); i<-i+1; { "ascendingnode"}
draw.vf.cell(tf, fi, "\\DE", i, nr); i<-i+1; { "descendingnode"}
draw.vf.cell(tf, fi, "\\s-", i, nr); i<-i+1; { "s1"}
draw.vf.cell(tf, fi, "\\dg", i, nr); i<-i+1; { "dagger"}
draw.vf.cell(tf, fi, "\\dd", i, nr); i<-i+1; { "daggerdbl"}
draw.vf.cell(tf, fi, "\\li", i, nr); i<-i+1; { "line integral"}
draw.vf.cell(tf, fi, "\\-+", i, nr); i<-i+1; { "minusplus"}
draw.vf.cell(tf, fi, "\\||", i, nr); i<-i+1; { "parallel"}
draw.vf.cell(tf, fi, "\\rn", i, nr); i<-i+1; { "overscore"}
draw.vf.cell(tf, fi, "\\ul", i, nr); i<-i+1; { "underscore"}

nr <- 25
nc <- 3
make.table(nr, nc)
i <- 0
code <- c(300:307,310:317,320:327,330:337,340:347,350:357,360:367,370:377,
          243,263)
string <- c(
"\300","\301","\302","\303","\304","\305","\306","\307",
"\310","\311","\312","\313","\314","\315",
"\316","\317","\320","\321","\322","\323",
"\324","\325","\326","\327","\330","\331",
"\332","\333","\334","\335","\336","\337",
"\340","\341","\342","\343","\344","\345","\346","\347",
"\350","\351","\352","\353","\354","\355",
"\356","\357","\360","\361","\362","\363",
"\364","\365","\366","\367","\370","\371",
"\372","\373","\374","\375","\376","\377","\243","\263")
draw.title("Cyrillic Octal Codes", nr ,nc)
for (i in 1:66)
    draw.vf.cell(tf, "cyrillic", string[i], i-1, nr,
                 raw.string=paste("\\", as.character(code[i]), sep=""))

nr <- 25
nc <- 3
make.table(nr, nc)
```

```
i <- 0
code <- c(252,254,256,262:269,275,278:281,284,745,746,750:768,796:802,
          804:807,809,814:828,830:834,840:844)
draw.title("Raw Hershey Escape Sequences", nr, nc)
for (i in 1:75)
    draw.vf.cell(tf, fi, paste("\\#H",formatC(code[i],wid=4,flag=0),sep=""),
                 i-1, nr)
make.table(nr, nc)
i <- 0
code <- c(845:847,850:856,860:874,899:909,2296:2299,2318:2332,2367:2382,
          4014,4109)
draw.title("More Raw Hershey Escape Sequences", nr, nc)
for (i in 1:73)
    draw.vf.cell(tf, fi, paste("\\#H",formatC(code[i],wid=4,flag=0),sep=""),
                 i-1, nr)

par(oldpar)
```

`hist` *Histograms* `hist`

Description

The generic function `hist` computes and plots (if `plot=T`) a histogram of the given data values.

Usage

```
hist(x, ...)
hist.default(x, breaks, freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     col = NULL, border = par("fg"),
     main = paste("Histogram of", deparse(substitute(x))),
     xlim = range(breaks),  ylim = range(counts, 0),
     xlab = deparse(substitute(x)),  ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, ...)
```

Arguments

`x`	a vector of values for which the histogram is desired.
`breaks`	either a single number giving the approximate number of cells for the histogram or a vector giving the breakpoints between histogram cells.
`freq`	logical; if `TRUE`, the histogram graphic is to present a representation of frequencies, i.e, the `counts` component of the result; if `FALSE`, *relative* frequencies ("probabilities"), the `rel.freqs`, are plotted. Defaults to `TRUE` *iff* `breaks` are equidistant.
`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.
`right`	logical; if `TRUE`, the histograms cells are right-closed (left open) intervals.

`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.
`main, xlab, ylab`	these arguments to `title` have useful defaults here.
`xlim, ylim`	the range of x and y values with sensible defaults.
`plot`	logical. If `TRUE` (default), a histogram is plotted, otherwise a list of breaks and counts is returned.
`labels`	logical. Additionally draw labels on top of bars, if `TRUE`.
`nclass`	numeric (integer). For S compatibility only, `nclass=n` is equivalent to `breaks=n` (n scalar).
`...`	further graphical parameters to `title` and `axis`.

Details

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*".

Value

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.
`intensities`	values $\hat{f}(x_i)$, as estimated density values. If `all(diff(breaks) == 1)`, they are the relative frequencies `counts/n` and in general satisfy $\sum_i \hat{f}(x_i)(b_{i+1} - b_i) = 1$, where $b_i =$ `breaks[i]`.
`mids`	the n cell midpoints; useful for plotting.

Note

The resulting value does *not* depend on the values of the arguments `freq` (or `probability`) or `plot`. This is intentionally different from S.

See Also

`stem`, `density`.

Examples

```
data(islands)
op <- par(mfrow=c(2,2))
hist(islands)
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$mids, r$intensities, r$counts, adj=c(.5,-.5), col='blue3')
```

```
sapply(r[2:3],sum)
sum(r$intensities * diff(r$breaks)) # == 1
par(op)

str(hist(islands, plot= F))
str(hist(islands, br=12, plot= F))
str(hist(islands, br=c(12,20,36,80,200,1000,17000), plot = F))
str(hist(islands, br=c(12,20,36,80,200,1000,17000), freq = TRUE))#warning
```

`hsv` *HSV Color Specification* `hsv`

Description

Create a vector of colors from vectors specifying hue, saturation and value.

Usage

```
hsv(h=1, s=1, v=1, gamma=1)
```

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.
`gamma`	a "gamma correction"

Value

This function creates a vector of "colors" corresponding to the given values in HSV space. The values returned by `hsv` can be used with a `col=` specification in graphics functions or in `par`.

See Also

`rainbow`, `rgb`, `gray`.

Examples

```
hsv(.5,.5,.5)

## Look at gamma effect:
n <- 20;  y <- -sin(3*pi*((1:n)-1/2)/n)
op <- par(mfrow=c(3,2),mar=rep(1.5,4))
for(gamma in c(.4, .6, .8, 1, 1.2, 1.5))
  plot(y,axes=F,frame.plot=T,xlab="",ylab="",pch=21,cex=30,
       bg = rainbow(n, start=.85, end=.1, gamma = gamma),
       main = paste("Red tones;  gamma=",format(gamma)))
par(op)
```

`httpclient` *Read text from an HTTP server* `httpclient`

Description

The function reads text from a URL on an HTTP server into the specified file. It checks for HTTP errors but will behave incorrectly if the data being read contains binary zeros. This function is used by the "socket" method of `download.file`.

Usage

```
httpclient(url, port=80, error.is.fatal=TRUE, check.MIME.type=TRUE,
           file=tempfile(), drop.ctrl.z=TRUE)
```

Arguments

`url`	The URL to read from
`port`	The port (usually 80)
`error.is.fatal`	Stop if an error is encountered
`check.MIME.type`	Require the URL to have a MIME type known to be plain ASCII
`file`	The file to store the text.
`drop.ctrl.z`	Drop ASCII EOF (CTRL-Z) characters from the text

Value

The name of the file containing the downloaded text.

See Also

`download.file`, `read.table.url`

`Hyperbolic` *Hyperbolic Functions* `Hyperbolic`

Description

These functions give the obvious hyperbolic functions. They respectively compute the hyperbolic cosine, sine, tangent, arc-cosine, arc-sine, arc-tangent.

Usage

```
cosh(x)
sinh(x)
tanh(x)
acosh(x)
asinh(x)
atanh(x)
```

See Also

cos, sin, tan, acos, asin, atan.

Examples

```
Ceps <- .Machine$double.eps # ‘‘Computer epsilon’’
x <- rnorm(500)
all(abs(cosh(x) - (exp(x) + exp(-x))/2) < 10*Ceps)
all(abs(sinh(x) - (exp(x) - exp(-x))/2) < 10*Ceps)
all(Mod(cosh(x) - cos(1i*x)) < 10*Ceps)
all(Mod(sinh(x) - sin(1i*x)/1i) < 10*Ceps)
all(abs(tanh(x)*cosh(x) - sinh(x)) < 10*Ceps)

## Inverse:
all(abs(asinh(sinh(x)) - x) < 10*Ceps)
x[abs(acosh(cosh(x)) - abs(x)) > 100*Ceps] #- imprecise for small x
all(abs(atanh(tanh(x)) - x) < 100*Ceps)

all(abs(asinh(x) - log(x + sqrt(x^2 + 1))) < 10*Ceps)
cx <- cosh(x)
all(abs(acosh(cx) - log(cx + sqrt(cx^2 - 1))) < 1000*Ceps)
```

Hypergeometric *The Hypergeometric Distribution* Hypergeometric

---

Description

Density, distribution function, quantile function and random generation for the hypergeometric distribution.

Usage

```
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	the number of observations to be generated.
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

$$p(x) = \binom{m}{x}\binom{n}{k-x} \Big/ \binom{m+n}{k}$$

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

Johnson, N. L., Kotz, S., and Kemp, A. W. (1992) *Univariate Discrete Distributions*, Second Edition. New York: Wiley.

Examples

```
m <- 10; n <- 7; k <- 8
x <- 0:m
rbind(phyper(x, m, n, k), dhyper(x, m, n, k))
all(phyper(x, m, n, k) == cumsum(dhyper(x, m, n, k)))# FALSE
## Error :
signif(phyper(x, m, n, k) - cumsum(dhyper(x, m, n, k)), dig=3)
```

`identify` *Identify Points in a Scatter Plot* `identify`

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 to the pointer, its index will be returned as part of the value of the call.

Usage

```
identify(x, ...)
identify.default(x, y = NULL, labels = seq(along = x), pos = FALSE,
                 n = length(x), plot = TRUE, offset = 0.5, ...)
```

Arguments

`x,y`	coordinates of points in a scatter plot. Alternatively, any object which defines coordinates (a plotting structure, time series etc.) 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 (1=below, 2=left, 3=above and 4=right).

`n`	the maximum number of points to be identified.
`plot`	if `plot` is `TRUE`, the labels are printed at the points and if `FALSE` they are omitted.
`offset`	the distance (in character widths) which separates the label from identified points.
`...`	further arguments to `par(.)`.

Details

If in addition, `plot` is `TRUE`, the point is labelled with the corresponding element of `text`.

The labels are placed either below, to the left, above or to the right of the identified point, depending on where the cursor was.

The identification process is terminated by pressing any mouse button other than the first, or by clicking outside the graphics window.

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.

`ifelse` *Conditional Element Selection* `ifelse`

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`. If `yes` or `no` are too short, their elements are recycled.

Usage

```
ifelse(test, yes, no)
```

See Also

`if`.

Examples

```
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)
```

`image` *Display a Color Image* `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".

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

```
image(x, y, z, zlim, col = heat.colors(12),
      add = FALSE, xaxs = "i", yaxs = "i", xlab, ylab, ...)
```

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`s are allowed). Note that `x` can be used instead of `z` for convenience.
`zlim`	the minimum and maximum `z` values for which colors will be plotted. Each of the given colors will be used to color an equispaced interval of this range.
`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.
`...`	graphical parameters for `plot` may also be passed as arguments to this function.

Details

The length of `x` should be equal to the `nrow(x)+1` or `nrow(x)`. 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.

Note

Based on a function by Thomas Lumley ⟨thomas@biostat.washington.edu⟩.

See Also

contour, heat.colors, topo.colors, terrain.colors, rainbow, hsv, par.

Examples

```
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=F, main="Math can be beautiful ...",
      xlab=expression(cos(r^2) * e^{-r/6}))
contour(z, add=T, drawlabels=F)

data(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)
```

`index.search` *Search Indices for Help Files* `index.search`

Description

Used to search the indices for help files, possibly under aliases.

Usage

```
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 searches.
`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 'help', 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

`infert` *Infertility after Spontaneous and Induced Abortion* `infert`

Description

This is a matched case-control study dating from before the availability of conditional logistic regression.

Usage

```
data(infert)
```

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	0 = 0
	induced abortions	1 = 1
		2 = 2 or more
5.	case status	1 = case
		0 = control
6.	number of prior	0 = 0
	spontaneous abortions	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

Trichopoulos et al. (1976) *Br. J. of Obst. and Gynaec.* **83**, 645–650.

Examples

```
data(infert)
model1 <- glm(case ~ spontaneous+induced, data=infert,family=binomial())
summary(model1)
## 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 equivalent to a Cox model :
if(require(survival5)){
  faketime <- rep(42,nrow(infert))
  model3 <- coxph(Surv(faketime,case)~spontaneous+induced+strata(stratum),
                data=infert,method="exact")
  summary(model3)
  detach()# survival5 (conflicts)
}
```

`influence.measures` *Regression Diagnostics* `influence.measures`

Description

This suite of functions can be used to compute some of the regression diagnostics discussed in Belsley, Kuh and Welsch (1980), and in Cook and Weisberg (1982).

Usage

```
influence.measures(lm.obj)
summary.infl (object, digits = max(2, getOption("digits") - 5), ...)
  print.infl (x, digits = max(3, getOption("digits") - 4), ...)

rstandard(lm.obj,
          infl = lm.influence(lm.obj),
          res = weighted.residuals(lm.obj),
          sd = sqrt(deviance(lm.obj)/df.residual(lm.obj)))
rstudent (lm.obj, infl = ..., res = ...)
dffits   (lm.obj, infl = ..., res = ...)
dfbetas  (lm.obj, infl = ...)
covratio (lm.obj, infl = ..., res = ...)
cooks.distance(lm.obj, infl = ..., res = ..., sd = ...)

hat(xmat)
```

Arguments

`lm.obj`	the resulting object returned by `lm`.
`infl`	influence structure as returned by `lm.influence`.
`res`	(possibly weighted) residuals, with proper default.
`sd`	standard deviation to use, see default.
`xmat`	the 'X' or design matrix.

Details

The primary 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.)

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`) are already available.

Note that cases with `weights == 0` are *dropped* from all these functions.

References

Belsley, D. A., Kuh, E. and Welsch, R. E. (1980) *Regression Diagnostics.* New York: Wiley.

Cook, R. D. and Weisber,g S. (1982) *Residuals and Influence in Regression.* London: Chapman and Hall.

See Also

`lm.influence`.

Examples

```
## Analysis of the life-cycle savings data
## given in Belsley, Kuh and Welsch.
data(LifeCycleSavings)
lm.SR <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)
summary(inflm.SR <- influence.measures(lm.SR))
inflm.SR
which(apply(inflm.SR$is.inf, 1, any)) # which observations 'are' influential
dim(dfb <- dfbetas(lm.SR))            # the 1st columns of influence.measures
all(dfb == inflm.SR$infmat[, 1:5])
rstandard(lm.SR)
rstudent(lm.SR)
dffits(lm.SR)
covratio(lm.SR)

## 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))
influence.measures(lmH)
```

`InsectSprays` *Effectiveness of Insect Sprays* `InsectSprays`

Description

The counts of insects in agricultural experimental units treated with different insecticides.

Usage

```
data(InsectSprays)
```

Format

A data frame with 72 observations on 2 variables.

[,1]	count	numeric	Insect count
[,2]	spray	factor	The type of spray

Source

Beall, G., (1942) The Transformation of data from entomological field experiments, *Biometrika*, **29**, 243–262.

References

McNeil, D. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(InsectSprays)
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)
```

INSTALL *Install Add-on Packages* INSTALL

Description

To install packages into the default library tree (which is rooted at '$R_HOME/library'), do `R INSTALL pkgs`.

To install into the library tree `lib` instead of the default one, use `R INSTALL -l lib pkgs`.

Usage

```
R INSTALL [options] [-l lib] pkgs
```

Arguments

`pkgs`	A list with the path names of the packages to be installed.
`lib`	the path name of the R library tree to install to.
`options`	a list of options through which in particular the build process for help files can be controlled. If `--no-docs` is given, no help files are built. `--no-text`, `--no-html`, and `--no-latex` suppress creating the text, HTML, and LaTeX versions, respectively. The default is to build help files in all three versions.

Details

Both `lib` and the elements of `pkgs` may be absolute or relative path names. `pkgs` can also contain name of package archive files of the form 'pkg_version.tar.gz' as obtained from CRAN, these are then extracted in a temporary directory.

For checking (via the examples in the help files 'man/*.Rd', use `R CMD check [-l lib] <pkg>`, use `R CMD check --help` for more information.

See Also

`REMOVE`, and `library` for information on using several library trees and creating packages; `update.packages` for automatic update of packages using the internet.

`integer` *Integer Vectors* `integer`

Description

Creates or tests for objects of type `"integer"`.

Usage

```
integer(length = 0)
as.integer(x)
is.integer(x)
```

Value

`integer` creates a integer vector of the specified length. Each element of the vector is equal to `0`. Integer vectors exist so that data can be passed to C or Fortran code which expects them.

`as.integer` attempts to coerce its argument to be of integer type.

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

`interaction` *Compute Factor Interactions* `interaction`

Description

`interaction` computes a factor which represents the interaction of the given factors. The result of `interaction` is always unordered.

Usage

```
interaction(..., drop=FALSE)
```

Arguments

`...`	The factors for which interaction is to be computed.
`drop`	If `drop` is `TRUE`, empty factor levels are dropped from the result. The default is to retain all factor levels.

Value

A factor which represents the interaction of the given factors.

See Also

`factor`.

Examples

```
a <- gl(2, 2, 8)
b <- gl(2, 4, 8)
interaction(a, b)
```

`interactive` *Is R Running Interactively?* `interactive`

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* `Internal`

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)
```

See Also

`.Primitive`, `.C`, `.Fortran`.

`invisible` *Change the Print Mode to Invisible* `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.

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
```

`IQR` *The Interquartile Range* `IQR`

Description

computes interquartile range of the `x` values.

Usage

```
IQR(x, na.rm = FALSE)
```

Details

Note that this function computes the quartiles using the `quantile` function rather than following Tukey's recommendations, i.e., `IQR(x) = quantile(x,3/4) - quantile(x,1/4)`.

For normally $N(m, 1)$ distributed X, the expected value of `IQR(X)` is `2*qnorm(3/4) = 1.3490`, i.e., for a normal-consistent estimate of the standard deviation, use `IQR(x) / 1.349`.

References

Tukey, J. W. (1977). *Exploratory Data Analysis.* Reading: Addison-Wesley.

See Also

`fivenum`, `mad` which is more robust, `range`, `quantile`.

Examples

```
data(rivers)
IQR(rivers)
```

`iris` *Edgar Anderson's Iris Data* `iris`

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

```
data(iris)
data(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

Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. *Annals of Eugenics,* **7**, Part II, 179–188.

The data were collected by Anderson, Edgar (1935). The irises of the Gaspe Peninsula, *Bulletin of the American Iris Society,* **59**, 2–5.

See Also

`matplot` some examples of which use `iris`.

`is.empty.model` *Check if a Model is Empty* `is.empty.model`

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

```
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

```
y <- rnorm(20)
is.empty.model(y ~ 0)
is.empty.model(y ~ -1)
is.empty.model(lm(y ~ 0))
```

`is.finite` *Finite, Infinite and NaN Numbers* `is.finite`

Description

`is.finite` and `is.infinite` return a vector of the same length as `x`, indicating which elements are finite or not.

`Inf` and `-Inf` are positive and negative 'infinity' whereas `NaN` means "Not a Number".

Usage

```
is.finite(x)
is.infinite(x)
Inf
NaN
is.nan(x)
```

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`).

`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`).

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 `Inf`s really are statements with a proper mathematical *limit*, see the many examples below.

References

ANSI/IEEE 754 Floating-Point Standard.

Currently (6/1999), Bill M.'s ⟨billm@melbpc.org.au⟩ tutorial and examples at
http://www.linuxsupportline.com/~billm/

See Also

`NA`, *'Not Available'* which is not a number as well, however usually used for missing values.

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

1/0 == Inf
1/Inf == 0
exp(-Inf) == 0
##               (actually, the last one seems to give NA on not-very-new
##                versions of Linux, which is a Linux bug and seems to be
##                corrected in newer 'libc6' based Linuxen).

is.na(0/0) # T
!is.na(Inf)# T
is.nan(0/0)# T

(!is.nan(NA))  &&  (!is.infinite(NA))  && (!is.finite(NA)) # TRUE!!
( is.nan(NaN)) &&  (!is.infinite(NaN)) && (!is.finite(NaN))# TRUE!!

!is.nan(NA)
all(!is.nan(c(1,NA)))
all(c(F,T,F) == is.nan(c   (1,NaN,NA)))
all(c(F,T,F) == is.nan(list(1,NaN,NA)))#-> FALSE  'BUGlet' [coerce.c]
                                # However, S is different anyway.

lgamma(Inf) == Inf
Inf + Inf == Inf
Inf - Inf == NaN # NA --- should test with  'is.nan()
```

```
(1/0) * (1/0)# Inf
(1/0) / (1/0)# NaN

pm <- c(-1,1) # 'pm' = plus/minus

log(0) == - 1/0
exp(-Inf) == 0

sin(Inf)
cos(Inf)
tan(Inf)
all(atan(Inf*pm) == pm*pi/2) # TRUE

x <- c(100,-1e-13,Inf,-Inf, NaN, pi, NA)
x #  1.000000 -3.000000       Inf      -Inf        NA  3.141593        NA
names(x) <- formatC(x, dig=3)
is.finite(x)
##-   100 -1e-13 Inf -Inf NaN 3.14 NA
##-     T      T   .    .   .    T  .
is.na(x)
##-   100 -1e-13 Inf -Inf NaN 3.14 NA
##-     .      .   .    .   T    .  T
which(is.na(x) & !is.nan(x))# only 'NA': 7

is.na(x) | is.finite(x)
##-   100 -1e-13 Inf -Inf NaN 3.14 NA
##-     T      T   .    .   T    T  T
is.infinite(x)
##-   100 -1e-13 Inf -Inf NaN 3.14 NA
##-     .      .   T    T   .    .  .

##-- either  finite or infinite  or  NA:
all(is.na(x) != is.finite(x) | is.infinite(x)) # TRUE
all(is.nan(x) != is.finite(x) | is.infinite(x)) # FALSE: have 'real' NA

##--- Integer
(ix <- structure(as.integer(x),names= names(x)))
##-   100 -1e-13        Inf        -Inf NaN 3.14 NA
##-   100      . 2147483647 -2147483648  NA    3 NA
all(is.na(ix) != is.finite(ix) | is.infinite(ix)) # TRUE (still)

ix[3] == (iI <- as.integer(Inf))#> warning: inaccurate integer conversion!
ix[4] == (imI<- as.integer(-Inf))
iI  == .Machine$integer.max # TRUE
imI == -.Machine$integer.max # TRUE

##--- Overflow in simple integer arithmetic:
as.integer(2)*iI #  -2
as.integer(3)*iI #  2147483645
as.integer(3)*iI == iI-2 # TRUE

storage.mode(ii <- -3:5)
storage.mode(zm <- outer(ii,ii, FUN="*"))# integer
storage.mode(zd <- outer(ii,ii, FUN="/"))# double
range(zd, na.rm=T)# -Inf  Inf
zd[,ii==0]
```

```
(storage.mode(print(1:1 / 0:0)))# Inf "double"
(storage.mode(print(1:1 / 1:1)))# 1 "double"
(storage.mode(print(1:1 + 1:1)))# 2 "integer"
(storage.mode(print(2:2 * 2:2)))# 4 "integer"
```

`is.function` *Is an Object of Type Function?* `is.function`

Description

Checks whether its argument is a function.

Usage

```
is.function(x)
```

Arguments

`x`	an R object.

Value

`TRUE` if `x` is a function, and `FALSE` otherwise.

`is.language` *Is an Object a Language Object?* `is.language`

Description

`is.language` returns `TRUE` if `x` is either a variable `name`, a `call`, or an `expression`.

Usage

```
is.language(x)
```

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)
sapply(ll, is.language)# all TRUE
```

`is.object` *Is an Object "internally classed"?* `is.object`

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)
```

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?* `is.R`

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

```
if (exists(is.R) && is.function(is.R) && is.R()) {
## R-specific code
} else {
## S-version of code
}
```

Value

`is.R` returns `TRUE` if we are using R and `FALSE` otherwise.

See Also

`R.version`, `system`.

Examples

```
x <- runif(20); small <- x < 0.4
# which only exists in R:
if(is.R()) which(small) else seq(along=small)[small]
```

`is.recursive` *Is an Object Atomic or Recursive?* `is.recursive`

Description

`is.atomic` returns `TRUE` if `x` does not have a list structure 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)
```

See Also

`is.list`, `is.language`, etc, and the `demo("is.things")`.

Examples

```
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)        #   "
is.a.r(y ~ x)     #   "
is.a.r(expression(x+1))# should be F-T (not in 0.62.3!)
```

`is.single` *Is an Object of Single Precision Type?* `is.single`

Description

`is.single` reports an error. There are no single precision values in R.

Usage

```
is.single(x)
```

`is.symbol` *Is an Object of Type Symbol?* `is.symbol`

Description

Checks whether its argument is a symbol.

Usage

```
is.symbol(x)
```

Arguments

`x`	an R object.

Value

`TRUE` if `x` is of type "symbol", and `FALSE` otherwise.

`islands` *Areas of the World's Major Landmasses* `islands`

Description

The areas in thousands of square miles of the landmasses which exceed 10,000 square miles.

Usage

```
data(islands)
```

Format

A named vector of length 48.

Source

The World Almanac and Book of Facts, 1975, page 406.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* Wiley.

Examples

```
data(islands)
dotplot(log(islands, 10),
   main = "islands data: log10(area) (log10(sq. miles))")
dotplot(log(islands[order(islands)], 10),
   main = "islands data: log10(area) (log10(sq. miles))")
```

`Japanese` *Japanese characters in R* `Japanese`

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.

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 `example(Japanese)` shows tables of the escape sequences for the available Japanese characters.

References

http://www.gnu.org/software/plotutils/plotutils.html

See Also

Hershey, text, contour

Examples

```
plot(1:9, type="n", axes=F, xlab="", ylab="")
box()
par(cex=3)
text(4, 2, "\\#J2438\\#J2421\\#J2451\\#J2473", vfont=c("serif", "plain"))
text(4, 4, "\\#J2538\\#J2521\\#J2551\\#J2573", vfont=c("serif", "plain"))
text(4, 6, "\\#J467c\\#J4b5c", vfont=c("serif", "plain"))
text(4, 8, "Japan", vfont=c("serif", "plain"))
par(cex=1)
text(8, 2, "Hiragana")
text(8, 4, "Katakana")
text(8, 6, "Kanji")
text(8, 8, "English")

######
# create tables of Japanese characters
######
make.table <- function(nr, nc) {
    savepar <- par(mar=rep(0, 4), pty="s")
    plot(c(0, nc*2 + 1), c(0, -(nr + 1)),
         type="n", xlab="", ylab="", axes=F)
    savepar
}

get.r <- function(i, nr) {
    i %% nr + 1
}
```

```
get.c <- function(i, nr) {
    i %/% nr + 1
}

draw.title <- function(title, nc) {
    text((nc*2 + 1)/2, 0, title, font=2)
}

draw.vf.cell <- function(typeface, fontindex, string, i, nr, raw.string=NULL) {
    r <- get.r(i, nr)
    c <- get.c(i, nr)
    if (is.null(raw.string))
        raw.string <- paste("\\", string, sep="")
    text((2*(c - 1) + 1), -r, raw.string, col="grey")
    text((2*c), -r, string, vfont=c(typeface, fontindex))
    rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5), border="grey")
}

make.table2 <- function(nr, nc) {
    savepar <- par(mar=rep(0, 4), pty="s")
    plot(c(0, nc*3 + 1), c(0, -(nr + 1)),
         type="n", xlab="", ylab="", axes=F)
    savepar
}

draw.title2 <- function(title, nc) {
    text((nc*3 + 1)/2, 0, title, font=2)
}

draw.vf.cell2 <- function(string, alt, i, nr) {
    r <- get.r(i, nr)
    c <- get.c(i, nr)
    text((3*(c - 1) + 1), -r, paste("\\", string, sep=""), col="grey")
    text((3*(c - 1) + 2), -r, paste("\\", alt, sep=""), col="grey", cex=.6)
    text((3*c), -r, string, vfont=c("serif", "plain"))
    rect((3*(c - 1) + .5), -(r - .5), (3*c + .5), -(r + .5), border="grey")
}

tf <- "serif"
fi <- "plain"
nr <- 25
nc <- 4
oldpar <- make.table(nr, nc)
index <- 0
digits <- c(0:9,"a","b","c","d","e","f")
draw.title("Hiragana", nc)
for (i in 2:7) {
    for (j in 1:16) {
        if (!((i == 2 && j == 1) || (i == 7 && j > 4))) {
            draw.vf.cell(tf, fi, paste("\\#J24", i, digits[j], sep=""),
                         index, nr)
            index <- index + 1
        }
    }
}
```

```
nr <- 25
nc <- 4
make.table(nr, nc)
index <- 0
digits <- c(0:9,"a","b","c","d","e","f")
draw.title("Katakana", nc)
for (i in 2:7) {
    for (j in 1:16) {
        if (!((i == 2 && j == 1) || (i == 7 && j > 7))) {
            draw.vf.cell(tf, fi, paste("\\#J25", i, digits[j], sep=""),
                         index, nr)
            index <- index + 1
        }
    }
}

nr <- 26
nc <- 3
make.table2(nr, nc)
i <- 0
draw.title2("Kanji", nc)
draw.vf.cell2("\\#J3021", "\\#N0043", i, nr); i <- i + 1
draw.vf.cell2("\\#J3026", "\\#N2829", i, nr); i <- i + 1
draw.vf.cell2("\\#J302d", "\\#N0062", i, nr); i <- i + 1
draw.vf.cell2("\\#J3035", "\\#N0818", i, nr); i <- i + 1
draw.vf.cell2("\\#J303f", "\\#N1802", i, nr); i <- i + 1
draw.vf.cell2("\\#J3045", "\\#N2154", i, nr); i <- i + 1
draw.vf.cell2("\\#J304c", "\\#N0401", i, nr); i <- i + 1
draw.vf.cell2("\\#J3057", "\\#N2107", i, nr); i <- i + 1
draw.vf.cell2("\\#J3059", "\\#N0138", i, nr); i <- i + 1
draw.vf.cell2("\\#J305b", "\\#N3008", i, nr); i <- i + 1
draw.vf.cell2("\\#J305e", "\\#N3579", i, nr); i <- i + 1
draw.vf.cell2("\\#J3061", "\\#N4214", i, nr); i <- i + 1
draw.vf.cell2("\\#J306c", "\\#N0001", i, nr); i <- i + 1
draw.vf.cell2("\\#J3070", "\\#N3294", i, nr); i <- i + 1
draw.vf.cell2("\\#J3078", "\\#N1026", i, nr); i <- i + 1
draw.vf.cell2("\\#J307a", "\\#N1562", i, nr); i <- i + 1
draw.vf.cell2("\\#J3122", "\\#N5006", i, nr); i <- i + 1
draw.vf.cell2("\\#J3126", "\\#N0878", i, nr); i <- i + 1
draw.vf.cell2("\\#J3127", "\\#N1280", i, nr); i <- i + 1
draw.vf.cell2("\\#J3129", "\\#N3673", i, nr); i <- i + 1
draw.vf.cell2("\\#J312b", "\\#N5042", i, nr); i <- i + 1
draw.vf.cell2("\\#J3132", "\\#N2629", i, nr); i <- i + 1
draw.vf.cell2("\\#J313b", "\\#N2973", i, nr); i <- i + 1
draw.vf.cell2("\\#J313f", "\\#N4725", i, nr); i <- i + 1
draw.vf.cell2("\\#J3140", "\\#N5046", i, nr); i <- i + 1
draw.vf.cell2("\\#J314a", "\\#N0130", i, nr); i <- i + 1
draw.vf.cell2("\\#J3155", "\\#N2599", i, nr); i <- i + 1
draw.vf.cell2("\\#J315f", "\\#N0617", i, nr); i <- i + 1
draw.vf.cell2("\\#J3173", "\\#N4733", i, nr); i <- i + 1
draw.vf.cell2("\\#J3176", "\\#N1125", i, nr); i <- i + 1
draw.vf.cell2("\\#J3177", "\\#N2083", i, nr); i <- i + 1
draw.vf.cell2("\\#J317e", "\\#N1504", i, nr); i <- i + 1
draw.vf.cell2("\\#J3221", "\\#N1885", i, nr); i <- i + 1
draw.vf.cell2("\\#J3223", "\\#N2361", i, nr); i <- i + 1
draw.vf.cell2("\\#J3226", "\\#N2922", i, nr); i <- i + 1
draw.vf.cell2("\\#J322b", "\\#N5399", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J322f", "\\#N0551", i, nr); i <- i + 1
draw.vf.cell2("\\#J3235", "\\#N0260", i, nr); i <- i + 1
draw.vf.cell2("\\#J3239", "\\#N2634", i, nr); i <- i + 1
draw.vf.cell2("\\#J323b", "\\#N5110", i, nr); i <- i + 1
draw.vf.cell2("\\#J323c", "\\#N0009", i, nr); i <- i + 1
draw.vf.cell2("\\#J323d", "\\#N0350", i, nr); i <- i + 1
draw.vf.cell2("\\#J323f", "\\#N0409", i, nr); i <- i + 1
draw.vf.cell2("\\#J3241", "\\#N0422", i, nr); i <- i + 1
draw.vf.cell2("\\#J3243", "\\#N0716", i, nr); i <- i + 1
draw.vf.cell2("\\#J3244", "\\#N0024", i, nr); i <- i + 1
draw.vf.cell2("\\#J3246", "\\#N0058", i, nr); i <- i + 1
draw.vf.cell2("\\#J3248", "\\#N1311", i, nr); i <- i + 1
draw.vf.cell2("\\#J324a", "\\#N3272", i, nr); i <- i + 1
draw.vf.cell2("\\#J324c", "\\#N0107", i, nr); i <- i + 1
draw.vf.cell2("\\#J324f", "\\#N2530", i, nr); i <- i + 1
draw.vf.cell2("\\#J3250", "\\#N2743", i, nr); i <- i + 1
draw.vf.cell2("\\#J3256", "\\#N3909", i, nr); i <- i + 1
draw.vf.cell2("\\#J3259", "\\#N3956", i, nr); i <- i + 1
draw.vf.cell2("\\#J3261", "\\#N4723", i, nr); i <- i + 1
draw.vf.cell2("\\#J3267", "\\#N2848", i, nr); i <- i + 1
draw.vf.cell2("\\#J3268", "\\#N0050", i, nr); i <- i + 1
draw.vf.cell2("\\#J3272", "\\#N4306", i, nr); i <- i + 1
draw.vf.cell2("\\#J3273", "\\#N1028", i, nr); i <- i + 1
draw.vf.cell2("\\#J3323", "\\#N2264", i, nr); i <- i + 1
draw.vf.cell2("\\#J3324", "\\#N2553", i, nr); i <- i + 1
draw.vf.cell2("\\#J3326", "\\#N2998", i, nr); i <- i + 1
draw.vf.cell2("\\#J3328", "\\#N3537", i, nr); i <- i + 1
draw.vf.cell2("\\#J332b", "\\#N4950", i, nr); i <- i + 1
draw.vf.cell2("\\#J332d", "\\#N4486", i, nr); i <- i + 1
draw.vf.cell2("\\#J3330", "\\#N1168", i, nr); i <- i + 1
draw.vf.cell2("\\#J3346", "\\#N1163", i, nr); i <- i + 1
draw.vf.cell2("\\#J334b", "\\#N2254", i, nr); i <- i + 1
draw.vf.cell2("\\#J3351", "\\#N4301", i, nr); i <- i + 1
draw.vf.cell2("\\#J3353", "\\#N4623", i, nr); i <- i + 1
draw.vf.cell2("\\#J3357", "\\#N5088", i, nr); i <- i + 1
draw.vf.cell2("\\#J3358", "\\#N1271", i, nr); i <- i + 1
draw.vf.cell2("\\#J335a", "\\#N2324", i, nr); i <- i + 1
draw.vf.cell2("\\#J3364", "\\#N0703", i, nr); i <- i + 1
draw.vf.cell2("\\#J3424", "\\#N2977", i, nr); i <- i + 1
draw.vf.cell2("\\#J3428", "\\#N1322", i, nr); i <- i + 1

make.table2(nr, nc)
i <- 0
draw.title2("Kanji", nc)
draw.vf.cell2("\\#J342c", "\\#N1466", i, nr); i <- i + 1
draw.vf.cell2("\\#J3433", "\\#N1492", i, nr); i <- i + 1
draw.vf.cell2("\\#J3434", "\\#N0790", i, nr); i <- i + 1
draw.vf.cell2("\\#J3436", "\\#N1731", i, nr); i <- i + 1
draw.vf.cell2("\\#J3437", "\\#N1756", i, nr); i <- i + 1
draw.vf.cell2("\\#J3445", "\\#N2988", i, nr); i <- i + 1
draw.vf.cell2("\\#J3449", "\\#N3416", i, nr); i <- i + 1
draw.vf.cell2("\\#J3454", "\\#N4750", i, nr); i <- i + 1
draw.vf.cell2("\\#J3456", "\\#N4949", i, nr); i <- i + 1
draw.vf.cell2("\\#J3458", "\\#N4958", i, nr); i <- i + 1
draw.vf.cell2("\\#J346f", "\\#N0994", i, nr); i <- i + 1
draw.vf.cell2("\\#J3470", "\\#N1098", i, nr); i <- i + 1
draw.vf.cell2("\\#J3476", "\\#N1496", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J347c", "\\#N3785", i, nr); i <- i + 1
draw.vf.cell2("\\#J3521", "\\#N2379", i, nr); i <- i + 1
draw.vf.cell2("\\#J3522", "\\#N1582", i, nr); i <- i + 1
draw.vf.cell2("\\#J3524", "\\#N2480", i, nr); i <- i + 1
draw.vf.cell2("\\#J3525", "\\#N2507", i, nr); i <- i + 1
draw.vf.cell2("\\#J352d", "\\#N4318", i, nr); i <- i + 1
draw.vf.cell2("\\#J3530", "\\#N4610", i, nr); i <- i + 1
draw.vf.cell2("\\#J3534", "\\#N5276", i, nr); i <- i + 1
draw.vf.cell2("\\#J3535", "\\#N5445", i, nr); i <- i + 1
draw.vf.cell2("\\#J3546", "\\#N3981", i, nr); i <- i + 1
draw.vf.cell2("\\#J3555", "\\#N4685", i, nr); i <- i + 1
draw.vf.cell2("\\#J355a", "\\#N0154", i, nr); i <- i + 1
draw.vf.cell2("\\#J355b", "\\#N0885", i, nr); i <- i + 1
draw.vf.cell2("\\#J355d", "\\#N1560", i, nr); i <- i + 1
draw.vf.cell2("\\#J3565", "\\#N2941", i, nr); i <- i + 1
draw.vf.cell2("\\#J3566", "\\#N3314", i, nr); i <- i + 1
draw.vf.cell2("\\#J3569", "\\#N3496", i, nr); i <- i + 1
draw.vf.cell2("\\#J356d", "\\#N2852", i, nr); i <- i + 1
draw.vf.cell2("\\#J356e", "\\#N1051", i, nr); i <- i + 1
draw.vf.cell2("\\#J356f", "\\#N1387", i, nr); i <- i + 1
draw.vf.cell2("\\#J3575", "\\#N4109", i, nr); i <- i + 1
draw.vf.cell2("\\#J3577", "\\#N4548", i, nr); i <- i + 1
draw.vf.cell2("\\#J357b", "\\#N5281", i, nr); i <- i + 1
draw.vf.cell2("\\#J357e", "\\#N0295", i, nr); i <- i + 1
draw.vf.cell2("\\#J3621", "\\#N0431", i, nr); i <- i + 1
draw.vf.cell2("\\#J3626", "\\#N0581", i, nr); i <- i + 1
draw.vf.cell2("\\#J362d", "\\#N1135", i, nr); i <- i + 1
draw.vf.cell2("\\#J362f", "\\#N1571", i, nr); i <- i + 1
draw.vf.cell2("\\#J3635", "\\#N2052", i, nr); i <- i + 1
draw.vf.cell2("\\#J3636", "\\#N2378", i, nr); i <- i + 1
draw.vf.cell2("\\#J364a", "\\#N0103", i, nr); i <- i + 1
draw.vf.cell2("\\#J364b", "\\#N2305", i, nr); i <- i + 1
draw.vf.cell2("\\#J364c", "\\#N2923", i, nr); i <- i + 1
draw.vf.cell2("\\#J3651", "\\#N1065", i, nr); i <- i + 1
draw.vf.cell2("\\#J3661", "\\#N4671", i, nr); i <- i + 1
draw.vf.cell2("\\#J3662", "\\#N4815", i, nr); i <- i + 1
draw.vf.cell2("\\#J3664", "\\#N4855", i, nr); i <- i + 1
draw.vf.cell2("\\#J3665", "\\#N0146", i, nr); i <- i + 1
draw.vf.cell2("\\#J3671", "\\#N3128", i, nr); i <- i + 1
draw.vf.cell2("\\#J3675", "\\#N3317", i, nr); i <- i + 1
draw.vf.cell2("\\#J367e", "\\#N1386", i, nr); i <- i + 1
draw.vf.cell2("\\#J3738", "\\#N0449", i, nr); i <- i + 1
draw.vf.cell2("\\#J3739", "\\#N0534", i, nr); i <- i + 1
draw.vf.cell2("\\#J373e", "\\#N2937", i, nr); i <- i + 1
draw.vf.cell2("\\#J373f", "\\#N1077", i, nr); i <- i + 1
draw.vf.cell2("\\#J3741", "\\#N1589", i, nr); i <- i + 1
draw.vf.cell2("\\#J3742", "\\#N1602", i, nr); i <- i + 1
draw.vf.cell2("\\#J374f", "\\#N0195", i, nr); i <- i + 1
draw.vf.cell2("\\#J3750", "\\#N3523", i, nr); i <- i + 1
draw.vf.cell2("\\#J3757", "\\#N4312", i, nr); i <- i + 1
draw.vf.cell2("\\#J375a", "\\#N4620", i, nr); i <- i + 1
draw.vf.cell2("\\#J3767", "\\#N2412", i, nr); i <- i + 1
draw.vf.cell2("\\#J3768", "\\#N2509", i, nr); i <- i + 1
draw.vf.cell2("\\#J376a", "\\#N3313", i, nr); i <- i + 1
draw.vf.cell2("\\#J376b", "\\#N3540", i, nr); i <- i + 1
draw.vf.cell2("\\#J376c", "\\#N4205", i, nr); i <- i + 1
draw.vf.cell2("\\#J376e", "\\#N2169", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J3777", "\\#N1045", i, nr); i <- i + 1
draw.vf.cell2("\\#J3824", "\\#N2868", i, nr); i <- i + 1
draw.vf.cell2("\\#J3826", "\\#N3180", i, nr); i <- i + 1
draw.vf.cell2("\\#J3828", "\\#N3543", i, nr); i <- i + 1
draw.vf.cell2("\\#J382b", "\\#N4284", i, nr); i <- i + 1
draw.vf.cell2("\\#J3833", "\\#N5220", i, nr); i <- i + 1

make.table2(nr, nc)
i <- 0
draw.title2("Kanji", nc)
draw.vf.cell2("\\#J3835", "\\#N0275", i, nr); i <- i + 1
draw.vf.cell2("\\#J3836", "\\#N0825", i, nr); i <- i + 1
draw.vf.cell2("\\#J3839", "\\#N1568", i, nr); i <- i + 1
draw.vf.cell2("\\#J383a", "\\#N2637", i, nr); i <- i + 1
draw.vf.cell2("\\#J383b", "\\#N2656", i, nr); i <- i + 1
draw.vf.cell2("\\#J383d", "\\#N2943", i, nr); i <- i + 1
draw.vf.cell2("\\#J3840", "\\#N4309", i, nr); i <- i + 1
draw.vf.cell2("\\#J3842", "\\#N4987", i, nr); i <- i + 1
draw.vf.cell2("\\#J3845", "\\#N0770", i, nr); i <- i + 1
draw.vf.cell2("\\#J3847", "\\#N1036", i, nr); i <- i + 1
draw.vf.cell2("\\#J384c", "\\#N1567", i, nr); i <- i + 1
draw.vf.cell2("\\#J384d", "\\#N1817", i, nr); i <- i + 1
draw.vf.cell2("\\#J384e", "\\#N2044", i, nr); i <- i + 1
draw.vf.cell2("\\#J385d", "\\#N5415", i, nr); i <- i + 1
draw.vf.cell2("\\#J385e", "\\#N0015", i, nr); i <- i + 1
draw.vf.cell2("\\#J3861", "\\#N0162", i, nr); i <- i + 1
draw.vf.cell2("\\#J3865", "\\#N1610", i, nr); i <- i + 1
draw.vf.cell2("\\#J3866", "\\#N1628", i, nr); i <- i + 1
draw.vf.cell2("\\#J386c", "\\#N4374", i, nr); i <- i + 1
draw.vf.cell2("\\#J3872", "\\#N0290", i, nr); i <- i + 1
draw.vf.cell2("\\#J3877", "\\#N1358", i, nr); i <- i + 1
draw.vf.cell2("\\#J3878", "\\#N0579", i, nr); i <- i + 1
draw.vf.cell2("\\#J387d", "\\#N0868", i, nr); i <- i + 1
draw.vf.cell2("\\#J387e", "\\#N0101", i, nr); i <- i + 1
draw.vf.cell2("\\#J3929", "\\#N1451", i, nr); i <- i + 1
draw.vf.cell2("\\#J3931", "\\#N1683", i, nr); i <- i + 1
draw.vf.cell2("\\#J393d", "\\#N2343", i, nr); i <- i + 1
draw.vf.cell2("\\#J3943", "\\#N0092", i, nr); i <- i + 1
draw.vf.cell2("\\#J394d", "\\#N3684", i, nr); i <- i + 1
draw.vf.cell2("\\#J3954", "\\#N4213", i, nr); i <- i + 1
draw.vf.cell2("\\#J3955", "\\#N1641", i, nr); i <- i + 1
draw.vf.cell2("\\#J395b", "\\#N4843", i, nr); i <- i + 1
draw.vf.cell2("\\#J395d", "\\#N4883", i, nr); i <- i + 1
draw.vf.cell2("\\#J395f", "\\#N4994", i, nr); i <- i + 1
draw.vf.cell2("\\#J3960", "\\#N1459", i, nr); i <- i + 1
draw.vf.cell2("\\#J3961", "\\#N5188", i, nr); i <- i + 1
draw.vf.cell2("\\#J3962", "\\#N5248", i, nr); i <- i + 1
draw.vf.cell2("\\#J3966", "\\#N0882", i, nr); i <- i + 1
draw.vf.cell2("\\#J3967", "\\#N0383", i, nr); i <- i + 1
draw.vf.cell2("\\#J3971", "\\#N1037", i, nr); i <- i + 1
draw.vf.cell2("\\#J3975", "\\#N5403", i, nr); i <- i + 1
draw.vf.cell2("\\#J397c", "\\#N5236", i, nr); i <- i + 1
draw.vf.cell2("\\#J397e", "\\#N4660", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a21", "\\#N2430", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a23", "\\#N0352", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a2c", "\\#N2261", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a38", "\\#N1455", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J3a39", "\\#N3662", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a42", "\\#N1515", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a46", "\\#N0035", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a47", "\\#N2146", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a59", "\\#N3522", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a5f", "\\#N1055", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a6e", "\\#N0407", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a72", "\\#N2119", i, nr); i <- i + 1
draw.vf.cell2("\\#J3a79", "\\#N2256", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b2e", "\\#N3113", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b30", "\\#N0008", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b33", "\\#N1407", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b36", "\\#N2056", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b3b", "\\#N3415", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b40", "\\#N4789", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b45", "\\#N0362", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b4d", "\\#N1025", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b4e", "\\#N1160", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b4f", "\\#N1208", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b52", "\\#N1264", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b54", "\\#N0284", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b57", "\\#N3001", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b58", "\\#N1904", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b59", "\\#N2039", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b5e", "\\#N2211", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b5f", "\\#N2429", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b60", "\\#N2439", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b61", "\\#N2478", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b64", "\\#N3265", i, nr); i <- i + 1

make.table2(nr, nc)
i <- 0
draw.title2("Kanji", nc)
draw.vf.cell2("\\#J3b65", "\\#N3492", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b66", "\\#N3510", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b6a", "\\#N3845", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b73", "\\#N2435", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b75", "\\#N5428", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b76", "\\#N0272", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b7a", "\\#N1281", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b7d", "\\#N1903", i, nr); i <- i + 1
draw.vf.cell2("\\#J3b7e", "\\#N2126", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c21", "\\#N0638", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c27", "\\#N3209", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c28", "\\#N3228", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c2a", "\\#N3697", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c2b", "\\#N3841", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c2d", "\\#N3860", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c2f", "\\#N5375", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c30", "\\#N1556", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c34", "\\#N4619", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c37", "\\#N0261", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c3c", "\\#N1300", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c3e", "\\#N2631", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c41", "\\#N4518", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c42", "\\#N1297", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c4d", "\\#N4603", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J3c50", "\\#N2074", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c54", "\\#N3685", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c56", "\\#N4608", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c5c", "\\#N1377", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c61", "\\#N4809", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c63", "\\#N3926", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c67", "\\#N0285", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c68", "\\#N3699", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c6a", "\\#N1827", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c6f", "\\#N3295", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c72", "\\#N2573", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c73", "\\#N5186", i, nr); i <- i + 1
draw.vf.cell2("\\#J3c7e", "\\#N0622", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d29", "\\#N3273", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d2a", "\\#N3521", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d2e", "\\#N3863", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d39", "\\#N4798", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d3d", "\\#N0768", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d3e", "\\#N1613", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d44", "\\#N3597", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d45", "\\#N0224", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d50", "\\#N0097", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d51", "\\#N1621", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d55", "\\#N2122", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d60", "\\#N0791", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d63", "\\#N3509", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d68", "\\#N1162", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d6b", "\\#N2138", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d71", "\\#N3719", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d77", "\\#N1185", i, nr); i <- i + 1
draw.vf.cell2("\\#J3d7c", "\\#N4993", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e26", "\\#N0321", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e2e", "\\#N1355", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e2f", "\\#N0166", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e3d", "\\#N2137", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e3e", "\\#N2212", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e46", "\\#N2772", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e4b", "\\#N3192", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e4e", "\\#N3280", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e57", "\\#N1638", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e5a", "\\#N4341", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e5d", "\\#N4472", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e65", "\\#N0798", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e68", "\\#N0223", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e6c", "\\#N1113", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e6f", "\\#N1364", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e75", "\\#N2839", i, nr); i <- i + 1
draw.vf.cell2("\\#J3e78", "\\#N4002", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f22", "\\#N2303", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f27", "\\#N3889", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f29", "\\#N5154", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f2d", "\\#N0403", i, nr); i <- i + 1

make.table2(nr, nc)
i <- 0
draw.title2("Kanji", nc)
draw.vf.cell2("\\#J3f34", "\\#N1645", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J3f36", "\\#N1920", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f37", "\\#N2080", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f39", "\\#N2301", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f3f", "\\#N0783", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f43", "\\#N3837", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f48", "\\#N4601", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f49", "\\#N4646", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f4a", "\\#N4709", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f4c", "\\#N5055", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f4d", "\\#N0339", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f5e", "\\#N1034", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f62", "\\#N0211", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f65", "\\#N2482", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f69", "\\#N3676", i, nr); i <- i + 1
draw.vf.cell2("\\#J3f74", "\\#N2057", i, nr); i <- i + 1
draw.vf.cell2("\\#J402d", "\\#N1666", i, nr); i <- i + 1
draw.vf.cell2("\\#J402e", "\\#N1799", i, nr); i <- i + 1
draw.vf.cell2("\\#J4030", "\\#N2436", i, nr); i <- i + 1
draw.vf.cell2("\\#J4031", "\\#N2121", i, nr); i <- i + 1
draw.vf.cell2("\\#J4032", "\\#N2143", i, nr); i <- i + 1
draw.vf.cell2("\\#J4035", "\\#N0027", i, nr); i <- i + 1
draw.vf.cell2("\\#J4038", "\\#N2991", i, nr); i <- i + 1
draw.vf.cell2("\\#J403e", "\\#N4273", i, nr); i <- i + 1
draw.vf.cell2("\\#J4044", "\\#N5076", i, nr); i <- i + 1
draw.vf.cell2("\\#J4045", "\\#N5077", i, nr); i <- i + 1
draw.vf.cell2("\\#J404e", "\\#N2108", i, nr); i <- i + 1
draw.vf.cell2("\\#J404f", "\\#N2194", i, nr); i <- i + 1
draw.vf.cell2("\\#J4050", "\\#N3176", i, nr); i <- i + 1
draw.vf.cell2("\\#J4051", "\\#N3306", i, nr); i <- i + 1
draw.vf.cell2("\\#J4056", "\\#N4534", i, nr); i <- i + 1
draw.vf.cell2("\\#J405a", "\\#N0667", i, nr); i <- i + 1
draw.vf.cell2("\\#J405c", "\\#N1951", i, nr); i <- i + 1
draw.vf.cell2("\\#J405e", "\\#N1855", i, nr); i <- i + 1
draw.vf.cell2("\\#J4063", "\\#N5044", i, nr); i <- i + 1
draw.vf.cell2("\\#J4064", "\\#N3539", i, nr); i <- i + 1
draw.vf.cell2("\\#J4065", "\\#N3855", i, nr); i <- i + 1
draw.vf.cell2("\\#J4068", "\\#N0571", i, nr); i <- i + 1
draw.vf.cell2("\\#J4069", "\\#N0156", i, nr); i <- i + 1
draw.vf.cell2("\\#J406e", "\\#N1447", i, nr); i <- i + 1
draw.vf.cell2("\\#J4070", "\\#N1823", i, nr); i <- i + 1
draw.vf.cell2("\\#J407e", "\\#N3580", i, nr); i <- i + 1
draw.vf.cell2("\\#J4125", "\\#N3873", i, nr); i <- i + 1
draw.vf.cell2("\\#J4130", "\\#N0595", i, nr); i <- i + 1
draw.vf.cell2("\\#J4133", "\\#N2770", i, nr); i <- i + 1
draw.vf.cell2("\\#J4134", "\\#N0384", i, nr); i <- i + 1
draw.vf.cell2("\\#J4147", "\\#N3511", i, nr); i <- i + 1
draw.vf.cell2("\\#J4148", "\\#N3520", i, nr); i <- i + 1
draw.vf.cell2("\\#J4150", "\\#N0859", i, nr); i <- i + 1
draw.vf.cell2("\\#J4158", "\\#N1402", i, nr); i <- i + 1
draw.vf.cell2("\\#J415b", "\\#N1728", i, nr); i <- i + 1
draw.vf.cell2("\\#J4161", "\\#N2100", i, nr); i <- i + 1
draw.vf.cell2("\\#J416a", "\\#N2241", i, nr); i <- i + 1
draw.vf.cell2("\\#J416d", "\\#N3567", i, nr); i <- i + 1
draw.vf.cell2("\\#J4170", "\\#N3939", i, nr); i <- i + 1
draw.vf.cell2("\\#J4175", "\\#N4234", i, nr); i <- i + 1
draw.vf.cell2("\\#J4176", "\\#N4539", i, nr); i <- i + 1
draw.vf.cell2("\\#J417c", "\\#N0540", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J417d", "\\#N1137", i, nr); i <- i + 1
draw.vf.cell2("\\#J4224", "\\#N4701", i, nr); i <- i + 1
draw.vf.cell2("\\#J4226", "\\#N0509", i, nr); i <- i + 1
draw.vf.cell2("\\#J422b", "\\#N0196", i, nr); i <- i + 1
draw.vf.cell2("\\#J422c", "\\#N2632", i, nr); i <- i + 1
draw.vf.cell2("\\#J422d", "\\#N4546", i, nr); i <- i + 1
draw.vf.cell2("\\#J422e", "\\#N4700", i, nr); i <- i + 1
draw.vf.cell2("\\#J4233", "\\#N3544", i, nr); i <- i + 1
draw.vf.cell2("\\#J4236", "\\#N0590", i, nr); i <- i + 1
draw.vf.cell2("\\#J4238", "\\#N1267", i, nr); i <- i + 1
draw.vf.cell2("\\#J423e", "\\#N0361", i, nr); i <- i + 1
draw.vf.cell2("\\#J423f", "\\#N1169", i, nr); i <- i + 1
draw.vf.cell2("\\#J4240", "\\#N1172", i, nr); i <- i + 1
draw.vf.cell2("\\#J424a", "\\#N2313", i, nr); i <- i + 1
draw.vf.cell2("\\#J424e", "\\#N0405", i, nr); i <- i + 1
draw.vf.cell2("\\#J4250", "\\#N2067", i, nr); i <- i + 1
draw.vf.cell2("\\#J4256", "\\#N1743", i, nr); i <- i + 1
draw.vf.cell2("\\#J4265", "\\#N0364", i, nr); i <- i + 1

make.table2(nr, nc)
i <- 0
draw.title2("Kanji", nc)
draw.vf.cell2("\\#J4267", "\\#N1171", i, nr); i <- i + 1
draw.vf.cell2("\\#J4268", "\\#N3385", i, nr); i <- i + 1
draw.vf.cell2("\\#J426a", "\\#N2164", i, nr); i <- i + 1
draw.vf.cell2("\\#J426c", "\\#N2655", i, nr); i <- i + 1
draw.vf.cell2("\\#J4274", "\\#N2503", i, nr); i <- i + 1
draw.vf.cell2("\\#J4323", "\\#N4721", i, nr); i <- i + 1
draw.vf.cell2("\\#J432b", "\\#N4458", i, nr); i <- i + 1
draw.vf.cell2("\\#J432f", "\\#N4384", i, nr); i <- i + 1
draw.vf.cell2("\\#J4331", "\\#N0139", i, nr); i <- i + 1
draw.vf.cell2("\\#J433a", "\\#N1418", i, nr); i <- i + 1
draw.vf.cell2("\\#J433b", "\\#N3172", i, nr); i <- i + 1
draw.vf.cell2("\\#J4346", "\\#N1575", i, nr); i <- i + 1
draw.vf.cell2("\\#J434b", "\\#N2996", i, nr); i <- i + 1
draw.vf.cell2("\\#J434d", "\\#N0488", i, nr); i <- i + 1
draw.vf.cell2("\\#J434e", "\\#N3169", i, nr); i <- i + 1
draw.vf.cell2("\\#J434f", "\\#N1056", i, nr); i <- i + 1
draw.vf.cell2("\\#J4356", "\\#N3644", i, nr); i <- i + 1
draw.vf.cell2("\\#J4359", "\\#N4722", i, nr); i <- i + 1
draw.vf.cell2("\\#J435d", "\\#N3366", i, nr); i <- i + 1
draw.vf.cell2("\\#J4362", "\\#N3325", i, nr); i <- i + 1
draw.vf.cell2("\\#J4363", "\\#N3940", i, nr); i <- i + 1
draw.vf.cell2("\\#J4365", "\\#N3665", i, nr); i <- i + 1
draw.vf.cell2("\\#J4366", "\\#N0081", i, nr); i <- i + 1
draw.vf.cell2("\\#J4368", "\\#N1291", i, nr); i <- i + 1
draw.vf.cell2("\\#J436b", "\\#N0053", i, nr); i <- i + 1
draw.vf.cell2("\\#J436c", "\\#N2236", i, nr); i <- i + 1
draw.vf.cell2("\\#J436e", "\\#N4115", i, nr); i <- i + 1
draw.vf.cell2("\\#J442b", "\\#N3788", i, nr); i <- i + 1
draw.vf.cell2("\\#J442c", "\\#N2702", i, nr); i <- i + 1
draw.vf.cell2("\\#J4436", "\\#N4543", i, nr); i <- i + 1
draw.vf.cell2("\\#J4439", "\\#N4938", i, nr); i <- i + 1
draw.vf.cell2("\\#J443b", "\\#N5340", i, nr); i <- i + 1
draw.vf.cell2("\\#J443e", "\\#N0775", i, nr); i <- i + 1
draw.vf.cell2("\\#J444c", "\\#N4703", i, nr); i <- i + 1
draw.vf.cell2("\\#J4463", "\\#N0406", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J446a", "\\#N1296", i, nr); i <- i + 1
draw.vf.cell2("\\#J446c", "\\#N1508", i, nr); i <- i + 1
draw.vf.cell2("\\#J446d", "\\#N1514", i, nr); i <- i + 1
draw.vf.cell2("\\#J4472", "\\#N1914", i, nr); i <- i + 1
draw.vf.cell2("\\#J4478", "\\#N3285", i, nr); i <- i + 1
draw.vf.cell2("\\#J4479", "\\#N3581", i, nr); i <- i + 1
draw.vf.cell2("\\#J4526", "\\#N1987", i, nr); i <- i + 1
draw.vf.cell2("\\#J452a", "\\#N3097", i, nr); i <- i + 1
draw.vf.cell2("\\#J452f", "\\#N0931", i, nr); i <- i + 1
draw.vf.cell2("\\#J4534", "\\#N4844", i, nr); i <- i + 1
draw.vf.cell2("\\#J4535", "\\#N0588", i, nr); i <- i + 1
draw.vf.cell2("\\#J4537", "\\#N0016", i, nr); i <- i + 1
draw.vf.cell2("\\#J453e", "\\#N4615", i, nr); i <- i + 1
draw.vf.cell2("\\#J4540", "\\#N0804", i, nr); i <- i + 1
draw.vf.cell2("\\#J4544", "\\#N2994", i, nr); i <- i + 1
draw.vf.cell2("\\#J4545", "\\#N5050", i, nr); i <- i + 1
draw.vf.cell2("\\#J454c", "\\#N1614", i, nr); i <- i + 1
draw.vf.cell2("\\#J4559", "\\#N1511", i, nr); i <- i + 1
draw.vf.cell2("\\#J455a", "\\#N1050", i, nr); i <- i + 1
draw.vf.cell2("\\#J455f", "\\#N1161", i, nr); i <- i + 1
draw.vf.cell2("\\#J4561", "\\#N0665", i, nr); i <- i + 1
draw.vf.cell2("\\#J4563", "\\#N1109", i, nr); i <- i + 1
draw.vf.cell2("\\#J4567", "\\#N0230", i, nr); i <- i + 1
draw.vf.cell2("\\#J456c", "\\#N0213", i, nr); i <- i + 1
draw.vf.cell2("\\#J4574", "\\#N2745", i, nr); i <- i + 1
draw.vf.cell2("\\#J4576", "\\#N1359", i, nr); i <- i + 1
draw.vf.cell2("\\#J4579", "\\#N3396", i, nr); i <- i + 1
draw.vf.cell2("\\#J4626", "\\#N4465", i, nr); i <- i + 1
draw.vf.cell2("\\#J4630", "\\#N0730", i, nr); i <- i + 1
draw.vf.cell2("\\#J4631", "\\#N0619", i, nr); i <- i + 1
draw.vf.cell2("\\#J4633", "\\#N1354", i, nr); i <- i + 1
draw.vf.cell2("\\#J463b", "\\#N4724", i, nr); i <- i + 1
draw.vf.cell2("\\#J463c", "\\#N4853", i, nr); i <- i + 1
draw.vf.cell2("\\#J4643", "\\#N2860", i, nr); i <- i + 1
draw.vf.cell2("\\#J4649", "\\#N4375", i, nr); i <- i + 1
draw.vf.cell2("\\#J465e", "\\#N2160", i, nr); i <- i + 1
draw.vf.cell2("\\#J4662", "\\#N0082", i, nr); i <- i + 1
draw.vf.cell2("\\#J466e", "\\#N0778", i, nr); i <- i + 1
draw.vf.cell2("\\#J4671", "\\#N5038", i, nr); i <- i + 1
draw.vf.cell2("\\#J4673", "\\#N0273", i, nr); i <- i + 1
draw.vf.cell2("\\#J4679", "\\#N3724", i, nr); i <- i + 1

make.table2(nr, nc)
i <- 0
draw.title2("Kanji", nc)
draw.vf.cell2("\\#J467c", "\\#N2097", i, nr); i <- i + 1
draw.vf.cell2("\\#J467e", "\\#N0574", i, nr); i <- i + 1
draw.vf.cell2("\\#J4721", "\\#N1189", i, nr); i <- i + 1
draw.vf.cell2("\\#J472e", "\\#N2797", i, nr); i <- i + 1
draw.vf.cell2("\\#J472f", "\\#N0188", i, nr); i <- i + 1
draw.vf.cell2("\\#J4733", "\\#N2808", i, nr); i <- i + 1
draw.vf.cell2("\\#J4734", "\\#N3472", i, nr); i <- i + 1
draw.vf.cell2("\\#J4748", "\\#N2529", i, nr); i <- i + 1
draw.vf.cell2("\\#J474f", "\\#N5191", i, nr); i <- i + 1
draw.vf.cell2("\\#J4769", "\\#N3275", i, nr); i <- i + 1
draw.vf.cell2("\\#J4772", "\\#N3095", i, nr); i <- i + 1
draw.vf.cell2("\\#J477e", "\\#N5385", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J4821", "\\#N0049", i, nr); i <- i + 1
draw.vf.cell2("\\#J482c", "\\#N0577", i, nr); i <- i + 1
draw.vf.cell2("\\#J482f", "\\#N3092", i, nr); i <- i + 1
draw.vf.cell2("\\#J483e", "\\#N0132", i, nr); i <- i + 1
draw.vf.cell2("\\#J483f", "\\#N0817", i, nr); i <- i + 1
draw.vf.cell2("\\#J4841", "\\#N1469", i, nr); i <- i + 1
draw.vf.cell2("\\#J484c", "\\#N3865", i, nr); i <- i + 1
draw.vf.cell2("\\#J4856", "\\#N4811", i, nr); i <- i + 1
draw.vf.cell2("\\#J4860", "\\#N1604", i, nr); i <- i + 1
draw.vf.cell2("\\#J4866", "\\#N2470", i, nr); i <- i + 1
draw.vf.cell2("\\#J4869", "\\#N3109", i, nr); i <- i + 1
draw.vf.cell2("\\#J4873", "\\#N5080", i, nr); i <- i + 1
draw.vf.cell2("\\#J4874", "\\#N5152", i, nr); i <- i + 1
draw.vf.cell2("\\#J4878", "\\#N1383", i, nr); i <- i + 1
draw.vf.cell2("\\#J4879", "\\#N1631", i, nr); i <- i + 1
draw.vf.cell2("\\#J487e", "\\#N3658", i, nr); i <- i + 1
draw.vf.cell2("\\#J4921", "\\#N5421", i, nr); i <- i + 1
draw.vf.cell2("\\#J492e", "\\#N3397", i, nr); i <- i + 1
draw.vf.cell2("\\#J4934", "\\#N0033", i, nr); i <- i + 1
draw.vf.cell2("\\#J4938", "\\#N2359", i, nr); i <- i + 1
draw.vf.cell2("\\#J4939", "\\#N0131", i, nr); i <- i + 1
draw.vf.cell2("\\#J493d", "\\#N0108", i, nr); i <- i + 1
draw.vf.cell2("\\#J4942", "\\#N3042", i, nr); i <- i + 1
draw.vf.cell2("\\#J4943", "\\#N3271", i, nr); i <- i + 1
draw.vf.cell2("\\#J494a", "\\#N0923", i, nr); i <- i + 1
draw.vf.cell2("\\#J4954", "\\#N0017", i, nr); i <- i + 1
draw.vf.cell2("\\#J495b", "\\#N1468", i, nr); i <- i + 1
draw.vf.cell2("\\#J4963", "\\#N2832", i, nr); i <- i + 1
draw.vf.cell2("\\#J4969", "\\#N4488", i, nr); i <- i + 1
draw.vf.cell2("\\#J4977", "\\#N5148", i, nr); i <- i + 1
draw.vf.cell2("\\#J497d", "\\#N1484", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a23", "\\#N4255", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a26", "\\#N0173", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a2a", "\\#N2857", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a2c", "\\#N0578", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a38", "\\#N2064", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a39", "\\#N4959", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a3f", "\\#N0026", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a42", "\\#N0589", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a44", "\\#N4945", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a46", "\\#N3461", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a50", "\\#N0511", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a51", "\\#N0306", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a52", "\\#N2842", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a55", "\\#N4661", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a6c", "\\#N2466", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a7c", "\\#N2084", i, nr); i <- i + 1
draw.vf.cell2("\\#J4a7d", "\\#N2082", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b21", "\\#N2535", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b26", "\\#N3749", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b4c", "\\#N0751", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b4f", "\\#N5404", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b5c", "\\#N0096", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b63", "\\#N5390", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b68", "\\#N2467", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b74", "\\#N0855", i, nr); i <- i + 1
draw.vf.cell2("\\#J4b7c", "\\#N0007", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J4c23", "\\#N0913", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c24", "\\#N0179", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c29", "\\#N1316", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c35", "\\#N2773", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c37", "\\#N3164", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c3e", "\\#N1170", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c40", "\\#N2110", i, nr); i <- i + 1

make.table2(nr, nc)
i <- 0
draw.title2("Kanji", nc)
draw.vf.cell2("\\#J4c4c", "\\#N5087", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c53", "\\#N2473", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c5a", "\\#N2170", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c5c", "\\#N3127", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c64", "\\#N4944", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c67", "\\#N4940", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c6b", "\\#N0298", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c70", "\\#N3168", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c72", "\\#N1598", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c74", "\\#N4074", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c78", "\\#N2233", i, nr); i <- i + 1
draw.vf.cell2("\\#J4c7d", "\\#N2534", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d2d", "\\#N3727", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d30", "\\#N2565", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d3a", "\\#N5030", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d3c", "\\#N1167", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d3e", "\\#N0408", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d4f", "\\#N2659", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d51", "\\#N2993", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d53", "\\#N3656", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d55", "\\#N4001", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d57", "\\#N4274", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d5b", "\\#N5012", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d63", "\\#N3680", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d68", "\\#N0202", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d6b", "\\#N5049", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d70", "\\#N3856", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d71", "\\#N0199", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d72", "\\#N1431", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d78", "\\#N3264", i, nr); i <- i + 1
draw.vf.cell2("\\#J4d7d", "\\#N2942", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e24", "\\#N4813", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e25", "\\#N5040", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e26", "\\#N5005", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e28", "\\#N0319", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e29", "\\#N3343", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e2e", "\\#N2576", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e32", "\\#N3191", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e33", "\\#N3471", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e35", "\\#N5440", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e3e", "\\#N0034", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e41", "\\#N3468", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e49", "\\#N3885", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e4c", "\\#N2141", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e4f", "\\#N0715", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e53", "\\#N2210", i, nr); i <- i + 1
```

```
draw.vf.cell2("\\#J4e55", "\\#N2807", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e58", "\\#N4630", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e60", "\\#N5138", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e63", "\\#N0428", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e64", "\\#N0642", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e6d", "\\#N5048", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e6e", "\\#N5056", i, nr); i <- i + 1
draw.vf.cell2("\\#J4e73", "\\#N2438", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f22", "\\#N4702", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f27", "\\#N2750", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f29", "\\#N4561", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f37", "\\#N3683", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f3b", "\\#N0283", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f40", "\\#N4391", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f42", "\\#N3268", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f43", "\\#N4358", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f44", "\\#N0054", i, nr); i <- i + 1
draw.vf.cell2("\\#J4f47", "\\#N1710", i, nr); i <- i + 1

draw.vf.cell2("\\#J534c", "\\#N0973", i, nr); i <- i + 1
draw.vf.cell2("\\#J5879", "\\#N1794", i, nr); i <- i + 1
draw.vf.cell2("\\#J5960", "\\#N1942", i, nr); i <- i + 1
draw.vf.cell2("\\#J626f", "\\#N3200", i, nr); i <- i + 1
draw.vf.cell2("\\#J6446", "\\#N3458", i, nr); i <- i + 1
draw.vf.cell2("\\#J6647", "\\#N5083", i, nr); i <- i + 1
draw.vf.cell2("\\#J6d55", "\\#N4633", i, nr); i <- i + 1

par(oldpar)
```

`jitter` *Add 'Jitter' (Noise) to Numbers* `jitter`

Description

Add a small amount of noise to a numeric vector.

Usage

```
jitter(x, factor=1, amount = NULL)
```

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

Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P.A. (1983) *Graphical Methods for Data Analysis.* Wadsworth; figures 2.8, 4.22, 5.4.

See Also

rug which you may want to combine with `jitter`.

Examples

```
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* — kappa

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

```
kappa(z, ...)
kappa.lm     (z, ...)
kappa.default(z, exact = FALSE)
kappa.qr     (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`	Should the result be exact?

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.

Value

The condition number, *kappa*, or an approximation if `exact=FALSE`.

Author(s)

B.D. Ripley

See Also

`svd` for the singular value decomposition and `qr` for the QR one.

Examples

```
kappa(x1 <- cbind(1,1:10))# 15.71
kappa(x1, exact=T)        # 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 Product of Arrays* `kronecker`

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 = "*", ...)
X %x% Y
```

Arguments

`X`	vector or array.
`Y`	vector or array.
`FUN`	a function, possibly specified as character (string).
`...`	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. `A` consists of 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

References

Searle, Shayle R. (1982) *Matrix Algebra Useful for Statistics*; John Wiley and Sons.

See Also

`outer` on which `kronecker` is built and `matmult` for usual matrix multiplication.

Examples

```
# simple scalar multiplication
( M <- matrix(1:6, ncol=2) )
all(kronecker(4, M) == 4 * M)

( A <- matrix(0:3, ncol=2) )
A %x% cbind(2:3)
# Block diagonal array:
kronecker(diag(3), M)
all(kronecker(diag(3), M) == diag(3) %x% M)
```

`labels` *Find Labels from Object* `labels`

Description

Find a suitable set of labels from an object for use in printing or plotting, for example.

Usage

```
labels(object, ...)
labels.default(object, ...)
labels.terms(object, ...)
labels.lm(object, ...)
```

Arguments

`object`	Any R object: the function is generic.

Value

A character vector or list of such vectors. For a vector the results is the names or `seq(along=x)`, for a data frame or array it is the dimnames (with `NULL` expanded to `seq(len=d[i])`), for a `terms` object it is the term labels and for an `lm` object it is the term labels for estimable terms.

Author(s)

B.D. Ripley

`lapply` *Apply a Function over a List or Vector* `lapply`

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` also accepting vectors as `X`, and returning a vector or array with `dimnames` if appropriate.

Usage

```
lapply(X, FUN, ...)
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

Arguments

`X`	list or vector 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`.
`simplify`	logical; should the result be simplified to a vector if possible?
`USE.NAMES`	logical; if `TRUE` and if `X` is character, use `X` as `names` for the result unless it had names already.

See Also

`apply`, `tapply`.

Examples

```
x <- list(a = 1:10, beta = exp(-3:3), logic = c(T,F,F,T))
# 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)
str(i39 <- sapply(3:9, seq))# list of vectors
sapply(i39, fivenum)
```

`Last.value` *Value of Last Evaluated Expression* `Last.value`

Description

Internal evaluation of an R expression is always assigned to `.Last.value` (in `package:base`) before further processing (e.g. printing).

Usage

```
.Last.value
```

Note

Do not assign to `.Last.value`, because this will always shadow the one in `package:base`.

See Also

`eval`

Examples

```
gamma(1:15)          # think of some intensive calculation...
fac14 <- .Last.value # keep them

library("eda") # returns invisibly
.Last.value    # shows what library(.) above returned
```

`layout` *Specifying Complex Plot Arrangements* `layout`

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

```
layout(mat,
       widths = rep(1, dim(mat)[2]),
       heights= rep(1, dim(mat)[1]),
       respect= FALSE)

layout.show(n = 1)
lcm(w)
```

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 $\{1, \ldots, N-1\}$ 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`.

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.

`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.

Author(s)

Paul R. Murrell

References

Chapter 5 of Paul Murrell's Ph.D. thesis.

See Also

`par(mfrow=..)`, `par(mfcol=..)` and `par(mfg=..)`

Examples

```
def.par <- par()# 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=T))
## 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=T), respect=T)
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 <- rnorm(50)
y <- rnorm(50)
xhist <- hist(x, breaks=seq(-3,3,0.5), plot=F)
yhist <- hist(y, breaks=seq(-3,3,0.5), plot=F)
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,T), c(3,1), c(1,3), T)
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$counts, axes=F, ylim=c(0, top), space=0)
par(mar=c(3,0,1,1))
barplot(yhist$counts, axes=F, xlim=c(0, top), space=0, horiz=T)

par(def.par)#- reset to default
```

`legend` *Add Legends to Plots* `legend`

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

```
legend(x, y, legend, fill, col = "black", lty, lwd = 1, pch,
       bty = "o", bg = par("bg"), cex = 1, xjust = 0, yjust = 1,
       x.intersp = 1, y.intersp = 1, adj = 0,
       text.width = NULL, merge = do.lines && has.pch)
```

Arguments

`x,y`	the x and y location of the legend. `x` can be a list with `x` and `y` components.
`legend`	a vector of text values or an `expression` of length ≥ 1 to appear in the legend.
`fill`	if specified, this argument will cause boxes filled with the specified colors to appear beside the legend text.
`col`	the color of points or lines appearing in the legend.
`lty,lwd`	the line types and widths for lines appearing in the legend. One of these two *must* be specified for line drawing.
`pch`	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.
`bty`	the type of box to be drawn around the legend.
`bg`	the background color for the legend box.
`cex`	character expansion factor **relative** to current `par("cex")`.
`xjust`	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.
`yjust`	the same as `xjust` for the legend y location.
`x.intersp`	character interspacing factor for horizontal (x) spacing.
`y.intersp`	the same for vertical (y) line distances.

`adj`	numeric of length 1 or 2; the string adjustment for legend text. Useful for y-adjustment when `labels` are plotmath expressions.
`text.width`	the width of the legend text in x (`"user"`) coordinates. Defaults to the proper value computed by `strwidth(legend)`.
`merge`	logical; if `TRUE`, "merge" points and lines but not filled boxes. Defaults to `TRUE` if there are points and lines.

Details

"Attribute" arguments such as `col`, `pch`, `lty`, etc, are recycled if necessary. `merge` is not.

See Also

`plot`, `barplot` which uses `legend()`, and `text` for more examples of math expressions.

Examples

```
# 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(.9,  y.leg[i], paste("cex=",formatC(cexv[i])),srt=90,adj=1, cex=.7)
  legend(1, 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),
       lty = c(2, -1, 1), pch = c(-1, 3, 4), merge = TRUE, bg='gray90')

##--- 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(2,3,3,2, 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,.3))
  legend(list(x=2,y=8), legend = leg.txt, col=2:3, pch=1:2,
         lty=1, merge=T)#, trace=T)
}
par(mfrow=c(1,1))

##-- Math expressions:  ------------------------------
```

```
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
legend(-3, .9, ex.cs1, lty=1:2, col=2:3, adj = c(0, .6))# adj y !

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),
                   hat(x) == median(x[i], i==1,n))
legend(4.1, 30, ex12, col = 2:3, lty=2:3, lwd=2)
```

`length` *Length of a Vector or List* `length`

Description

Get or set the length of vectors (including lists).

Usage

```
length(x)
length(x) <- n
```

Arguments

`x`	a vector or list.
`n`	an integer.

Details

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 `NA`s.

Value

If `x` is (or can be coerced to) a vector or list, `length` returns the length of `x`. Otherwise, `length` returns `NA`.

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
```

`levels` *Levels Attributes* `levels`

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 `factor`s:

Usage

```
levels(x)
levels(x) <- value
```

See Also

`levels<-.factor`, `nlevels`.

`levels.factor` *Factor Levels Assignment* `levels.factor`

Description

`levels<-` provides a way to alter the levels attribute of factor. `value` can be a vector of character strings with length equal to the number of levels of `x`, or a named list specifying how to rename the levels.

Usage

```
levels(x) <- value
```

See Also

`factor`, `levels`, `levels<-`, `nlevels`.

Examples

```
# assign individual levels
x <- gl(2, 4, 8)
levels(x)[1] <- "low"
levels(x)[2] <- "high"

# or as a group
y <- gl(2, 4, 8)
levels(y) <- c("low", "high")

# combine some levels
z <- gl(3, 2, 12)
levels(z) <- c("A", "B", "A")

# same, using a named list
z <- gl(3, 2, 12)
levels(z) <- list(A=c(1,3), B=2)
```

`library` *Loading and Listing of Packages* `library`

Description

`library` and `require` load add-on packages; `provide` allows code to register services that packages provide. `.First.lib` is called when a package is loaded; `.packages` returns information about package availability. `.path.package` returns information about where a package was loaded from.

Usage

```
library(package, help = NULL, lib.loc = .lib.loc,
        character.only = FALSE, logical.return = FALSE,
        warn.conflicts = TRUE)
require(package, quietly = FALSE, warn.conflicts = TRUE)
provide(package)

.First.lib(libname, pkgname)

.packages(all.available = FALSE, lib.loc = .lib.loc)
.path.package(package = .packages())
.lib.loc
.Library
.Provided
.Autoloaded
```

Arguments

`package, help`	`name` or character string giving the name of a package.
`lib.loc`	a character vector describing the location of R library trees to search through.
`character.only`	a logical indicating whether `package` or `help` can be assumed to be character strings.
`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`.
`quietly`	a logical. If `TRUE`, a warning will not be printed if the package cannot be found.
`libname`	a character string giving the library directory where the package was found.
`pkgname`	a character string giving the name of the package.
`all.available`	logical; if `TRUE` return `character` vector of all available packages in `lib.loc`.

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 optionally gives a warning, rather than giving an error, 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. `require` also checks the list `.Provided`.

`provide` allows code to register services that it provides. The argument is stored in the list `.Provided`. `provide` returns `FALSE` if the package was already present in `.Provided` or among the packages in `search()`. The main use for `provide` is when multiple packages share code. This is most likely when the code implements features present in S(-PLUS) but not in R.

If `library` is called with no `package` or `help` argument, it gives a list of all available packages in `lib.loc` and invisibly returns their names (same as `.packages(all = TRUE)`).

`library(help = somename)` prints information on the package `somename`, typically by listing the most important user level objects it contains.

`.First.lib()` is called when a package is loaded by `library()`. It is called with two arguments, the name of the library tree where the package was found (i.e., the corresponding element of `lib.loc`), and the name of the package (in that order). 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 `search()` has been updated, so `pos.to.env(match("package:name"), search())` will return the environment in which the package is stored.

`.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`.

`.path.package` returns the paths from which the named packages were loaded, or if none were named, for all currently loaded packages. It will warn if the packages named are not loaded.

`.Autoloaded` contains the "base names" of the packages for which autoloading has been promised.

`.Library` is a character string giving the location of the default library, the "library" subdirectory of `R_HOME`. `.lib.loc` is a character vector with the locations of all library trees that R should use. It is initialized at startup from the environment variable `R_LIBS` (`RLIBS` as used by older versions of R is no longer accepted) (which should be a colon-separated list of directories at which R library trees are rooted) followed by `.Library`.

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.

Author(s)

R core; Guido Masarotto for the `all.available=TRUE` part of `.packages`.

See Also

`attach`, `detach`, `search`, `objects`, `autoload`, `library.dynam`, `data`, `install.packages`. `INSTALL`, `REMOVE`

Examples

```
.packages()                 # maybe just "base"
.packages(all = TRUE)       # return all available as char.vector
library()                   # list all available packages
library(lib = .Library)     # list all packages in the default library
library(help = eda)         # documentation on package "eda"
library(eda)                # load package "eda"
require(eda)                # the same
.packages()                 # "eda", too
require(nonexistent)        # FALSE
## Suppose a package needs to call a shared library named "foo.EXT",
## where "EXT" is the system-specific extension.  Then you should use
.First.lib <- function(lib, pkg) {
  library.dynam("foo", pkg, lib)
}
```

`library.dynam` *Loading Shared Libraries* `library.dynam`

Description

Load the specified file of compiled code if it has not been loaded already.

Usage

```
library.dynam(chname, package = .packages(), lib.loc = .lib.loc)
```

Arguments

`chname`	a character string naming a shared library to load.
`package`	a character vector with the names of packages to search through.
`lib.loc`	a character vector describing the location of R library trees to search through.

Details

This is designed to be used inside a package rather than at the command line, and should really only be used inside `.First.lib()`. The system-specific extension for shared libraries (e.g., '.so' or '.sl' on Unix systems) should not be added.

Value

The `.Dyn.libs` vector with the names of packages which have used `library.dynam(..)` in the current R session.

It is returned as `invisible`, unless the `chname` argument is missing.

Users should never set `.Dyn.libs` directly.

See Also

`.First.lib`, `library`, `dyn.load`, `.packages`, `.lib.loc`

Examples

```
library.dynam()# which packages have been ‘‘dynamically loaded’’
```

`license` *The R License Terms* `license`

Description

The license terms under which R. is distributed.

Details

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in

effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution

of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do

not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and ”any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY

MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

References

A copy of license is in '$R_HOME/COPYING' and you may also want to look at the R FAQ in http://www.ci.tuwien.ac.at/~hornik/R/R-FAQ.html.

`LifeCycleSavings` *Intercountry Life-Cycle Savings Data* `LifeCycleSavings`

Description

Data on the savings ratio 1960–1970.

Usage

```
data(LifeCycleSavings)
```

Format

A data frame with 50 observations on 5 variables.

[,1]	sr	numeric	aggregate personal savings
[,2]	pop15	numeric	% of population under 15
[,3]	pop75	numeric	% of population over 75
[,4]	dpi	numeric	real per-capita disposable income
[,5]	ddpi	numeric	% growth rate of dpi

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

Sterling, Arnie (1977) Unpublished BS Thesis. Massachusetts Institute of Technology.

Belsley, D. A., E. Kuh. E. and Welsch, R. E. (1980) *Regression Diagnostics.* New York: Wiley.

Examples

```
data(LifeCycleSavings)
pairs(LifeCycleSavings, panel = panel.smooth,
      main = "LifeCycleSavings data")
fm1 <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)
summary(fm1)
```

`lines` *Add Connected Line Segments to a Plot* `lines`

Description

A generic function taking coordinates given in various ways and joining the corresponding points with line segments.

Usage

```
lines(x, ...)
lines.default(x, y=NULL, type="l", col=par("col"), ...)
```

Arguments

`x, y`	coordinate vectors of points to join.
`type`	character indicating the type of plotting; actually any of the `type`s as in `plot(..)`.
`col`	color 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 Also

`points`, `plot`, and the underlying "primitive" `plot.xy`.

Examples

```
data(cars)
# draw a smooth line through a scatter plot
plot(cars, main="Stopping Distance versus Speed")
lines(lowess(cars))
```

`list` *Lists – Generic and Dotted Pairs* `list`

Description

Functions to construct, coerce and check for all kinds of R lists.

Usage

```
list(...)
pairlist(...)

as.list(x)
as.list.default(x)
as.pairlist(x)

is.list(x)
is.pairlist(x)

alist(...)
```

Details

Since version 0.63, 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.

An empty pairlist, `pairlist()` is the same as `NULL`. This is different from `list()`.

See Also

`vector(., mode="list")`, `c`, for concatenation; `formals`.

Examples

```
data(cars)
# create a plotting structure
pts <- list(x=cars[,1], y=cars[,2])
plot(pts)

# Argument lists
f <- function()x
# Note the specification of a "..." argument:
formals(f) <- al <- alist(x=, y=2, ...=)
f
str(al)

str(pl <- as.pairlist(ps.options()))

## These are all TRUE:
is.list(pl) && is.pairlist(pl)
!is.null(list())
```

```
is.null(pairlist())
!is.list(NULL)
is.pairlist(pairlist())
is.null(as.pairlist(list()))
is.null(as.pairlist(NULL))
```

`list.files` *List the Files in a Directory/Folder* `list.files`

Description

This function produces a list containing the names of files in the named directory. `dir` is an alias.

Usage

```
list.files(path, pattern=NULL, all.files=FALSE,
    full.names=FALSE)
dir(path, pattern=NULL, all.files=FALSE,
    full.names=FALSE)
```

Arguments

`path`	a character vector of full path names.
`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.

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.

Author(s)

Ross Ihaka

Examples

```
list.files(R.home())
```

`lm` *Fitting Linear Models* `lm`

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

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE contrasts = NULL, offset = NULL, ...)

lm.fit (x, y,     offset = NULL, method = "qr", tol = 1e-7, ...)
lm.wfit(x, y, w, offset = NULL, method = "qr", tol = 1e-7, ...)
lm.fit.null (x, y,     method = "qr", tol = 1e-7, ...)
lm.wfit.null(x, y, w, method = "qr", tol = 1e-7, ...)
```

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. By default the variables are taken from the environment which `lm` is called from.
`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 `NA`s. 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`.
`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, defaulting to `TRUE`. *`FALSE` is not yet implemented.*
`method`	currently, only `method="qr"` is supported.
`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.
`tol`	tolerance for the `qr` decomposition. Default is 1e-7.
`...`	currently disregarded.

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 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`.

Value

`lm` returns an object of `class` `"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`.

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.

See Also

`summary.lm` for summaries and `anova.lm` for the ANOVA table. `aov` for a different interface.

The generic functions `coefficients`, `effects`, `residuals`, `fitted.values`; `lm.influence` for regression diagnostics, and `glm` for **generalized** linear models.

Examples

```
## 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)
```

`lm.influence` *Regression Diagnostics* `lm.influence`

Usage

```
lm.influence(lm.obj)
```

Arguments

`lm.obj`	an object as returned by `lm`.

Details

The functions listed in **See Also** give a more direct way of computing a variety of regression diagnostics.

Value

A list containing the following components:

`hat`	a vector containing the diagonal of the "hat" matrix.
`coefficients`	the change in the estimated coefficients which results when the i-th case is dropped from the regression is contained in the i-th row of this 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.

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.

Note that cases with `weights == 0` are *dropped* (contrary to the situation in S).

References

Belsley, D. A., Kuh, E. and Welsch, R. E. (1980) *Regression Diagnostics.* New York: Wiley.

See Also

`summary.lm` for `summary` and related methods;
`influence.measures`,
`hat` for the hat matrix diagonals,
`dfbetas`, `dffits`, `covratio`, `cooks.distance`, `lm`.

Examples

```
## Analysis of the life-cycle savings data given in Belsley, Kuh
## and Welsch.
data(LifeCycleSavings)
summary(lm.SR <- lm(sr ~ pop15 + pop75 + dpi + ddpi,
                    data = LifeCycleSavings),
        corr = TRUE)
rstudent(lm.SR)
dfbetas(lm.SR)
dffits(lm.SR)
covratio(lm.SR)
```

`lm.summaries` *Accessing Linear Model Fits* `lm.summaries`

Description

All these functions are `methods` for class `lm` or `summary.lm` and `anova.lm` objects.

Usage

```
anova(object, ...)
anovalist.lm(object, ..., test = NULL)
summary(object, correlation = FALSE)
coefficients(x) ; coef(x)
df.residual(x)
family(x)
formula(x)
fitted.values(x)
residuals(x, type = c("working", "pearson", "deviance"), ...)
weights(x)
plot(x)

print(summary.lm.obj, digits = max(3, getOption("digits") - 3),
      symbolic.cor = p > 4,
      signif.stars= getOption("show.signif.stars"), ...)
```

Arguments

`object, x`	an object of class `lm`, usually, a result of a call to `lm`.

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`.

`anova.lm` produces an analysis of variance (`anova`) table.

The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` can be used to extract various useful features of the value returned by `lm`.

Value

The function `summary.lm` computes and returns a list of summary statistics of the fitted linear model given in `lm.obj`, 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 $p \times 4$ matrix with columns for the estimated coefficient, its standard error, t-statistic and corresponding (two-sided) p-value.
`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 $(p, n-p, p*)$.

`fstatistic` 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.

Warning

The comparison between two or more models by `anova` or `anovalist.lm` 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.

See Also

The model fitting function `lm`.

`anova` for the ANOVA table, `coefficients`, `deviance`, `effects`, `fitted.values`, `glm` for **generalized** linear models, `lm.influence` for regression diagnostics, `weighted.residuals`, `residuals`, `residuals.glm`, `summary`.

Examples

```
##-- Continuing the  lm(.) example:
coef(lm.D90)# the bare coefficients
sld90 <- summary(lm.D90 <- lm(weight ~ group -1))# omitting intercept
sld90
coef(sld90)# much more

## 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))
```

`load` — *Reload Saved Datasets* — `load`

Description

This function will reload the datasets written to a file with the function `save`.

Usage

```
load(filename, envir = sys.frame(sys.parent()))
```

Arguments

`filename`	a character string giving the name of the file to load.
`envir`	the environment where the data should be loaded

See Also

`save`.

Examples

```
# save all data
save(list = ls(), file= "all.Rdata")

# restore the saved values to the current environment
load("all.Rdata")

# restore the saved values to the global environment
load("all.Rdata",globalenv())
```

`locator` *Graphical Input* `locator`

Description

Reads the position of the graphics cursor when the (first) mouse button is pressed.

Usage

```
locator(n = 512, type = "n")
```

Arguments

`n`	the maximum number of points to locate.
`type`	One of `"n"`, `"p"`, `"l"` or `"o"`. If `"p"` or `"o"` the points are plotted; if `"l"` or `"o"` they are joined by lines.

Details

Unless the process is terminated prematurely by the user (see below) at most `n` positions are determined.

The input process can be terminated prematurely by pressing any mouse button other than the first.

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.

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.

`log` *Logarithms and Exponentials* `log`

Description

`log` computes natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base` (`log10` and `log2` are only special cases).

`log1p(x)` computes $\log 1 + x$ accurately also for $|x| \ll 1$ (and less accurately when x = -1).

`exp` computes the exponential function.

Usage

```
log(x)
log(x, base)
log10(x)
log2(x)
exp(x)
log1p(x)
```

Arguments

`x`	a numeric or complex vector.
`base`	positive number. The base with respect to which logarithms are computed. Defaults to e=`exp(1)`.

Value

A vector of the same length as `x` containing the transformed values. `log(0)` gives `-Inf` (when available).

See Also

Trig, Math, Arithmetic.

Examples

```
log(exp(3))
all.equal(log(1:10), log(1:10, exp(1)))
log10(30) == log(30, 10)
log10(1e7)# = 7
log2(2^pi) == 2^log2(pi)
Mod(pi - log(exp(pi*1i)) / 1i) < .Machine$double.eps
Mod(1+exp(pi*1i)) < .Machine$double.eps

x <- 10^-(1+2*1:9)
cbind(x, log(1+x), log1p(x))
```

`Logic` *Logical Operators* `Logic`

Description

These operators act on logical vectors.

Usage

```
! x
x & y
x && y
x | y
x || y
xor(x, y)
```

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.

`xor` indicates elementwise exclusive OR.

See Also

`TRUE` or `logical`.

Examples

```
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"
```

`logical` *Logical Vectors* `logical`

Description

Create or test for objects of type `"logical"`, and the basic logical "constants".

Usage

```
TRUE
FALSE
T; F

logical(length = 0)
as.logical(x)
is.logical(x)
```

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.

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 `factor`s, this uses the `levels` (labels) and not the `codes`.

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

`Logistic` *The Logistic Distribution* `Logistic`

Description

Density, distribution function, quantile function and random generation for the logistic distribution with parameters `location` and `scale`.

Usage

```
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 to generate.
`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 μ 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.

Examples

```
eps <- 100 * .Machine$double.eps
x <- c(0:4, rlogis(100))
all.equal(plogis(x),                1 / (1 + exp(-x)), tol = eps)
all.equal(plogis(x, lower=F),   exp(-x)/ (1 + exp(-x)), tol = eps)
all.equal(plogis(x, lower=F, log=T), -log(1 + exp(x)),  tol = eps)
all.equal(dlogis(x), exp(x) * (1 + exp(x))^-2, tol = eps)

var(rlogis(4000, 0, s = 5))# approximately (+/- 3)
pi^2/3 * 5^2
```

`loglin` *Fitting Log-Linear Models* `loglin`

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 term 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 chisquare 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`.

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

Haberman, S. J. (1972) Log-linear fit for contingency tables—Algorithm AS51. *Applied Statistics*, **21**, 218–225.

Agresti, A. (1990) *Categorical data analysis.* New York: Wiley.

See Also

`table`

Examples

```
## Currently no appropriate data sets are available.
```

`Lognormal` *The Log Normal Distribution* `Lognormal`

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 to generate.
`meanlog, sdlog`	mean and standard deviation of the distribution on the log scale.
`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 `meanlog` or `sdlog` are not specified they assume the default values of `0` and `1` respectively.

The log normal distribution has density

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\log(x)-\mu)^2/2\sigma^2}$$

where μ and σ are the mean and standard deviation of the logarithm.

Value

`dlnorm` gives the density, `plnorm` gives the distribution function, `qlnorm` gives the quantile function, and `rlnorm` generates random deviates.

See Also

`dnorm` for the normal distribution.

Examples

```
dlnorm(1) == dnorm(0)
x <- rlnorm(1000)       # not yet always :
all(abs(x  -  qlnorm(plnorm(x))) < 1e4 * .Machine$double.eps * x)
```

`longley` *Longley's Regression Data* `longley`

Description

A macroeconomic data set which provides a well-known example for a highly collinear regression.

Usage

```
data(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 Producr.
Unemployed:	number of unemployed
Armed.Forces:	number of .. in the armed forces
Population:	'noninstitutionalized' population $\geq$ 14 years of age.
Year:	the year (time).
Employed:	number of people employed.

The regression `lm(Employed ~ .)` is known to be highly collinear.

Source

J. W. Longley (1967) An appraisal of least-squares programs from the point of view of the user. *Journal of the American Statistical Association*, **62**, 819–841.

Examples

```
# give the data set in the form it is used in S-plus:
data(longley)
longley.x <- data.matrix(longley[, 1:6])
longley.y <- longley[, "Employed"]
pairs(longley, main = "longley data")
summary(fm1 <- lm(Employed ~ ., data = longley))
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)
```

`lower.tri` *Lower and Upper Triangular Part of a Matrix* `lower.tri`

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 <- ma <- matrix(1:20, 4, 5)
m2[lower.tri(m2)] <- NA
m2
all(lower.tri(ma) == !upper.tri(ma, diag=TRUE))
```

`lowess` *Scatter Plot Smoothing* `lowess`

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, f=2/3, iter=3, delta=.01*diff(range(x)))
```

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 replaced by a single value in the output from `lowess`.

References

Cleveland, W. S. (1979) Robust locally weighted regression and smoothing scatterplots. *J. Amer. Statist. Assoc.* **74**, 829–836.

Cleveland, W. S. (1981) LOWESS: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, **35**, 54.

Examples

```
data(cars)
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` *List Objects* `ls`

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

```
ls(name, pos= -1, envir=pos.to.env(pos),
       all.names=FALSE, pattern)
objects(name, pos= -1, envir=pos.to.env(pos),
       all.names=FALSE, pattern)
```

Arguments

`name`	the name of an attached object appearing in the vector of names returned by `search`.
`pos`	the index of an attached object in the list returned by `search`. Defaults to the *current* environment.
`envir`	an evaluation environment. Defaults to the one corresponding to `pos`.
`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, see `grep`. Only names matching `pattern` are returned.

See Also

`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 = T)    # 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"
```

`ls.diag` *Compute Diagnostics for 'lsfit' Regression Results* `ls.diag`

Description

Computes basic statistics, including standard errors, t- and p-values for the regression coefficients.

Usage

```
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 σ.
`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

Belsley, D. A., Kuh, E. and Welsch, R. E. (1980) *Regression Diagnostics.* New York: Wiley.

See Also

`hat` for the hat matrix diagonals, `ls.print`, `lm.influence`, `summary.lm`, `anova`.

Examples

```
##-- 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* `ls.print`

Description

Computes basic statistics, including standard errors, t- and p-values for the regression coefficients and prints them if `print.it` is `TRUE`.

Usage

```
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* `lsfit`

Description

The least squares estimate of β in the model

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

is found.

Usage

```
lsfit(x, y, wt, 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 matrix valued 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`	an unused parameter for compatibility.

Details

If weights are specified then a weighted least squares is performed with the weight given to the jth case specified by the jth 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 (stated below).
`residuals`	residuals from the fit.
`intercept`	indicates whether an intercept was fitted.
`qr`	the QR decomposition of the design matrix.

See Also

`lm` which usually is preferable; `ls.print`, `ls.diag`.

Examples

```
##-- Using the same data as the lm(.) example:
lsD9 <- lsfit(x = codes(gl(2,10)), y = weight)
ls.print(lsD9)
```

`Machine` *Machine Characteristics* `Machine`

Description

`Machine()` returns information on numeric characteristics of the machine R is running on, such as the largest double or integer and the machine's precision.

`.Machine` is a variable holding this information.

Usage

```
Machine()
.Machine
```

Details

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

Value

`Machine()` returns 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 != 1`. It equals `base^ulp.digits` if either `base` is 2 or `rounding` is 0; otherwise, it is `(base^ulp.digits) / 2`.

`double.neg.eps`
a small positive floating-point number `x` such that `1 - x != 1`. It equals `base^neg.ulp.digits` if `base` is 2 or `round` is 0; otherwise, it is `(base^neg.ulp.digits) / 2`. As `neg.ulp.digits` is bounded below by `-(digits + 3)`, `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., `base^min.exp`.

`double.xmax` the largest finite floating-point number. Typically, it is equal to `(1 - neg.eps) * base^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 `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 + base^i != 1`, except that it is bounded below by `-(digits + 3)`.

`double.neg.ulp.digits`
the largest negative integer `i` such that `1 - base^i != 1`, except that it is bounded below by `-(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 `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.

References

Cody, W. J. (1988) MACHAR: A subroutine to dynamically determine machine parameters. *Transactions on Mathematical Software*, **14**, 4, 303–311.

See Also

`machine` to determine the computer type which R is running on.

Examples

```
str(Machine())
1 +     .Machine$double.eps != 1
1 + .5* .Machine$double.eps == 1
```

`machine` *Determine the Machine R is Running On* `machine`

Description

This function returns a string which specifies what kind of environment R is being run in.

Usage

```
machine()
```

See Also

`Machine` for the computer's characteristics in arithmetics.

Examples

```
machine()# to see yours
if (machine() == "Macintosh")
  cat("You are using a Macintosh computer\n")
```

`mad` *Median Absolute Deviation* `mad`

Description

Compute a scale estimate based on the median absolute deviation.

Usage

```
mad(x, center, constant = 1.4826, na.rm = FALSE)
```

Arguments

`x`	a numeric vector.
`center`	Optionally, the centre: defauls to the median.
`constant`	scale factor.
`na.rm`	if `TRUE` then `NA` values are stripped from `x` before computation takes place.

Details

The actual value calculated is `constant * (median(abs(x - center)))` with the default value of `center` being `median(x)`.

The default `constant = 1.4826` (approximately $1/\Phi^{-1}(\frac{3}{4}) =$ `1/qnorm(3/4)`) ensures consistency, i.e.,

$$E[mad(X_1, \ldots, X_n)] = \sigma$$

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
```

`mahalanobis` *Mahalanobis Distance* `mahalanobis`

Description

Returns the Mahalanobis distance of all rows in `x` and the vector μ =`center` with respect to Σ =`cov`. This is (for vector `x`) defined as

$$D^2 = (x - \mu)' \Sigma^{-1} (x - \mu)$$

Usage

```
mahalanobis(x, center, cov, inverted=FALSE)
```

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`, `cov` is supposed to contain the *inverse* of the covariance matrix.

Author(s)

Friedrich Leisch

See Also

`cov`, `var`

Examples

```
ma <- cbind(1:6, 1:3)
(S <-  var(ma))
mahalanobis(c(0,0), 1:2, S)

x <- matrix(rnorm(100*3), ncol = 3)
all(mahalanobis(x, 0, diag(ncol(x)))
    == apply(x*x, 1, sum)) ##- Here, D^2 = usual Euclidean distances
Sx <- cov(x)
D2 <- mahalanobis(x, apply(x, 2, mean), Sx)
plot(density(D2, bw=.5), main="Mahalanobis distances, n=100, p=3"); rug(D2)
qqplot(qchisq(ppoints(100), df=3), D2,
       main = expression("Q-Q plot of Mahalanobis" * ~D^2 *
                         " vs. quantiles of" * ~ chi[3]^2))
abline(0, 1, col = 'gray')
```

`make.link` *Create a Link for GLM families* `make.link`

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 λ 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))
```

`make.names` *Make Legal R Names Out of Strings* `make.names`

Description

Make legal R names out of every `names[i]` string. Invalid characters are translated to `"."`.

Usage

```
make.names(names, unique=FALSE)
```

Arguments

`names`	character (vector) to be coerced to proper names.
`unique`	logical; if `TRUE`, the resulting strings are unique. This may be desired for, e.g., column names.

Value

character vector of same length as `names` with each changed to a legal name.

See Also

`names`,`character`,`data.frame`

Examples

```
make.names(c("a and b", "a_and_b"), unique=TRUE)#-> "a.and.b"  "a.and.b1"

all(make.names(letters) == letters)# TRUE

data(state)
state.name[make.names(state.name) != state.name]# those 10 with a space
```

`make.socket` *Create a Socket Connection* `make.socket`

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

```
make.socket(host = "localhost", port, fail = TRUE, server = FALSE)
print.socket(socket)
```

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

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)
}
daytime()  ## only works if your computer runs this service
daytime("ntp.demon.co.uk")
```

`make.tables` *Create model.tables* `make.tables`

Description

These are support functions for (the methods of) `model.tables` and probably not much of use otherwise.

Usage

```
make.tables.aovproj    (proj.cols, mf.cols, prjs, mf,
                        fun = "mean", prt = FALSE, ...)
make.tables.aovprojlist(proj.cols, strata.cols, model.cols, projections,
                        model, eff, fun = "mean", prt = FALSE, ...)
```

See Also

`model.tables`

`margin.table` *Compute table margin* `margin.table`

Description

Compute the sum of table entries over a given index.

Usage

```
margin.table(x, margin)
```

Arguments

`x`	an array
`margin`	index number (1 for rows, etc.)

Details

This is really just `apply(x,margin,sum)` packaged up for newbies.

Value

The relevant marginal table.

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* `mat.or.vec`

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)
```

Examples

```
mat.or.vec(3, 1)
mat.or.vec(3, 2)
```

match *Value Matching* match

Description

match: If x[i] is found to equal table[j] then the value returned in the i-th position of the return value is j. If no match is found, the value is nomatch.

%in%: A utility function, currently defined as
"%in%" <- function(x, table) match(x, table, nomatch = 0) > 0 allowing an intuitive usage and returning a logical vector of length length(x).

Usage

```
match(x, table, nomatch=NA)
x %in% table
```

Arguments

x	the values to be matched.
table	the values to be matched against.
nomatch	the value to be returned in the case when no match is found.

See Also

pmatch and charmatch for (*partial*) string matching, match.arg, etc for function argument matching.

Examples

```
## 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* match.arg

Description

match.arg matches arg against a table of candidate values as specified by choices.

Usage

```
match.arg(arg)
match.arg(arg, choices)
```

Arguments

`arg`	a character string
`choices`	a character vector of candidate values

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.

See Also

`pmatch`, `match.fun`, `match.call`.

Examples

```
## 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
center(x, "m")       # Error
```

`match.call` *Argument Matching* `match.call`

Description

`match.call` returns a call in which all of the arguments are specified by their names. The most common use is to get the call of the current function, with all arguments named.

Usage

```
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?

Value

An object of class `call`.

See Also

`call`, `pmatch`, `match.arg`, `match.fun`.

Examples

```
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)
```

`match.fun` *Function Verification for "Function Variables"* `match.fun`

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 `attach`es 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

match.fun(outer, descend = FALSE) #-> Error:  not a function

match.fun(outer) # finds it anyway
is.function(match.fun("outer")) # as well
```

`matmult` *Matrix Multiplication* `matmult`

Description

Multiplies two matrices, if they are conformable. If one argument is a vector, it will be coerced to a 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
```

Value

The matrix product. Use `drop` to get rid of

See Also

`matrix`, `Arithmetic`, `diag`.

Examples

```
x <- 1:4
x%*%x # scalar ("inner") product (1 x 1 matrix)
drop(.Last.value)# as scalar

y <- diag(x)
z <- matrix(1:12,ncol=3,nrow=4)
y%*%z
y%*%x
x%*%z
```

`matplot` *Plot Columns of Matrices* `matplot`

Description

Plot the columns of one matrix against the columns of another.

Usage

```
matplot(x, y, type = "p", lty = 1:5, lwd = 1, pch = NULL, col = 1:6,
        xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
        ..., add = FALSE)
matpoints(x, y, lty = 1:5, lwd = 1, pch = NULL, col = 1:6)
matlines(x, y, 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, telling which type of plot (`"p"`, points; `"l"`, lines; `"b"`, both; `"o"`, overplotted both; `"n"`, none; or `"h"`, high-density) should be done for each column of `y`. 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 (length 1 vector) or vector of 1-character strings for plotting-characters. 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.
`xlim, ylim`	ranges of x and y axes, as in `plot`.
`xlab, ylab`	titles for x and y axes, as in `plot`.
`...`	Graphical parameters (see `par`) may also be supplied as arguments to this function. In addition, 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.)

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.

See Also

`plot`, `points`, `lines`, `matrix`, `par`.

Examples

```
matplot((-4:5)^2, main = "Quadratic")
sines <- outer(1:20, 1:4, function(x, y) sin(x / 20 * pi * y))
matplot(sines, type = "o", col = rainbow(ncol(sines)))

x <- 0:100/100
matplot(x, outer(x, 1:8, function(x, k) sin(k * pi * x)),
        type = "l", col = 1:8, ylim = c(-2,2))

data(iris)                  # is data.frame with  'Species' factor
table(iris$Species)
iS <- iris$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", "    Setosa Sepals",
               "Versicolor Petals", "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])

matplot(iris.S[,"Petal.Length",], iris.S[,"Petal.Width",], pch="SCV",
        col = rainbow(3, start = .8, end = .1),
        sub = paste(c("S", "C", "V"), dimnames(iris.S)[[3]],
                    sep = "=", collapse= ",  "),
        main = "Fisher's Iris Data")
```

`matrix` *Matrices* `matrix`

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.

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.
`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.

`is.matrix` returns `TRUE` if `x` is a matrix (i.e., it *not* a `data.frame` and has a `dim` attribute of length 2) and `FALSE` otherwise.

See Also

`data.matrix`.

Examples

```
is.matrix(as.matrix(1:10))
data(warpbreaks)
!is.matrix(warpbreaks)# data.frame, NOT matrix!
str(warpbreaks)
str(as.matrix(warpbreaks))#using as.matrix.data.frame(.) method
```

`mean` *Arithmetic Mean* `mean`

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, trim=0, na.rm=FALSE)
```

Arguments

`x`	a numeric vector containing the values whose mean is to be computed.
`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.

Value

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.

See Also

`weighted.mean`

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = .10))
```

`median` *Median Value* `median`

Description

Compute the sample median of the vector of values given as its argument.

Usage

```
median(x, na.rm=FALSE)
```

Arguments

`x`	a numeric vector containing the values whose median is to be computed.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.

See Also

`quantile` for general quantiles.

Examples

```
median(1:4)# = 2.5 [even number]
median(c(1:3,100,1000))# = 3 [odd, robust]
```

`Memory` *Memory Available for Data Storage* `Memory`

Description

Use command line options to set the memory available for R.

Usage

```
R --vsize=v --nsize=n
```

Arguments

`v`	Use `v` bytes of heap memory
`n`	Use `n` cons cells.

Details

R (currently) uses a static memory model. This means that when it starts up, it asks the operating system to reserve a fixed amount of memory for it. The size of this chunk cannot be changed subsequently. Hence, it can happen that not enough memory was allocated, e.g., when trying to read large data sets into R.

In these cases, you should restart R (after saving your current workspace) with more memory available, using the command line options `--nsize` and `--vsize`. To understand these 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" (see `gc()["Vcells","total"]`) of 8 bytes each. Effectively, the input `v` is therefore truncated to the nearest multiple of 8.

The `--nsize` option can be used to specify the number of cons cells (each occupying 20 bytes on a 32-bit machine) which R is to use (the default is 250000), and the `--vsize` option to specify the size of the vector heap in bytes (the default is 6 MB). Both options must be integers or integers ending with `M`, `K`, or `k` meaning *Mega* ($2^{20} = 1048576$), (computer) *Kilo* ($2^{10} = 1024$), or regular *kilo* (1000). (The minimum allowed values are 200000 and 2M.)

E.g., to read in a table of 10000 observations on 40 numeric variables, `R --vsize=10M` should do; for sourcing a large file, you would use `R --nsize=500k`.

Note that the information on where to find vectors and strings on the heap is stored using cons cells. Thus, it may also be necessary to allocate more space for cons cells in order to perform computations with very "large" variable-size objects.

You can find out the current memory consumption (the proportion of heap and cons cells used) by typing `gc()` at the R prompt. This may help you in finding out whether to increase

`--vsize` or `--nsize`. Note that following `gcinfo(TRUE)`, automatic garbage collection always prints memory use statistics.

R will tell you whether you ran out of cons or heap memory.

The defaults for `--nsize` and `--vsize` can be changed by setting the environment variables `R_NSIZE` and `R_VSIZE` respectively, perhaps most conveniently in the file '.Renviron' or '~/.Renviron'.

When using `read.table`, the memory requirements are in fact higher than anticipated, because the file is first read in as one long string which is then split again. Use `scan` if possible in case you run out of memory when reading in a large table.

See Also

`gc` for information on the garbage collector.

Examples

```
# Start R with 15MB of heap memory and 1 million cons cells

## Unix
R --vsize=15M --nsize=1000k
```

`memory.profile` *Profile the Usage of Cons Cells* `memory.profile`

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'. There will be blanks in the list corresponding to types that are no longer in use (types 11 and 12 at the time of writing). Also `FUNSXP` is not included.

Value

A vector of counts, named by the types.

See Also

`gc` for the overall usage of cons cells.

Examples

```
memory.profile()
```

`menu` *Menu Interaction Function* `menu`

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. Currently unused.
`title`	a character string to be used as the title of the menu

Value

The number corresponding to the selected item, or 0 if no choice was made.

Examples

```
switch(menu(c("List letters", "List LETTERS")) + 1,
       cat("Nothing done\n"), letters, LETTERS)
```

`merge` *Merge Two Data Frames* `merge`

Description

Merge two data frames by common columns or row names.

Usage

```
merge(x, y, by, by.x, by.y, sort = TRUE)
```

Arguments

`x, y`	data frames, or objects to be coerced to one
`by, by.x, by.y`	specifcations of the common columns. See Details.
`sort`	logical. Should the results be sorted on the `by` columns?

Details

By default the data frames are merged on the columns with names they both have, but separate specifcations 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 remaining columns in the data frames have any common names, these have `".x"` and `".y"` appended to make the names of the result unique.

Value

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 `x`. 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

```
authors <- data.frame(
    surname = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil"),
    nationality = c("US", "Australia", "US", "UK", "Australia"),
    retired = c("yes", rep("no", 4)))
books <- data.frame(
    name = c("Tukey", "Venables", "Tierney", "Ripley", "Ripley", "McNeil"),
    title = c("Exploratory Data Analysis",
              "Modern Applied Statistics ...",
              "LISP-STAT",
              "Spatial Statistics", "Stochastic Simulation",
               "Interactive Data Analysis"),
    other.author = c(NA, "Ripley", NA, NA, NA, NA))

merge(authors, books, by.x="surname", by.y="name")
merge(books, authors, by.x="name", by.y="surname")
```

`Methods` *Internal and Group Methods and Generic Functions* `Methods`

Description

Many R-internal functions are *generic* and allow methods to be written for. *Group* methods in particular are available for the `"Math"`, `"Ops"`, and `"Summary"` group.

Usage

```
Math.data.frame(x, ...)
Math.factor(x, ...)

Ops.data.frame(e1, e2 = NULL)
Ops.factor(e1, e2)
Ops.ordered(e1, e2)

Summary.data.frame(x, ...)
Summary.factor(x, ...)

.Method
.Generic
.Group
.Class
```

Group Dispatching

There are three *groups* for which methods can be written, namely the `"Math"`, `"Ops"` and `"Summary"` groups.

A function *f* belonging to one of these groups must be `.Internal` or `.Primitive` and will automatically be using *<grp>.<class> (ob)* when *f(<ob>)* is called, *f* belongs to group *<grp>* and *<ob>* is of `class` *<class>*.

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`, `tetragamma`, `pentagamma`
   - `cumsum`, `cumprod`, `cummax`, `cummin`
2. Group `"Ops"`:
   - `"+"`, `"-"`, `"*"`, `"/"`, `"^"`, `"%%"`, `"%/%"`
   - `"&"`, `"|"`, `"!"`
   - `"=="`, `"!="`, `"<"`, `"<="`, `">="`, `">"`
3. Group `"Summary"`:
   - `all`, `any`
   - `sum`, `prod`
   - `min`, `max`

Simple Dispatching

The following builtin functions are *generic* as well, i.e., you can write `methods` for them:

`[`, `[[`

`dimnames<-`, `dimnames`, `dim<-`, `dim`

`c`, `unlist`, `as.vector`, `is.na`, `is.nan`

References

Appendix A, *Classes and Methods* of
Chambers, J. M. and Hastie, T. J. eds (1992) *Statistical Models in S.* Wadsworth & Brooks/Cole.

See Also

`methods` for methods of non-Internal generic functions.

Examples

```
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. ...
```

`methods` *Class Methods* `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, object, ...)
methods(generic.function, class)
```

Details

An R "object" is a data object which has a `class` attribute. A class attribute is a vector of character strings giving the names of the classes which the object "inherits" from. 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.

`methods` can be used to find out about the methods for a particular generic function or class. See the examples below for details.

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 (!?). Any statements after the call to `UseMethod` will not be evaluated as `UseMethod` does not return.

`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.

Note

The `methods` function was written by Martin Maechler.

See Also

`class`

Examples

```
methods(summary)

methods(print)

methods(class = data.frame)

methods("[")#- does not list the C-internal ones...
```

`missing` *Does a Formal Argument have a Value?* `missing`

Description

`missing` can be used to test whether a value was specified as an argument to a function. The following 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.

Usage

```
missing(x)
```

See Also

`substitute` for argument expression; `NA` for "missing values" in data.

Examples

```
myplot <- function(x,y) {
                if(missing(y)) {
                        y <- x
                        x <- 1:length(y)
                }
                plot(x,y)
        }
```

mode	*The (Storage) Mode of an Object*	mode

Usage

```
mode(x)
mode(x) <- "<mode>"
storage.mode(x)
storage.mode(x) <- "<mode>"
```

Details

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`.

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 `call`s have mode `"("` which is S compatible.

See Also

`typeof` for the R-internal "mode", `attributes`.

Examples

```
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((1))[[1]]", "(y~x)[[1]]", "expression(x <- pi)[[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)
```

`model.extract` *Extract Components from a Model Frame* `model.extract`

Description

Returns the response, offset, subset, weights or other special components of a model frame passed as optional arguments to `model.frame`.

Usage

```
model.extract(frame, component)
model.offset(frame)
model.response(frame)
model.weights(x)
```

Arguments

`frame, x`	A model frame
`component`	The name of a components to extract, such as `"weights"`, `"subset"`

Details

`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

```
data(esoph)
a <- model.frame(cbind(ncases,ncontrols) ~ agegp+tobgp+alcgp, data=esoph)
model.extract(a, "response")
all(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)
all(model.extract(a, "something") == esoph$tobgp) # TRUE
```

`model.frame` *Extracting the "Environment" of a Model Formula* `model.frame`

Description

`model.frame` (generic function) and its methods return a `data.frame` with the variables from `formula`.

Usage

```
model.frame(formula, data, na.action, ...)
model.frame.default(formula, data = sys.frame(sys.parent()), na.action,
                    drop.unused.levels = FALSE, xlev = NULL, ...)

Methods for
        lm      glm     aovlist
```

Arguments

`formula`	a model formula
`data`	`data.frame`, list, `environment` or object coercible to `data.frame` containing the variables in `formula`.
`na.action`	how `NA`s 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`.
`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 `subset`, `offset` and `weights`. `NULL` arguments are treated as missing.

Value

A `data.frame` containing the variables used in `formula`.

See Also

`model.matrix` for the "design matrix".

Examples

```
data(cars)
data.class(model.frame(dist ~ speed, data = cars))
```

model.matrix *Construct Design Matrices* model.matrix

Description

`model.matrix` creates a design matrix.

Usage

```
model.matrix   (object, ...)
model.matrix.lm(object, ...)
model.matrix.default(formula, data, contrasts.arg = NULL, xlev = NULL)
```

Arguments

`formula`	a model formula or terms object.
`data`	a data frame created with `model.frame`.
`contrasts.arg`	A list, whose entries are contrasts suitable for input to the `contrasts` function and whose names are the names of columns of `data` containing `factor`s.
`xlev`	to be used as argument of `model.frame` if `data` has no `"terms"` attribute.
`x`	a model frame.

Details

`model.matrix` creates a design matrix from the description given in `terms(formula)`, using the data in `data` which must contain columns with the same names as would be created by a call to `model.frame(formula)` or, more precisely, by evaluating `attr(terms(formula), "variables")`. There may be other columns and the order is not important. If `contrasts` is specified it overrides the default factor coding for that variable.

Value

The design matrix for a regression model with the specified formula and data.

References

Chambers, J. M. and Hastie, T. J. eds (1992) *Statistical Models in S.* Chapman & Hall, London.

See Also

`model.frame`, `model.extract`, `terms`

Examples

```
data(trees)
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.* `model.tables`

Description

Computes summary tables for model fits, especially complex `aov` fits.

Usage

```
model.tables(x, ...)
model.tables.aov(x, type = "effects", se = FALSE, cterms)
model.tables.aovlist(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.

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.

Author(s)

B.D. Ripley

See Also

aov

Examples

```
## From Venables and Ripley (1997) p.210.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
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(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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)
npk.aov <- aov(yield ~ block + N*P*K, npk)
model.tables(npk.aov, "means", se=T)

## as a test, not particularly sensible statistically
options(contrasts=c("contr.helmert", "contr.treatment"))
npk.aovE <- aov(yield ~  N*P*K + Error(block), npk)
model.tables(npk.aovE,  se=T)
model.tables(npk.aovE, "means")
```

`mosaicplot` *Mosaic Plots* `mosaicplot`

Description

Plots a mosaic on the current graphics device.

Usage

```
mosaicplot(x, ...)
mosaicplot.default(X, main = NULL, xlab = NULL, ylab = NULL,
                   sort = NULL, off = NULL, dir = NULL,
                   color = FALSE)
mosaicplot.formula(formula, data = NULL, subset, na.action, ...)
```

Arguments

`x`	an R object.
`X`	a contingency table, with optional category labels specified in the `dimnames(x)` attribute. The table is best created by the `table()` command, which produces an object of type array.
`main`	character string for the mosaic title.
`xlab,ylab`	x- and y-axis labels; by default, `names(dimnames(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 10 at each level). There should be one offset for each dimension of the contingency table.

`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`	(`TRUE` or vector of integer colors) for color shading or (`FALSE`, the default) for empty boxes with no shading.
`formula`	a formula, such as `y ~ x`.
`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 in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s.
`...`	further arguments to the default mosaicplot method.

Details

This is a generic function. It currently has a default method (`mosaicplot.default`) and a formula interface (`mosaicplot.formula`).

See Emerson (1998) for more information and a case study with television viewer data from Nielsen Media Research.

Author(s)

S-PLUS original by John Emerson ⟨emerson@stat.yale.edu⟩. Slightly modified for R by KH.

References

Hartigan, J.A., and Kleiner, B. (1984) A mosaic of television ratings. *The American Statistician*, **38**, 32–35.

Emerson, J. W. (1998) Mosaic displays in S-PLUS: a general implementation and a case study. *Statistical Computing and Graphics Newsletter (ASA)*, **9**, 1, 17–23.

Friendly, M. (1994) Mosaic displays for multi-way contingency tables. *Journal of the American Statistical Association*, **89**, 190–200.

The home page of Michael Friendly (http://hotspur.psych.yorku.ca/SCS/friendly.html) provides information on various aspects of graphical methods for analyzing categorical data, including mosaic plots.

Examples

```
data(Titanic)
mosaicplot(Titanic, main = "Survival on the Titanic", color = TRUE)

## Formula interface: visualize crosstabulation of numbers of gears and
## carburettors in Motor Trend car data.
data(mtcars)
mosaicplot(~ gear + carb, data = mtcars, color = TRUE)
```

`mtcars` *Motor Trend Road Tests* `mtcars`

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

```
data(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 carburettors

Source

Henderson and Velleman (1981), Building multiple regression models interactively. *Biometrics*, **37**, 391–411.

Examples

```
data(mtcars)
pairs(mtcars, main = "mtcars data")
coplot(mpg ~ disp | as.factor(cyl), data = mtcars,
       panel = panel.smooth, rows = 1)
```

`mtext` *Write Text into the Margins of a Plot* `mtext`

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

```
mtext(text, side = 3, line = 0, outer = FALSE, at = NULL,
      adj = NA, ...)
```

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. For strings parallel to the axes, `adj=0` means left or bottom alignment, and `adj=1` means right or top aligment. If `adj` is not a finite value (the default), the value `par("las")` determines the adjustment. For strings plotted parallel to the axis the default is to centre the string.
`...`	Further graphical parameters; currently supported are:
`cex`	character expansion factor (default = 1).
`col`	color to use.
`font`	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).

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 justifcation is to place the end of the string nearest the axis on the specified line.

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.

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)
```

NA *Not Available / "Missing" Values* NA

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.

The generic function 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.

Usage

```
NA
is.na(x)
is.na.data.frame(x)
```

Details

is.na(x) works elementwise when x is a list.

The method dispatching is C-internal, rather than via UseMethod.

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

```
is.na(c(1,NA))       #> F TRUE
is.na(paste(c(1,NA)))#> F FALSE
```

`na.action` *NA Action* `na.action`

Description

`na.action` is a generic function, and `na.action.default` its default method.

Usage

```
na.action(x, ...)
na.action.default(x)
```

Arguments

`x`	any object whose `NA` action is given.
`...`	further arguments special methods could assume.

Value

The "NA action" which should be applied to `x` whenever `NA`s are not desired.

See Also

`options("na.action")`, `na.omit`, `na.fail`

Examples

```
na.action(c(1, NA))
```

`na.fail` *Handle Missing Values in Objects* `na.fail`

Description

These generic functions are useful for dealing with `NA`s in e.g., data frames. `na.fail` returns the data frame if it does not contain any missing values, and signals an error otherwise. `na.omit` returns the data frame with incomplete cases removed.

Usage

```
na.fail(object)
na.omit(object)
```

Arguments

`object`	an R object, typically a data frame

See Also

`na.action`, `options(na.action=..)` for setting "NA actions", and `lm` and `glm` for functions using these.

Examples

```
DF <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA))
na.omit(DF)
na.fail(DF)#> Error: missing values in ..

options("na.action")
```

`name` *Variable Names* `name`

Description

`as.name` coerces its argument to be a name. The argument must be of mode `"character"`.

`is.name` returns `TRUE` or `FALSE` depending on whether its argument is a name or not.

Usage

```
as.name(x)
is.name(x)
```

See Also

`call`, `is.language`. For the internal object mode, `typeof`.

Examples

```
an <- as.name("arrg")
is.name(an) # TRUE
str(an)# symbol
```

`names` *The Names Attribute of an Object* `names`

Description

`names` is a generic accessor function to the `names` attribute of an R object, typically a `vector`. The first form prints the names of the observations and the second sets the names. In this case, `value` must be a vector of character strings of the same length as `x`.

Usage

```
names(x, ...)
names.default(x)
names(x) <- value
```

Examples

```
# print the names attribute of the islands data set
names(islands)

# remove the names attribute
names(islands) <- NULL
```

nargs *The Number of Arguments to a Function* nargs

Description

When used inside a function body, `nargs` returns the number of arguments supplied to that function.

Usage

```
nargs()
```

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

nargs()# not really meaningful
```

nchar *Count the Number of Characters* nchar

Description

`nchar` takes a character vector as an argument and returns a vector whose elements contain the number of characters in the corresponding element of `x`. It only accepts character vectors as arguments if you want to operate on other objects passing them through `deparse` first will be required.

Usage

```
nchar(x)
```

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))
# 23  1 16 45 11 64  2 17 50 43  2 17  1
```

NegBinomial *The Negative Binomial Distribution* NegBinomial

Description

Density, distribution function, quantile function and random generation for the negative binomial distribution with parameters `size` and `prob`.

Usage

```
dnbinom(x, size, prob, log = FALSE)
pnbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qnbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rnbinom(n, size, prob)
```

Arguments

`x, q`	vector of quantiles representing the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached, or alternately the probability distribution of a compound Poisson process whose intensity is distributed as a gamma (`pgamma`) distribution with scale parameter `(1-prob)/prob` and shape parameter `size` (this definition allows non-integer values of `size`).
`x`	vector of (non-negative integer) quantiles.
`q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations to generate.
`size`	target for number of successful trials, or shape parameter of gamma distribution.
`prob`	probability of success in each trial, or scale of gamma distribution (`prob` = `scale/(1+scale)`).
`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 `size` $= n$ and `prob` $= p$ has density

$$p(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!}p^n(1-p)^x$$

for $x = 0, 1, 2, \ldots$

If an element of `x` is not integer, the result of `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

`dnbinom` gives the density, `pnbinom` gives the distribution function, `qnbinom` gives the quantile function, and `rnbinom` generates random deviates.

See Also

`dbinom` for the binomial, `dpois` for the Poisson and `dgeom` for the geometric distribution, which is a special case of the negative binomial.

Examples

```
x <- 0:11
dnbinom(x, size = 1, prob = 1/2) * 2^(1 + x) # == 1
126 /  dnbinom(0:8, size  = 2, 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
size <- (1:20)/4
persp(x,size, dnb <- outer(x,size,function(x,s)dnbinom(x,s, pr= 0.4)),
      xlab = "x", ylab = "s", zlab="density", 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)
```

`nextn` *Highly Composite Numbers* `nextn`

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` to 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

```
nextn(1001) # 1024
table(sapply(599:630, nextn))
```

`nhtemp` *Average Yearly Temperatures in New Haven* `nhtemp`

Description

The mean annual temperature in degrees Fahrenheit in New Haven, Connecticut, from 1912 to 1971.

Usage

```
data(nhtemp)
```

Format

A time series of 60 observations.

Source

Vaux, J. E. and Brinker, N. B. (1972) *Cycles*, **1972**, 117–121.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(nhtemp)
plot(nhtemp, main = "nhtemp data",
  ylab = "Mean annual temperature in New Haven, CT (deg. F)")
```

`nlevels` *The Number of Levels of a Factor* `nlevels`

Usage

```
nlevels(x)
```

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
```

nlm *Non-Linear Minimization* nlm

Description

This function carries out a minimization of the function `f` using a Newton-type algorithm. See the references for details.

This is a preliminary version of this function and it will probably change.

Usage

```
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.
`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.

Value

A list containing the following components:

`minimum`	the value of the estimated minimum of `f`.
`estimate`	the point at which the mininum 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 `stepmax` is too small.

References

Dennis, J. E. and Schnabel, R. B. (1983) *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, NJ.

Schnabel, R. B., Koontz, J. E. and Weiss, B. E. (1985) A modular system of algorithms for unconstrained minimization. *ACM Trans. Math. Software*, **11**, 419–440.

See Also

`optimize` for one-dimensional minimization and `uniroot` for root finding. `deriv` to calculate analytical derivatives.

Examples

```
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))

## more examples, including the use of derivatives.
demo(nlm)
```

`noquote` *Class for "no quote" Printing of Strings* `noquote`

Description

These functions exist both as utilities and as an example of using `class` and object orientation.

Usage

```
noquote(obj)
print.noquote(obj, ...)
obj[j]
```

Arguments

`obj`	any R object; typically a vector of `character` strings.
`...`	further options for `print`.

Value

`noquote` returns its argument as an object of class `"noquote"`. The function `"[.noquote"` ensures that the class is not lost by subsetting.

For (default) printing, `print.noquote` will be used which prints characters *without* quotes (`"..."`).

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)
  noquote(if(length(log.v)==0)"()" else c(".","|")[1+log.v])
}
cmp.logical(runif(20) > 0.8)
```

`Normal` *The Normal Distribution* `Normal`

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.
`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

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

where μ is the mean of the distribution and σ the standard deviation.

Value

`dnorm` gives the density, `pnorm` gives the distribution function, `qnorm` gives the quantile function, and `rnorm` generates random deviates.

See Also

`runif` and `.Random.seed` about random number generation, and `dlnorm` for the *Log*normal distribution.

Examples

```
dnorm(0) == 1/ sqrt(2*pi)
dnorm(1) == exp(-1/2)/ sqrt(2*pi)
dnorm(1) == 1/ sqrt(2*pi*exp(1))

## 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)
mtext("dnorm(x, log=TRUE)", adj=0); mtext("log(dnorm(x))", col="red", adj=1)

plot(function(x)pnorm(x, log=TRUE), -50, 10, main = "log { Normal Cumulative }")
curve(log(pnorm(x)), add=TRUE, col="red",lwd=2)
mtext("pnorm(x, log=TRUE)", adj=0); mtext("log(pnorm(x))", col="red", adj=1)
```

NotYet *Not Yet Implemented Functions and Unused Arguments* NotYet

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)

symbols(...)
```

Arguments

arg	an argument of a function that is not yet used.

See Also

the contrary, Deprecated and Defunct for outdated code.

Examples

```
symbols                     # to see how the ‘‘NotYetImplemented’’
.NotYetImplemented          # reference is made automagically

barplot(1:5, inside = TRUE) # ‘inside’ is not yet used
```

nrow *The Number of Rows/Columns of an Array* nrow

Description

nrow and ncol return the number of rows or columns present in x. NCOL and NROW do the same, but treat a vector as 1-column matrix.

Usage

```
nrow(x)
ncol(x)
NCOL(x)
NROW(x)
```

Arguments

x	a vector, array or data frame

See Also

dim which returns *all* dimensions; array, matrix.

Examples

```
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
```

NULL *The Null Object* NULL

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

```
NULL
as.null(x)
is.null(x)
```

Examples

```
is.null(list())    # TRUE (not so in S)
is.null(integer(0))# F
is.null(logical(0))# F
as.null(list(a=1,b='c'))
```

---

numeric *Numeric Vectors* numeric

Description

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 numeric type (either integer or real).

is.numeric returns TRUE if its argument is of type real or type integer and FALSE otherwise.

Usage

```
numeric(length = 0)
as.numeric(x)
is.numeric(x)
```

Note

R *has no single precision data type. All real numbers are stored in double precision format.*

`as.numeric` for factors yields the codes underlying the factor levels.

Examples

```
as.numeric(c("-.1"," 2.7 ","B")) # (-0.1, 2.7, NA)  +  warning
as.numeric(factor(5:10))
```

`offset` *Include an Offset in a Model Formula* `offset`

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

```
offset(x)
```

Arguments

`x`	An offset to be included in a model frame

Value

The input value.

See Also

`model.offset`, `model.frame`, `glm`

`on.exit` *Function Exit Code* `on.exit`

Description

`on.exit` records the expression given as its argument as needing to be executed when then current function exits (either naturally or as the result of an error). This is useful for resetting graphical parameters or performing other cleanup actions.

Usage

```
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.

Examples

```
opar <- par(mai=c(1,1,1,1))
on.exit(par(opar))
```

`optim` *General-purpose Optimization* `optim`

Description

General-purpose optimization based on Nelder–Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization.

Usage

```
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, 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. Not needed for the `"Nelder-Mead"` and `"SANN"` method. If it is `NULL` and it is needed, a finite-difference approximation will be used. It is guaranteed that `gr` will be called immediately after a call to `fn` at the same parameter values.
`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.* (1994) 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.

Method `"SANN"` is 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. The next candidate point is generated from a Gaussian Markov kernel with scale proportional to the actual temperature. 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` Logical. If true, tracing information on the progress of the optimization is produced.

`fnscale` An overall scaling to be applied to the value of `fn` and `gr` during optimization. If negative, turns the problem into a maximization problem. Optimization is performed on `fn(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 criteria. 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 * (abs(val) + reltol)` at a step.

`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"` method in `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

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 make freely available.

The code for method `"L-BFGS-B"` is based on Fortran code by Zhu, Byrd, Lu-Chen and Nocedal obtained from Netlib.

The code for method `"SANN"` was contributed by A. Trapletti.

References

Belisle, C. J. P. (1992) Convergence theorems for a class of simulated annealing algorithms on R^d. *J Applied Probability*, **29**, 885–895.

Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C. (1995) A limited memory algorithm for bound constrained optimization. *SIAM J. Scientific Computing*, **16**, 1190–1208.

Fletcher, R. and Reeves, C. M. (1964) Function minimization by conjugate gradients. *Computer Journal* **7**, 148–154.

Nash, J. C. (1990) *Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation.* Adam Hilger.

Nelder, J. A. and Mead, R. (1965) A simplex algorithm for function minimization. *Computer Journal* **7**, 308–313.

See Also

nlm, optimize

Examples

```
## Rosenbrock Banana function
fr <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}

grr <- function(x) {
    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)
     sum(c(1, rep(4, length(x)-1))*(x - c(1, x[-length(x)])^2)^2)
optim(rep(3, 25), flb, NULL, "L-BFGS-B",
      lower=rep(2, 25), upper=rep(4, 25))

## "wild" function
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)  # global minimum at about -15.81515
res <- optim(50, fw, method="SANN",
             control=list(maxit=20000, temp=20, parscale=20))
res
optim(res$par, fw, method="BFGS")
```

`optimize` *One Dimensional Optimization* `optimize`

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.

It uses Fortran code (from Netlib) based on algorithms given in the reference.

`optimise` is an alias for `optimize`.

Usage

```
optimize(f = , interval = , lower = min(interval),
        upper = max(interval), maximum = FALSE,
        tol = .Machine$double.eps^0.25, ...)
optimise(f = , interval = , lower = min(interval),
        upper = max(interval), maximum = FALSE,
        tol = .Machine$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`	the upper end point of the interval to be searched.
`tol`	the desired accuracy.
`...`	additional arguments to `f`.

Value

A list with components `minimum` (or `maximum`) and `objective` which give the location of the minimum (or maximum) and the value of the function at that point.

References

Brent, R. (1973) *Algorithms for Minimization without Derivatives.* Englewood Cliffs N.J.: Prentice-Hall.

See Also

`nlm`, `uniroot`.

Examples

```
f <- function (x,a) (x-a)^2
xmin <- optimize(f, c(0, 1), tol = 0.0001, a = 1/3)
xmin
```

`options` *Options Settings* `options`

Description

`options` allows 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`. However, only the ones below are used in "base R". Further, `options('name') == options()['name']`, see the example.
`prompt`	a string, used for R's prompt; should usually end in a blank (`" "`).
`continue`	a string setting the prompt used for lines which continue over one line.
`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.
`digits`	controls the number of digits to print when printing numeric values. It is a suggestion only.
`editor`	sets the default text editor, e.g., for `edit`. Set from the environment variable `VISUAL` on UNIX.
`pager`	the (stand-alone) program used for displaying ASCII files on R's console. Defaults to '$R_HOME/bin/pager' on UNIX.
`browser`	default HTML browser used by `help.start()` on UNIX.
`mailer`	default mailer used by `bug.report()`. can be `"none"`.
`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.
`expressions`	sets a limit on the number of nested expressions that will be evaluated. This is especially important on the Macintosh since stack overflow is likely if this is set too high.
`keep.source`	When `TRUE`, the default, the source code for functions loaded by is stored in their `"source"` attribute, allowing comments to be kept in the right places. This does not apply to functions loaded by `library`.
`na.action`	the name of a function for treating missing values (`NA`'s) for certain situations.
`papersize`	the paper format used for graphics printing; currently **read-only**, set by environment variable `R_PAPERSIZE`, or in 'config.site'.
`printcmd`	the command used for graphics printing; currently **read-only**, set by environment variable `R_PRINTCMD`, or in 'config.site'.
`latexcmd, dvipscmd`	character strings giving commands to be used in off-line printing of help pages.
`show.signif.stars, show.coef.Pvalues`	logical, affecting P value printing, see `print.coefmat`.
`ts.eps`	the relative tolerance for certain time series (`ts`) computations.
`error`	an expression governing the handling of non-catastrophic errors such as those generated by `stop` as well as by signals and internally detected errors. The default expression is `NULL`: see `stop` for the behaviour in that case. The function `dump.frames` provides one alternative that allows post-mortem debugging.
`show.error.messages`	a logical. Should error messages be printed? Intended for use with `try` or a user-installed error handler.

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.
echo	logical. Only used in non-interactive mode, when it controls whether input is echoed. Command-line options --quiet and --slave set this initially to FALSE.
verbose	logical. Should R report extra information on progress? Set to TRUE by the command-line option --verbose.
device	a character string giving the default device for that session.
X11colortype	The default colour type for X11 devices.
CRAN	The URL of the preferred CRAN node for use by update.packages. Defaults to http://cran.r-project.org.
unzip	the command used unzipping help files.
x	a character string holding one of the above option names.

Details

Invoking options() with no arguments returns a list with the current values of the options. 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.

The default settings of some of these options are

prompt	"> "	continue	"+ "
width	80	digits	7
expressions	500	keep.source	TRUE
show.signif.stars	TRUE	show.coef.Pvalues	TRUE
na.action	na.omit	ts.eps	1e-5
error	NULL	warn	0
echo	TRUE	verbose	FALSE

Others are set from environment variables or are platform-dependent.

Value

A list (in any case) with the previous values of the options changed, or all options when no arguments were given.

Examples

```
options() # printing all current options
op <- options(); str(op) # nicer printing

# .Options is the same:
all(sapply(1:length(op), function(i) all(.Options[[i]] == op[[i]])))

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(op)     # reset (all) initial options
options('digits')

## set contrast handling to be like S
options(contrasts=c("contr.helmert", "contr.poly"))

## on error, terminate the R session with error status 66
options(error=quote(q("no", status=66, runLast=FALSE)))
stop("test it")
```

`OrchardSprays` *Potency of Orchard Sprays* `OrchardSprays`

Description

An experiment was conducted to assess the potency of various constituents of orchard sprays in repelling honeybees, using a Latin square design.

Usage

```
data(OrchardSprays)
```

Format

A data frame with 64 observations on 4 variables.

[,1]	rowpos	numeric	Row of the design
[,2]	colpos	numeric	Column of the design
[,3]	treatment	factor	Treatment level
[,4]	decrease	numeric	Response

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 $* \times 8$ 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
.	
.	
.	

G	lowest level of lime sulphur
H	no lime sulphur

Source

Finney, D. J. (1947) *Probit Analysis.* Cambridge.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(OrchardSprays)
pairs(OrchardSprays, main = "OrchardSprays data")
```

---

order — *Ordering Permutation* — order

Description

`order` returns a permutation which rearranges its first argument into ascending order, breaking ties by further arguments.

`sort.list` is the same, using only one argument put allowing partial sorting.

Usage

```
order(...)
sort.list(x, partial)
```

Arguments

`...`	a sequence of vectors, all of the same length.
`x`	a vector.
`partial`	vector of indices for partial sorting.

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).

`NA` values are treated as greater than any other values so that permutations returned by `order` move `NA` values to the top end of the array.

`partial` is supplied for compatibility with S, but the sorting is always complete.

See Also

`sort` and `rank`.

Examples

```
(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)

## 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])
```

`outer` *Outer Product of Arrays* `outer`

Description

The outer product of the arrays `X` and `Y` is the array `A` with dimension `c(dim(X), dim(Y))` where element `A[i, j, .., k, l, ..] = FUN(X[i, j, ..], Y[k, l,..], ..)`.

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

See Also

`matmult` for usual (*inner*) matrix vector multiplication; `kronecker` which is based on `outer`.

Examples

```
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]
```

`p.adjust` *Adjust p values for multiple comparisons* `p.adjust`

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", "bonferroni","none")
```

Arguments

`p`	vector of p values
`method`	correction method
`n`	number of comparisons

Details

The adjustment methods include the Bonferroni correction in which the p values are multiplied by the number of comparisons. Two less conservative corrections by Holm, respectively Hochberg, are also included. 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`.

Value

A vector of corrected p values.

Note

The Hochberg method is only proved to work if the p values are independent, although simulations have indicated that it works in correlated cases as well. Hence the Holm method is the default.

References

S Paul Wright: Adjusted P-values for simultaneous inference, Biometrics **48**, 1005–1013

See Also

`pairwise.*` functions in the `ctest` package, such as `pairwise.t.test`.

Examples

```
p <- runif(50)
p.adjust(p)
```

`package.contents` *Package Contents and Description* `package.contents`

Description

Parses and returns the 'CONTENTS' and 'DESCRIPTION' file of a package.

Usage

```
package.contents(pkg, lib = .lib.loc)
package.description(pkg, lib = .lib.loc, fields = NULL)
```

Arguments

`pkg`	A character string with the package name.
`lib`	A character vector with path names of R package libraries.
`fields`	A character vector of fields to return (if other fields occur in the file they are ignored.

See Also

`parse.dcf`

Examples

```
package.contents("mva")
```

`package.dependencies` *Check Packahe Dependencies* `package.dependencies`

Description

Parses and checks the dependencies of a package against the currently installed version of R [and other packages].

Usage

```
package.dependencies(x, check=FALSE)
```

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.

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

---

page *Invoke a Pager on an R Object* page

Description

Displays the object named by x in a pager

Usage

```
page(x)
```

Arguments

x	the name of an R object.

Author(s)

B. D. Ripley

See Also

file.show, edit, fix.

---

pairs *Scatterplot Matrices* pairs

Description

A matrix of scatterplots is produced.

Usage

```
pairs(x, ...)
pairs.default(x, labels = colnames(x), 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,
              row1attop = TRUE)
```

Arguments

`x`	the coordinates of points given as columns of a matrix.
`labels`	the names of the variables.
`panel`	`function(x,y,...)` which is used to plot the contents of each panel of the display.
`...`	graphical parameters can be given as arguments to `plot`.
`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.
`row1attop`	logical. Should the layout be matrix-like with row 1 at the top, or graphi-like with row 1 at the bottom?

Details

The ijth 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.

Author(s)

Enhancements for R 1.0.0 contributed by Dr. Jens Oehlschlaegel-Akiyoshi and R-core members.

Examples

```
data(USJudgeRatings)
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,
      diag.panel=panel.hist, cex.labels=1.5, 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)
```

`pairs.formula` *Formula Notation for Scatterplot Matrices* `pairs.formula`

Description

Produce a matrix of scatterplots using formula notation.

Usage

```
pairs.formula(formula, data = NULL, subset, na.action, ...)
```

Arguments

`formula`	a formula, such as `y ~ x`.
`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 in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s.
`...`	graphical parameters may also be passed as arguments, see `par`.

Details

This is a method of the generic function `pairs`. It operates by setting up the data from the formula specification, and then calling `pairs.default`.

See Also

`pairs.default`

Examples

```
data(swiss)
pairs(~ Fertility + Education + Catholic,
      data = swiss, subset = Education < 20)
```

`palette` *Set or View the Graphics Palette* `palette`

Description

View or manipulate the color palette which is used when a `col=` has a numeric index.

Usage

```
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.

Examples

```
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` *Color Palettes* `Palettes`

Description

These functions create a vector of `n` "contiguous" colors.

Usage

```
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 `hsv(.., gamma)`.

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`.

See Also

`colors`, `palette`, `hsv`, `rgb`, `gray`.

Examples

```
# A Color Wheel
piechart(rep(1,12), col=rainbow(12))

##------ Some palettes ------------
ch.col <- c("rainbow(n, start=.7, end=.1)", "heat.colors(n)",
            "terrain.colors(n)", "topo.colors(n)", "cm.colors(n)")

n <- if(.Device == "postscript") 64 else 16
     # Since for screen, larger n may give color allocation problem
nt <- length(ch.col)
i <- 1:n; j <- n / nt; d <- j/6; dy <- 2*d
plot(i,i+d, type="n", yaxt="n", ylab="", main=paste("color palettes;  n=",n))
for (k in 1:nt) {
  rect(i-.5,(k-1)*j+ dy, i+.4,  k*j, col=eval(parse(text=ch.col[k])))
  text(2*j,  k * j +dy/4, ch.col[k])
}
```

`panel.smooth` *Simple Panel Plot* `panel.smooth`

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, pch, span=2/3, iter=3, ...)
```

Arguments

`x,y`	numeric vectors of the same length
`col,pch`	numeric or character codes for the color and point type; see `par`
`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

```
data(swiss)
pairs(swiss, panel = panel.smooth, pch = ".")# emphasize the smooths
```

`par` *Set or Query Graphical Parameters* `par`

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

```
par(..., no.readonly = FALSE)

<highlevel plot> (..., <tag> = <value>)
```

Arguments

`no.readonly`	logical; if `TRUE` and there are no other arguments, only parameters are returned which can be set by a subsequent `par(.)` call.
`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. Note that the `adj` argument of `text` and `mtext` 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.
`bty`	A character string which determined the type of box which is drawn about plots. If `bty` is one of `"o"`, `"l"`, `"7"`, `"c"`, or `"]"` 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.
`cex.axis`	The magnification to be used for axis annotation relative to the current.
`cex.lab`	The magnification to be used for x and y labels relative to the current.
`cex.main`	The magnification to be used for main titles relative to the current.
`cex.sub`	The magnification to be used for sub-titles relative to the current.
`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.***. The height of (default sized) characters in inches.

`cxy` ***R.O.***; size of default character `(width,height)` in user coordinate units. `par("cxy")` is `par("cin")/par("pin")` scaled to user coordinates. Note that `c(strwidth(ch),strwidth(ch))` for a given string `ch` is usually much more precise.

`din` ***R.O.***. The device dimensions in inches.

`err` (*Unimplemented*; R is silent when points outside the plot region are *not* plotted.) The degree of error reporting desired.

`fg` The color to be used for the foreground of plots. This is the default color is 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.

`fin` A numerical vector of the form `c(x, y)` which gives the size of the figure region in inches.

`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.

`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 `hsv(.., gamma)`.

`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. *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,

3: always vertical.

Note that other string/character rotation (via `par(srt = ..)`) does *not* affect the axis labels.

`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(0:9, "A":"F")`) may be given, giving the length (typically in points/pixels) of line segments which are alternatively drawn and skipped. For example, `"44"` is dashed and `"13"` is dotted.

`lwd` The line width, a *positive* numerical, defaulting to `1`.

`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 lines of margin to be specified on the four sides of the plot. The default is `c(5, 4, 4, 2) + 0.1`.

`mex` `mex` is a character size expansion factor which is used to describe coordinates in the margins of plots.

`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.

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 `nc` 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**.

`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.

`pin` The width and height of the current plot 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 arc should be.

`srt` The string rotation in degrees.

`tck` The length of tick marks as a fraction of the smaller of the width or height of the plotting region. If `tck=1`, grid lines are drawn. The default setting 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`.

`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.

`xaxp` 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.

`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 `"r"` 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.

`xlog` ***R.O.***. A logical value (see `log` in `plot.default`). If `TRUE` a logarithmic scale is in use. 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.

`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` ***R.O.***. A logical value (see `log` in `plot.default`). See `xlog` above.

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 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. Arguments := **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 arguments** can be set as well in high-level and mid-level plot functions, such as `plot`, `points`, `lines`, `axis`, `title`, `text`, `mtext`:

- `"ask"`

- "fig", "fin"
- "mai", "mar", "mex"
- "mfrow", "mfcol", "mfg"
- "new"
- "oma", "omd", "omi"
- "pin", "plt", "ps", "pty"
- "usr"
- "xlog", "ylog"

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 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.

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 there 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. This provides compatibility with S.

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, 3=dotted) or directly as the lengths of on/off stretches of line. This is done with a string of up to eight characters which give the lengths in consecutive positions in the string. For example, the string `"33"` specifies three pixels on followed by three off and `"3313"` specifies three pixels on followed by three off followed by one on and finally three off.

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

```
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

( 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

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()
```

Paren *Parentheses and Braces* Paren

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

```
( ... )

{ ... }
```

See Also

`if`, `return`, etc for other objects used in the R language itself.

Examples

```
f <- get("(")
e <- expression(3 + 2 * 4)
f(e) == e           # TRUE

do <- get("{")
do(x_ 3, y_ 2*x-3, 6-x-y); x; y
```

parse *Parse Expressions* parse

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 = NULL, white = FALSE)
```

Arguments

file	the name of a file to read the expressions from. Note that the ASCII file *must* end with a newline (" n"). If file is "" and text is missing or NULL then input is taken from the keyboard.
n	the number of statements to parse. If n is negative the file is parsed in its entirety.
text	character. The text to parse, quoted.
prompt	the prompt to print when parsing from the keyboard. The default, NULL, is to use R's prompt, options("prompt")[[1]].
white	if TRUE then any white space separates expressions otherwise only newlines or semicolons do.

See Also

scan, source, eval, deparse.

Examples

```
cat("x <- c(1,4)\n  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")
```

parse.dcf *Parse Debian Control File Format* parse.dcf

Description

Parses text read from a file in Debian control file format, e.g., the 'DESCRIPTION' or 'CONTENTS' of a package.

Usage

```
parse.dcf(desc = NULL, file = "", fields = NULL, versionfix = FALSE)
```

Arguments

`text`	A character vector containing one line of the file in each element.
`file`	Name of the file to be parsed.
`fields`	A character vector of fields to return (if other fields occur in the file they are ignored.
`versionfix`	Logical, if `TRUE` than the version field is truncated at the first whitespace character.

Value

If `fields = NULL`, a list with one element per entry is returned. Each entry is a list of character vectors, one per field. If `fields` are specified, then a character matrix with one row per entry and one column per field is returned.

See Also

`package.contents`, `library`

`paste` *Concatenate Strings* `paste`

Description

Concatenate character vectors.

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.

See Also

`as.character`, `substr`, `nchar`, `strsplit`.

Examples

```
paste(1:12) # same as as.character(..)
paste("A", 1:6, sep = "")
paste("Today is", date())
```

`persp` *Perspective Plots* `persp`

Description

This function draws perspective plots of surfaces over the x–y plane.

Usage

```
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),
      xlab = NULL, ylab = NULL, zlab = NULL,
      theta = 0, phi = 15, r = sqrt(3), d = 1,
      scale = TRUE, expand = 1,
      col = NULL, 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.
`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 of the surface facets.
`border`	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.
`ltheta, lphi`	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`.
`shade`	the shade at a surface facet is computed as `((1+d)/2)^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.
`box`	should the bounding box for the surface be displayed. The default is `TRUE`.
`axes`	should ticks and labels be added to the box. The default is `TRUE`. If `box` is `FALSE` then no ticks or labels are drawn.
`ticktype`	character: "simple" draws just an arrow parallel to the axis to indicate direction of increase; "detailed" draws normal ticks as per 2D plots.
`nticks`	the (approximate) number of tick marks to draw on the axes. Has no effect if `ticktype` is "simple".
`...`	additional graphical parameters (see `par`) and the arguments to `title` may also be supplied.

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.

See Also

`contour` and `image`.

Examples

```
# (1) The Obligatory Mathematical surface.
#     Rotated sinc function.

x <- seq(-10, 10, length=50)
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
par(bg = "white")
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue",
      xlab = "X", ylab = "Y", zlab = "Z")
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 = "Z")

# (2) Visualizing a simple DEM model

data(volcano)
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)
persp(x, y, z, theta = 120, phi = 15, scale = FALSE, axes = FALSE)

# (3) Now something more complex
#     We border the surface, to make it more "slice like"
#     and color the top and sides of the surface differently.

zmin <- min(z) - 20
z <- rbind(zmin, cbind(zmin, z, zmin), zmin)
x <- c(min(x) - 1e-10, x, max(x) + 1e-10)
y <- c(min(y) - 1e-10, y, max(y) + 1e-10)

fill <- matrix("green3", nr = nrow(z)-1, nc = ncol(z)-1)
fill[,1] <- "gray"
fill[,ncol(fill)] <- "gray"
fill[1,] <- "gray"
fill[nrow(fill),] <- "gray"

par(bg = "lightblue")
persp(x, y, z, theta = 120, phi = 15, col = fill, scale = FALSE, axes = FALSE)
title(main = "Maunga Whau\nOne of 50 Volcanoes in the Auckland Region.",
      font.main = 4)

par(bg = "slategray")
persp(x, y, z, theta = 135, phi = 30, col = fill, scale = FALSE,
      ltheta = -120, lphi = 15, shade = 0.65, axes = FALSE)
persp(x, y, z, theta = 135, phi = 30, col = "green3", scale = FALSE,
      ltheta = -120, shade = 0.75, border = NA, box = FALSE)
```

---

phones *The World's Telephones* phones

Description

The number of telephones in various regions of the world (in thousands).

Usage

```
data(phones)
```

Format

A matrix with 7 rows and 8 columns.

Details

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.

The years are: 1951, 1956, 1957, 1958, 1959, 1960, 1961.

Source

AT&T (1961) *The World's Telephones*.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(phones)
matplot(rownames(phones), phones, type = "b", log = "y",
        xlab = "Year", ylab = "Number of telephones (1000's)")
legend(1951.5, 80000, colnames(phones), col = 1:7, lty = 1:7, pch = rep(21, 7))
title(main = "phones data: log scale for response")
```

---

pictex — *A PicTeX Graphics Driver* — pictex

Description

This function produces graphics suitable for inclusion in TeX and LaTeX documents.

Usage

```
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.
`fg`	the foreground color for the plot.

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

Knuth, D. E. (1984) *The TeXbook.* Reading, MA: Addison-Wesley.

Lamport, L. (1994) *LATEX: A Document Preparation System.* Reading, MA: Addison-Wesley.

Goossens, M., Mittelbach, F. and Samarin, A. (1994) *The LATEX Companion.* Reading, MA: Addison-Wesley.

See Also

postscript, Devices.

Examples

```
pictex()
plot(1:11,(-5:5)^2, type='b', main="Simple Example Plot")
dev.off()
##--------------------

%% LaTeX Example
\documentclass{article}
\usepackage{pictex}
\begin{document}
%...
\begin{figure}[h]
  \centerline{\input{Rplots.tex}}
  \caption{}
\end{figure}
%...
\end{document}

%%-- TeX Example --
\input pictex
$$ \input Rplots.tex $$
```

---

piechart *Pie Charts* piechart

Description

Draw a pie chart.

Usage

```
piechart(x, labels=names(x), shadow=FALSE,
        edges=200, radius=0.8, fill=NULL, main=NULL, ...)
```

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.

`shadow`	a logical vector indicating whether a shadow effect should be attempted for the chart. This only makes sense if the slices are filled with colors.
`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.
`col`	a vector of colors to be used in filling the slices.
`main`	an overall title for the plot.
`...`	graphical parameters can be given as arguments to `piechart`.

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.

See Also

`dotplot`.

Examples

```
piechart(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")
piechart(pie.sales,
    col=c("purple", "violetred1", "green3",
    "cornsilk", "cyan", "white"))
piechart(pie.sales,
    col=gray(seq(0.4,1.0,length=6)))
```

`PlantGrowth` *Results from an Experiment on Plant Growth* `PlantGrowth`

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

```
data(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

Dobson, A. J. (1983) *An Introduction to Statistical Modelling.* London: Chapman and Hall.

Examples

```
## One factor ANOVA example from Dobson's book, cf. Table 7.4:
data(PlantGrowth)
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))
```

`Platform` *Platform Specific Variables* `Platform`

Description

`.Platform` is a list with functions and variables as components. This provides a possibility to write OS portable R code. However, this interface is still somewhat experimental.

Currently, `.Platform <- Platform()` when R starts up. This is even more experimental.

Usage

```
.Platform
Platform()
```

Value

`.Platform` is list with at least the following components:

`OS.type`	character, giving the **O**perating **S**ystem (family) of the computer. One of the following values is returned: `"unix"`, `"mac"`, or `"windows"` (in historical order).
`file.sep`	character, giving the **file sep**arator, used on your platform, e.g., `"/"` on Unix alikes.
`dynlib.ext`	character, giving the file name **ext**ension of **dyn**amically loadable **lib**raries, e.g., `".dll"` on Windows.

See Also

`system` for invoking platform specific system commands.

Examples

```
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
}
```

`plot` *Generic X-Y Plotting* `plot`

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see `par`.

Usage

```
plot(x, ...)
plot(x, y, xlim=range(x), ylim=range(y), type="p",
        main, xlab, ylab, ...)
plot(y ~ x, ...)
```

Arguments

`x`	the coordinates of points in the plot. Alternatively, a single plotting structure or *any* R *object with a* ***`plot`*** *method* can be provided.
`y`	the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.
`xlim, ylim`	the ranges to be encompassed by the x and y axes.
`type`	what type of plot should be drawn. Possible types are • `"p"` for **p**oints, • `"l"` for **l**ines, • `"b"` for **b**oth, • `"o"` for both "**o**verplotted", • `"h"` for "**h**istogram" like vertical lines, • `"s"` for **s**teps, • `"S"` for other **s**teps, • `"n"` for no plotting.
`main`	an overall title for the plot.
`xlab`	a title for the x axis.
`ylab`	a title for the y axis.
`...`	graphical parameters can be given as arguments to `plot`.

Details

For simple scatter plots, `plot.default` will be used. However, there are `plot` methods for many R objects, including `function`s, `data.frame`s, `density` objects, etc. Use `methods(plot)` and the documentation for these.

See Also

`plot.default`, `plot.formula` and other methods; `points`, `lines`, `par`.

Examples

```
data(cars)
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)")
```

`plot.default` *The Default Scatterplot Function* `plot.default`

Description

Draw a scatter plot with "decorations" such as axes and titles in the active graphics window.

Usage

```
plot.default(x, y = NULL, type = "p",  xlim = NULL, ylim = NULL,
             log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
             ann = par("ann"), axes = TRUE, frame.plot = axes,
             panel.first = NULL, panel.last = NULL,
             col = par("fg"), bg = NA, pch = par("pch"),
             cex = par("cex"), lty = par("lty"), lwd = par("lwd"),
             asp = NA, ...)
```

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 `xy.coords` for details.
`type`	the type of plot desired. The following possibilities are possible; `"p"` for points, `"l"` for lines, `"o"` for overplotted points and lines, `"b"` or `c` for points joined by lines, `"s"` and `"S"` for step functions and `"h"` for *high density* vertical lines.
`xlim`	the x limits (min,max) of the plot.
`ylim`	the y limits of the plot.
`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.
`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.
`ann`	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
`axes`	a logical value indicating whether axes should be drawn on the plot.

`frame.plot`	a logical indicating whether a box should be drawn around the plot.
`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.
`panel.last`	an expression to be evaluated after plotting has taken place.
`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.
`bg`	background color of ???
`pch`	a vector of plotting characters or symbols.
`cex`	a numerical value giving the amount by which plotting text and symbols should be scaled relative to the default
`lty`	the line type, see `par`.
`lwd`	the line width **not yet supported for postscript**.
`asp`	the y/x aspect ratio, see `plot.window`.
`...`	graphical parameters as in `par(...)` may also be passed as arguments.

References

Cleveland, W. S. (1985) *The Elements of Graphing Data.* Monterey, CA: Wadsworth.

See Also

`plot`, `plot.window`, `xy.coords`.

Examples

```
data(cars)
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")

##--- 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.factor *Plotting Factor Variables* plot.factor

Description

This functions implements a "scatterplot" method for `factor` arguments of the *generic* `plot` function. Actually, `boxplot` or `barplot` are used when appropriate.

Usage

```
plot.factor(x, y, legend.text = levels(y), ...)
```

Arguments

`x,y`	numeric or factor. `y` may be missing.
`legend.text`	a vector of text used to construct a legend for the plot. Only used if `y` is present and a factor.
`...`	Further arguments to `plot`, see also `par`.

See Also

`plot.default`, `plot.formula`, `barplot`, `boxplot`.

Examples

```
data(PlantGrowth)
plot(PlantGrowth)                                 # -> plot.data.frame
plot(weight ~ group, data = PlantGrowth)          # numeric vector ~ factor
plot(cut(weight, 2) ~ group, data = PlantGrowth) # factor ~ factor

plot(PlantGrowth$group, axes=FALSE, main="no axes")# extremly silly
```

plot.formula *Formula Notation for Scatterplots* plot.formula

Description

Specify a scatterplot or add points or lines via a formula.

Usage

```
plot.formula(formula, ..., data = parent.frame(), subset,
             ylab = varnames[response], ask = TRUE)
points.formula(formula, ..., data = parent.frame(), subset)
lines.formula(formula, ..., data = parent.frame(), subset)
plot(y ~ x, ...)
points(y ~ x, ...)
lines(y ~ x, ...)
```

Arguments

`formula`	a `formula`, such as `y ~ x`.
`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 in the fitting process.
`ylab`	the y label of the plot(s)
`...`	Further graphical parameters may also be passed as arguments, see `par`
`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)`.

Value

These functions are invoked for their side effect of drawing in the active graphics device.

See Also

`plot.default`, `plot.factor`.

Examples

```
data(airquality)
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)
```

`plot.lm` *Plot Diagnostics for an lm Object* `plot.lm`

Description

Four plots (choosable by `which`) are currently provided: a plot of residuals against fitted values, a Scale-Location plot of $\sqrt{|residuals|}$ against fitted values, a Normal Q-Q plot, and a plot of Cook's distances versus row labels.

Usage

```
plot.lm(x, which = 1:4,
        caption = c("Residuals vs Fitted", "Normal Q-Q plot",
                    "Scale-Location plot", "Cook's distance plot"),
        panel = points,
        sub.caption = deparse(x$call), main = "",
        ask = interactive() && one.fig && .Device != "postscript",
        ...,
        id.n = 3, labels.id = names(residuals(x)), cex.id = 0.25)
```

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 `1:4`.
`caption`	Captions to appear above the plots
`panel`	Panel function. A useful alternative to `points` is `panel.smooth`.
`sub.caption`	common title—above figures if there are multiple; used as `sub` (s.`title`) otherwise.
`main`	title to each plot—in addition to the above `caption`.
`ask`	logical; if `TRUE`, the user is *ask*ed before each plot, see `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. `NULL` uses observation numbers.
`cex.id`	magnification of point labels.

Details

`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).

This 'S-L' and the Q-Q 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, `lm.influence()$hat`, see also `hat`.

Author(s)

John Maindonald and Martin Maechler.

References

Belsley, D. A., Kuh, E. and Welsch, R. E. (1980) *Regression Diagnostics.* New York: Wiley.

Cook, R. D. and Weisberg, S. (1982) *Residuals and Influence in Regression.* London: Chapman and Hall.

Hinkley, D. V. (1975) On power transformations to symmetry. *Biometrika* **62**, 101–111.

McCullagh, P. and Nelder, J. A. (1989) *Generalized Linear Models.* London: Chapman and Hall.

See Also

`lm.influence`, `cooks.distance`

Examples

```
## Analysis of the life-cycle savings data
## given in Belsley, Kuh and Welsch.
data(LifeCycleSavings)
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

## Fit a smmooth 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))
## Warnings:  panel = panel.smoth, 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.window` *Set up World Coordinates for Graphics Window* `plot.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

```
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 **asp**ect 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` × 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.

Usually, one should rather use the higher level functions such as `plot`, `hist`, `image`, ..., instead and refer to their help pages for explanation of the arguments.

See Also

`xy.coords`, `plot.xy`, `plot.default`.

Examples

```
##--- An example for the use of  'asp' :
library(mva)
data(eurodist)
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=0.5)
```

`plot.xy` *Basic Internal Plot Function* `plot.xy`

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

```
plot.xy(xy, type, pch=1, lty="solid", col=par("fg"), bg=NA, cex=1, ...)
```

Arguments

`xy`	A four-element list as results from `xy.coords(..)`.
`type`	1 character code.
`pch`	character or integer code for kind of points/lines, see `points.default`.
`lty`	line type code, see `lines`.
`col`	color code or name, see `colors`, `palette`.
`bg`	background ("fill") color for open plot symbols.
`cex`	character expansion
`...`	further graphical parameters

See Also

`plot`, `plot.default`, `points`, `lines`.

Examples

```
points.default # to see how it calls "plot.xy(xy.coords(x, y), ....)"
```

`plotmath` *Mathematical Annotation in R* `plotmath`

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.

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 `example(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:

Syntax	**Meaning**
`x + y`	x plus y
`x - y`	x minus y
`x*y`	juxtapose x and y
`x/y`	x forwardslash y
`x %+-% y`	x plus or minus y
`x %/% y`	x divided by y
`x %*% y`	x times y
`x[i]`	x subscript i
`x^2`	x superscript 2
`paste(x, y, z)`	juxtapose x, y, and z
`sqrt(x)`	square root of x
`sqrt(x, y)`	yth root of x
`x == y`	x equals y
`x != y`	x is not equal to y
`x < y`	x is less than y
`x <= y`	x is less than or equal to y
`x > y`	x is greater than y
`x >= y`	x is greater than or equal to y
`x %~~% y`	x is approximately equal to y
`x %=~% y`	x and y are congruent
`x %==% y`	x is defined as y
`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 %subseteq% y`	x is a subset of y
`x %notsubset% y`	x is not a subset of y
`x %supset% y`	x is a proper superset of y
`x %supseteq% y`	x is 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
`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
`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
`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
`x^{y + z}`	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

Murrell, P. and Ihaka, R. (2000) An approach to providing mathematical annotation in plots. *Journal of Computational and Graphical Statistics* (In press).

See Also

`axis`, `mtext`, `text`, `title`

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))

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)

######
# create tables of mathematical annotation functionality
######
make.table <- function(nr, nc) {
    savepar <- par(mar=rep(0, 4), pty="s")
    plot(c(0, nc*2 + 1), c(0, -(nr + 1)),
         type="n", xlab="", ylab="", axes=F)
    savepar
}

get.r <- function(i, nr) {
    i %% nr + 1
}
```

```
get.c <- function(i, nr) {
    i %/% nr + 1
}

draw.title.cell <- function(title, i, nr) {
    r <- get.r(i, nr)
    c <- get.c(i, nr)
    text(2*c - .5, -r, title)
    rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5))
}

draw.plotmath.cell <- function(expr, i, nr, string = NULL) {
    r <- get.r(i, nr)
    c <- get.c(i, nr)
    if (is.null(string)) {
        string <- deparse(expr)
        string <- substr(string, 12, nchar(string) - 1)
    }
    text((2*(c - 1) + 1), -r, string, col="grey")
    text((2*c), -r, expr, adj=c(.5,.5))
    rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5), border="grey")
}

nr <- 20
nc <- 2
oldpar <- make.table(nr, nc)
i <- 0
draw.title.cell("Arithmetic Operators", i, nr); i <- i + 1
draw.plotmath.cell(expression(x + y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x - y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x * y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x / y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %+-% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %/% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %*% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(-x), i, nr); i <- i + 1
draw.plotmath.cell(expression(+x), i, nr); i <- i + 1
draw.title.cell("Sub/Superscripts", i, nr); i <- i + 1
draw.plotmath.cell(expression(x[i]), i, nr); i <- i + 1
draw.plotmath.cell(expression(x^2), i, nr); i <- i + 1
draw.title.cell("Juxtaposition", i, nr); i <- i + 1
draw.plotmath.cell(expression(x * y), i, nr); i <- i + 1
draw.plotmath.cell(expression(paste(x, y, z)), i, nr); i <- i + 1
draw.title.cell("Lists", i, nr); i <- i + 1
draw.plotmath.cell(expression(list(x, y, z)), i, nr); i <- i + 1
# even columns up
i <- 20
draw.title.cell("Radicals", i, nr); i <- i + 1
draw.plotmath.cell(expression(sqrt(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(sqrt(x, y)), i, nr); i <- i + 1
draw.title.cell("Relations", i, nr); i <- i + 1
draw.plotmath.cell(expression(x == y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x != y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x < y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x <= y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x > y), i, nr); i <- i + 1
```

```
draw.plotmath.cell(expression(x >= y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %~~% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %=~% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %==% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %prop% y), i, nr); i <- i + 1
draw.title.cell("Typeface", i, nr); i <- i + 1
draw.plotmath.cell(expression(plain(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(italic(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(bold(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(bolditalic(x)), i, nr); i <- i + 1

# Need fewer, wider columns for ellipsis ...
nr <- 20
nc <- 2
make.table(nr, nc)
i <- 0
draw.title.cell("Ellipsis", i, nr); i <- i + 1
draw.plotmath.cell(expression(list(x[1], ..., x[n])), i, nr); i <- i + 1
draw.plotmath.cell(expression(x[1] + ... + x[n]), i, nr); i <- i + 1
draw.plotmath.cell(expression(list(x[1], cdots, x[n])), i, nr); i <- i + 1
draw.plotmath.cell(expression(x[1] + ldots + x[n]), i, nr); i <- i + 1
draw.title.cell("Set Relations", i, nr); i <- i + 1
draw.plotmath.cell(expression(x %subset% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %subseteq% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %supset% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %supseteq% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %notsubset% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %in% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %notin% y), i, nr); i <- i + 1
draw.title.cell("Accents", i, nr); i <- i + 1
draw.plotmath.cell(expression(hat(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(tilde(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(ring(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(bar(xy)), i, nr); i <- i + 1
draw.plotmath.cell(expression(widehat(xy)), i, nr); i <- i + 1
draw.plotmath.cell(expression(widetilde(xy)), i, nr); i <- i + 1
draw.title.cell("Arrows", i, nr); i <- i + 1
draw.plotmath.cell(expression(x %<->% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %->% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %<-% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %up% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %down% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %<=>% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %=>% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %<=% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %dblup% y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x %dbldown% y), i, nr); i <- i + 1
draw.title.cell("Symbolic Names", i, nr); i <- i + 1
draw.plotmath.cell(expression(Alpha - Omega), i, nr); i <- i + 1
draw.plotmath.cell(expression(alpha - omega), i, nr); i <- i + 1
draw.plotmath.cell(expression(infinity), i, nr); i <- i + 1
draw.plotmath.cell(expression(32 * degree), i, nr); i <- i + 1
draw.plotmath.cell(expression(60 * minute), i, nr); i <- i + 1
draw.plotmath.cell(expression(30 * second), i, nr); i <- i + 1

# Need even fewer, wider columns for typeface and style ...
nr <- 20
```

```
nc <- 1
make.table(nr, nc)
i <- 0
draw.title.cell("Style", i, nr); i <- i + 1
draw.plotmath.cell(expression(displaystyle(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(textstyle(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(scriptstyle(x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(scriptscriptstyle(x)), i, nr); i <- i + 1
draw.title.cell("Spacing", i, nr); i <- i + 1
draw.plotmath.cell(expression(x ~~ y), i, nr); i <- i + 1

# Need fewer, taller rows for fractions ...
# cheat a bit to save pages
par(new =T)
nr <- 10
nc <- 1
make.table(nr, nc)
i <- 4
draw.plotmath.cell(expression(x + phantom(0) + y), i, nr); i <- i + 1
draw.plotmath.cell(expression(x + over(1, phantom(0))), i, nr); i <- i + 1
draw.title.cell("Fractions", i, nr); i <- i + 1
draw.plotmath.cell(expression(frac(x, y)), i, nr); i <- i + 1
draw.plotmath.cell(expression(over(x, y)), i, nr); i <- i + 1
draw.plotmath.cell(expression(atop(x, y)), i, nr); i <- i + 1

# Need fewer, taller rows and fewer, wider columns for big operators ...
nr <- 10
nc <- 1
make.table(nr, nc)
i <- 0
draw.title.cell("Big Operators", i, nr); i <- i + 1
draw.plotmath.cell(expression(sum(x[i], i=1, n)), i, nr); i <- i + 1
draw.plotmath.cell(expression(prod(plain(P)(X == x), x)), i, nr); i <- i + 1
draw.plotmath.cell(expression(integral(f(x) * dx, a, b)), i, nr); i <- i + 1
draw.plotmath.cell(expression(union(A[i], i==1, n)), i, nr); i <- i + 1
draw.plotmath.cell(expression(intersect(A[i], i==1, n)), i, nr); i <- i + 1
draw.plotmath.cell(expression(lim(f(x), x %->% 0)), i, nr); i <- i + 1
draw.plotmath.cell(expression(min(g(x), x >= 0)), i, nr); i <- i + 1
draw.plotmath.cell(expression(inf(S)), i, nr); i <- i + 1
draw.plotmath.cell(expression(sup(S)), i, nr); i <- i + 1

make.table(nr, nc)
i <- 0
draw.title.cell("Grouping", i, nr); i <- i + 1
draw.plotmath.cell(expression((x + y)*z), i, nr); i <- i + 1
draw.plotmath.cell(expression(x^y + z), i, nr); i <- i + 1
draw.plotmath.cell(expression(x^(y + z)), i, nr); i <- i + 1
# have to do this one by hand
draw.plotmath.cell(expression(x^{y + z}), i, nr, string="x^{y + z}"); i <- i + 1
draw.plotmath.cell(expression(group("(", list(a, b), "]")), i, nr); i <- i + 1
draw.plotmath.cell(expression(bgroup("(", atop(x, y), ")")), i, nr); i <- i + 1
draw.plotmath.cell(expression(group(lceil, x, rceil)), i, nr); i <- i + 1
draw.plotmath.cell(expression(group(lfloor, x, rfloor)), i, nr); i <- i + 1
draw.plotmath.cell(expression(group("|", x, "|")), i, nr); i <- i + 1

par(oldpar)
```

`pmatch` *Partial String Matching* `pmatch`

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.
`table`	the values to be matched against.
`nomatch`	the value returned at non-matching or mutliply 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. 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 differentiates 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`.

Note

Versions of R prior to 1.0.0 had a different behaviour that was seriously incompatible with S (and the current version) when `duplicates.ok = TRUE`.

Author(s)

Of this version, B. D. Ripley.

See Also

match, charmatch and match.arg, match.fun, match.call, for function argument matching etc.

Examples

```
pmatch("", "")                             # returns NA
pmatch("m",   c("mean", "median", "mode")) # returns NA
pmatch("med", c("mean", "median", "mode")) # returns 2

pmatch(c("", "ab", "ab"), c("abc", "ab"), dup=F)
pmatch(c("", "ab", "ab"), c("abc", "ab"), dup=T)
## compare
charmatch(c("", "ab", "ab"), c("abc", "ab"))
```

---

points *Add Points to a Plot* points

Description

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, ...)
points.default(x, y=NULL, type="p", pch=1, col="black", 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(..).
pch	plotting 'character', i.e. symbol to use. pch can either be a character or an integer code for a set of graphics symbols. The full set of S symbols is available with pch=0:18. In addition, there is a special set of R plotting symbols which can be obtained with pch=19:25 and can be colored and filled with different colors: • pch=19: solid circle, • pch=20: bullet, • pch=21: circle, • pch=22: square, • pch=23: diamond, • pch=24: triangle point-up, • pch=25: triangle point down.
col	color code or name, see colors, palette.
bg	background ("fill") color for open plot symbols
cex	character expansion
...	Further graphical parameters (see plot.xy and par) may also be supplied as arguments.

Details

Graphical parameters are permitted as arguments to this function.

See Also

`plot`, `lines`, and the underlying "primitive" `plot.xy`.

Examples

```
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)

##-------- Showing all the extra & some char graphics symbols ------------
Pex <- 3 ## now (0.90) 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 = F, 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)
}
```

`Poisson` *The Poisson Distribution* `Poisson`

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 $x = 0, 1, 2, \ldots$.

If an element of `x` is not integer, the result of `dpois` is zero, with a warning.

The quantile is left continuous: `qgeom(q, 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, lambda=1) * gamma(1+ 0:7)) # == 1
Ni <- rpois(50, lam= 4); table(factor(Ni, 0:max(Ni)))

1 - ppois(10*(15:25), lambda=100)               # becomes 0 (cancellation)
    ppois(10*(15:25), lambda=100, lower=FALSE)  # no cancellation

par(mfrow = c(2, 1))
x <- seq(-0.01, 5, 0.01)
plot(x, ppois(x, 1), type="s", ylab="F(x)", main="Poisson(1) CDF")
plot(x, pbinom(x, 100, 0.01),type="s", ylab="F(x)",
     main="Binomial(100, 0.01) CDF")
```

---

poly — *Compute Orthogonal Polynomials* — poly

Description

Returns orthogonal polynomials of degree 1 to `degree` over the specified set of points `x`. These are all orthogonal to the constant polynomial of degree 0.

Usage

```
poly(x, degree=1)
```

Arguments

`x`	a numeric vector at which to evaluate the polynomial
`degree`	the degree of the polynomial

Value

A matrix with rows corresponding to points in `x` and columns corresponding to the degree, with attributes `"degree"` specifying the degrees of the columns and `"coefs"` which contains the centring and normalization constants used in constructing the orthogonal polynomials.

Note

This routine is intended for statistical purposes such as `contr.poly`: it does not attempt to orthogonalize to machine accuracy.

Author(s)

B.D. Ripley

See Also

`contr.poly`

Examples

```
poly(1:10, 3)
```

`polygon` *Polygon Drawing* `polygon`

Description

`polygon` draws the polygons whose vertices are given in `x` and `y`.

Arguments

`x,y`	vectors containing the coordinates of the vertices of the polygon.
`density`	density of fill pattern. *NOT YET* implemented.
`angle`	angle of fill pattern. *NOT YET* implemented.
`border`	the color to draw the border. Defaults to `par("fg")`. Use `border=0` to omit borders.
`...`	graphical parameters can be given as arguments to `polygon`, must importantly,
`col`	an integer specifying the color to be used in filling the polygon. The default is to leave polygons unfilled.

See Also

`segments` for even more flexibility, `lines`, `rect`, `box`, `abline`.

Examples

```
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")
```

`polyroot` *Find Zeros of a Real or Complex Polynomial* `polyroot`

Description

Find zeros of a real or complex polynomial.

Usage

```
polyroot(z)
```

Arguments

`z`	the vector of polynomial coefficients in decreasing 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.

Value

A complex vector of length $n-1$, where n is `length(z)`.

References

Jenkins and Traub (1972) TOMS Algorithm 419. *Comm. ACM*, **15**, 97–99.

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)
```

`pos.to.env` *Convert Positions in the Search Path to Environments* `pos.to.env`

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)
pos.to.env(length(search()))
```

`postscript` *PostScript Graphics* `postscript`

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

```
postscript(file = "Rplots.ps", ...)
ps.options(..., reset = FALSE, override.check = FALSE)
.PostScript.Options
```

Arguments

`file`	a character string giving the name of the file to print to.
`...`	further options for `postscript()`.
`paper`	the size of paper in the printer. The choices are `"a4"`, `"letter"`, `"legal"` and `"executive"`. Also, `"special"` can be used, when the width and height specify the paper size.
`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.
`onefile`	logical: if true (the default) allow multiple figures in one file. If false, assume only one figure and give EPSF header and no `DocumentMedia` comment.
`pagecentre`	logical: should the device region be centred on the page: defaults to true.
`print.it, append`	logical; are currently **disregarded**; just there for compatibility reasons.

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.

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, onefile=FALSE, paper="special"`.

See Also

Devices, check.options which is called from both `ps.options` and `postscript`.

Examples

```
 # open the file "foo.ps" for graphics output
 postscript("foo.ps")


dev.off()               # turn off the postscript device

all(unlist(ps.options()) == unlist(.PostScript.Options))
ps.options(bg = "pink")
str(ps.options(reset = TRUE))
```

```
##- error checking of arguments:
ps.options(width=0:12, onefile=0, bg=pi)

## override the check for 'onefile', but not the others:
str(ps.options(width=0:12, onefile=1, bg=pi, override.check = c(F,T,F)))
```

---

power *Create a Power Link Object* power

Description

Creates a link object based on the link function $\eta = \mu^{\lambda}$.

Usage

```
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.

See Also

`make.link`, `family`

Examples

```
power()
quasi(link=power(1/3))[c("linkfun", "linkinv")]
```

`ppoints` *Ordinates for Probability Plotting* `ppoints`

Description

Generates the sequence of "probability" points `(1:m - a)/(m + (1-a)-a)` where `m` is either `n`, if `length(n)==1`, or `length(n)`.

Usage

```
ppoints(n, a = ifelse(n <= 10, 3/8, 1/2))
```

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 + rev(p) == 1`.

`ppoints()` is used in `qqplot` and `qqnorm` to generate the set of probabilities at which to evaluate the inverse distribution.

See Also

`qqplot`, `qqnorm`.

Examples

```
ppoints(4) # the same as  ppoints(1:4)
ppoints(10)
ppoints(10, a=1/2)
```

`precip` *Annual Precipitation in US Cities* `precip`

Description

The average amount of precipitation (rainfall) in inches for each of 70 United States (and Puerto Rico) cities.

Usage

```
data(precip)
```

Format

A named vector of length 70.

Source

Statistical Abstracts of the United States, 1975.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(precip)
dotplot(precip[order(precip)], main = "precip data")
title(sub = "Average annual precipitation (in.)")
```

`predict` *Model Predictions* `predict`

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.

The function `predict.lm` makes predictions based on the results produced by `lm`.

Usage

```
predict (object, ...)
```

Arguments

`object`	a model object for which prediction is desired.
`...`	additional arguments affecting the predictions produced.

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.

See Also

`predict.lm`.

Examples

```
## All the "predict" methods available in your current search() path:
for(fn in methods("predict"))
   cat(fn,":\n\t",deparse(args(get(fn))),"\n")
```

predict.glm *Predict Method for GLM Fits* predict.glm

Description

Obtains predictions and optionally estimates standard errors of those predictions from a fitted generalized linear model object.

Usage

```
predict.glm(object, newdata = NULL, type = c("link", "response", "terms"),
            se.fit = FALSE, dispersion = NULL, terms = NULL, ...)
```

Arguments

`object`	A fitted object of class inheriting from `"glm"`.
`newdata`	Optionally, a new data frame from which to make the predictions. If omitted, the fitted linear predictors are used.
`type`	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.
`se.fit`	A switch indicating if standard errors are required.
`dispersion`	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 vector of strings specifies which terms are to be returned

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.

Author(s)

B.D. Ripley

See Also

glm

Examples

```
## example from Venables and Ripley (1997, pp. 231-3.)
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.lm` *Predict method for Linear Model Fits* `predict.lm`

Description

Predicted values based on linear model object

Usage

```
predict[.lm](object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
             interval = c("none", "confidence", "prediction"),
             level = 0.95, type = c("response", "terms"),
             terms = NULL, ...)
```

Arguments

`object`	Object of class inheriting from `"lm"`
`newdata`	Data frame in which to predict
`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 model term prediction, which term

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 `intervals` specifies computation of confidence or prediction (tolerance) intervals at the specified `level`.

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 predictions
`residual.scale`	residual standard deviations
`df`	degrees of freedom for residual

Note

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`.

Examples

```
## 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[,-1]),
        lty=c(1,2,2,3,3), type="l", ylab="predicted y")
```

`preplot` *Pre-computations for a Plotting Objeect* `preplot`

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`.

`presidents` *Approval Rating of US Presidents* `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

```
data(presidents)
```

Format

A time series of 120 values.

Source

The Gallup Organisation.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(presidents)
plot(presidents, las = 1, ylab = "Approval rating (%)",
     main = "presidents data")
```

pressure — *Vapor Pressure of Mercury as a Function of Temperature* — pressure

Description

Data on the relation between temperature in degrees Celsius and vapor pressure of mercury in millimeters (of mercury).

Usage

```
data(pressure)
```

Format

A data frame with 19 observations on 2 variables.

[, 1]	temperature	numeric	temperature (deg C)
[, 2]	pressure	numeric	pressure (mm)

Source

Weast, R. C., ed. (1973) *Handbook of Chemistry and Physics.* CRC Press.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(pressure)
plot(pressure, xlab = "Temperature (deg C)",
     ylab = "Pressure (mm of Hg)",
     main = "pressure data: Vapor Pressure of Mercury")
plot(pressure, xlab = "Temperature (deg C)",  log = "y",
     ylab = "Pressure (mm of Hg)",
     main = "pressure data: Vapor Pressure of Mercury")
```

pretty — *Pretty Breakpoints* — pretty

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.
`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

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 $10^{\lfloor \log_{10}(c) \rfloor}$ such that $b \leq c < 10b$.

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`.

. . . . . .

Examples

```
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* — Primitive

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 R function.

See Also

`.Internal`.

Examples

```
mysqrt <- .Primitive("sqrt")
c
.Internal # this one *must* be primitive!
get("if") # just 'if' or 'print(if)' are not syntactically ok.
```

print — *Print Values* — print

Description

`print` prints its argument and returns it *invisibly* (via `invisible(x)`). It is a generic function which means that new printing methods can be easily added for new `class`es.

Usage

```
print(x, ...)

print.factor(x, quote = FALSE)
print.ordered(x, quote = FALSE)
```

Details

The default method, `print.default` has its own help page. Use `methods("print")` to get all the methods for the `print` generic.

See `noquote` as an example of a class whose main purpose is a specific `print` method.

See Also

The default method `print.default`, and help for the methods above; further `options`, `noquote`.

Examples

```
ts(1:20)#-- print  is the ‘‘Default function’’ --> print.ts(.) is called
rr <- for(i in 1:3) print(1:i)
rr
```

`print.coefmat` *Print Coefficient Matrices* `print.coefmat`

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

```
print.coefmat(x, digits=max(3, getOption("digits") - 2),
              signif.stars = getOption("show.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(",
              na.print = "", ...)
```

Arguments

`x`	a numeric matrix like object, to be printed.
`digits`	number of 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(.)`.
`dig.tst`	number of significant digits for the test statistics, see `tst.ind`.
`cs.ind`	indices (integer) of column numbers which are (like) **c**oefficients and **s**tandard 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 `link{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 *iff* `P.values` (above).
`na.print`	a character string to code `NA` values in printed output.
`...`	Further arguments for `print(..)`.

Details

Despite its name, this is **not** (yet) a method for the generic `print` function, because there is no class `"coefmat"`.

Value

Invisibly returns its argument, `x`.

Author(s)

Martin Maechler

See Also

`print.summary.lm`, `format.pval`, `format`

Examples

```
cmat <- cbind(rnorm(3, 10), sqrt(rchisq(3, 12)))
cmat <- cbind(cmat, cmat[,1]/cmat[,2])
cmat <- cbind(cmat, 2*pnorm(-cmat[,3]))
colnames(cmat) <- c("Estimate", "Std.Err", "Z value", "Pr(>z)")
print.coefmat(cmat[,1:3])
print.coefmat(cmat)
options(show.coef.Pvalues = FALSE)
print.coefmat(cmat, digits=2)
print.coefmat(cmat, digits=2, P.values = TRUE)
options(show.coef.Pvalues = TRUE)# revert
```

`print.default` *Default Printing* `print.default`

Description

`print.default` is the *default* method of the generic `print` function which prints its argument.

`print.atomic` is almost the same and exists purely for compatibility reasons.

Usage

```
print.default(x, digits = NULL, quote = TRUE, na.print = "NA",
              print.gap = 1, ...)
print.atomic(x, quote = TRUE, ...)
```

Arguments

`digits`	a non-null value for `digits` specifies the number of significant digits to be printed in values. If `digits` is `NULL`, the value of `digits` set by `options` is used.
`quote`	logical, indicating whether or not strings (`characters`) should be printed with surrounding quotes.
`na.print`	a character string which is used to indicate `NA` values in printed output.

`print.gap`	an integer, giving the spacing between adjacent columns in printed matrices and arrays.
`...`	(further arguments, currently disregarded)

See Also

The generic `print`, `options`. The `"noquote"` class and print method.

Examples

```
pi
print(pi, digits = 16)
LETTERS[1:16]
print(LETTERS, quote = FALSE)
```

`print.matrix` *Print Matrices* `print.matrix`

Description

Pseudo-method for the `print` generic. Especially useful with the `right` argument which does not (yet) exist for `print.default`.

Usage

```
print.matrix(x, rowlab=character(0), collab=character(0),
             quote=TRUE, right=FALSE)
```

Arguments

`x`	numeric or character matrix.
`rowlab,collab`	(optional) character vectors giving row or column names respectively. By default, these are taken from `dimnames(x)`.
`quote`	logical; if `TRUE` and `x` is of `mode character`, *quotes* (`".."`) are used.
`right`	if `TRUE` and `x` is of `mode character`, the output columns are *right*-justified.

Details

`print.matrix` and `print.default` both print matrices, and each has at least an optional argument that the other lacks. Also, both directly dispatch into `.Internal` code directly instead of relying on each other. This mainly stems from historic compatibility and similar reasons should be changed in the future.

`prmatrix` is currently just an `.Alias` for `print.matrix`.

Value

Invisibly returns its argument, `x`.

See Also

`print.default`, and other `print` methods.

Examples

```
print.matrix(m6 <- diag(6), row = rep("",6), coll=rep("",6))

chm <- matrix(scan(file.path(system.file("help", pkg="eda"),"AnIndex"),
what=""),,2, byrow=T)
chm  #-> print.default(.)  = 'same' as  print.matrix(chm)
print.matrix(chm, collab = paste("Column",1:3), right=TRUE, quote=FALSE)
```

`proc.time` *Running Time of R* `proc.time`

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 the elapsed time to be recorded to the nearest second, and CPU times to of the order of 1/100 second.

It is most useful for "timing" the evaluation of R expressions, which can be done conveniently with `system.time`.

Note

It is possible to compile R without support for `proc.time`, when the function will not exist.

See Also

`system.time` for timing a valid R expression.

Examples

```
ptm <- proc.time()
for (i in 1:50) mad(runif(500))
proc.time() - ptm
```

`prod` *Product of Vector Elements* `prod`

Description

`prod` returns the product of all the values present in its arguments.

Usage

```
prod(..., na.rm=TRUE)
```

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.

See Also

`sum`, `cumprod`, `cumsum`.

Examples

```
print(prod(1:7)) == print(gamma(8))
```

`profile` *Generic Function for Profiling Models* `profile`

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.

proj *Projections of Models* proj

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

```
proj     (object, ...)
proj.aov    (object, onedf = FALSE, unweighted.scale = FALSE)
proj.aovlist(object, onedf = FALSE, unweighted.scale = FALSE)
proj.default(object, onedf = TRUE, ...)
proj.lm     (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 observations 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)

B.D. Ripley

See Also

aov, lm, model.tables

Examples

```
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
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(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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)
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)
```

`prompt` *Produce Prototype of an R Documentation File* `prompt`

Description

Facilitate the constructing of files documenting R functions.

Usage

```
prompt(object, ...)

prompt.default(object,
    filename = paste(name, ".Rd", sep = ""),
    force.function = FALSE)

prompt.data.frame(object,
    filename = paste(name, ".Rd", sep = ""))
```

Arguments

`object`	an R object, typically a function
`filename`	name of the output file
`force.function`	treat `object` as function in any case

Details

An ASCII file `filename` is produced containing the proper function and argument names of `object`. You have to edit it before adding the documentation to the source tree, i.e., (currently) to '$R_HOME/src/library/base/man/'.

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` library. We are trying to settle on a preferred format for the documentation.

Author(s)

Douglas Bates for `prompt.data.frame`

See Also

`help` and the chapter on "Writing R documentation" in "Writing R Extensions" (see the 'doc/manual' subdirectory of the R source tree).

Examples

```
prompt(plot.default)
prompt(interactive, force.function = TRUE)
unlink("plot.default.Rd")
unlink("interactive.Rd")

data(women)
prompt(women)
unlink("women.Rd")
```

---

prop.table — *Express table entries as fraction of marginal table* — prop.table

Description

This is really `sweep(x, margin, margin.table(x, margin), "/")` for newbies.

Usage

```
prop.table(x, margin)
```

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)
```

qqnorm *Quantile-Quantile Plots* qqnorm

Description

qqnorm 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, ylim, main = "Normal Q-Q Plot", xlab = "Theoretical Quantiles",
       ylab = "Sample Quantiles", plot.it = TRUE, ...)
qqline(y, ...)
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.

Value

For qqnorm and qqplot, a list with components

x	The x coordinates of the points that were/would be plotted
y	The corresponding y coordinates

See Also

ppoints.

Examples

```
y <- rt(200, df = 5)
qqnorm(y); qqline(y, col = 2)
qqplot(y, rt(300, df = 5))
data(precip)
qqnorm(precip, ylab = "Precipitation [in/yr] for 70 US cities")
```

qr *The QR Decomposition of a Matrix* qr

Description

`qr` computes the QR decomposition of a matrix. It provides an interface to the techniques used in the LINPACK routine DQRDC.

Usage

```
qr(x, tol=1e-07)
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)

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`.
`qr`	a QR decomposition of the type computed by `qr`.
`y, b`	a vector or matrix of right-hand sides of equations.
`a`	A matrix or QR decomposition.

Details

The QR decomposition plays an important role in many statistical techniques. In particular it can be used to solve the equation $\boldsymbol{Ax} = \boldsymbol{b}$ for given matrix $\boldsymbol{A}$, and vector $\boldsymbol{b}$. 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 %*% y` and `t(Q) %*% y`, where `Q` is the $\boldsymbol{Q}$ matrix.

`qr.solve` solves systems of equations via the QR decomposition.

`is.qr` returns `TRUE` if `x` is a list with a component named `qr` 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. The components in the returned value correspond directly to the values returned by DQRDC.

`qr`	a matrix with the same dimensions as `x`. The upper triangle contains the $\boldsymbol{R}$ of the decomposition and the lower triangle contains information on the $\boldsymbol{Q}$ of the decomposition (stored in compact form).

`qraux`	a vector of length `ncol(x)` which contains additional information on $\boldsymbol{Q}$.
`rank`	the rank of `x` as computed by the decomposition.
`pivot`	information on the pivoting strategy used during the decomposition.

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 `det2` in the examples below.

References

Dongarra, J. J., Bunch, J. R., Moler, C. B. and Stewart, G. W. (1978) *LINPACK Users Guide.* Philadelphia: SIAM Publications.

See Also

`qr.Q`, `qr.R`, `qr.X` for reconstruction of the matrices. `solve.qr`, `lsfit`, `eigen`, `svd`.

Examples

```
## The determinant of a matrix  -- if you really must have it
det2 <- function(x) prod(diag(qr(x)$qr))*(-1)^(ncol(x)-1)
det2(print(cbind(1,1:3,c(2,0,1))))

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
```

`QR.Auxiliaries` *Reconstruct the Q, R, or X Matrices from a QR Object* `QR.Auxiliaries`

Description

Returns the original matrix from which the object was constructed or the components of the decomposition.

Usage

```
qr.X(qrstr, complete = FALSE, ncol =)
qr.Q(qrstr, complete = FALSE, Dvec = 1)
qr.R(qrstr, complete = FALSE)
```

Arguments

`qrstr`	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 $\boldsymbol{Q}$ or $\boldsymbol{X}$ matrices is to be made, or whether the $\boldsymbol{R}$ matrix is to be completed by binding zero-value rows beneath the square upper triangle.
`ncol`	integer in the range `1:nrow(qrstr$qr)`. The number of columns to be in the reconstructed $\boldsymbol{X}$. The default when `complete` is `FALSE` is the original $\boldsymbol{X}$ from which the qr object was constructed. The default when `complete` is `TRUE` is a square matrix with the original $\boldsymbol{X}$ 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 $\boldsymbol{Q}$ will be multiplied by the corresponding diagonal value.

Value

`qr.X` returns $\boldsymbol{X}$, the original matrix from which the qr object was constructed. 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 $\mathbf{Q}$, the order-nrow(X) orthogonal (unitary) transformation represented by qrstr. If `complete` is `TRUE`, $\mathbf{Q}$ has `nrow(X)` columns. If `complete` is `FALSE`, $\mathbf{Q}$ has `ncol(X)` columns. When `Dvec` is specified, each column of $\mathbf{Q}$ is multiplied by the corresponding value in `Dvec`.

`qr.R` returns $\mathbf{R}$, the upper triangular matrix such that `X == Q %*% R`. The number of rows of $\mathbf{R}$ is `nrow(X)` or `ncol(X)`, depending on whether `complete` is `TRUE` or `FALSE`.

Examples

```
data(LifeCycleSavings)
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 :
all((1 - X /( Q %*% R))< 100*.Machine$double.eps)#TRUE

dim(Qc <- qr.Q(qrstr, complete=TRUE)) # Square: dim(Qc) == rep(nrow(x),2)
all((crossprod(Qc) - diag(nrow(x))) < 10*.Machine $double.eps)

QD <- qr.Q(qrstr, D=1:p)      # QD == Q %*% diag(1:p)
all(QD - Q %*% diag(1:p)  < 8* .Machine$double.eps)

dim(Rc <- qr.R(qrstr, complete=TRUE)) # == dim(x)
dim(Xc <- qr.X(qrstr, complete=TRUE)) # square: nrow(x) ^ 2
all(Xc[,1:p] == X)
```

`quakes` *Locations of Earthquakes off Fiji* `quakes`

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

```
data(quakes)
```

Format

A data frame with 1000 observations on 5 variables.

[,1]	lat	numeric	Latitude of event
[,2]	long	numeric	Longitude
[,3]	depth	numeric	Depth (km)
[,4]	mag	numeric	Richter Magnitude
[,5]	stations	numeric	Number of stations reporting

Details

There are two clear planes of seismic activity. One is a major plate junction; the other is a 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

```
data(quakes)
pairs(quakes, main = "Fiji Earthquakes, N = 1000", cex.main=1.2, pch=".")
```

`quantile` *Sample Quantiles* `quantile`

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, probs=seq(0, 1, 0.25), na.rm=FALSE, names = TRUE)
```

Details

A vector of length `length(probs)` is returned; if `names = TRUE`, it has a `names` attribute.

`quantile(x,p)` as a function of `p` linearly interpolates the points ((i-1)/(n-1), ox[i]), where `ox <- order(x)` (the "order statistics") and `n <- length(x)`.

This gives `quantile(x, p) == (1-f)*ox[i] + f*ox[i+1]`, where `r <- 1 + (n-1)*p`, `i <- floor(r)`, `f <- r - i` *and* `ox[n+1] := ox[n]`.

Examples

```
quantile(x <- rnorm(1001))# Extremes & Quartiles by default
quantile(x,  probs=c(.1,.5,1,2,5,10,50)/100)

n <- length(x) ## the following is exact, because 1/(1001-1) is exact:
all(abs(sort(x) == quantile(x, probs = ((1:n)-1)/(n-1), names=F)))# TRUE

n <- 777
ox <- sort(x <- round(rnorm(n),1))# round() produces ties
ox <- c(ox, ox[n]) #- such that ox[n+1] := ox[n]
p <- c(0,1,runif(100))
i <- floor(r <- 1 + (n-1)*p)
f <- r - i
all(abs(quantile(x,p) - ((1-f)*ox[i] + f*ox[i+1])) < 20*.Machine$double.eps)
```

---

quit — *Terminate an R Session* — quit

Description

The function `quit` or its alias `q` terminate the current R session.

Usage

```
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.

See Also

`.First` for setting things on startup.

Examples

```
## Unix-flavour example
.Last <- function() {
  cat("Now sending PostScript graphics to the printer:\n")
  system("lpr Rplots.ps")
  cat("bye bye...\n")
}
quit("yes")
```

`R.home` *Return the R Home Directory* `R.home`

Description

Return the R home directory.

Usage

```
R.home()
```

Value

A character string giving the current home directory.

`R.Version` *Version Information* `R.Version`

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 an `.Alias` to it for S compatibility), whereas `R.version.string` is simple `character` string, useful for plotting, etc.

Usage

```
R.Version()
R.version
R.version.string
```

Value

`R.Version` returns a list with components

`platform`	the platform for which R was built. Under Unix, a triplet of the form CPU-VENDOR-OS, as determined by the configure script. E.g, `"i586-unknown-linux"`.
`arch`	the architecture (CPU) R was built on/for.
`os`	the underlying operating system
`system`	CPU and OS.
`status`	the status of the version (e.g., `"Alpha"`)
`status.rev`	the status revision level
`major`	the major version number
`minor`	the minor version number
`year`	the year the version was released
`month`	the month the version was released
`day`	the day the version was released
`language`	always `"R"`.

See Also

`machine`.

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
```

`Random` *Random Number Generation* `Random`

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.

`set.seed` is the recommended way to specify seeds.

Usage

```
.Random.seed <- c(rng.kind, n1, n2, ...)
save.seed <- .Random.seed

RNGkind(kind = NULL, normal.kind = NULL)
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.
`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 `"Marsaglia-Multicarry"`.

`"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 6.9536×10^{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 `.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

$$X_j = (X_{j-100} - X_{j-37}) \bmod 2^{30}$$

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}.

`"user-supplied"`: Use a user-supplied generator. See `Random.user` for details.

`normal.kind` can be `"Kinderman-Ramage"` (the default) or `"Ahrens-Dieter"` or `"Box-Muller"`.

`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 `"Knuth-TAOCP"`).

Value

`.Random.seed` is an `integer` vector whose first element *codes* the kind of RNG and normal generator. The lowest two decimal digits are in in `0:(k-1)` where `k` is the number of available RNGs. The thousands represent the type of normal generator (starting at `0`).

In the underlying C, `.Random.seed[-1]` is `unsigned`; therefore in `R` `.Random.seed[-1]` can be negative.

`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`.

`set.seed` returns `NULL`, invisibly.

Note

`.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 `set.seed` rather than set `.Random.seed`.

Author(s)

of RNGkind: Martin Maechler. Current implementation, B. D. Ripley

References

Wichmann, B. A. and Hill, I. D. (1982) *Algorithm AS 183: An Efficient and Portable Pseudo-random Number Generator*, Applied Statistics, **31**, 188–190; Remarks: **34**, 198 and **35**, 89.

De Matteis, A. and Pagnutti, S. (1993) *Long-range Correlation Analysis of the Wichmann-Hill Random Number Generator*, Statist. Comput., **3**, 67–70.

Marsaglia, G. (1997) *A random number generator for C.* Discussion paper, posting on Usenet newsgroup `sci.stat.math` on September 29, 1997.

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.)

Marsaglia, G. and Zaman, A. (1994) Some portable very-long-period random number generators. *Computers in Physics*, **8**, 117–121.

Matsumoto, M. and Nishimura, T. (1998) Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Transactions on Modeling and Computer Simulation*, **8**, 3–30.
Source code at http://www.math.keio.ac.jp/~matumoto/emt.html.

Knuth, D. E. (1997) *The Art of Computer Programming.* Volume 2, third edition.
Source code at http://www-cs-faculty.stanford.edu/~knuth/taocp.html.

See Also

runif, rnorm, ...

Examples

```
runif(1); .Random.seed; runif(1); .Random.seed
## If there is no seed,  a ‘‘random’’ new one is created:
rm(.Random.seed); runif(1); .Random.seed

RNGkind("Wich")# (partial string matching on 'kind')
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 }

## This shows how 'runif(.)' works for Wichmann-Hill, using only R functions:
rs <- .Random.seed
(WHs <- next.WHseed(rs[-1]))
u <- runif(1)
all(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])
```

`Random.user` *User-supplied Random Number Generation* `Random.user`

Description

Function RNGkind allows a user-coded random number generator to be supplied. The details are give here.

Details

A user-specified 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 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`.

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

```
## Marsaglia's conguential 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; }
```

`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

```
data(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

```
## We could re-generate the dataset by the following R code
seed <- as.double(1)
RANDU <- function() {
    seed <<- ((2^16 + 3) * seed)
    seed/(2^31)
}
for(i in 1:400) {
    U <- c(RANDU(), RANDU(), RANDU(), RANDU(), RANDU())
    print(round(U[1:3], 6))
}
```

`range` *Range of Values* `range`

Description

`range` returns a vector containing the minimum and maximum of all the given arguments.

Usage

```
range(..., na.rm = FALSE)
range.default(..., na.rm = FALSE, finite = FALSE)
```

Arguments

`...`	any `numeric` objects.
`na.rm`	logical, indicating if `NA`'s should be omitted.
`finite`	logical, indicating if all non-finite elements should be omitted.

Details

This is a generic function; currently, it has only a default method (`range.default`).

It is also a member of the `Summary` group of functions, see `Methods`.

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`.

See Also

`min`, `max`, `Methods`.

Examples

```
print(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* `rank`

Description

`rank` returns the sample ranks of the values in `x`. Ties result in ranks being averaged. `NA` values are not allowed in this function.

Usage

```
rank(x)
```

See Also

`order` and `sort`.

Examples

```
(r1 <- rank(x1 <- c(3,1,4,59,26)))
all(rank(r1) == r1) ## rank() is "idempotent": rank(rank(x)) == rank(x)
(r2 <- rank(x2 <- c(3,1,4,5,9,2,6,5,3,5)))
all(rank(r2) == r2)
```

`read.fwf` *Read Fixed Width Format Files* `read.fwf`

Description

Read a "table" of **f**ixed **w**idth **f**ormatted data into a `data.frame`.

Usage

```
read.fwf(file, widths, sep=" ", as.is = FALSE,
         skip = 0, row.names, col.names)
```

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.
`widths`	integer vector, giving the widths of the fixed-width fields (of one line).
`sep`	character; the separator used internally; should be irrelevant.
`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`.

Value

A `data.frame` as by `read.table` which is called internally.

Note

This function currently relies on the `Perl` script '$R_HOME/bin/fwf2table'. Therefore, it will only work in an environment with `Perl 5` installed, see 'INSTALL' in the top-level directory of the sources.

Author(s)

Kurt Hornik ⟨hornik@ci.tuwien.ac.at⟩

See Also

`scan` and `read.table`.

Examples

```
ff <- tempfile()
cat(file=ff, "123456", "987654", sep="\n")
read.fwf(ff, width=c(1,2,3))    #> 1 23 456 \ 9 87 654
unlink(ff)
```

`read.socket` *Read from or Write to a Socket* `read.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

```
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

```
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)
    }
    close.socket(socket)
    if (print) cat(output)
    invisible(output)
}
finger("root")  ## only works if your site provides a finger daemon
```

---

read.table *Data Input* read.table

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

```
read.table(file, header = FALSE, sep = "", dec = ".", quote = "\"'",
           row.names, col.names, as.is = FALSE, na.strings = "NA",
           skip = 0)

read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         row.names, col.names, as.is=FALSE, na.strings="", skip=0)

read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",",
          row.names, col.names, as.is=FALSE, na.strings="", skip=0)
```

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()`.
`header`	a logical value indicating whether the file contains the names of the variables as its first line.
`sep`	the field separator character. Values on each line of the file are separated by this character.
`quote`	the set of quoting characters. To disable quoting altogether, use `quote=""`
`dec`	the decimal point
`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.
`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 non-numeric variables to factors. The variable `as.is` controls this conversion. Its value is either a vector of logicals (values are recycled if necessary), or a vector of numeric indices which specify which columns should be left as character strings.
`na.strings`	a vector strings which are to be interpreted as `NA` values.
`skip`	the number of lines of the data file to skip before beginning to read data.

Details

`read.csv` and `read.csv2` are identical to `read.table` except for the defaults. They are intended for reading "comma separated variable" files (.csv) or the variant used in countries that use comma as decimal point and consequently semicolon as field separator. Notice that `header=TRUE` in these variants.

Value

A data frame (`data.frame`) containing a representation of the data in the file.

This function is the principal means of reading tabular data into R.

Note

The implementation of `read.table` currently reads everything as character using `scan` and subsequently defines `"numeric"` or `factor` variables.

This is quite memory consuming for files of thousands of records and may need larger memory, see `Memory`.

See Also

`scan`, `read.fwf` for reading *f*ixed *w*idth *f*ormatted input; `write.table`; `data.frame`.

`read.table.url` *Read Data and Code from a URL* `read.table.url`

Description

Extensions of `read.table`, `scan`, `source` and `file.show` to read text files on a remote server.

Usage

```
read.table.url(url, method="auto",...)
scan.url(url, method="auto", ...)
source.url(url, method="auto", ...)
url.show(url, title = url, file = tempfile(),
         delete.file = TRUE, ...)
```

Details

These functions call `download.file` to create a temporary local file. The file can be downloaded by `lynx` or `wget` if these are available on the system. Another option is a direct `HTTP` socket connection, if the local machine allows this.

Value

The same value as the respective file-based functions.

See Also

`read.table`, `scan`, `source`, `make.socket`, `read.socket`, `file.show`,`download.file`

Examples

```
read.table.url("http://lib.stat.cmu.edu/jcgs/tu",skip=4,header=T)
url.show("http://lib.stat.cmu.edu/datasets/csb/ch11b.txt")
beaver<-read.table.url("http://lib.stat.cmu.edu/datasets/csb/ch11b.dat",
  col.names=c("obsnum","day","time","temperature","activity"),row.names=1)
# the next two examples will only work if socket connections to
# statlib are allowed from your site.
url.show("http://lib.stat.cmu.edu/datasets/csb/ch3a.txt",
                 method="socket")
ozone<-read.table.url("http://lib.stat.cmu.edu/datasets/csb/ch3a.dat",
  col.names=c("date","day.cts","day.passive","night.cts","night.passive"),
  na.strings=".", method="socket")
```

`readline` *Read a Line from the Terminal* `readline`

Description

`readline` reads a line from the terminal, returning it as a character string.

Usage

```
readline(prompt="")
```

Arguments

`prompt`	the string printed when prompting the user for input. Should usually end with a space `" "`.

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()
```

`real` *Real Vectors* `real`

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)
```

Note

R *has no single precision data type. All real numbers are stored in double precision format.*

Recall — *Recursive Calling* — Recall

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.

See Also

`do.call` and `call`.

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 <- .Alias(fib) ## renaming wouldn't work without Recall
fibonacci(10) # 55
```

rect — *Draw a Rectangle* — rect

Usage

```
rect(xleft, ybottom, xright, ytop,
     col=NULL, border=par("fg"), lty=NULL, lwd=par("lwd"), xpd=FALSE)
```

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.
`col`	color(s) to fill the rectangle(s) with.
`border`	color for rectangle border(s).
`lty`	line type for borders; defaults to `"solid"`.
`lwd`	width for borders.
`xpd`	logical (*"expand"*); if `FLASE`, everything is clipped to the plot region.

Details

The positions supplied, i.e., `xleft, ...`, are relative to the current plotting region. If the x-axis goes from 100 to 200 then `xleft` must be larger than 100 and `xright` must be less than 200.

See Also

`box` for the "standard" box around the plot; `polygon` and `segments` for flexible line drawing.

Examples

```
## set up the plot region:
plot(c(100, 250), c(300, 450), type = "n",
     main = "2 x 11 rectangles; `rect(100+i,300+i,  150+i,380+i)'")
i <- 4*(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)
```

---

relevel *Reorder Levels of Factor* relevel

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

```
relevel(x, ref, ...)
```

Arguments

`x`	An unordered factor.
`ref`	The reference level.
`...`	Additional arguments for future methods.

Value

A factor of the same length as `x`.

Author(s)

B. D. Ripley

See Also

`factor`, `contr.treatment`

Examples

```
data(warpbreaks)
warpbreaks$tension <- relevel(warpbreaks$tension, ref="M")
summary(lm(breaks ~ wool + tension, data=warpbreaks))
```

REMOVE *Remove Add-on Packages* REMOVE

Description

Use `R 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 REMOVE -l lib pkgs`.

Usage

```
R 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.

See Also

INSTALL

`remove` *Remove Objects from a Specified Environment* `remove`

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 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 = pos.to.env(pos),
       inherits = FALSE)
rm    (..., list = character(0), pos = -1, envir = pos.to.env(pos),
       inherits = FALSE)
```

See Also

ls, objects

`rep` *Replicate Elements* `rep`

Description

`rep` replicates the values in `x` according to the values given in `times` and `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.

Usage

```
rep(x, times, length.out)
```

See Also

`seq`, `sequence`.

Examples

```
rep(1:4,2)
rep(1:4,c(2,2,2,2))
```

`replace` *Replace Values in a Vector* `replace`

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)
```

replications *Number of Replications of Terms* replications

Description

Returns a vector or a list of the number of replicates for each term in the formula.

Usage

```
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)

B. D. Ripley

See Also

model.tables

Examples

```
## From Venables and Ripley (1997) p.210.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
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(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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)
replications(~ . - yield, npk)
```

`residuals` *Extract Model Residuals* `residuals`

Description

`residuals` is a generic function which extracts model residuals from objects returned by modeling functions.

The abbreviated form `resid` is an alias for `residuals`. It is intended to encourage users to access object components through an accessor function rather than by directly referencing an object slot.

All object classes which are returned by model fitting functions should provide a `residuals` method. (Note that the method is 'residuals' and not 'resid'.)

Usage

```
residuals(x, ...)
resid(x, ...)
```

Arguments

`x`	an object for which the extraction of model residuals is meaningful.
`...`	other arguments.

Value

Residuals extracted from the object `x`.

See Also

`coefficients`, `fitted.values`, `glm`, `lm`.

`restart` *Restart an Expression* `restart`

Description

`restart` performs a type of non-local return.

`try` is a user-friendly wrapper to run an expression that might fail.

Usage

```
restart(on = TRUE)
try(expr, first = T)
```

Arguments

`on`	if true a jump point is set; if false the jump point is removed
`expr`	an R expression to try
`first`	not for user use!

Details

When `restart` is called with `on = TRUE` the evaluator marks that function as a return point. Any errors or signals (such as control-C on Unix) cause control to return to the start of the function containing the call to `restart`. The most recently established function is always entered first.

Value

`try` returns the value of the expression if it succeeds, and an invisible object of class `"try_error"` containing the error message if it if fails. The normal error handling will print the same message unless `options("show.error.messages")` is false.

Note

The direct use of `restart` is likely to result in an infinite loop. Use `try` unless you are sure you know what you are doing.

See Also

`options` for setting error handlers and suppressing the printing of error messages; `geterrmessage` for retrieving the last error message.

Examples

```
## 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)
```

---

rev *Reverse a Vector's Elements* rev

Description

`rev` provides a reversed version of its argument. It can be used in combination with `sort` to obtain vectors sorted into descending order.

Usage

```
rev(x)
```

See Also

`seq`, `sort`.

Examples

```
x <- c(1:5,5:3)
# sort into descending order
rev(sort(x))
all(rev(1:7) == 7:1)#- don't need 'rev' here
```

`rgb` *RGB Color Specification* `rgb`

Description

This function creates "colors" corresponding to the given intensities (between 0 and 1) of the red, green and blue primaries. The names argument may be used to provide names for the colors.

The values returned by `rgb` can be used with a `col=` specification in graphics functions or in `par`.

Usage

```
rgb(red, green, blue, names=NULL)
```

See Also

`rainbow`, `hsv`, `gray`.

Examples

```
rgb(0,1,0)
u01 <- seq(0,1, length=11); all(rgb(u01,u01,u01) == gray(u01))
reds <- rgb((0:15)/15, g=0,b=0, names=paste("red",0:15,sep="."))
reds
```

`RHOME` *R Home Directory* `RHOME`

Description

Returns the location of the R home directory, which is the root of the installed R tree.

Usage

```
R RHOME
```

`rivers` *Lengths of Major North American Rivers* `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

```
data(rivers)
```

Format

A vector containing 141 observations.

Source

World Almanac and Book of Facts, 1975, page 406.

References

McNeil, D. R. (1977) *Interactive Data Analysis*. New York: Wiley.

`rle` *Run Length Encoding* `rle`

Description

Compute the lengths and values of runs of equal values in a vector.

Usage

```
rle(x)
```

Arguments

`x`	a (numerical, logical or character) vector.

Value

A list with components

`lengths`	a vector containing the length of each run.
`values`	a vector of the same length as `lengths` with the corresponding values.

Examples

```
x <- rev(rep(6:10, 1:5))
rle(x)
## $lengths
## [1] 5 4 3 2 1
## $values
## [1] 10  9  8  7  6
z <- c(T,T,F,F,T,F,T,T,T)
rle(z)
rle(as.character(z))
```

`Round` *Rounding of Numbers* `Round`

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`.

`round` rounds the values in its first argument to the specified number of decimal places (default 0). Note that for rounding off a 5, the IEEE standard is used, *"go to the even digit"*. Therefore `round(0.5)` is `0` and `round(-1.5)` is `-2`.

`signif` rounds the values in its first argument to the specified number of significant digits.

`trunc` takes a single numeric argument `x` and returns a numeric vector containing the integers by truncating the values in `x` toward `0`.

`zapsmall` determines a `digits` argument `dr` for calling `round(x, digits = dr)` such that values "close to zero" values are "zapped", i.e., treated as `0`.

Usage

```
ceiling(x)
floor(x)
round(x, digits = 0)
signif(x, digits = 6)
trunc(x)
zapsmall(x, digits= getOption("digits"))
```

See Also

`as.integer`.

Examples

```
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)]
all(trunc(x1) == as.integer(x1))# TRUE
non.int <- ceiling(x1) != floor(x1)
all(non.int == (ceiling(x1) != trunc(x1) | trunc(x1) != floor(x1)))
# TRUE
all((signif(x1, 1) != round(x1,1)) == (non.int & abs(x1)>1)) # TRUE

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))
```

row *Row Indexes* row

Description

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

Usage

```
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.

See Also

col to get columns.

Examples

```
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/colnames *Row and Columnn Names* row/colnames

Description

Retrieve or set the row or column names of an object (the first or second component of its dimnames).

Usage

```
rownames(x, do.NULL = TRUE, prefix = "row")
rownames(x) <- namevector

colnames(x, do.NULL = TRUE, prefix = "col")
colnames(x) <- namevector
```

Details

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 `dimnames(x)[[i]]` (`i` = 1 or 2) is `NULL`.

See Also

`dimnames`, `case.names`, `variable.names`.

Examples

```
m0 <- matrix(NA, 4, 0)
m2 <- cbind(1,1:4)
rownames(m0)

colnames(m2, do.NULL = FALSE)
colnames(m2) <- c("x","Y")
rownames(m2) <- rownames(m2, do.NULL = FALSE, prefix = "Obs.")
m2
```

`rowsum` *Give Row Sums of a Matrix, Based on a Grouping Variable* `rowsum`

Description

Compute sums across rows of a matrix for each level of a grouping variable.

Usage

```
rowsum(x, group, reorder = TRUE)
```

Arguments

`x`	a matrix or vector of numeric data. Missing values are allowed.
`group`	a vector giving the grouping, with one element per row of `x`. Missing values are not allowed.
`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 (and may run faster for large matrices). The default is to reorder the data, so as to agree with `tapply` (see example below).

Value

a matrix containing the sums. There will be one row per unique value of `group`.

Author(s)

Terry Therneau

See Also

`tapply`

Examples

```
x <- matrix(runif(100), ncol=5)
group <- sample(1:8, 20, T)
xsum <- rowsum(x, group)

## same result another way, slower, and temp may be much larger than x
temp <- model.matrix(~ a - 1, data.frame(a=as.factor(group)))
xsum2<- t(temp) %*% x

## same as last one, but really slow
xsum3 <- tapply(x, list(group[row(x)], col(x)), sum)
```

`rug` *Add a Rug to a Plot* `rug`

Description

Adds a *rug* representation (1-d plot) of the data to the plot.

Usage

```
rug(x, ticksize=0.03, side=1, lwd=0.5)
```

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.

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, bu non-finite values are omitted silently.

Author(s)

B. D. Ripley

See Also

`jitter` which you may want for ties in `x`.

Examples

```
data(faithful)
attach(faithful)
plot(density(eruptions, bw=0.15))
rug(eruptions)
detach("faithful")
```

`sample` *Random Samples and Permutations* `sample`

Description

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

Usage

```
sample(x, size, replace=FALSE, prob)
```

Arguments

`x`	Either a (numeric, complex, character or logical) vector of more than one elements from which to choose, or a positive integer.
`size`	A positive integer giving the number of items to choose.
`replace`	Should sampling be with replacement?
`prob`	A vector of probabilities of obtaining the elements of the vector being sampled.

Details

If `x` has length 1, sampling takes place from `1:x`.

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 probabilities of obtaining the elements of the vector being sampled. 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.

Examples

```
x <- 1:12
# a random permutation
sample(x)
# bootstrap sampling
sample(x,replace=TRUE)

# 100 Bernoulli trials
sample(c(0,1), 100, replace = TRUE)
```

`save` *Save R Objects* `save`

Description

`save` writes a 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`.

`save.image()` is just a short-cut for "save my current environment", i.e., `save(list = ls(all=T), file = ".RData")`. It is what also happens with `q("yes")`.

Usage

```
save(..., list = character(0), file = "", ascii = FALSE, oldstyle = FALSE)
save.image(file = ".Rdata", oldstyle = FALSE)
```

Arguments

`...`	the names of the objects to be saved.
`list`	A character vector containing the names of objects to be saved.
`file`	the name of the file where the data will be saved.
`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.
`oldstyle`	logical. Should the old (0.90.1 or earlier) format be used?

See Also

`dput`, `dump`, `load`.

Examples

```
x <- runif(20)
y <- list(a = 1, b = TRUE, c = "oops")
save(x, y, file = "xy.Rdata")
save.image()
unlink("xy.Rdata")
unlink(".RData")
```

`scale` *Scaling and Centering of Matrices* `scale`

Description

Center and/or scale the columns of a numeric matrix.

Usage

```
scale(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	a numeric matrix.
`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 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

The centered, scaled matrix.

See Also

`sweep` which allows centering (and scaling) with arbitrary statistics.

Examples

```
x <- matrix(1:10, nc=2)
(centered.x <- scale(x, scale=FALSE))
cov(centered.scaled.x <- scale(x))# all 1
```

`scan` *Read Data Values* `scan`

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 = "", dec = ".", skip = 0, nlines = 0, na.strings = "NA",
     flush = FALSE, strip.white = FALSE, quiet = 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 (in this case input can be terminated by a blank line). Otherwise, the file name is *relative* to the current working directory, `getwd()`, unless it specifies an *absolute* path.
`what`	the type of `what` gives the type of data to be read. If `what` is a list, it is assumed that the lines of the data file are records each containing `length(what)` items ("fields").
`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, `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.
`quote`	the set of quoting characters as a single character string.
`dec`	decimal point character.
`skip`	this many lines of the input file should be skipped before starting to read data values.
`nlines`	the maximum number of lines of data to be read.
`na.strings`	character string, indicating *which* character fields in the file should translate to missing (`NA`) values.
`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.
`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 to strip 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 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, telling what fields have been read.

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 `sep` is nondefault, the fields may be quoted in the style of .csv format files where separators inside quotes (■ or `""`) are ignored and quotes may be put inside strings by doubling them.

Keyboard entry is terminated by typing a blank line.

See Also

`read.table` for more user-friendly reading of data matrices; `write`.

Examples

```
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
str(scan("ex.data", what = list("","",""))) # flush is F -> read "7"
str(scan("ex.data", what = list("","",""), flush = TRUE))
unlink("ex.data") # tidy up
```

`screen` *Creating and Controlling Multiple Screens on a Single Device* `screen`

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.

`close.screen` removes the specified screen definition(s).

Usage

```
split.screen(figs, screen = <<see below>>, erase = TRUE)
screen(n = <<see below>>, new = TRUE)
erase.screen(n = <<see below>>)
close.screen(n = <<see below>>, all = TRUE)
```

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.
`n`	A number indicating which screen to prepare for drawing (`screen`), erase (`erase.screen`), or close (`close.screen`).
`new`	A logical value indicating whether the screen should be erased as part of the preparation for drawing in the screen.
`all`	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 WARNINGS below). Split-screen mode is exited by the command `close.screen(all = TRUE)`

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.

Warning

These functions are totally incompatible with the other mechanisms for arranging plots on a device: `par(mfrow)`, `par(mfcol)`, and `layout()`.

The functions are also incompatible with some plotting functions, such as `coplot`, which make use of these other mechanisms.

The functions should not be used with multiple devices.

See Also

`par`, `layout`, `Devices`, `dev.*`

Examples

```
if (interactive()) {
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 = T) # 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")
screen(1, F)              #return to screen 1, but do not clear
plot(10:1, axes=F, lty=2, ylab="")  # overlay second plot
axis(4)                   # add tic marks to right-hand axis
title("Plot 1")
close.screen(all = T) # exit split-screen mode
}
```

`sd` *Standard Deviation* `sd`

Description

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 dataframe, a vector of the standard deviation of the columns is returned.

Usage

```
sd(x, na.rm = FALSE)
```

See Also

`var` for its square, and `mad`, the most robust alternative.

Examples

```
sd(1:2) ^ 2
```

`se.aov` *Internal Functions Used by model.tables* `se.aov`

Description

Internal function for use by `model.tables`.

Usage

```
se.aov(object, n, type = "means")
se.aovlist(object, dn.proj, dn.strata, factors, mf, efficiency,
           n, type = "diff.means", ...)
```

Author(s)

B. D. Ripley

See Also

`model.tables`

`se.contrast` *Standard Errors for Contrasts in Model Terms* `se.contrast`

Description

Returns the standard errors for one or more contrasts in an `aov` object.

Usage

```
se.contrast(object, ...)
se.contrast.aov(x, contrast.obj,
  coef = contr.helmert(ncol(contrast))[, 1], data = NULL)
se.contrast.aovlist(...)
```

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 as returned by `contrasts`.
`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`.

Details

Contrasts are usually used to test if certain means are significantly different; it can be easier to use `se.contrast` than compute directly with the coefficients.

In multistratum models, the contrasts can appear in more than one stratum; the contrast and standard error are computed in the lowest stratum and adjusted for efficiencies and comparisons between strata.

Value

A vector giving the standard errors for each contrast.

Author(s)

B.D. Ripley

Examples

```
## From Venables and Ripley (1997) p.210.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
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(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
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)
options(contrasts=c("contr.treatment", "contr.poly"))
npk.aov1 <- aov(yield ~ block + N + K, npk)
se.contrast(npk.aov1, list(N=="0", N=="1"), data=npk)
```

`search` *Give Search Path for R Objects* `search`

Description

Gives a list of `attach`ed *packages* (see `library`), and R objects, usually `data.frame`s.

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.

See Also

`attach` and `detach` to change the search "path", `objects` to find R objects in there.

Examples

```
search()
searchpaths()
```

`segments` *Add Line Segments to a Plot* `segments`

Description

Draw line segments between pairs of points.

Usage

```
segments(x0, y0, x1, y1,
       col = par("fg"), lty = par("lty"), xpd = FALSE)
```

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.
`col, lty, xpd`	usual graphical parameters as in `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).

See Also

`arrows`, `polygon` for slightly easier and less flexible line drawing and `lines` for the usual polygons.

Examples

```
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')
```

`seq` *Sequence Generation* `seq`

Description

Generate regular sequences.

Usage

```
from:to
seq(from, to)
seq(from, to, by=)
seq(from, to, length=)
seq(along)
```

Details

The operator `:` and the first `seq(.)` form generate the sequence `from, from+1`, ..., `to`. `seq` is a generic function.

The second form generates `from, from+by`, ..., `to`.

The third generates a sequence of `length` equally spaced values from `from` to `to`.

The last generates the sequence `1, 2`, ..., `length(along)`.

If `from` and `to` are factors of the same length, then `from : to` returns the "cross" of the two.

Value

The result is of `mode "integer"` if `from` is (numerically equal to an) integer.

See Also

rep, sequence, row, col.

Examples

```
1:4
pi:6 # float
6:pi # integer

seq(0,1, length=11)
str(seq(rnorm(20)))
seq(1,9, by = 2) # match
seq(1,9, by = pi)# stay below
seq(1,6, by = 3)
seq(1.575, 5.125, by=0.05)


for (x in list(NULL, letters[1:6], list(1,pi)))
  cat("x=",deparse(x),";  seq(along = x):",seq(along = x),"\n")

f1 <- gl(2,3); f1
f2 <- gl(3,2); f2
f1:f2 # a factor, the ‘‘cross’’  f1 x f2
```

---

sequence *Create A Vector of Sequences* sequence

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* `sets`

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`.

Author(s)

B. D. Ripley

See Also

`%in%`

Examples

```
(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)))
```

```
is.element(x, y)# length 10
is.element(y, x)# length  8
```

SHLIB *Build Shared Library for Dynamic Loading* SHLIB

Description

The given source files are first compiled using `R COMPILE`. All specified object files are then linked into a shared library which can be loaded into R using `dyn.load()`.

Usage

```
R SHLIB [-o libname] files
```

Arguments

`files`	a list of names of (typically) source files to be compiled and included in the library. You can also include the name of object files which are automagically made from their sources.
`libname`	the full name of the shared library to be built, including the extension (typically '.so' on Unix systems). If not given, the name of the library is taken from the first source file.

See Also

COMPILE, dyn.load

---

sign *Sign Function* sign

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

See Also

abs

Examples

```
sign(pi) # == 1
sign(-2:3)# -1 -1 0 1 1 1
```

SignRank *Distribution of the Wilcoxon Signed Rank Statistic* SignRank

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

```
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 to generate.
`n`	numbers of observations in the sample. Must be positive integers less than 50.
`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.

Value

`dsignrank` gives the density, `psignrank` gives the distribution function, `qsignrank` gives the quantile function, and `rsignrank` generates random deviates.

Author(s)

Kurt Hornik ⟨hornik@ci.tuwien.ac.at⟩

See Also

`dwilcox` etc, for the *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,")"))
}
```

`sink` *Send R Output to a File* `sink`

Description

Diverts all R output to `file`, overwriting the file unless `append` is `TRUE`.

Usage

```
sink(file = NULL, append = FALSE)
```

Arguments

`file`	a character string naming the file to write to, or `NULL` to stop "sink"ing.
`append`	if `TRUE`, output will be appended to `file`; otherwise, it will overwrite the contents of `file`.

Details

Only prompts and error messages continue to appear on the terminal.

`sink()` or `sink(file=NULL)` ends the diversion.

Examples

```
sink("sink-examp.txt")
i <- 1:10
outer(i,i,"*")
sink()
unlink("sink-examp.txt")
```

`sleep` *Students' Sleep Data* `sleep`

Description

Data which show the effect of two soporific drugs (increase in hours of sleep) on groups consisting of 10 patients each.

Usage

```
data(sleep)
```

Format

A data frame with 20 observations on 2 variables.

[, 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

Scheffé, Henry (1959) *The Analysis of Variance.* New York, NY: Wiley.

Examples

```
data(sleep)
## ANOVA
anova(lm(extra ~ group, data = sleep))
```

`solve` *Solve a System of Equations* `solve`

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, tol = 1e-7)
```

Arguments

`a`	a numeric matrix containing the coefficients of the linear system.
`b`	a numeric vector or matrix giving the right-hand side(s) of the linear system. If omitted, `b` is taken to be an identity matrix and `solve` will return the inverse of `a`.
`tol`	the tolerance for detecting linear dependencies in the columns of `a`.

See Also

`backsolve`, `qr.solve`.

Examples

```
hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
h8 <- hilbert(8); h8
solve(h8) # gives error: 'singular'
sh8 <- solve(h8, tol = 1e-10)
round(sh8 %*% h8, 3)
```

`sort` *Sort a Vector* `sort`

Usage

```
sort(x, partial = NULL, na.last = NA)
```

Details

If `partial` is non `NULL`, it is taken to contain indexes 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.

If `na.last` is `NA`, `sort` removes missing values. If `na.last` is `TRUE`, missing values are put last, if `FALSE`, missing values are put first.

See Also

`order`, `rank`.

Examples

```
data(swiss)
x <- swiss$Education[1:25]
x; sort(x); sort(x, partial = c(10,15))
median # shows you another example for 'partial'
```

`source` *Redirect Input* `source`

Description

`source` causes R to accept its input from the named file (the name must be quoted). Input is read from that file until the end of the file is reached. `parse` is used to scan the expressions in, they are then evaluated sequentially in the chosen environment.

Usage

```
source(file, local = FALSE, echo = verbose, print.eval = echo,
       verbose = getOption("verbose"),  prompt.echo = getOption("prompt"),
       max.deparse.length = 150, chdir = FALSE)
```

Arguments

`file`	character; the name of the file to read from (quoted).
`local`	if `local` is `FALSE`, the statements scanned are evaluated in the global environment, otherwise in the local 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`, the R working directory is changed to the directory containing `file` for evaluating.

See Also

`demo` which uses `source`; `eval`, `parse` and `scan`; `options("keep.source")`.

`Special` *Special Functions of Mathematics* `Special`

Description

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 functions `gamma` and `lgamma` return the gamma function $\Gamma(x)$ and the natural logarithm of the absolute value of the gamma function.

The functions `digamma`, `trigamma`, `tetragamma` and `pentagamma` return the first, second, third and fourth derivatives of the logarithm of the gamma function.

$$\texttt{digamma(x)} = \psi(x) = \frac{d}{dx}\ln\Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

The functions `choose` and `lchoose` return binomial coefficients and their logarithms.

Usage

```
beta(a, b)
lbeta(a, b)
gamma(x)
lgamma(x)
digamma(x)
trigamma(x)
tetragamma(x)
pentagamma(x)
choose(n,k)
lchoose(n,k)
```

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions.* New York: Dover. Chapter 6: Gamma and Related Functions.

See Also

`Arithmetic` for simple, `Math` for miscellaneous mathematical functions and `Bessel` for the real Bessel functions.

Examples

```
choose(5, 2)
for (n in 0:10) print(choose(n, k = 0:n))

curve(gamma(x),-3,4, n=1001, ylim=c(-10,100),
      col="red", lwd=2, main="gamma(x)")
abline(h=0,v=0, lty=3, col="midnightblue")

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
  if (is.deriv) dy <- diff(y) / dx
  nm <- paste(ch, "gamma", sep = "")
  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)
}
par(mfrow = c(2, 2))
```

`splinefun` *Interpolating Splines* `splinefun`

Usage

```
splinefun(x, y, method = "fmm")
spline(x, y, 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.
`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

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"`.

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

Forsythe, G. E., Malcolm, M. A. and Moler, C. B. (1977) *Computer Methods for Mathematical Computations*.

See Also

`approx` and `approxfun` for constant and linear interpolation.

Examples

```
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)
ls(envir = environment(f))
splinecoef <- eval(expression(z), envir = environment(f))
curve(f(x), 1, 10, col = "green", lwd = 1.5)
points(splinecoef, col = "purple", cex = 2)
par(op)
```

---

split — *Divide into Groups* — split

Description

`split` divides the data in the vector `x` into the groups defined by the factor `f`.

Usage

```
split(x, f)
split.default(x, f)
split.data.frame(x, f)
```

Arguments

`x`	vector containing the values to be divided into groups.
`f`	a "factor" such that `as.factor(f)` defines the grouping.

Details

`f` is recycled as necessary and if the length of `x` is not a multiple of the length of `f` a warning is printed.

Value

The value returned is a list of vectors containing the values for the groups. The components of the list are named by the factor levels of `f`. If `f` is longer than `x` some of these will be of zero length.

See Also

`cut`

Examples

```
n <- 10; nn <- 100
g <- factor(round(n * runif(n * nn)))
x <- rnorm(n * nn) + sqrt(codes(g))
xg <- split(x, g)
boxplot(xg, col = "lavender", notch = TRUE, varwidth = TRUE)
sapply(xg, length)
sapply(xg, mean)

## 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)
```

`stackloss` *Brownlee's Stack Loss Plant Data* `stackloss`

Description

Operational data of a plant for the oxidation of ammonia to nitric acid.

Usage

```
data(stackloss)
```

Format

A data frame with 21 observations on 4 variables.

[,1]	x1	`Air.Flow`	Flow of cooling air
[,2]	x2	`Water.Temp`	Cooling Water Inlet Temperature
[,3]	x3	`Acid.Conc.`	Concentration of acid [per 1000, minus 500]
[,4]	y	`stack.loss`	Stack loss

Details

"Obtained from 21 days of operation of a plant for the oxidation of ammonia (NH_3) to nitric acid (HNO_3). The nitric oxides produced are absorbed in a countercurrent absorption tower." (Brownlee, cited by Dodge, slightly reformatted by MM.)

`x1` represents the rate of operation of the plant. `x2` is the temperature of cooling water circulated through coils in the absorption tower. `x3` is the concentration of the acid circulating, minus 50, times 10: that is, 89 corresponds to 58.9 per cent acid. `y` (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

Brownlee, K. A. (1960, 2nd ed. 1965) *Statistical Theory and Methodology in Science and Engineering.* New York: Wiley. pp. 491–500.

References

Dodge, Y. (1996) The guinea pig of multiple regression. In: *Robust Statistics, Data Analysis, and Computer Intensive Methods; In Honor of Peter Huber's 60th Birthday*, 1996, *Lecture Notes in Statistics* **109**, Springer-Verlag, New York.

Examples

```
data(stackloss)
summary(lm.stack <- lm(stack.loss ~ stack.x))
```

`stars` *Star Plots and Segment Diagrams of Multivariate Data* `stars`

Description

Star plots or segment diagrams are created on the current graphics device.

Usage

```
stars(x, full = TRUE, scale = TRUE, radius = TRUE, labels =
      dimnames(x)[[1]], locations = NULL, xlimit = NULL, ylimit = NULL,
      len = 1, colors = NULL, key.loc = NULL, key.labels = NULL,
      draw.segments = FALSE, draw.axes = FALSE, ...)
```

Arguments

`x`	matrix of data. One segment plot will be produced for each row of the matrix. Missing values (`NA`) are allowed, but they are treated as if they were 0.
`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`	two column matrix with the x and y coordinates used to place each of the segment plots. If `locations` is `NULL` the segment plots will be placed in a rectangular grid.
`xlimit`	vector with the range of x coordinates to plot.
`ylimit`	vector with the range of y coordinates to plot.
`len`	scale factor for the length of radii or segments.
`colors`	vector of integers, each of which specifies a color for one of the segments. Ignored if `draw.segments = FALSE`
`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.
`draw.axes`	logical flag: if `TRUE` axes are added to the plot.
`...`	any other arguments, typically arguments to `par`.

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.stat.rice.edu/~andrewsd/software/software.html.

Author(s)

Thomas S. Dye

Examples

```
data(mtcars)
stars(mtcars[, 1:7], len = 0.8, key.loc = c(12, 2),
      main = "Motor Trend Cars", full = FALSE)

stars(mtcars[, 1:7], len = 0.8, key.loc = c(12, 2),
      main = "Motor Trend Cars", draw.segments = TRUE)

data(USJudgeRatings)
stars(USJudgeRatings, labels = abbreviate(case.names(USJudgeRatings)),
      key.loc = c(13, 1.5), main = "Judge not ...", len = 0.8)
```

`start` *Encode the Terminal Times of Time Series* `start`

Description

Extract and encode the times the first and last observations were taken. Provided only for compatibility with S version 2.

Usage

```
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 `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`.

`Startup` *Initialization at Start of an R Session* `Startup`

Description

In R, the startup mechanism is as follows. If a file '.Renviron', or failing that ' /.Renviron', exists, it is sourced, usually to set environmental variables. This is suppressed by the flag `--no-environ`. 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 that variable is unset, the default is '$R_HOME/etc/Rprofile'.

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.

It also loads a saved image from '.RData' if there is one (unless `--no-restore` was specified).

Finally, if a function `.First` exists, it is executed as `.First()`.

The functions `.First` and `.Last` can be defined in the appropriate startup profiles or reside in '.RData'.

The command-line flag `--vanilla` implies `--no-init-file`, `--no-restore` and `--no-environ`.

Usage

```
.First <- function() { ..... }

.Rprofile <startup file>
```

See Also

`.Last` for final actions before termination.

`stat.anova` *GLM Anova Statistics* `stat.anova`

Description

This is a utility function, used in `anova.glm(..., test != NULL)` and should not be used by the average user.

Usage

```
stat.anova(table, test, 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.

See Also

`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)
```

`state` *States of the U.S.A.* `state`

Description

Data sets related to the 50 states of the United States of America.

Usage

```
data(state)
```

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

U.S. Department of Commerce, Bureau of the Census (1977) *Statistical Abstract of the United States*.

U.S. Department of Commerce, Bureau of the Census (1977) *County and City Data Book*.

`stem` *Stem-and-Leaf Plots* `stem`

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

```
stem(x, scale = 1, width = 80, atom = 1e-08)
```

Examples

```
data(islands)
stem(islands)
stem(log10(islands))
```

`step` *Choose a model by AIC in a Stepwise Algorithm* `step`

Description

Select a formula-based model by AIC.

Usage

```
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. This is used as the initial model in the stepwise search.
`scope`	defines the range of models examined in the stepwise search.
`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`	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`	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`	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.

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 function `extractAIC.glm` 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.

Author(s)

B. D. Ripley

See Also

`add1`, `drop1`

Examples

```
example(lm)
step(lm.D9)

data(swiss)
```

```
summary(lm1 <- lm(Fertility ~ ., data = swiss))
slm1 <- step(lm1)
summary(slm1)
slm1$anova
```

`stop` *Stop Function Execution* `stop`

Description

`stop` stops execution of the current expression, prints the message given as its argument, then executes an error action.

`geterrmessage` gives the last error message.

Usage

```
stop(message)
geterrmessage()
```

Arguments

`message`	a character vector (of length 1) or `NULL`.

Details

The error action is controlled by the current error handler set by `options(error=)`. The default behaviour (the `NULL` error-handler) in interactive use is to return to the top level prompt, and in non-interactive use to (effectively) call `q("no", status=1, runLast=FALSE)`.

Value

`geterrmessage` gives the last error message, as character string ending in "
n".

See Also

`warning`, `restart` to catch errors and retry, and `options` for setting error handlers.

Examples

```
if(iter > 10) stop("too many iterations")
```

`str` *Compactly Display the Structure of an Arbitrary R Object* `str`

Description

This is 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.

`ls.str` and `lsf.str` are useful "versions" of `ls`, calling `str` on each object. They are not foolproof and should rather not be used for programming, but are provided for their usefulness.

Usage

```
str(object, ...)
str.data.frame(object, ...)
str.default(object, max.level = 0, vec.len = 4, digits.d = 3,
    give.attr = TRUE, give.length = TRUE,
    wid = getOption("width"),
    nest.lev = 0,
    indent.str = paste(rep(" ", max(0, nest.lev + 1)), collapse = ".."))

ls.str(name, pattern, mode = "any", max.level = 1, give.attr = FALSE)
lsf.str(...)
```

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 0: Display all nesting levels.
`vec.len`	numeric 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`).
`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.

Value

Nothing, 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

summary, args.

Examples

```
## The following examples show some of 'str' capabilities
str(1:12)
str(ls)
str(args)#- more useful than  args(args) !
data(freeny); str(freeny)
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()

lsf.str()#- how do the functions look like which I am using?
ls.str(mode = "list")#- what are the structured objects I have defined?
```

`stripplot` *1-D Scatter Plots* `stripplot`

Description

`stripplot` produces one dimensional scatter plots (or dot plots) of the given data. These plots are are good alternative to boxplots when sample sizes are small.

Extensive examples of the use of this kind of plot can be found in Box, Hunter and Hunter or Seber and Wild.

Usage

```
stripplot(x, method="overplot", jitter=0.1, offset=1/3,
        vertical=FALSE, group.names,
        xlim=NULL, ylim=NULL, main="", ylab="", xlab="",
        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`. `NA`s 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 jittering 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.
`...`	Graphical parameters can also be specified as arguments.

Examples

```
x <- round(rnorm(50), 1)
stripplot(x)
```

`strsplit` *Split the Strings in a Vector* `strsplit`

Description

Split the strings in `x` into substrings according to the presence of substring `split` within them.

Usage

```
strsplit(x, split)
```

Arguments

`x`	character vector, to be split.
`split`	character string containing a regular expression to use as "split". If empty matches occur, in particular if `split` has length 0, `x` is split into single characters. If `split` is a vector, it is re-cycled along `x`.

Value

A list of length `length(x)` the `i`-th element of which contains the vector of splits of `x[i]`.

See Also

`paste` for the reverse, `grep` and `sub` for string search and manipulation; further `nchar`, `substr`.

Examples

```
noquote(strsplit("A text I want to display with spaces", NULL)[[1]])

x <- c("asfef", "qwerty", "yuiop[", "b", "stuff.blah.yech")
# split x on the letter e
strsplit(x,"e")

unlist(strsplit("a.b.c", "."))
## [1] "" "" "" "" ""
## Note that 'split' is a regexp!
## If you really want to split on '.', use
unlist(strsplit("a.b.c", "\\."))
## [1] "a" "b" "c"
```

`structure` *Attribute Specification* `structure`

Description

`structure` returns the given object with its attributes set.

Usage

```
structure(data, ...)
```

Arguments

`data`	an object which will have various attributes attached to it.
`...`	attributes, specified in `tag=value` form, which will be attached to data.

Examples

```
structure(1:6, dim = 2:3)
```

`strwidth` *Plotting Dimensions of Character Strings and Math Expressions* `strwidth`

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

```
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.
`units`	character indicating in which units `s` is measured; must be one of `"user"`, `"inches"`, `"figure"` .
`cex`	character expansion to which is applies. Per default, the current `par("cex")` is used.

Value

integer vector with the same length as `s`, giving the width for each `s[i]`.

See Also

`text`, `nchar`

Examples

```
str.ex <- c("W","w","I",".","WwI.")
op <- par(pty='s'); plot(1:100,1:100); par('usr')
sw <- strwidth(str.ex); sw
sum(sw[1:4] == sw[5])#- since the last string contains the others
sw / strwidth(str.ex, cex = .5)
# between 1.5 and 4.2 (!),  font dependent

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")
table(shL)# all have same heights ..
mean(shL) / par("cin")[2] # should be 1 (exactly?)

swL <- strwidth(all.lett)
swL <- 3 * swL / min(swL)
all(swL == round(swL))#- TRUE !
swL <- as.integer(swL)
n.classes <- length(tL <- table(swL)); tL
iL <- order(swL)
structure(swL[iL], names = all.lett[iL])
lett.classes <- structure(vector("list", n.classes), names= names(tL))
for(i in 1:n.classes)
  lett.classes[[i]] <- all.lett[swL == as.numeric(names(tL)[i])]
lett.classes

sumex <- expression(sum(x[i], i=1,n), e^{i * pi} == -1)
strwidth(sumex)
strheight(sumex)

rm(sumex); par(op)#- reset to previous setting
```

`subset` *Subsetting Vectors and Data Frames* `subset`

Description

Return subsets of vectors or data frames which meet conditions.

Usage

```
subset(x, ...)
subset.default(x, subset)
subset.data.frame(x, subset, select)
```

Arguments

`x`	object to be subsetted
`...`	how to subset, depends on object

`subset`	logical expression
`select`	expression, indicating variables to select from a data frame

Details

For ordinary vectors, the result is simply `x[subset & !is.na(subset)]`.

For dataframes, the `subset` argument works similarly on the rows. Note that `subset` will be evaluated in the dataframe.

The `select` argument exists only for dataframes. It works by first replacing variable names in the selection expression with the corresponding column numbers in the dataframe and then using the resulting integer vector to index the columns. This allows the use of the standard indexing conventions so that for examples ranges of variables can be specified easily.

Value

Selected rows and columns of the object `x`.

Author(s)

Peter Dalgaard

See Also

`[`, `transform`

Examples

```
data(airquality)
subset(airquality, Temp > 80, select = c(Ozone, Temp))
subset(airquality, Day == 1, select = -Temp)
subset(airquality, select = Ozone:Wind)

attach(airquality)
subset(Ozone, Temp > 80)
```

`substitute` *Substituting and Quoting Expressions* `substitute`

Description

`substitute` returns the parse tree for the (unevaluated) expression `expr`, substituting any variables bound in `env`.

`quote` simply returns the parse tree for the expression.

Usage

```
substitute(expr, env=<<see below>>)
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 `delay()`, 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.

See Also

`missing` for argument "missingness".

Examples

```
(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` *Extract Substrings from a Character Vector* `substr`

Description

Extract substrings from a character vector returning a vector whose elements contain the substring starting with the character at position `start` up to the character at position `stop`.

Usage

```
substr(x, start, stop)
substring(text, first, last = 1000000)
```

Details

If `start` is larger than the string length then `NA` is returned. If `stop` is longer than `start` an error is signalled.

`substring` is compatible with S, with `first` and `last` instead of `start` and `stop`. For vector arguments, it expands the arguments cyclically.

See Also

`strsplit`, `paste`, `nchar`.

Examples

```
substr("abcdef",2,4)
print(ss <- substring("abcdef",1:6,1:6))
all(  ss == strsplit ("abcdef",NULL)[[1]])# strsplit is more efficient..

substr(rep("abcdef",4),1:4,4:5)
x <- c("asfef", "qwerty", "yuiop[", "b", "stuff.blah.yech")
all(substr(x, 2, 5) == substring(x, 2, 5)) #> TRUE
substr(x, 2, 5)
substring(x, 2, 4:6)
```

sum — *Sum of Vector Elements* — sum

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)
```

summary — *Object Summaries* — summary

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, ...)

summary.default   (object, ..., digits = max(3, getOption("digits") -3))
summary.data.frame(object, maxsum = 7, ...)
summary.factor    (object, maxsum = 100, ...)
summary.matrix    (object, ...)
```

Arguments

object	an object for which a summary is desired.
maxsum	integer, indicating how many levels should be shown for factors.
...	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.

See Also

anova,summary.glm,summary.lm.

Examples

```
options(digits=5)
data(attenu)
summary(attenu) #-> summary.data.frame(..)
summary(attenu $ station, maxsum = 20) #-> summary.factor(..)
```

`sunflowerplot` *Produce a Sunflower Scatter Plot* `sunflowerplot`

Description

Multiple points are plotted as "sunflowers" with multiple leaves such that overplotting is visualized instead of accidental and invisible.

Usage

```
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,
              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[],y[]`.
`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 computes.
`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.
`size`	of sunflower leaves in inches, 1[in] := 2.54[cm]. Default: $1/8^{¨} = 3.2$mm.

`seg.col`	color to be used for the **seg**ments which make the sunflowers leaves, see `par(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)

Andreas Ruckstuhl, Werner Stahel, Martin Maechler, Tim Hesterberg, 1989–1993. Port to R by Martin Maechler ⟨maechler@stat.math.ethz.ch⟩.

References

Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P. A. (1983) *Graphical Methods for Data Analysis.* Wadsworth.

Schilling, M. F. and Watkins, A. E. (1994) A suggestion for sunflower plots. *The American Statistician*, **48**, 303–305.

See Also

`density`

Examples

```
data(iris)
## '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")
```

`sunspots` *Monthly Mean Relative Sunspot Numbers* `sunspots`

Description

Monthly sunspot numbers from 1749 to 1983. Collected at Swiss Federal Observatory, Zurich until 1960, then Tokyo Astronomical Observatory.

Usage

```
data(sunspots)
```

Format

A time series of monthly data from 1749 to 1983.

Source

Andrews, D. F. and Herzberg, A. M. (1985) *Data: A Collection of Problems from Many Fields for the Student and Research Worker*. New York: Springer-Verlag.

Examples

```
data(sunspots)
plot(sunspots, main = "sunspots data", xlab = "Year",
     ylab = "Monthly sunspot numbers")
```

`svd` *Singular Value Decomposition of a Matrix* `svd`

Description

Compute the singular-value decomposition of a rectangular matrix.

Usage

```
svd(x, nu = min(n,p), nv = min(n,p))
```

Arguments

`x`	a matrix whose SVD decomposition is to be computed.
`nu`	the number of left eigenvectors to be computed. This must be one of `0`, `nrow(x)` and `ncol(x)`.
`nv`	the number of right eigenvectors to be computed. This must be one of `0`, and `ncol(x)`.

Details

`svd` provides an interface to the LINPACK routine DSVDC. The singular value decomposition plays an important role in many statistical techniques.

Value

The SVD decomposition of the matrix as computed by LINPACK,

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}',$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal, $\boldsymbol{V}'$ means V *transposed*, and $\boldsymbol{D}$ is a diagonal matrix with the singular values D_{ii}. Equivalently, $\boldsymbol{D} = \boldsymbol{U}'\boldsymbol{X}\boldsymbol{V}$, which is verified in the examples, below.

The components in the returned value correspond directly to the values returned by DSVDC.

`d`	a vector containing the singular values of `x`.
`u`	a matrix whose columns contain the left eigenvectors of `x`.
`v`	a matrix whose columns contain the right eigenvectors of `x`.

References

Dongarra, J. J., Bunch, J. R., Moler, C. B. and Stewart, G. W. (1978) *LINPACK Users Guide.* Philadelphia: SIAM Publications.

See Also

`eigen`, `qr`.

Examples

```
hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
str(X <- hilbert(9)[,1:6])
str(s <- svd(X))
Eps <- 10 * .Machine$double.eps

D <- diag(s$d)
all(abs(X - s$u %*% D %*% t(s$v)) < Eps)# TRUE:  X = U D V'
all(abs(D - t(s$u) %*% X %*% s$v) < Eps)# TRUE:  D = U' X V

X <- cbind(1, 1:7)
str(s <- svd(X)); D <- diag(s$d)
all(abs(X - s$u %*% D %*% t(s$v)) < Eps)# TRUE:  X = U D V'
all(abs(D - t(s$u) %*% X %*% s$v) < Eps)# TRUE:  D = U' X V
```

`sweep` — *Sweep out Array Summaries* — `sweep`

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 giving the extents of `x` which correspond to `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.

See Also

`apply` on which `sweep` is based; `scale` for centering and scaling.

Examples

```
data(attitude)
med.att <- apply(attitude, 2, median)
sweep(data.matrix(attitude), 2, med.att)# subtract the column medians
```

`swiss` *Swiss Fertility and Socioeconomic Indicators (1888) Data* `swiss`

Description

Standardized fertility measure and socio-economic indicators for each of 47 French-speaking provinces of Switzerland at about 1888.

Usage

```
data(swiss)
```

Format

A data frame with 47 observations on 6 variables, *each* of which is in percent, i.e., in $[0, 100]$.

[,1]	Fertility	I_g, "common standardized fertility measure"
[,2]	Agriculture	% involved in agriculture as occupation
[,3]	Examination	% "draftees" receiving highest mark on army examination
[,4]	Education	% education beyond primary school.
[,5]	Catholic	% catholic (as opposed to "protestant").
[,6]	Infant.Mortality	live births who live less than 1 year.

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 seven French-speaking "provinces" at about 1888.

Here, all variables are scaled to $[0, 100]$, where in the original, all but `"Catholic"` were scaled to $[0, 1]$.

Source

Project "16P5", pages 549–551 in

Mosteller, F. and Tukey, J. W. (1977) *Data Analysis and Regression: A Second Course in Statistics.* Addison-Wesley, Reading Mass.

indicating their source as "Data used by permission of Franice van de Walle. Office of Population Research, Princeton University, 1976. Unpublished data assembled under NICHD contract number No 1-HD-O-2077."

Examples

```
data(swiss)
pairs(swiss, panel = panel.smooth, main = "swiss data")
summary(lm(Fertility ~ . , data = swiss))
```

`switch` — *Select One of a List of Alternatives* — `switch`

Description

`switch` evaluates `EXPR` and accordingly chooses one of the further arguments (in `...`).

Usage

```
switch(EXPR, ...)
```

Arguments

`EXPR`	an expression evaluating to a number or a character string.
`...`	the list of alternatives, given explicitly.

Details

If the value of `EXPR` is an integer between 1 and `nargs()-1` then the corresponding element of `...` is evaluated and the result returned.

If `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., `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`.

Examples

```
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","QQ","a","A","bb")
for(ch in ccc) cat(ch,":",switch(ch, a=1,    b=2:3),            "\n")
for(ch in ccc) cat(ch,":",switch(ch, a=,A=1, b=2:3, "Otherwise: last"),"\n")

## Numeric EXPR don't allow an 'otherwise':
for(i in c(-1:3,9))  print(switch(i, 1,2,3,4))
```

`symnum` *Symbolic Number Coding* `symnum`

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,
       lower.triangular = corr & 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 minmal cutpoint is coded especially.

`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 `= cutpoints[j]` and s_j `= symbols[j]`.

Author(s)

Martin Maechler ⟨maechler@stat.math.ethz.ch⟩

See Also

`as.character`

Examples

```
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:
data(attitude)
symnum(cor(attitude), diag = FALSE)

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[c(3,6), 2] <- 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))
```

---

sys.parent *Functions to Access the Function Call Stack* sys.parent

---

Description

These functions provide access to environments ("frames" in S terminology) associated with functions further up the calling stack.

Usage

```
sys.call(which=<<see below>>)
sys.frame(which=<<see below>>)
sys.nframe()
sys.function(n=<<see below>>)
sys.parent(n=1)

sys.calls()
sys.frames()
sys.parents()
sys.on.exit()
sys.status()
parent.frame(n=1)
```

Arguments

which	the frame number.
n	the number of frame generations to go back.

Details

.GlobalEnv is given number 0 in the list of frames. Each subsequent function evaluation increases the frame stack by 1 and the environment for evaluation of that function is returned by sys.frame with the appropriate index.

The parent of a function evaluation is the environment in which the function was called. It is not necessarily one less than the frame number of the current evaluation, nor the environment from which it is called. sys.parent returns the number of the parent frame if n is 1 (the default), the grandparent if n is 2, and so forth. sys.frame returns the environment associated with a given frame number.

sys.call and sys.frame both accept either positive or negative values for the argument which. Positive values of which are normal frame numbers whereas negative values are counted back from the frame number of the current evaluation.

Notice that even though the sys.*xxx* functions 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.

sys.on.exit() retrieves the expression stored for use by on.exit in the function currently being evaluated.

parent.frame(n) is a convenient shorthand for sys.frame(sys.parent(n)).

See Also

eval for the usage of sys.frame.

Examples

```
ff <- function(x) gg(x)
gg <- function(y) sys.status()
str(ff(1))

t1 <- function() {
  aa <- "here"
  t2 <- function() {
    ## in frame 2 here
    cat("current frame is", sys.nframe(), "\n")
    str(sys.calls()) ## list with two components t1() and t2()
    cat("parents are frame nos", 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
```

---

sys.source — *Parse and Evaluate Expressions from a File* — sys.source

Description

Parses expressions in the given file, and then successively evaluates them in the specified environment. By default, evaluation is done in the global environment.

Usage

```
sys.source(file, envir = NULL)
```

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.

Details

The keep.source option is set to FALSE when reading from the file, so the source code is not kept (see options for more details).

See Also

source

---

system *Invoke a System Command* system

Description

system invokes the system command specified by command.

Usage

```
system(command, intern = FALSE, ignore.stderr = 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.
ignore.stderr	a logical indicating whether error messages (written to 'stderr') should be ignored. This probably only affects Unix-like systems and is ignored otherwise.

Details

If intern is TRUE then popen is used to invoke the command and the output collected, line by line, into an R character vector which is returned as the value of system. If intern is FALSE then the C function system is used to invoke the command and the value returned by system is the exit status of this function. This function provides users with the ability to invoke system commands on whatever platform they are using.

unix is a *deprecated* alternative, available for backwards compatibility.

See Also

.Platform for platform specific variables.

Examples

```
# list all files in the current directory using the -F flag
system("ls -F")

# t1 contains a vector of strings, each one
# representing a separate line of output from who
t1 <- system("who", TRUE)

system("ls fizzlipuzzli",TRUE,TRUE)# empty since file doesn't exist
```

system.file — *Find Names of R System Files* — system.file

Description

Finds the full file names of files in packages etc.

Usage

```
system.file(..., pkg = .packages(), lib = .lib.loc)
```

Arguments

`...`	character strings, specifying subdirectory and file(s) within some package. The default, none or `""`, returns the root of package(s). Wildcards are not supported.
`pkg`	a character vector with package names.
`lib`	a character vector with path names of R package libraries, see `.lib.loc` for the default.

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(pkg="lqs")   # The root of package 'lqs'
system.file("INDEX")
system.file("help/AnIndex", pkg = c("stepfun", "mva"))
```

system.time — *CPU Time Used* — system.time

Description

Return CPU (and other) times that `expr` used.

Usage

```
system.time(expr)
unix.time(expr)
```

Arguments

expr	Valid R expression to be "timed"

Details

system.time calls the builtin proc.time, evaluates expr, and then calls proc.time once more, returning the difference between the two proc.time calls.

The values returned by the proc.time are (on Unix) those returned by the C library function times(3v).

unix.time is an .Alias of system.time, for compatibility reasons.

Value

A numeric vector of length 5 containing the user cpu, system cpu, elapsed, subproc1, subproc2 times. The subproc times are the the user and system cpu time used by child processes (and so are usually zero).

The resolution of the times will be system-specific; it is common for the elapsed time to be recorded to the nearest second, and CPU times to of the order of 1/100 second.

Note

It is possible to compile R without support for system.time, when all the values will be NA.

See Also

proc.time, time which is for time series.

Examples

```
system.time(for(i in 1:50) mad(runif(500)))
exT <- function(n = 100) {
  # 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):
exT()                 #- '1.4' on -O-optimized Ultra1
system.time(exT())    #~ +/- same
```

---

t *Matrix Transpose* t

Description

Given a matrix or data.frame x, t returns the transpose (matrix or data.frame) of x.

Usage

```
t(x)
```

Examples

```
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(! a[,j] == ta[j,]) stop("wrong transpose")
```

`table` *Cross Tabulation* `table`

Description

`table` uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

Usage

```
table(..., exclude = c(NA, NaN), dnn, deparse.level = 1)
```

Arguments

`...`	objects which can be interpreted as factors (including character strings), or a list (or data frame) whose components can be so interpreted
`exclude`	values to use in the exclude argument of `factor` when interpreting non-factor objects
`dnn`	the names to be given to the dimensions in the result ('the dimname names'
`deparse.level`	controls how the default `dnn` is constructed. See details.

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.

Examples

```
## Simple frequency distribution
table(rpois(100,5))
data(warpbreaks)
attach(warpbreaks)
## Check the design:
table(wool, tension)
data(state)
table(state.division, state.region)

data(airquality)
attach(airquality)
# simple two-way contingency table
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)")
```

`tabulate` *Tabulation for Vectors* `tabulate`

Description

`tabulate` takes the integer valued vector `bin` and counts the number of times each integer occurs in it. `tabulate` is used as the basis of the `table` function.

Usage

```
tabulate(bin, nbin=max(1,bin))
```

Arguments

`bin`	a vector of integers, or a factor.
`nbins`	the number of bins to be used.

Details

If `bin` is a factor, its internal integer representation is tabulated. If the elements of `bin` are not integers, they are rounded to the nearest integer. Elements outside the range `1,...,nbin` are (silently) ignored in the tabulation.

See Also

`factor`, `table`.

Examples

```
tabulate(c(2,3,5))
tabulate(c(2,3,3,5), nb = 10)
tabulate(c(-2,0,2,3,3,5), nb = 3)
tabulate(factor(letters[1:10]))
```

`tapply` *Apply a Function Over a "Ragged" Array* `tapply`

Description

Apply a function to each cell of a ragged array, i.e., for 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.
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.
...	optional arguments to FUN.

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.

See Also

the convenience function aggregate (using tapply); apply, lapply with its version sapply.

Examples

```
groups <- as.factor(rbinom(32, n = 5, p = .4))
tapply(groups, groups, length) #- is almost the same as
table(groups)

data(warpbreaks)
## contingency table from data.frame : array with named dimnames
tapply(warpbreaks$breaks, warpbreaks[,-1], sum)
tapply(warpbreaks$breaks, warpbreaks[,3,drop=F], 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)

ind <- list(c(1, 2, 2), c("A", "A", "B"))
table(ind)
tapply(1:3, ind) #-> the split vector
tapply(1:3, ind, sum)
```

`TDist` *The Student t Distribution* `TDist`

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, log = FALSE)
pt(q, df, ncp=0, lower.tail = TRUE, log.p = FALSE)
qt(p, df,        lower.tail = TRUE, log.p = FALSE)
rt(n, df)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations to generate.
`df`	degrees of freedom (> 0, maybe non-integer).
`ncp`	non-centrality parameter δ; currently `ncp <= 37.62`.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p).
`lower.tail`	logical; if TRUE (default), probabilities are $P[X \le x]$, 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 (ν, δ) `= (df, ncp)` is defined as a the distribution of $T_\nu(\delta) := \frac{U+\delta}{\chi_\nu/\sqrt{\nu}}$ where U and χ_ν are independent random variables, $U \sim \mathcal{N}(0,1)$, and χ^2_ν is chi-squared, see `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 **n**on-**c**entrality **p**arameter `ncp`$= \mu - \mu_0$.

Value

`dt` gives the density, `pt` gives the distribution function, `qt` gives the quantile function, and `rt` generates random deviates.

References

Lenth, R. V. (1989). *Algorithm AS 243* — Cumulative distribution function of the non-central t distribution, *Appl. Statist.* **38**, 185–189.

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)")
```

---

tempfile *Create Names for Temporary Files* tempfile

Description

Returns a vector of character strings which can be used as names for temporary files.

Usage

```
tempfile(pattern = "file")
```

Arguments

pattern	a non-empty character vector giving the initial part of the 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.

The file name is made of the pattern, the process number in hex and a random suffix in hex. The filenames will be in the directory given by the first found of the environment variables TMP, TEMP, or "/tmp".

Value

A character vector giving the names of possible (temporary) files.

Note that no files are generated by tempfile.

See Also

unlink for deleting files.

Examples

```
tempfile(c("ab", "a b c"))   # give file name with spaces in!
## One possibility of getting ALL environment variables;
## compare with 'getenv':
fun <- function() {
  FILE <- tempfile("fun")
  on.exit(unlink(FILE))
  system(paste("printenv >", FILE))
  x <- strsplit(scan(FILE, what = ""), "=")
  v <- n <- character(LEN <- length(x))
  for (i in 1:LEN) {
    n[i] <- x[[i]][1]
    v[i] <- paste(x[[i]][-1], collapse = "=")
  }
  structure(v, names = n)
}

fun()
```

`terms` *Model Terms* `terms`

Description

The function `terms` is a generic function which can be used to extract *terms* objects from various kinds of R data objects.

Usage

```
terms(object, ...)
terms.formula(formula.obj, keep.order=FALSE)
terms.terms(terms.obj)
terms.default(anything)
```

Value

All the functions above produce an object of class `terms` which contains the *terms* representation of a symbolic model.

See Also

`lm`, `glm`, `formula`.

`text` *Add Text to a Plot* `text`

Description

`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

```
text (x, ...)
text.default (x, y = NULL, labels = seq(along = x), adj = NULL,
              pos = NULL, offset = 0.5, vfont = NULL, ...)
```

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.
`adj`	one or two values in $[0, 1]$ which specify the x (and optionally y) adjustment of the labels.
`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.
`...`	further graphical parameters (from `par`), such as `cex`, etc.

Details

`labels` 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.

`adj` allows *adj*ustment 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 = c(0.5, 0.5)`. 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).

See Also

mtext, title, Hershey for details on Hershey vector fonts, plotmath for details and more examples on mathematical annotation.

Examples

```
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 l', but not ■ ..")
mtext("ńISO-accents■: ś 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
text(5,10.2,"Le français, c'est façile: Règles, Liberté, Egalité, Fraternité..")
text(5,9.8, "Jetz no chli züritüütsch: (noch ein biSSchen Zürcher deutsch)")
```

`time` *Sampling Times of Time Series* `time`

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, 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.

See Also

`ts`, `start`, `tsp`, `window`.

`date` for clock time, `system.time` for CPU usage.

Examples

```
data(presidents)
cycle(presidents)
# a simple series plot: c() makes the x and y arguments into vectors
plot(c(time(presidents)), c(presidents), type="l")
```

`Titanic` *Survival of passengers on the Titanic* `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

```
data(Titanic)
```

Format

A 4-dimensional array resulting from cross-tabulating 2201 observations on 4 variables. The variables and their levels are as follows:

No	Name	Levels
1	Class	1st, 2nd, 3rd, Crew
2	Sex	Male, Female
3	Age	Child, Adult
4	Survived	No, Yes

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 inter-

est in the Titanic. Very detailed data about the passengers is now available on the Internet, at sites such as *Encyclopedia Titanica* (http://www.rmplc.co.uk/eduweb/sites/phind).

Source

Dawson, Robert J. MacG. (1995), The 'Unusual Episode' Data Revisited. *Journal of Statistics Education*, **3**. http://www.amstat.org/publications/jse/v3n3/datasets.dawson.html

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:

British Board of Trade (1990), *Report on the Loss of the 'Titanic' (S.S.)*. British Board of Trade Inquiry Report (reprint). Gloucester, UK: Allan Sutton Publishing.

Examples

```
data(Titanic)
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 ...
```

`title` *Plot Annotation* `title`

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

```
title(main = "", sub = "", xlab = "", ylab = "", ...)
```

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`.
`...`	further graphical parameters (from `par`).

See Also

`mtext`, `text`; `plotmath` for details on mathematical annotation.

Examples

```
plot(cars, main = "")
title(main = "Stopping Distance versus Speed")

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))
abline(h = 0, v = pi/2 * c(-1,1), lty = 2, lwd = .1, col = "gray70")
```

`ToothGrowth` *The Effect of Vitamin C on Tooth Growth in Guinea Pigs* `ToothGrowth`

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

```
data(ToothGrowth)
```

Format

A data frame with 60 observations on 3 variables.

[,1]	len	numeric	Tooth length
[,2]	supp	factor	Supplement type (VC or OJ).
[,3]	dose	numeric	Dose in milligrams.

Source

C. I. Bliss (1952) *The Statistics of Bioassay.* Academic Press.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(ToothGrowth)
coplot(len ~ dose | supp, data = ToothGrowth, panel = panel.smooth,
       xlab = "ToothGrowth data: given is supplement type")
```

`trace` *Trace All Calls to a Function.* `trace`

Description

These are both in very primitive form. If `trace` is called the function named is marked and each time it is entered the call is printed on the console. To stop tracing a function use `untrace`.

Usage

```
trace(fun)
untrace(fun)
```

Arguments

`fun`	any interpreted R function (not quoted).

See Also

`debug`.

Examples

```
trace(names)
data.frame(x=1:2, y=3:4)
untrace(names)
```

`traceback` *Print Call Stack of Last Error* `traceback`

Description

`traceback()` prints the call stack of the last error, i.e., the sequence of calls that lead to the error. This is useful when an error occurs with an unidentifiable error message. Currently, this stack is given as a list in `.Traceback`.

Usage

```
traceback()
```

Examples

```
foo <- function(x) { print(1); bar(2) }
bar <- function(x) { x + a.variable.which.does.not.exist }

foo(2) # gives a strange error
traceback()
## [1] "bar(2)" "foo(2)"
bar
## Ah, this is the culprit ...
```

`transform` *Transform an Object, e.g. a Data Frame* `transform`

Description

`transform` is a generic function, which—at least currently—only does anything useful with dataframes. `transform.default` converts its first argument to a dataframe if possible and calls `transform.data.frame`.

Usage

```
transform(x, ...)
transform.default(x, ...)
transform.data.frame(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 dataframe `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`.

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

```
data(airquality)
transform(airquality, Ozone = -Ozone)
transform(airquality, new = -Ozone, Temp = (Temp-32)/1.8)

attach(airquality)
transform(Ozone, logOzone = log(Ozone)) # marginally interesting ...
detach(airquality)
```

`trees` *Girth, Height and Volume for Black Cherry Trees* `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 at 4 ft 6 in above the ground.

Usage

```
data(trees)
```

Source

Ryan, T. A., Joiner, B. L. and Ryan, B. F. (1976) *The Minitab Student Handbook.* Duxbury Press.

References

Atkinson, A. C. (1985) *Plots, Transformations and Regression.* Oxford University Press.

Examples

```
data(trees)
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))
```

`Trig` *Trigonometric Functions* `Trig`

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

```
cos(x)
sin(x)
tan(x)
acos(x)
asin(x)
atan(x)
atan2(y, x)
```

Details

The arc-tangent of two arguments `atan2(y,x)` returns the angle between the x-axis and the vector from the origin to (x, y), i.e., for positive arguments `atan2(y,x) == atan(y/x)`.

Angles are in radians, not degrees (i.e. a right angle is $\pi/2$).

Examples

```
cos(0) == 1
sin(3*pi/2) == cos(pi)
x <- rnorm(99)
all.equal( sin(-x), - sin(x))
all.equal( cos(-x), cos(x))
x <- abs(x); y <- abs(rnorm(x))
all(abs(atan2(y, x) - atan(y/x)) <= .Machine$double.eps)# TRUE
table(abs(atan2(y, x) - atan(y/x)) / .Machine$double.eps) # depends!

x <- 1:99/100
all(Mod(1 - (cos(x) + 1i*sin(x)) / exp(1i*x)) < 1.1 * .Machine$double.eps)
 2* abs(1 - x / acos(cos(x))) / .Machine$double.eps #-- depends ?
all(abs(1 - x / asin(sin(x))) <= .Machine$double.eps) # TRUE
all(abs(1 - x / atan(tan(x))) <= .Machine$double.eps) # TRUE
```

---

ts *Time-Series Objects* ts

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)

print(ts.obj, calendar, ...)
plot(ts.obj, plot.type=c("multiple", "single"), ...)
lines(ts.obj, ...)
```

Arguments

`data`	a vector or matrix of the observed time-series values.
`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`, ...
`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.
`plot.type`	for multivariate time series, should the series by plotted separately (with a common time axis) or on a single plot?
`...`	additional arguments to print or plot.

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.

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.

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` will use the `tsp` attribute of the object if it has one to set the start and end times and frequency.

See Also

`tsp`, `frequency`, `start`, `end`, `time`, `window`

Standard package `ts` for many additional time-series functions.

Examples

```
ts(1:10, frequency = 4, start = c(1959, 2)) # 2nd Quarter of 1959
print( 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)
plot(z)
plot(z, plot.type="single", lty=1:3)

## A phase plot:
data(nhtemp)
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
library(ts)
plot(nhtemp, lag(nhtemp, 1), cex = .8, col="blue",
     main = "Lag plot of New Haven temperatures")
```

`tsp` *Tsp Attribute of Time-Series-like Objects* `tsp`

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`.

See Also

`ts`, `time`, `start`.

`Tukey` *The Studentized Range Distribution* `Tukey`

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, 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 n_g =`nranges` is greater than one, R is the *maximum* of n_g groups of `nmeans` observations each.

Value

`ptukey` gives the distribution function and `qtukey` its inverse, the quantile function.

Note

A Legendre 16-point formula is used for the integral of `ptukey`. The computations are relatively expensive, especially for `qtukey` which uses a simple secant method for finding the inverse of `ptukey`. `qtukey` will be accurate to the 4th decimal place.

References

Copenhaver, Margaret Diponzio and Holland, Burt S. (1988) Multiple comparisons of simple effects in the two-way analysis of variance with fixed effects. *Journal of Statistical Computation and Simulation*, **30**, 1–15.

See Also

`pnorm` and `qnorm` for the corresponding functions for the normal distribution.

Examples

```
system.time(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)))
```

typeof *The Type of an Object* typeof

Description

`typeof` determines the (R internal) type or storage mode of any object `x`. It returns a character string.

Usage

```
typeof(x)
```

See Also

`mode`, `storage.mode`.

Examples

```
typeof(2)
mode(2)
```

UCBAdmissions *Student Admissions at UC Berkeley* UCBAdmissions

Description

Aggregate data on applicants to graduate school at Berkeley for the six largest departments in 1973 classified by admission and sex.

Usage

```
data(UCBAdmissions)
```

Format

A 3-dimensional array resulting from cross-tabulating 4526 observations on 3 variables. The variables and their levels are as follows:

No	Name	Levels
1	Admit	Admitted, Rejected
2	Gender	Male, Female
3	Dept	A, B, C, D, E, F

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://hotspur.psych.yorku.ca/SCS/friendly.html) for further information.

References

Bickel, P. J., Hammel, E. A., and O'Connell, J. W. (1975) Sex bias in graduate admissions: Data from Berkeley. *Science*, **187**, 398–403.

Examples

```
data(UCBAdmissions)
## 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)
```

---

undoc *Find Undocumented Objects* undoc

Description

Finds the objects in a package which are undocumented, in the sense that they are visible to the user (or data objects provided by the package), but no documentation entry exists.

Usage

```
undoc(pkg, dir)
```

Arguments

pkg	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.

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

A character vector containing the names of the undocumented objects.

Examples

```
undoc("eda")                    # Undocumented objects in 'eda'
```

`Uniform` *The Uniform Distribution* `Uniform`

Description

These functions provide information about the uniform distribution on the interval from `min` to `max`. `dunif` gives the density, `punif` gives the distribution function `qunif` gives the quantile function and `runif` generates random deviates.

Usage

```
dunif(x, min=0, max=1, log = FALSE)
punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
runif(n, min=0, max=1)
```

Arguments

`x,q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations to generate.
`min,max`	lower and upper limits of the distribution.
`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 `min` or `max` are not specified they assume the default values of `0` and `1` respectively.

The uniform distribution has density

$$f(x) = \frac{1}{max - min}$$

for $min \leq x \leq max$.

For the case of $u := min == max$, the limit case of $X \equiv u$ is assumed.

See Also

`.Random.seed` about random number generation, `rnorm`, etc for other distributions.

Examples

```
u <- runif(20)
all(punif(u) == u) # T
all(dunif(u) == 1) # T

var(runif(10000))#- ~ = 1/12 = .08333
all(runif(100, 2,2) == 2)#-> TRUE [exhibits bug in R version <= 0.63.1]
```

`unique` *Extract Unique Elements* `unique`

Description

`unique` returns a vector like `x` but with duplicate elements removed. If an element is equal to one with a smaller index, it is removed.

Usage

```
unique(x)
```

Arguments

`x`	an atomic vector

See Also

`duplicated` which gives the indices of duplicated elements.

Examples

```
unique(c(3:5, 11:8, 8 + 0:5))
length(unique(sample(100,100,replace=T)))# ~= 100(1 - 1/e) = 63.21
my.unique <- function(x) x[!duplicated(x)]
for(i in 1:4)
 { x <- rpois(100, pi); print(all(unique(x) == my.unique(x))) }
```

`uniroot` *One Dimensional Root Finding* `uniroot`

Description

The function `uniroot` searches the interval from `lower` to `upper` for a 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

Brent, R. (1973) *Algorithms for Minimization without Derivatives.* Englewood Cliffs, NJ: Prentice-Hall.

See Also

`polyroot` for all complex roots of a polynomial; `optimize`, `nlm`.

Examples

```
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)
```

`units` *Graphical Units* `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 `par(log=..)`).

`xyinch` does the same for a pair of numbers `xy`, simultaneously.

`cm` translates inches in to cm (centimeters).

Usage

```
xinch(x=1, warn.log=TRUE)
yinch(y=1, warn.log=TRUE)
xyinch(xy=1, warn.log=TRUE)
cm(x)
```

Arguments

x,y	numeric vector

Examples

```
all(c(xinch(),yinch()) == xyinch()) # TRUE
xyinch()
xyinch #- to see that is really   delta{"usr"} / "pin"

cm(1)# = 2.54

## plot labels offset 0.12 inches to the right
## of plotted symbols in a plot
data(mtcars)
attach(mtcars)
plot(mpg, disp, pch=19, main= "Motor Trend Cars")
text(mpg + xinch(0.12), disp, rownames(mtcars),adj=0, cex = .7, col='blue')
detach(mtcars)
```

`unlink` *Delete Files and Directories* `unlink`

Description

`unlink` deletes the file(s) or directories specified by `x`.

Usage

```
unlink(x)
```

Arguments

x	a character vector with the names of the file(s) or directories to be deleted. Wildcards (normally '*' and '?') are allowed.

Value

The return value of the corresponding system command, `rm -f`, normally `0` for success, `1` for failure. Not deleting a non-existent file is not a failure.

See Also

`tempfile` for file creation.

`unlist` *Flatten Lists* `unlist`

Description

Given a list structure `x`, `unlist` produces a vector which contains all the atomic components which occur in `x`.

Usage

```
unlist(x, recursive = TRUE, use.names = TRUE)
```

Arguments

`x`	A list.
`recursive`	logical. Should unlisting be applied to list components of `x`?
`use.names`	logical. Should names be preserved?

Details

If `recursive=FALSE`, the function will not recurse beyond the first level items in `x`.

By default, `unlist` tries to retain the naming information present in `x`. If `use.names = FALSE` all naming information is dropped.

See Also

`c`, `as.list`.

Examples

```
unlist(options())
unlist(unlist(options(), use.names=F))# works for vectors or lists
l.ex <- list(a = list(1:5, LETTERS[1:5]), b = "Z", c = NA)
unlist(l.ex, rec = F)
unlist(l.ex, rec = T)
```

`unname` *Remove 'names' or 'dimnames'* `unname`

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.frame`s. *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)))
str(table(breaks)) ## The names are quite larger than the data..
barplot(unname(table(breaks)), axes= FALSE)
```

`update` *Update and Re-fit a Model Call* `update`

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

```
update(object, ...)
update.default(object, formula, ..., evaluate = TRUE)
```

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.

Author(s)

B.D. Ripley

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 <- c(ctl, trt)
lm.D9 <- lm(weight ~ group)
lm.D9
summary(lm.D90 <- update(lm.D9, . ~ . - 1))
options(contrasts = c("contr.helmert", "contr.poly"))
update(lm.D9)
options(oldcon)
```

`update.formula` *Model Updating* `update.formula`

Description

`update.formula` is used to update model formulae. This typically involves adding or dropping terms, but updates can be more general.

Usage

```
update.formula(old, new)
```

Arguments

`old`	a model formula to be updated.
`new`	a formula giving a template which specifies how to update.

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 occurence of "`.`" on the left of `new`, and substitutes the *rhs* of the `old` formula for any occurence of "`.`" on the right of `new`.

Value

The updated formula is returned.

See Also

`terms`, `model.matrix`.

update.packages *Download Packages from CRAN* update.packages

Description

These functions can be used to automatically compare the version numbers of installed packages with the newest available version on CRAN and update outdated packages on the fly.

Usage

```
update.packages(lib.loc = .lib.loc, CRAN = options("CRAN"),
                contriburl = contrib.url(CRAN),
                method = "auto", instlib = NULL,
                ask=TRUE, available=NULL)

installed.packages(lib.loc = .lib.loc)
CRAN.packages(CRAN = options("CRAN"), method = "auto",
              contriburl = contrib.url(CRAN))
old.packages(lib.loc = .lib.loc, CRAN = getOption("CRAN"),
             contriburl = contrib.url(CRAN),
             method = "auto", available = NULL)

download.packages(pkgs, destdir, available = NULL,
                  CRAN = options("CRAN"), method = "auto",
                  contriburl = contrib.url(CRAN))
install.packages(pkgs, lib, available = NULL,
                  CRAN = options("CRAN"), method = "auto",
                  contriburl = contrib.url(CRAN))
```

Arguments

lib.loc	A character vector describing the location of R library trees to search through (and update packages therein).
CRAN	The base URL of the CRAN mirror to use, i.e., the URL of a CRAN root such as "http://www.r-project.org" (the default) or its Statlib mirror, "http://lib.stat.cmu.edu/R/CRAN".
contriburl	URL of the contrib section of CRAN. Use this argument only if your CRAN mirror is incomplete, e.g., because you burned only the contrib section on a CD. Overrides argument CRAN.
method	Download method, see download.file.
pkgs	A character vector of the short names of packages whose current versions should be downloaded from CRAN.
destdir	Directory where downloaded packages are stored.
available	List of packages available at CRAN as returned by CRAN.packages.
lib,instlib	A character string giving the library directory where to install the packages.
ask	If TRUE, ask before packages are actually downloaded and installed.

Details

`installed.packages` scans the 'DESCRIPTION' files of each package found along `lib.loc` and returns a list of package names, library paths and version numbers. `CRAN.packages` returns a similar list, but corresponding to packages currently available in the contrib section of CRAN, the comprehensive R archive network. The current list of packages is downloaded over the internet (or copied from a local CRAN mirror). Both functions use `parse.dcf` for parsing the description files. `old.packages` compares the two lists and reports installed packages that have newer versions on CRAN.

`download.packages` takes a list of package names and a destination directory, downloads the newest versions of the package sources and saves the in `destdir`. If the list of available packages is not given as argument, it is also directly obtained from CRAN. If CRAN is local, i.e., the URL starts with `"file:"`, then the packages are not downloaded but used directly.

The main function of the bundle is `update.packages`. First a list of all packages found in `lib.loc` is created and compared with the packages available on CRAN. Outdated packages are reported and for each outdated package the user can specify if it should be automatically updated. If so, the package sources are downloaded from CRAN and installed in the respective library path (or `instlib` if specified) using the R `INSTALL` mechanism.

`install.packages` can be used to install new packages, it takes a vector of package names and a destination directory, downloads the packages from CRAN and installs them. If the destination directory is omitted it defaults to the first directory in `.lib.loc`, with a warning.

See Also

`INSTALL`, `REMOVE`, `library`, `.packages`, `parse.dcf`, `download.file`

`USArrests` *Violent Crime Rates by US State* `USArrests`

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

```
data(USArrests)
```

Format

A data frame with 50 observations on 5 variables.

[,1]	Murder	numeric	Murder arrests (per 100,000)
[,2]	Assault	numeric	Assault arrests (per 100,000)
[,3]	UrbanPop	numeric	Percent urban population
[,4]	Rape	numeric	Rape arrests (per 100,000)

Source

World Almanac and Book of facts 1975. (Crime rates).

Statistical Abstracts of the United States 1975. (Urban rates).

References

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(USArrests)
pairs(USArrests, panel = panel.smooth, main = "USArrests data")
```

`USJudgeRatings` *Lawyers' Ratings of State Judges in the US Superior Court* `USJudgeRatings`

Description

Lawyers' ratings of state judges in the US Superior Court

Usage

```
data(USJudgeRatings)
```

Format

A data frame containing 43 observations on 12 numeric variables.

[,1]	CONT	Number of contacts of lawyer with judge.
[,2]	INTG	Judicial integrity.
[,3]	DMNR	Demeanor.
[,4]	DILG	Diligence.
[,5]	CFMG	Case flow managing.
[,6]	DECI	Prompt decisions.
[,7]	PREP	Preparation for trial.
[,8]	FAMI	Familiarity with law.
[,9]	ORAL	Sound oral rulings.
[,10]	WRIT	Sound written rulings.
[,11]	PHYS	Physical ability.
[,12]	RTEN	Worthy of retention.

Source

New Haven Register, 14 January, 1977 (from John Hartigan).

Examples

```
data(USJudgeRatings)
pairs(USJudgeRatings, main = "USJudgeRatings data")
```

`USPersonalExpenditure` *Personal Expenditure Data* `USPersonalExpenditure`

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, and 1960.

Usage

```
data(USPersonalExpenditure)
```

Format

A matrix with 5 rows and 5 columns.

Source

The World Almanac and Book of Facts, 1962, page 756.

References

Tukey, J. W. (1977) *Exploratory Data Analysis.* Addison-Wesley.

McNeil, D. R. (1977) *Interactive Data Analysis.* Wiley.

Examples

```
data(USPersonalExpenditure)
USPersonalExpenditure
require(eda)
medpolish(log10(USPersonalExpenditure))
```

`uspop` *The Population of the United States* `uspop`

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

```
data(uspop)
```

Format

A time series of 19 values.

Source

McNeil, D. R. (1977) *Interactive Data Analysis.* New York: Wiley.

Examples

```
data(uspop)
plot(uspop, log = "y", main = "uspop data", xlab = "Year",
     ylab = "U.S. Population (millions)")
```

`VADeaths` *Death Rate Data* `VADeaths`

Description

Death rates per 100 in Virginia in 1940.

Usage

```
data(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

Moyneau, L., Gilliam, S. K., and Florant, L. C.(1947) Differences in Virginia death rates by color, sex, age, and rural or urban residence. *American Sociological Review*, **12**, 525–535.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* Wiley.

Examples

```
data(VADeaths)
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))
plot(aov.VAD)
par(opar)
```

var *Covariance Matrices* var

Description

var computes the variance of x and the covariance of x and y if x and y are vectors. If x and y are matrices then the covariance between the columns of x and the the columns of y are computed.

Usage

```
var(x, y = x, na.rm = FALSE, use)
```

Arguments

x	a numeric matrix or vector.
y	a numeric matrix or vector.
na.rm	logical.
use	an optional character string giving a method for computing covariances in the presence of missing values. This must be one of "all.obs", "complete.obs" or "pairwise.complete.obs", with abbreviation being permitted.

Details

If na.rm is TRUE then the complete observations (rows) are used to compute the variance. If na.rm is FALSE and there are missing values, then var will fail.

The argument use can also be used for describing how to handle missing values. Specifying use = "all" is equivalent to specifying na.rm = FALSE and specifying use = "pair" is equivalent to na.rm = TRUE. If use = "pair", then all the observations which are complete for a pair of variables are used to compute the covariance for that pair of variables. This can result in covariance matrices which are not positive semidefinite.

See Also

cov with the same functionality for the multivariate case.

Examples

```
var(1:10)# 9.166667

var(1:5,1:5)# 2.5
```

`vector` *Vectors* `vector`

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 or character if not specified) and `FALSE` otherwise.

Usage

```
vector(mode = "logical", length = 0)
as.vector(x, mode = "any")
is.vector(x, mode = "any")
```

Arguments

`mode`	A character string giving an atomic mode, or `"any"`.
`length`	A non-negative integer specifying the desired length.
`x`	An object.

Details

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` and character vector elements to `""`.

See Also

`c`, `is.numeric`, `is.list`, etc.

Examples

```
df <- data.frame(x=1:3, y=5:7)
## Error:
  as.vector(data.frame(x=1:3, y=5:7), mode="numeric")

###-- All the following are TRUE:
is.list(df)
! is.vector(df)
! is.vector(df, mode="list")

is.vector(list(), mode="list")
is.vector(NULL,   mode="NULL")
```

volcano *Topographic Information for the Maunga Whau Volcano* 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

```
data(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

```
data(volcano)
filled.contour(volcano, color = terrain.colors, asp = 1)
title(main = "volcano data: filled contour map")
```

warning *Warning Messages* warning

Description

Generates a warning message that corresponds to its argument and the expression or function from which it was called.

Usage

```
warning(message)
```

Arguments

`message`	character string (of length 1) or `NULL`.

Details

The result *depends* on the value of `options("warn")`.

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.

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`.

See Also

`stop` for fatal errors, `warnings`, and `options(warn=..)`.

Examples

`warnings` *Print Warning Messages* `warnings`

Description

`warnings` prints the top-level variable `last.warning` in a pleasing form.

Usage

```
warnings(...)
```

Arguments

`...`	arguments to be passed to `cat`.

See Also

`warning`.

Examples

```
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(ow) # reset
```

`warpbreaks` *The Number of Breaks in Yarn during Weaving* `warpbreaks`

Description

This data set gives the number of warp breaks per loom, where a loom corresponds to a fixed length of yarn.

Usage

```
data(warpbreaks)
```

Format

A data frame with 54 observations on 3 variables.

`[,1]`	`breaks`	numeric	The number of breaks
`[,2]`	`wool`	factor	The type of wool (A or B)
`[,3]`	`tension`	factor	The level of tension (L, M, H)

Source

Tippet, L. H. C. (1950) *Technological Applications of Statistics.* Wiley. Page 106.

References

Tukey, J. W. (1977) *Exploratory Data Analysis.* Addison-Wesley.

McNeil, D. R. (1977) *Interactive Data Analysis.* Wiley.

Examples

```
data(warpbreaks)
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)
```

`Weibull` *The Weibull Distribution* `Weibull`

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 to generate.
`shape, scale`	shape 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 `scale` is omitted it assumes the default value of `1`.

The Weibull distribution with `shape` parameter a and `scale` parameter b has density given by

$$f(x) = (a/b)(x/b)^{a-1} \exp(-(x/b)^a)$$

for $x > 0$.

Value

`dweibull` gives the density, `pweibull` gives the distribution function, `qweibull` gives the quantile function, and `rweibull` generates random deviates.

See Also

`dexp` for the Exponential which is a special case of a Weibull distribution.

Examples

```
x <- 1:10
all.equal(dweibull(x, shape = 1), dexp(x))
all.equal(pweibull(x, shape = 1, scale = pi), pexp(x, rate = 1/pi))
all.equal(qweibull(x/11, shape = 1, scale = pi), qexp(x/11, rate = 1/pi))
```

`weighted.mean` *Weighted Arithmetic Mean* `weighted.mean`

Description

Computed 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(5/15,5/15,4/15,1/15)
x <- c(3.7,3.3,3.5,2.8)
xm <- weighted.mean(x,wt)
```

`weighted.residuals` *Compute Weighted Residuals* `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 the usual 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 `lm.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

```
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)
```

`which` *Which indices are TRUE ?* `which`

Description

Give the `TRUE` indices of a logical object, allowing for array indices.

Usage

```
which(x, arr.ind = FALSE)
```

Arguments

`x`	a `logical` vector or array. `NA`s are allowed and omitted (treated as if `FALSE`).
`arr.ind`	logical; should **arr**ay **ind**ices be returned when `x` is an array?

Value

If `arr.ind == FALSE` (the default), an integer vector with `length` equal to `sum(x)`, i.e., to the number of `TRUE`s in `x`; Basically, the result is `(1:length(x))[x]`.

If `arr.ind == TRUE` and `x` is an `array` (has a `dim` attribute), the result is a matrix who's rows each are the indices of one element of `x`; see Examples below.

Author(s)

Werner Stahel and Peter Holzer ⟨holzer@stat.math.ethz.ch⟩, for the array case.

See Also

Logic

Examples

```
which(LETTERS == "R")
which(ll <- c(T,F,T,NA,F,F,T))#> 1 3 7
names(ll) <- letters[seq(ll)]
which(ll)
which((1:12)%%2 == 0) # which are even?
str(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)
```

`Wilcoxon` *Distribution of the Wilcoxon Rank Sum Statistic* `Wilcoxon`

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

```
dwilcox(x, m, n, log = FALSE)
pwilcox(q, m, n, lower.tail = TRUE, log.p = FALSE)
qwilcox(p, m, n, lower.tail = TRUE, log.p = FALSE)
rwilcox(nn, m, n)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`nn`	number of observations to generate.
`m, n`	numbers of observations in the first and second sample, 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

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 `m * n`, and its mean and variance are `m * n / 2` and `m * n * (m + n + 1) / 12`, respectively.

Value

`dwilcox` gives the density, `pwilcox` gives the distribution function, `qwilcox` gives the quantile function, and `rwilcox` generates random deviates.

Author(s)

Kurt Hornik ⟨hornik@ci.tuwien.ac.at⟩

See Also

`dsignrank` etc, for the *one-sample* Wilcoxon rank statistic.

Examples

```
x <- -1:(4*6 + 1)
fx <- dwilcox(x, 4, 6)
all(fx == dwilcox(x, 6, 4))
Fx <- pwilcox(x, 4, 6)
all(abs(Fx - cumsum(fx)) < 10 * .Machine$double.eps)

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)",sep="\n"))
n <- as.numeric(names(print(tU <- table(U))))
text(n+.2, n+.5, labels=tU, col="red")
```

`window` *Time Windows* `window`

Description

`window` is a generic function which extracts the subset of the object `x` observed between the times `start` and `end`. Methods are available for time series and point-processes.

Usage

```
window(x, start=NULL, end=NULL, warn=TRUE)
```

Arguments

`x`	a time-series or other object.
`start`	the start time of the period of interest.
`end`	the end time of the period of interest.
`warn`	if `FALSE` omit warnings about `start` and `end` being out of range.

Note

The `warn` argument is for use by lazy R programmers whose code requires `window` to do the checking silently.

See Also

`time`, `ts`.

Examples

```
data(presidents)
sixties <- window(presidents, 1960, c(1969,4))
```

`women` *Average Heights and Weights for American Women* `women`

Description

This data set gives the average heights and weights for American women aged 30–39.

Usage

```
data(women)
```

Format

A data frame with 15 observations on 2 variables.

`[,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.

Source

The World Almanac and Book of Facts, 1975.

References

McNeil, D. R. (1977) *Interactive Data Analysis.* Wiley.

Examples

```
data(women)
plot(women, xlab = "Height (in)", ylab = "Weight (lb)",
     main = "women data: American women aged 30-39")
```

`write` *Write Data to a File* `write`

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

```
write(x, file="data",
        ncolumns=if(is.character(x)) 1 else 5,
        append=FALSE)
```

Arguments

`x`	the data to be written out.
`file`	the name of the file (quoted) to write to.
`ncolumns`	the number of columns to write the data in.
`append`	if `TRUE` the data `x` is appended to file `file`.

See Also

`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))

# the file data now contains the data in x,
# two rows, five cols but the first row is 1 2 3 4 5
write(x)
unlink("data") # tidy up
```

`write.table` *Data Output* `write.table`

Description

`write.table` prints its required argument `x` (after converting it to a data frame if it is not one already) to `file`. The entries in each line (row) are separated by the value of `sep`.

Usage

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = NA, row.names = TRUE, col.names = TRUE)
```

Arguments

`x`	the object to be written, typically a data frame. If not, it is attempted to create one from it.
`file`	the name of the file which the data are to be written to.
`sep`	the field separator string. Values on each line of the file are separated by this string.
`col.names`	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.
`row.names`	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.
`quote`	a logical or a numeric vector. If `TRUE`, any strings in the data will be surrounded by double quotes. If a numeric vector, its elements are taken as the indices of the variable (columns) to quote. In both cases, row and columns names are always quoted if they are written.
`na`	the string to use for missing values in the data.
`eol`	the character(s) to print at the end of each line (row).

See Also

`read.table`.

`x11` *X Window System Graphics* `x11`

Description

`X11` starts a graphics device driver for the X Window System (version 11). This can only be done on machines that run X. `x11` is recognized as a synonym for `X11`.

Usage

```
X11(display = "", width = 7, height = 7, pointsize = 12,
    gamma = 1, colortype = getOption("X11colortype"),
    maxcubesize = 256)
```

Arguments

`display`	the display on which the graphics window will appear. The default is to use the value in the user's environment variable `DISPLAY`.
`width`	the width of the plotting window in inches.
`height`	the height of the plotting window in inches.
`pointsize`	the default pointsize to be used.
`gamma`	the gamma correction factor. This value is used to ensure that the colors displayed are linearly related to RGB values. A value of around 0.5 is appropriate for many PC displays. A value of 1.0 (no correction) is usually appropriate for high-end displays or Macintoshs.
`colortype`	the kind of color model to be used. The possibilities are `"mono"`, `"gray"`, `"pseudo"`, `"pseudo.cube"` and `"true"`.
`maxcubesize`	can be used to limited the size of color cube allocated for pseudocolor devices.

Details

By default, an X11 device will use the best color rendering strategy that it can. The choice can be overriden with the `colortype` parameter. A value of `"mono"` results in black and white graphics, `"gray"` in grayscale and `"true"` in truecolor graphics (if this is possible). The values `"pseudo"` and `"pseudo.cube"` provide color strategies for pseudocolor displays. The first strategy provides on-demand color allocation which produces exact colors until the color resources of the display are exhausted. The second causes a standard color cube to be set up, and requested colors are approximated by the closest value in the cube. The default strategy for pseudocolor displays is `"pseudo"`.

See Also

`Devices`.

`xy.coords` *Extracting Plotting Structures* `xy.coords`

Description

`xy.coords` is used by many function to obtain x and y coordinates for plotting. The use of this common mechanism across all R functions produces a measure of consistency.

`plot.default` and `lowess` are examples of functions which use this mechanism.

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 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 yvar ~ xvar, xvar and yvar are used as x and y variables; if the argument is a list containing components x and y, these are used are assumed to define plotting coordinates; if the argument contains a time series, the x values are taken to be time and the y values to be the time series; if the argument is a matrix with two columns, the first is assumed to contain the x values and the second the y values; in any other case, the argument is coerced to a vector and the values plotted against their indices.
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.

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.

Examples

```
xy.coords(fft(c(1:10)), NULL)
data(cars) ; attach(cars)
xy.coords(dist ~ speed, NULL)$xlab # = "speed"

str(xy.coords(1:3, 1:2, recycle=TRUE))
str(xy.coords(-2:10,NULL, log="y"))
##> warning: 3 y values <=0 omitted ..
```

---

zip.file.extract — *Extract File from a Zip Archive* — zip.file.extract

---

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, zipname="R.zip")
```

Arguments

file	A file name.
zipname	The file name of a zip archive.

Details

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 used by `tempfile`.

Value

The name of the original or extracted file.

Warning

This function is intended for internal use only: it may be altered at any time, and may differ between platforms. Do NOT use in user code.

Note

The implementation differs by platform: it might do nothing.

Author(s)

B. D. Ripley

Chapter 2

The `ctest` package

`ansari.test` *Ansari-Bradley Test* `ansari.test`

Description

Performs the Ansari-Bradley two-sample test for a difference in scale parameters.

Usage

```
ansari.test(x, y, alternative = c("two.sided", "less", "greater"),
            exact = NULL)
```

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.

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 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.

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.
`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.

References

Myles Hollander & Douglas A. Wolfe (1973), *Nonparametric statistical inference.* New York: John Wiley & Sons. Pages 83–92.

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

```
## 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,
            100, 96, 108, 103, 104, 114, 114, 113, 108, 106, 99)
ansari.test(ramsay, jung.parekh)
```

`bartlett.test` *Bartlett Test for Homogeneity of Variances* `bartlett.test`

Description

Performs Bartlett's test of the null that the variances in each of the groups (samples) are the same.

Usage

```
bartlett.test(x, g)
```

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.

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-square test statistic.
`parameter`	the degrees of freedom of the approximate chi-square distribution of the test statistic.
`p.value`	the p-value of the test.
`method`	the string `"Bartlett test for homogeneity of variances"`.
`data.name`	a character string giving the names of the data.

References

Bartlett, M. S. (1937). Properties of sufficiency and statistical tests. *Proceedings of the Royal Statistical Society Series A* **160**, 268–282.

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

```
data(InsectSprays)
plot(InsectSprays$count ~ InsectSprays$spray)
bartlett.test(InsectSprays$count, InsectSprays$spray)
```

`binom.test` *Exact Binomial Test* `binom.test`

Description

Performs an exact test of the null that the probability of success in a Bernoulli experiment of length `n` is `p`, based on the number `x` of successes observed.

Usage

```
binom.test(x, n, p = 0.5, alternative = c("two.sided", "less", "greater"),
           conf.level = 0.95)
```

Arguments

`x`	number of successes.
`n`	number of trials.
`p`	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.

Value

A list with class `"htest"` containing the following components:

`statistic`	the number of successes, `x`.
`parameter`	the number of trials, `n`.
`p.value`	the p-value of the test.
`conf.int`	a confidence interval for the probability of success.
`estimate`	the estimated probability of success, `x / n`.
`null.value`	the probability of success under the null, `p`.
`alternative`	a character string describing the alternative hypothesis.
`method`	the string `"Exact binomial test"`.
`data.name`	a character string giving the names of the data.

References

Conover, W. J. (1971), *Practical nonparametric statistics*. New York: John Wiley & Sons. Pages 97–104.

Myles Hollander & Douglas A. Wolfe (1973), *Nonparametric statistical inference*. New York: John Wiley & Sons. Pages 15–22.

See Also

`prop.test` for a general (approximate) test for equal or given proportions.

Examples

```
## Conover (1971), p. 97f.
## 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 =
##  3/4 and the alternative that p != 3/4.
binom.test(682, 682 + 243, p = 3/4)
## => Data are in agreement with the null hypothesis.
```

`chisq.test` *Pearson's Chi-square Test for Count Data* `chisq.test`

Description

`chisq.test` performs chi-square tests on contingency tables.

Usage

```
chisq.test(x, y = NULL, correct = TRUE,
           p = rep(1/length(x), length(x)),
           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`.
`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, `x` is treated as a one-dimensional contingency table. 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, and hence its entries should be nonnegative 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-square 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-square 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. This is done by random sampling from the set of all contingency tables with given marginals, and works only if the marginals are positive.

Value

A list with class `"htest"` containing the following components:

`statistic`	the value the chi-square test statistic.
`parameter`	the degrees of freedom of the approximate chi-square 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.

Examples

```
data(InsectSprays)              # 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$spray)$obs
                                # 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
x <- trunc(5 * runif(100))
chisq.test(table(x))            # NOT 'chisq.test(x)'!
```

`cor.test` *Test for Zero Correlation* `cor.test`

Description

Tests whether two samples come from uncorrelated (independent) populations, using Pearson's product moment correlation coefficient, Kendall's tau, or Spearman's rho.

Usage

```
cor.test(x, y,
         alternative = c("two.sided", "less", "greater"),
         method = c("pearson", "kendall", "spearman"), exact = NULL)
```

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.
`method`	a string indicating which correlation coefficient is 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.

Details

If `method` is `"pearson"`, the test statistic is based on Pearson's product moment correlation coefficient `cor(x, y)` and follows a t distribution with `length(x)-2` degrees of freedom.

If `method` is `"kendall"` or `"spearman"`, Kendall's tau or Spearman's rho, respectively, are used to estimate the correlation. These tests should be used if the data do not necessarily come from a bivariate normal distribution.

For Kendall's test, 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, the standardized estimate is used as the test statistic, 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 correlation coefficient, with names attribute `"cor"`, `"tau"`, or `"rho"`, correspoding to the method employed.
`null.value`	the value of the correlation coefficient under the null hypothesis, hence `0`.
`alternative`	a character string describing the alternative hypothesis.
`method`	a string indicating how the correlation was estimated
`data.name`	a character string giving the names of the data.

References

D. J. Best & D. E. Roberts (1975), Algorithm AS 89: The Upper Tail Probabilities of Spearman's ρ. *Applied Statistics*, **24**, 377–379.

Myles Hollander & Douglas A. Wolfe (1973), *Nonparametric statistical inference*. New York: John Wiley & Sons. Pages 185–194 (Kendall and Spearman tests).

Examples

```
## Hollander & Wolfe (1973), p. 187f.
## 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 null is that the Hunter L value is positively associated
##  with the panel score.
x <- c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 60.1)
y <- c( 2.6,  3.1,  2.5,  5.0,  3.6,  4.0,  5.2,  2.8,  3.8)
cor.test(x, y, method = "kendall", alternative = "greater")
## => p=0.05972
##
cor.test(x, y, method = "kendall", alternative = "greater",
         exact = FALSE) # using large sample approximation
## => p=0.04765
## Compare this to
```

```
cor.test(x, y, method = "spearm", alternative = "g")
cor.test(x, y,                    alternative = "g")
```

`fisher.test` *Fisher's Exact Test for Count Data* `fisher.test`

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,
            or = 1, alternative = "two.sided", 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-square probabilies are only used provided that the "Cochran" conditions are satisfied.
`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.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.

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).

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 string `"Fisher's Exact Test for Count Data"`.
`data.name`	a character string giving the names of the data.

References

Alan Agresti (1990). *Categorical data analysis*. New York: Wiley. Pages 59–66.

Fisher, R. A. (1935). The logic of inductive inference. *Journal of the Royal Statistical Society Series A* **98**, 39–54.

Fisher, R. A. (1962). Confidence limits for a cross-product ratio. *Australian Journal of Statistics* **4**, 41.

Cyrus R. Mehta & Nitin R. Patel (1986). Algorithm 643. FEXACT: A Fortran subroutine for Fisher's exact test on unordered $r * c$ contingency tables. *ACM Transactions on Mathematical Software*, **12**, 154–161.

Douglas B. Clarkson, Yuan-an Fan & Harry Joe (1993). A Remark on Algorithm 643: FEXACT: An Algorithm for Performing Fisher's Exact Test in $r \times c$ Contingency Tables. *ACM Transactions on Mathematical Software*, **19**, 484–488.

See Also

`chisq.test`

Examples

```
## 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),
       nr = 2,
```

```
        dimnames =
        list(c("Dizygotic", "Monozygotic"),
             c("Convicted", "Not convicted")))
Convictions
fisher.test(Convictions, alternative = "less")
fisher.test(Convictions, conf.level = 0.95)$conf.int
fisher.test(Convictions, conf.level = 0.99)$conf.int
```

`fligner.test` *Fligner-Killeen Test for Homogeneity of Variances* `fligner.test`

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, g)
```

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.

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 $a(i) = qnorm((1 + i/(n + 1))/2)$. 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-square distribution of the test statistic.
`p.value`	the p-value of the test.
`method`	the string `"Fligner-Killeen test for homogeneity of variances"`.
`data.name`	a character string giving the names of the data.

References

W. J. Conover & Mark E. Johnson & Myrie M. Johnson (1981). A comparative study of tests for homogeneity of variances, with applications to the outer continental shelf bidding data. *Technometrics* **23**, 351–361.

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

```
data(InsectSprays)
plot(InsectSprays$count ~ InsectSprays$spray)
fligner.test(InsectSprays$count, InsectSprays$spray)
## Compare this to bartlett.test()
```

`friedman.test` *Friedman Rank Sum Test* `friedman.test`

Description

Performs a Friedman rank sum test with unreplicated blocked data.

Usage

```
friedman.test(y, groups, blocks)
```

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.

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-square statistic.
`parameter`	the degrees of freedom of the approximate chi-square distribution of the test statistic.
`p.value`	the p-value of the test.
`method`	the string `"Friedman rank sum test"`.
`data.name`	a character string giving the names of the data.

References

Myles Hollander & Douglas A. Wolfe (1973), *Nonparametric statistical inference.* New York: John Wiley & Sons. Pages 139–146.

Examples

```
## Hollander & Wolfe (1973), p. 140ff.
## 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,
         5.55, 5.50, 5.40,
         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

data(warpbreaks)
```

```
wb <- aggregate(warpbreaks$breaks,
                by = list(w = warpbreaks$wool,
                          t = warpbreaks$tension),
                FUN = mean)
wb
friedman.test(wb$x, wb$w, wb$t)
```

`kruskal.test` *Kruskal-Wallis Rank Sum Test* `kruskal.test`

Description

Performs a Kruskal-Wallis rank sum test.

Usage

```
kruskal.test(x, g)
```

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.

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-square distribution of the test statistic.
`p.value`	the p-value of the test.
`method`	the string `"Kruskal-Wallis rank sum test"`.
`data.name`	a character string giving the names of the data.

References

Myles Hollander & Douglas A. Wolfe (1973), *Nonparametric statistical inference.* New York: John Wiley & Sons. Pages 115–120.

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

```
## 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)
```

`ks.test` *Kolmogorov-Smirnov Tests* `ks.test`

Description

Performs one or two sample Kolmogorov-Smirnov tests.

Usage

```
ks.test(x, y, ..., alternative = c("two.sided", "less", "greater"))
```

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 by `y`.
`alternative`	indicates the alternative hypothesis and must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter.

Details

If `y` is numeric, a two sample test of the null that `x` and `y` were drawn from the same distribution is performed.

Alternatively, `y` can be a character string naming a distribution function. In this case, a one sample test of the null that the distribution function underlying `x` is `y` with parameters specified by `...` is carried out.

Currently, no exact p-values are available. The approximation by the limiting distribution may be inaccurate in small samples.

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

Conover, W. J. (1971), *Practical nonparametric statistics*. New York: John Wiley & Sons. Pages 295–301 (one-sample "Kolmogorov" test), 309–314 (two-sample "Smirnov" test).

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 scale 2?
ks.test(x+2, "pgamma", 3, 2) # two-sided
ks.test(x+2, "pgamma", 3, 2, alternative = "gr")
```

`mantelhaen.test` *Mantel-Haenszel Chi-Square Test for Count Data* `mantelhaen.test`

Description

`mantelhaen.test` performs a Mantel-Haenszel chi-square test of the null that `x` and `y` are conditionally independent in each stratum.

Usage

```
mantelhaen.test(x, y = NULL, z = NULL, correct = TRUE)
```

Arguments

`x`	either an array of dimension 2 by 2 by s, where s is the number of strata, or a dichotomous factor object.
`y`	a dichotomous factor object; ignored if x is an array.
`z`	a factor object idenitifying to which stratum the corresponding elements in `x` and `y` belong; ignored if `x` is an array.
`correct`	a logical indicating whether to apply continuity correction when computing the test statistic.

Details

If `x` is an array, it must be of dimension 2 by 2 by s, 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. Both `x` and `y` must be dichotomous (take exactly 2 values).

Value

A list with class `"htest"` containing the following components:

`statistic`	the Mantel-Haenszel chi-square statistic.
`parameter`	always 1, the degrees of freedom of the approximate chi-square distribution of the test statistic.
`p.value`	the p-value of the test.
`method`	a string indicating the method employed, and whether or not continuity correction was used.
`data.name`	a character string giving the names of the data.

References

Alan Agresti (1990). *Categorical data analysis*. New York: Wiley. Pages 230–234.

Examples

```
data(UCBAdmissions)
mantelhaen.test(UCBAdmissions)
## No evidence for association between admission and gender
## when adjusted for department
```

`mcnemar.test` *McNemar's Chi-square Test for Count Data* `mcnemar.test`

Description

Performs McNemar's chi-square 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 `correct` is `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-square 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

Alan Agresti (1990). *Categorical data analysis*. New York: Wiley. Pages 350–354.

Examples

```
## Agresti (1990), p. 350.
## 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
```

`mood.test` *Mood Two-Sample Test of Scale* `mood.test`

Description

Performs Mood's two-sample test for a difference in scale parameters.

Usage

```
mood.test(x, y, alternative = c("two.sided", "less", "greater"))
```

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.

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 string `"Mood two-sample test of scale"`.
`data.name`	a character string giving the names of the data.

References

Conover, W. J. (1971), *Practical nonparametric statistics*. New York: John Wiley & Sons. Pages 234f.

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

```
## 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,
            100, 96, 108, 103, 104, 114, 114, 113, 108, 106, 99)
mood.test(ramsay, jung.parekh)
## Compare this to ansari.test(ramsay, jung.parekh)
```

`pairwise.prop.test` *Pairwise comparisons of proportions* `pairwise.prop.test`

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* `pairwise.t.test`

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

```
data(airquality)
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* `pairwise.table`

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

Value

Table of p values in lower triangular form.

See Also

`pairwise.t.test`, et al.

`pairwise.wilcox.test` *Pairwise Wilcoxon rank sum tests* `pairwise.wilcox.test`

Description

Calculate pairwise comparisons between group levels with corrections for multiple testing

Usage

```
pairwise.wilcox.test(x, g, p.adjust.method=p.adjust.methods, ...)
```

Arguments

`x`	Response vector
`g`	Grouping vector or factor
`p.adjust.method`	Method for adjusting p values (see `p.adjust`)
`...`	Additional arguments to pass to `t.test`

Value

Object of class `"pairwise.htest"`

See Also

`wilcox.test`, `p.adjust`

Examples

```
data(airquality)
attach(airquality)
Month <- factor(Month, labels = month.abb[5:9])
pairwise.wilcox.test(Ozone, Month)
pairwise.wilcox.test(Ozone, Month, p.adj = "bonf")
detach()
```

`power.prop.test` *Power calculations two sample test for of proportions* `power.prop.test`

Description

Compute power of test, or determine parameters to obtain target power.

Usage

```
power.prop.test(n=NULL, p1=NULL, p2=NULL, sig.level=0.05,
                power=NULL,
                alternative=c("two.sided", "one.sided"))
```

Arguments

n	Number of observations (per group)
p1	probability in one group
p2	probability in other group
sig.level	Significance level (Type I error probability)
power	Power of test (1 minus Type II error probability)
alternative	One- or two-sided test

Details

Exactly one of the parameters n, p1, p2 power, and sig.level must be passed as NULL, and that parameter is determined from the others. Notice that sig.level has a non-NULL default so NULL must be explicitly passed if you want it computed.

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. If one of them is computed p1 < p2 will hold, although this is not enforced when both are specified.

Author(s)

Peter Dalgaard. Based on previous work by Claus Ekstrøm

See Also

prop.test, uniroot

Examples

```
power.prop.test(n=50, p1=.50, p2=.75)
power.prop.test(p1=.50, p2=.75, power=.90)
power.prop.test(n=50, p1=.5, power=.90)
```

---

power.t.test *Power calculations for one and two sample t tests* power.t.test

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"))
```

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

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.

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`, `uniroot`

Examples

```
power.t.test(n=20, delta=1)
power.t.test(power=.90, delta=1)
power.t.test(power=.90, delta=1, alt="one.sided")
```

`print.pairwise.htest` *Print method for pairwise tests* `print.pairwise.htest`

Description

Display results of pairwise comparison procedures

Usage

```
print.pairwise.htest(x)
```

Arguments

x	Object of class "pairwise.htest"

Value

None

See Also

pairwise.t.test, et al.

---

print.power.htest *Print method for power calculation object* print.power.htest

---

Description

Print power.htest object in nice layout

Usage

```
print.power.htest(x)
```

Arguments

x	Object of class power.htest

Details

A power.htest object is just a named list of numbers and 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

Author(s)

Peter Dalgaard

See Also

power.t.test, power.prop.test

`prop.test` *Test for Equal or Given Proportions* `prop.test`

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

```
prop.test(x, n = NULL, p = NULL, 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`	the alternative hypothesis; must be one of `"two.sided"` (default), `"greater"` or `"less"` and can be abbreviated by 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.

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-square test statistic.
`parameter`	the degrees of freedom of the approximate chi-square 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.

Examples

```
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.

smokers  <- c( 83, 90, 129, 70 )
patients <- c( 86, 93, 136, 82 )
prop.test(smokers, patients)
```

`shapiro.test` *Shapiro-Wilk Normality Test* `shapiro.test`

Description

Performs the Shapiro-Wilk test for 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 string "Shapiro-Wilk normality test".
data.name	a character string giving the name(s) of the data.

References

Patrick Royston (1982) An Extension of Shapiro and Wilk's W Test for Normality to Large Samples. *Applied Statistics*, **31**, 115–124.

Patrick Royston (1982) Algorithm AS 181: The W Test for Normality. *Applied Statistics*, **31**, 176–180.

Patrick Royston (1995) A Remark on Algorithm AS 181: The W Test for Normality. *Applied Statistics*, **44**, 547–551.

See Also

qqnorm for producing a normal quantile-quantile plot.

Examples

```
shapiro.test(rnorm(100, mean = 5, sd = 3))
shapiro.test(runif(100, min = 2, max = 4))
```

---

t.test *Student's t-Test* t.test

Description

t.test performs one and two sample t-tests on vectors of data.

Usage

```
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE, conf.level = 0.95)
```

Arguments

x	a numeric vector of data values.
y	an optional numeric vector data values.
alternative	character 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 approximation to the degrees of freedom is used.
conf.level	confidence level of the interval.

Details

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.

Value

A list with class "htest" containing the following components:

statistic	the value of the t-statistic.
parameters	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.

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
```

`var.test` *F Test to Compare Two Variances* `var.test`

Description

Performs an F test to compare the variances of two samples from normal populations.

Usage

```
var.test(x, y, ratio = 1, alternative = c("two.sided", "less", "greater"),
         conf.level = 0.95)
```

Arguments

`x, y`	numeric vectors of data values, or fitted linear model objects (inheriting from class `"lm"`).
`ratio`	the hypothesized ratio of the population variances of `x` and `y`.
`alternative`	the alternative hypothesis; must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter.
`conf.level`	confidence level for the returned confidence interval.

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 `ratio`.

Value

A list with class `"htest"` containing the following components:

`statistic`	the value of the F test statistic.
`parameter`	the degrees of the freedom of the F distribtion 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 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

```
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.
```

---

wilcox.test *Wilcoxon Rank Sum and Signed Rank Tests* wilcox.test

Description

Performs one and two sample Wilcoxon tests on vectors of data.

Usage

```
wilcox.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, exact = NULL, correct = TRUE)
```

Arguments

`x`	numeric vector of data values.
`y`	an optional numeric vector of data values.
`alternative`	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.

Details

If only `x` is given, or if both `x` and `y` are given and `paired` is `TRUE`, a Wilcoxon signed rank test of the null that the median of `x` (in the one sample case) or of `x-y` (in the paired two sample case) equals `mu` is performed.

Otherwise, if both `x` and `y` are given and `paired` is `FALSE`, a Wilcoxon rank sum test (equivalent to the Mann-Whitney test) is carried out. In this case, the null hypothesis is that the location of the distributions of `x` and `y` differ by `mu`.

By default (if `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.

Value

A list with class `"htest"` containing the following components:

`statistic`	the value of the test statistic with a name describing it.
`parameter`	the parameter(s) for the exact distribution of the test statistic.
`p.value`	the p-value for the test.
`null.value`	the location parameter `mu`.

`alternative`	a character string describing the alternative hypothesis.
`method`	the type of test applied.
`data.name`	a character string giving the names of the data.

References

Myles Hollander & Douglas A. Wolfe (1973), *Nonparametric statistical inference.* New York: John Wiley & Sons. Pages 27–33 (one-sample), 68–75 (two-sample).

See Also

`kruskal.test` for testing homogeneity in location parameters in the case of two or more samples; `t.test` for a parametric alternative under normality assumptions.

Examples

```
## 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
```

Chapter 3

The eda package

line	*Robust Line Fitting*	line

Description

This function fits a line robustly as recommended in *Exploratory Data Analysis*.

Usage

```
line(x, y)

coefficients(tukeyline.obj)
residuals(tukeyline.obj)
fitted.values(tukeyline.obj)
print(tukeyline.obj)
```

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 `coefficients`, `residuals`, `fitted.values`, and `print`.

References

Tukey, J. W. (1977). *Exploratory Data Analysis*, Reading Massachusetts: Addison-Wesley.

See Also

`lm`.

Examples

```
library(eda)
data(cars)
plot(cars)
z <- line(cars)
abline(coef(z))
```

`medpolish` *Median Polish of a Matrix* `medpolish`

Description

Fits an additive model using Tukey's *median polish* procedure.

Usage

```
medpolish(x, eps = 0.01, maxiter = 10, trace.iter = TRUE)

plot(medpolish.obj)
print(medpolish.obj)
```

Arguments

`x`	a numeric matrix.
`eps`	real number greater than 0. A tolerance for convergence: see **Details**.
`maxiter`	the maximum number of iterations
`trace.iter`	logical. Should progress in convergence be reported?
`medpolish.obj`	object of class `medpolish`.

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 `eps` or until there have been `maxiter` iterations. The sum of absolute residuals is printed at each iteration of the fitting process, if `trace.iter` is `TRUE`.

`medpolish` returns an object of class `medpolish` (see below). There are printing and plotting methods for this class, which are invoked via by the generics `print` and `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

Tukey, J. W. (1977). *Exploratory Data Analysis*, Reading Massachusetts: Addison-Wesley.

See Also

`median`; `aov` for a *mean* instead of *median* decomposition.

Examples

```
## 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)
```

`smooth` *Median Smoothing* `smooth`

Description

This function applies Tukey's smoothing method *3RSR* to `x`.

Usage

```
smooth(x)
```

Arguments

`x`	a vector or time series

Value

A vector or time series containing the smoothed values. Note that there are other smoothing methods which provide rather better results. This one was designed for hand calculations.

Note

3R stands for **R**epeated running `median`s of **3**.

References

Tukey, J. W. (1977). *Exploratory Data Analysis*, Reading Massachusetts: Addison-Wesley.

See Also

lowess; loess, supsmu and smooth.spline in package 'modreg'.

Examples

```
library(eda)
data(presidents)
presidents[is.na(presidents)] <- 0
plot(presidents)
lines(smooth(presidents))
```

Chapter 4

The lqs package

Description

Compute a multivariate location and scale estimate with a high breakdown point – this can be thought of as estimating the mean and covariance of the good part of the data. cov.mve and cov.mcd are compatibility wrappers.

Usage

```
cov.rob(x, cor = FALSE, quantile.used = floor((n + p + 1)/2),
        method = c("mve", "mcd", "classical"), nsamp = "best", seed)
cov.mve(x, cor = FALSE, quantile.used = floor((n + p + 1)/2),
        nsamp = "best", seed)
cov.mcd(x, cor = FALSE, quantile.used = floor((n + p + 1)/2),
        nsamp = "best", seed)
```

Arguments

x	a matrix or data frame.
cor	should the returned result include a correlation matrix?
quantile.used	the minimum number of the data points regarded as good points.
method	the method to be used – minimum volume ellipsoid, minimum covariance determinant or classical product-moment. Using cov.mve or cov.mcd forces mve or mcd respectively.
nsamp	the number of samples or "best" or "exact" or "sample". If "sample" the number chosen is min(5*p, 3000), taken from Rousseeuw and Hubert (1997). If "best" exhaustive enumeration is done up to 5000 samples: if "exact" exhaustive enumeration will be attempted however many samples are needed.
seed	the seed to be used for random sampling: see RNGkind. The current value of .Random.seed will be preserved if it is set.

Details

For method "mve", an approximate search is made of a subset of size quantile.used with an enclosing ellipsoid of smallest volume; in method "mcd" it is the volume of the Gaussian confidence ellipsoid, equivalently the determinant of the classical covariance matrix, that is minimized. The mean of the subset provides a first estimate of the location, and the rescaled covariance matrix a first estimate of scatter. The Mahalanobis distances of all the points from the location estimate for this covariance matrix are calculated, and those points within the 97.5% point under Gaussian assumptions are declared to be good. The final estimates are the mean and rescaled covariance of the good points.

The rescaling is by the appropriate percentile under Gaussian data; in addition the first covariance matrix has an *ad hoc* finite-sample correction given by Marazzi.

For method "mve" the search is made over ellipsoids determined by the covariance matrix of p of the data points. For method "mcd" an additional improvement step suggested by Rousseeuw and van Driessen (1997) is used, in which once a subset of size quantile.used is selected, an ellipsoid based on its covariance is tested (as this will have no larger a determinant, and may be smaller).

Value

A list with components

center	the final estimate of location.
cov	the final estimate of scatter.
cor	(only is cor = TRUE) the estimate of the correlation matrix.
sing	message giving number of singular samples out of total
crit	the value of the criterion on log scale. For MCD this is the determinant, and for MVE it is proportional to the volume.
best	the subset used. For MVE the best sample, for MCD the best set of size quantile.used.
n.obs	total number of observations.

Author(s)

B.D. Ripley

References

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection.* Wiley.

A. Marazzi (1993) *Algorithms, Routines and S Functions for Robust Statistics.* Wadsworth and Brooks/Cole.

P. J. Rousseeuw and B. C. van Zomeren (1990) Unmasking multivariate outliers and leverage points, *Journal of the American Statistical Association*, **85**, 633–639.

P. J. Rousseeuw and K. van Driessen (1997) A fast algorithm for the minimum covariance determinant estimator. Technical Report, Department of Mathematics and Computer Science, Universitaire Instelling Antwerpen.

P. Rousseeuw and M. Hubert (1997) Recent developments in PROGRESS. In *L1-Statistical Procedures and Related Topics* ed Y. Dodge, IMS Lecture Notes volume **31**, pp. 201–214.

See Also

lqs

Examples

```
data(stackloss)
.Random.seed <- 1:4
cov.rob(stackloss)
cov.rob(stack.x, method = "mcd", nsamp = "exact")
```

---

lqs *Resistant Regression* lqs

Description

Fit a regression to the good points in the dataset, thereby achieving a regression estimator with a high breakdown point. lmsreg and ltsreg are compatibility wrappers.

Usage

```
lqs(x, ...)
lqs.formula(formula, data = NULL, ...,
            method = c("lts", "lqs", "lms", "S", "model.frame"),
            subset, na.action = na.fail, model = TRUE,
            x = FALSE, y = FALSE, contrasts = NULL)
lqs.default(x, y, intercept, method = c("lts", "lqs", "lms", "S"),
            quantile, control = lqs.control(...), k0 = 1.548, seed, ...)
lmsreg(...)
ltsreg(...)
```

Arguments

formula	a formula of the form y ~ x1 + x2 + ...{}{}.
data	data frame from which variables specified in formula are preferentially to be taken.
subset	An index vector specifying the cases to be used in fitting. (NOTE: If given, this argument must be named exactly.)
na.action	A function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is na.omit, which leads to omission of cases with missing values on any required variable. (NOTE: If given, this argument must be named exactly.)
x	a matrix or data frame containing the explanatory variables.
y	the response: a vector of length the number of rows of x.
intercept	should the model include an intercept?
method	the method to be used. model.frame returns the model frame: for the others see the Details section. Using lmsreg or ltsreg forces "lms" and "lts" respectively.
quantile	the quantile to be used: see Details. This is over-ridden if method = "lms".

control	additional control items: see Details.
seed	the seed to be used for random sampling: see .Random.seed. The current value of .Random.seed will be preserved if it is set..
...	arguments to be passed to lqs.default or lqs.control.

Details

Suppose there are n data points and p regressors, including any intercept.

The first three methods minimize some function of the sorted squared residuals. For methods "lqs" and "lms" is the quantile squared residual, and for "lts" it is the sum of the quantile smallest squared residuals. "lqs" and "lms" differ in the defaults for quantile, which are floor((n+p+1)/2) and floor((n+1)/2) respectively. For "lts" the default is floor(n/2) + floor((p+1)/2).

The "S" estimation method solves for the scale s such that the average of a function chi of the residuals divided by s is equal to a given constant.

The control argument is a list with components:

psamp: the size of each sample. Defaults to p.

nsamp: the number of samples or "best" or "exact" or "sample". If "sample" the number chosen is min(5*p, 3000), taken from Rousseeuw and Hubert (1997). If "best" exhaustive enumeration is done up to 5000 samples: if "exact" exhaustive enumeration will be attempted however many samples are needed.

adjust: should the intercept be optimized for each sample?

Value

An object of class "lqs".

Note

There seems no reason other than historical to use the lms and lqs options. LMS estimation is of low efficiency (converging at rate $n^{-1/3}$) whereas LTS has the same asymptotic efficiency as an M estimator with trimming at the quartiles (Marazzi, 1993, p.201). LQS and LTS have the same maximal breakdown value of (floor((n-p)/2) + 1)/n attained if floor((n+p)/2) <= quantile <= floor((n+p+1)/2). The only drawback mentioned of LTS is greater computation, as a sort was thought to be required (Marazzi, 1993, p.201) but this is not true as a partial sort can be used (and is used in this implementation).

Adjusting the intercept for each trial fit does need the residuals to be sorted, and may be significant extra computation if n is large and p small.

Opinions differ over the choice of psamp. Rousseeuw and Hubert (1997) only consider p; Marazzi (1993) recommends p+1 and suggests that more samples are better than adjustment for a given computational limit.

The computations are exact for a model with just an intercept and adjustment, and for LQS for a model with an intercept plus one regressor and exhaustive search with adjustment. For all other cases the minimization is only known to be approximate.

Author(s)

B.D. Ripley

References

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection.* Wiley.

A. Marazzi (1993) *Algorithms, Routines and S Functions for Robust Statistics.* Wadsworth and Brooks/Cole.

P. Rousseeuw and M. Hubert (1997) Recent developments in PROGRESS. In *L1-Statistical Procedures and Related Topics*, ed Y. Dodge, IMS Lecture Notes volume **31**, pp. 201–214.

See Also

`predict.lqs`

Examples

```
data(stackloss)
.Random.seed <- 1:4
lqs(stack.loss ~ ., data = stackloss)
lqs(stack.loss ~ ., data = stackloss, method = "S", nsamp = "exact")
```

`predict.lqs` *Predict from an lqs Fit* `predict.lqs`

Description

Predict from an resistant regression fitted by `lqs`.

Usage

```
predict.lqs(object, newdata)
```

Arguments

`object`	object inheriting from class `lqs`
`newdata`	matrix or data frame of cases to be predicted or, if object has a formula, a data frame with columns of the same names as the variables used. A vector will be interpreted as a row vector. If `newdata` is missing, an attempt will be made to retrieve the data used to fit the `lqs` object.

Details

This function is a method for the generic function `predict()` for class `lqs`. It can be invoked by calling `predict(x)` for an object `x` of the appropriate class, or directly by calling `predict.lqs(x)` regardless of the class of the object.

Missing values in `newdata` are handled by returning `NA` if the linear discriminants cannot be evaluated. If `newdata` is omitted and the `na.action` of the fit omitted cases, these will be omitted on the prediction.

Value

A vector of predictions.

Author(s)

B.D. Ripley

See Also

lqs

Examples

```
data(stackloss)
.Random.seed <- 1:4
fm <- lqs(stack.loss ~ ., data = stackloss, method = "S", nsamp = "exact")
predict(fm, stackloss)
```

Chapter 5

The modreg package

---

ksmooth — *Kernel Regression Smoother* — ksmooth

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.

Author(s)

B. D. Ripley

Examples

```
data(cars)
attach(cars)
plot(speed, dist)
lines(ksmooth(speed, dist, "normal", bandwidth=2), col=2)
lines(ksmooth(speed, dist, "normal", bandwidth=5), col=3)
lines(ksmooth(speed, dist, "normal", bandwidth=10), col=4)
```

`loess` *Local Polynomial Regression Fitting* `loess`

Description

Fit a polynomial surface determined by one or more numerical predictors, using local fitting.

Usage

```
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 response and one or more 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 to stop.
`model`	should the model frame be returned?
`span`	the parameter α 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 know 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 α (set by `span` or `enp.target`). For $\alpha < 1$, the neighbourhood includes proportion α of the points, and these have tricubic weighting (proportional to $(1 - (\text{dist}/\text{maxdist})^3)^3$. For $\alpha > 1$, all points are used, with the 'maximum distance' assumed to be α times the actual maximum distance.

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)

B.D. Ripley, based on the `cloess` package of Cleveland, Grosse and Shyu.

References

W.S. Cleveland, E. Grosse and W.M. Shyu (1992) Local regression models. Chapter 8 of *Statistical Models in S* eds J.M. Chambers and T.J. Hastie, Wadsworth & Brooks/Cole.

See Also

`loess.control`, `predict.loess`

Examples

```
data(cars)
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)
```

`loess.control` *Set Parameters for Loess* `loess.control`

Description

Set control parameters for `loess` fits.

Usage

```
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.
`trace.hat`	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.
`cell`	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 `floor(n*span*cell)` points are subdivided.
`iterations`	the number of iterations used in robust fitting.

Value

A list with components

`surface`

`statistics`

`trace.hat`

`cell`

`iterations`

Author(s)

B.D. Ripley

See Also

loess

`modreg-internal` *Internal modreg functions* `modreg-internal`

Description

Internal modreg functions.

Usage

```
predLoess(y, x, newx, s, weights, robust, span, degree, normalize,
    parametric, drop.square, surface, cell, family, kd, divisor,
    se = F)
simpleLoess(y, x, weights, span = 0.75, degree = 2, parametric = FALSE,
    drop.square = FALSE, normalize = TRUE, statistics = "approximate",
    surface = "interpolate", cell = 0.2, iterations = 1,
    trace.hat = "exact")
pointwise(results, coverage)
```

Details

These are not to be called by the user.

`plot.ppr` *Plot Ridge Functions for Projection Pursuit Regression Fit* `plot.ppr`

Description

Plot ridge functions for projection pursuit regression fit.

Usage

```
plot.ppr(fit, ask, type = "o", ...)
```

Arguments

`fit`	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.

See Also

ppr, par

Examples

```
data(rock)
attach(rock)
area1 <- area/10000; peri1 <- peri/10000
par(mfrow=c(3,2))# maybe: , pty="s")
rock.ppr <- ppr(log(perm) ~ area1 + peri1 + 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)")
```

---

ppr *Projection Pursuit Regression* ppr

Description

Fit a projection pursuit regression model.

Usage

```
ppr(formula, data = sys.parent(), weights,
    subset, na.action, contrasts = NULL,
    ww = rep(1,q), nterms, max.terms=nterms, optlevel = 2,
    sm.method = c("supsmu", "spline", "gcvspline"),
    bass = 0, span = 0, df = 5, gcvpen = 1)

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 regression formula specifying one or more response variables and the explanatory variables.
`x`	matrix of explanatory 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 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 ww_j (y_ij - 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 `NA`s are found. The default action is for the procedure to fail. An alternative is `na.omit`, which leads to rejection of cases with missing values on any required variable. (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 `sm.method` is `"spline"` specifies the smoothness of each ridge term via the requested equivalent degrees of freedom.
`gcvpen`	if `sm.method` is `"gcvspline"` this is the penalty used in the GCV selection for each degree of freedom used.

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 `nterm` terms are left.

The levels of optimization (argument `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
`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 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.

References

Friedman, J. H. and Stuetzle, W. (1981) Projection pursuit regression. *Journal of the American Statistical Association*, **76**, 817–823.

Friedman, J. H. (1984) SMART User's Guide. Laboratory for Computational Statistics, Stanford University Technical Report No. 1.

See Also

plot.ppr, supsmu, smooth.spline

Examples

```
# Note: your numerical values may differ
data(rock)
attach(rock)
area1 <- area/10000; peri1 <- peri/10000
rock.ppr <- ppr(log(perm) ~ area1 + peri1 + shape,
                data = rock, nterms = 2, max.terms = 5)
rock.ppr
# Call:
# ppr.formula(formula = log(perm) ~ area1 + peri1 + 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
# area1  0.34357179  0.37071027
# peri1 -0.93781471 -0.61923542
```

```
# shape  0.04961846  0.69218595
#
# Coefficients of ridge terms:
#    term 1    term 2
# 1.6079271 0.5460971

par(mfrow=c(3,2))# maybe: , pty="s")
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)")
```

`predict.loess` *Predict Loess Curve or Surface* `predict.loess`

Description

Predictions from a `loess` fit, optionally with standard errors.

Usage

```
predict.loess(object, newdata = NULL, se = FALSE)
```

Arguments

`object`	an object fitted by `loess`.
`newdata`	an optional data frame specifying points at which to do the predictions. If missing, the original data points are used.
`se`	should standard errors be computed?

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.

Author(s)

B.D. Ripley, based on the `cloess` package of Cleveland, Grosse and Shyu.

See Also

loess

Examples

```
data(cars)
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)
```

---

predict.smooth.spline *Predict from Smoothing Spline Fit* predict.smooth.spline

---

Description

Predict a smoothing spline fit at new points, return the derivative if desired. The predicted fit is linear beyond the original data.

Usage

```
predict.smooth.spline(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.

Value

A list with components

x	The input x.
y	The fitted values or derivatives at x.

Author(s)

B.D. Ripley

See Also

smooth.spline

Examples

```
data(cars)
attach(cars)
cars.spl <- smooth.spline(speed, dist, df=6.4)


## "Proof" that the derivatives are okay, by comparing with approximation
diff.quot <- function(x,y) {
  ## Difference quotient (central differences where available)
  n <- length(x); i1 <- 1:2; i2 <- (n-1):n
  c(diff(y[i1]) / diff(x[i1]), (y[-i1] - y[-i2]) / (x[-i1] - x[-i2]),
    diff(y[i2]) / diff(x[i2]))
}

xx <- unique(sort(c(seq(0,30, by = .2), kn <- unique(speed))))
i.kn <- match(kn, xx)# indices of knots within xx
op <- par(mfrow = c(2,2))
plot(speed, dist, xlim = range(xx), main = "Smooth.spline & derivatives")
lines(pp <- predict(cars.spl, xx), col = "red")
points(kn, pp$y[i.kn], pch = 3, col="dark red")
mtext("s(x)", col = "red")
for(d in 1:3){
  n <- length(pp$x)
  plot(pp$x, diff.quot(pp$x,pp$y), type = 'l', xlab="x", ylab="",
       col = "blue", col.main = "red",
       main= paste("s",paste(rep("'",d), collapse=""),"(x)", sep=""))
  mtext("Difference quotient approx.(last)", col = "blue")
  lines(pp <- predict(cars.spl, xx, deriv = d), col = "red")

  points(kn, pp$y[i.kn], pch = 3, col="dark red")
  abline(h=0, lty = 3, col = "gray")
}
detach(); par(op)
```

`rock` *Measurements on Petroleum Rock Samples* `rock`

Description

Measurements on 48 rock samples from a petroleum reservoir.

Usage

```
data(rock)
```

Format

A data frame with 48 rows and 4 numeric columns.

[,1]	area	area of pores space, in pixels out of 256 by 256
[,2]	peri	perimeter in pixels
[,3]	shape	perimeter/sqrt(area)
[,4]	perm	permeability in milli-Darcies

Details

Twelve core samples from petroleum reservoirs were sampled by 4 cross-sections. Each core sample was measured for permeability, and each cross-section has total area of pores, total perimeter of pores, and shape.

Source

Data from BP Research, image analysis by Ronit Katz, U. Oxford.

`scatter.smooth` *Scatter Plot with Smooth Curve Fitted by Loess* `scatter.smooth`

Description

Plot and add a smooth curve computed by `loess` to a scatter plot.

Usage

```
scatter.smooth(x, y, span = 2/3, degree = 1,
    family = c("symmetric", "gaussian"),
    xlab = deparse(substitute(x)), ylab = deparse(substitute(y)),
    ylim = range(y, prediction$y), evaluation = 50, ...)
loess.smooth(x, y, span = 2/3, degree = 1,
    family = c("symmetric", "gaussian"), evaluation=50, ...)
```

Arguments

`x`	x coordinates for scatter plot.
`y`	y coordinates for scatter plot.
`span`	smoothness parameter for `loess`.
`degree`	degree of local polynomial used.
`family`	if `"gaussian"` fitting is by least-squares, and if `family="symmetric"` a re-descending M estimator is used.
`xlab`	label for x axis.
`ylab`	label for y axis.
`ylim`	the y limits of the plot.
`evaluation`	number of points at which to evaluate the smooth curve.
`...`	graphical parameters.

Details

`loess.smooth` is an auxiliary function.

Value

None.

Author(s)

B.D. Ripley

See Also

loess

Examples

```
data(cars)
attach(cars)
scatter.smooth(speed, dist)
detach()
```

`smooth.spline` *Fit a Smoothing Spline* `smooth.spline`

Description

Fits a cubic smoothing spline to the supplied data.

Usage

```
smooth.spline(x, y, w = rep(1, length(x)), df = 5, spar = 0,
              cv = FALSE, all.knots = FALSE, df.offset = 0, penalty = 1)
```

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
`df`	the desired equivalent number of degrees of freedom (trace of the smoother matrix).
`spar`	the coefficient λ of the integral of the squared second derivative in the fit (penalized log lik.) criterion.
`cv`	ordinary (`TRUE`) or 'generalized' (`FALSE`) cross-validation.
`all.knots`	if `TRUE`, all points in `x` are uses as knots. If `FALSE`, a suitably fine grid of knots is used.
`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.

Details

The `x` vector should contain at least ten distinct values.

If `spar` is missing or 0, the value of `df` is used to determine the degree of smoothing. If both are missing, leave-one-out cross-validation is used to determine λ.

Value

An object of class `"smooth.spline"` with components

`x`	the distinct `x` values in increasing order.
`y`	the fitted values corresponding to `x`.
`w`	the weights used at the unique values of `x`.
`yin`	the y values used at the unique `y` values.
`lev`	leverages, the diagonal values of the smoother matrix.
`cv.crit`	(generalized) cross-validation score.
`pen.crit`	penalized criterion
`df`	equivalent degrees of freedom used.
`spar`	the value of λ chosen.
`fit`	list for use by `predict.smooth.spline`.
`call`	

Author(s)

B.D. Ripley

See Also

`predict.smooth.spline`

Examples

```
data(cars)
attach(cars)
plot(speed, dist, main = "data(cars)  &  smoothing splines")
cars.spl <- smooth.spline(speed, dist)
(cars.spl)
all(cars.spl $ w == table(speed)) # TRUE (weights = multiplicities)

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()
```

---

supsmu *Friedman's SuperSmoother* supsmu

Description

Smooth the (x, y) values by Friedman's "super smoother".

Usage

```
supsmu(x, y, wt = rep(1, length(y)), span = "cv", periodic = FALSE,
       bass = 0)
```

Arguments

`x`	x values for smoothing
`y`	y values for smoothing
`wt`	case weights
`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 indicating increasing smoothness.

Details

`supsmu` is a running lines smoother which chooses 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 0.5n, 0.2n and 0.05n. If `span` is specified, as 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.

Author(s)

B. D. Ripley

References

Friedman, J. H. (1984) SMART User's Guide. Laboratory for Computational Statistics, Stanford University Technical Report No. 1.

Friedman, J. H. (1984) A variable span scatterplot smoother. Laboratory for Computational Statistics, Stanford University Technical Report No. 5.

See Also

ppr

Examples

```
data(cars)
attach(cars)
plot(speed, dist)
lines(supsmu(speed, dist))
lines(supsmu(speed, dist, bass=7), lty=2)
detach()
```

Chapter 6

The mva package

---

as.hclust *Convert Objects to Class hclust* as.hclust

Description

Converts objects from other hierarchical clustering functions to class "hclust".

Usage

```
as.hclust(x, ...)
as.hclust.twins(x)
```

Arguments

x	Hierarchical clustering object

Details

Currently there is only support for converting objects of class "twins" as produced by the functions diana and agnes from the package 'cluster'.

Value

An object of class "hclust".

See Also

hclust, diana, agnes

Examples

```
x <- matrix(rnorm(30), ncol=3)
hc <- hclust(dist(x), method="complete")

library(cluster)
ag <- agnes(x, method="complete")

x11()
```

```
par(mfrow=c(1,2))
plot(hc)
mtext("hclust", side=1)
plot(as.hclust(ag))
mtext("agnes", side=1)
```

`biplot` *Biplot of Multivariate Data* `biplot`

Description

Plot a biplot on the current graphics device.

Usage

```
biplot(x, ...)

biplot.default(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, ...)
```

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 variables.
`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.
`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 get the two sets on 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`.

Details

A biplot is plot which aims to represent both the observations and variables of a matrix of multivariate 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.

Author(s)

B.D. Ripley

References

K. R. Gabriel (1971). The biplot graphical display of matrices with application to principal component analysis. *Biometrika* **58**, 453–467.

J.C. Gower and D. J. Hand (1996). *Biplots*. Chapman & Hall.

See Also

`biplot.princomp`, also for examples.

`biplot.princomp` *Biplot for Principal Components* `biplot.princomp`

Description

Produces a biplot (in the strict sense) from the output of `princomp`.

Usage

```
biplot.princomp(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

Gabriel, K. R. (1971). The biplot graphical display of matrices with applications to principal component analysis. *Biometrika*, **58**, 453–467.

Gabriel, K. R. and Odoroff, C. L. (1990). Biplots in biomedical research. *Statistics in Medicine*, **9**, 469–485.

See Also

`biplot`, `princomp`.

Examples

```
data(USArrests)
biplot(princomp(USArrests))
```

---

cancor — *Canonical Correlations* — cancor

Description

Compute the canonical correlations between two data matrices.

Usage

```
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.

Details

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.

Value

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 `x` variables.

References

Hotelling H. (1936). Relations between two sets of variables. *Biometrika*, **28**, 321–327.

Seber, G. A. F. (1984). *Multivariate Analysis*. New York: Wiley, p. 506f.

See Also

`qr`, `svd`.

Examples

```
data(LifeCycleSavings)
pop <- LifeCycleSavings[, 2:3]
oec <- LifeCycleSavings[, -(2:3)]
str(cancor(pop, oec))

x <- matrix(rnorm(150), 50, 3)
y <- matrix(rnorm(250), 50, 5)
str(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)
```

`cmdscale` *Classical (Metric) Multidimensional Scaling* `cmdscale`

Description

Classical muiltidimensional scaling of a data matrix.

Usage

```
cmdscale(d, k = 2, eig = 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.
`eig`	indicates whether eigenvalues 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.

Value

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`	if requested, the eigenvalues computed during the scaling process.

Note

The S version of this function provides for computing an additional "fiddle" factor suggested by Torgerson. R does not provide this option.

References

Seber, G. A. F. (1984). *Multivariate Analysis*. New York: Wiley.

Torgerson, W. S. (1958). *Theory and Methods of Scaling*. New York: Wiley.

See Also

`dist`. Also `isoMDS` and `sammon` in package 'MASS'.

Examples

```
data(eurodist)
loc <- cmdscale(eurodist)
x <- loc[,1]
y <- -loc[,2]
plot(x, y, type="n", xlab="", ylab="")
text(x, y, names(eurodist), cex=0.5)
```

`cutree` — *Cut a tree into groups of data* — `cutree`

Description

Cuts a tree, e.g., as resulting from `hclust`, into several groups either by specifying the desired number of groups or the cut height.

Usage

```
cutree(tree, k=NULL, h=NULL)
```

Arguments

`tree`	a tree as produced by `hclust`
`k`	An integer scalar or vector with the desired number of groups
`h`	A 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 an vector with group meberships if `k` or `h` are scalar, otherwise a matrix with group meberships is returned where each column corresponds to the elements of `k` or `h`, respectively (which are also used as column names).

See Also

`hclust`

Examples

```
require(mva)
data(USArrests)

hc <- hclust(dist(USArrests))

cutree(hc, k=2:5)
cutree(hc, h=250)
```

`dist` *Distance Matrix Computation* `dist`

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)

print.dist(dist.obj, diag = NULL, upper = NULL)
as.matrix.dist(dist.obj)
as.dist(m, diag = NULL, upper = NULL)
```

Arguments

`x`	A matrix or (data frame). Distances between the rows of `x` will be computed.
`method`	The distance measure to be used. This must be one of `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"` or `"binary"`. Any unambiguous substring can be given.
`diag`	A logical value indicating whether the diagonal of the distance matrix should be printed by `print.dist`.

`upper`	A logical value indicating whether the upper triangle of the distance matrix should be printed by `print.dist`.
`m`	A distance matrix to be converted to a dist object (only lower triangle is used, the rest is ignored).

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: $\sum_i |x_i - y_i|/|x_i + y_i|$
- Count the number of different bits in x and y where at least one of the two bits is 1, i.e., components where both bits are zero are ignored.

The functions `as.matrix.dist()` and `as.dist()` can be used for conversion between objects of class `"dist"` and conventional distance matrices and vice versa.

Value

The lower triangle of the distance matrix stored by columns in a single vector. The vector has the attributes `"Size"`, `"Diag"`, `"Upper"`, `"Labels"` and `"class"` equal to `"dist"`.

References

Mardia, K. V., J. T. Kent and J. M. Bibby (1979). *Multivariate Analysis*, London: Academic Press.

See Also

`hclust`.

Examples

```
x <- matrix(rnorm(100),nrow=5)
dist(x)
dist(x, diag = TRUE)
dist(x, upper = TRUE)
m <- as.matrix(dist(x))
as.dist(m)
```

`hclust` *Hierarchical Clustering* `hclust`

Description

Performs a hierarchical cluster analysis on a set of dissimilarities.

Usage

```
hclust(d, method = "complete")

plot.hclust(hclust.obj, labels, hang = 0.1, ...)
```

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"`.
`hclust.obj`	an object of the type produced by `hclust`.
`hang`	The fraction of the plot height 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 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.

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.

An number of different clustering methods are provided. *Ward's* minimum variance method aims at finding compact, spherical clusters. The *complete linkage* method finds similar clusters. The *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.

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 `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 **hclust** which describes the tree produced by the clustering process. The object is a list with components:

`merge`	an $n-1$ by 2 matrix. Row i of `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 `merge` indicate agglomerations of singletons, and positive entries indicate agglomerations of non-singletons.
`height`	a set of $n-1$ non-decreasing real values. The clustering *height*: that is, the value of the criterion associated with the clustering `method` for the particular agglomeration.
`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 `merge` will not have crossings of the branches.

`labels`	labels for each of the objects being clustered.
`call`	the call which produced the result.
`method`	the cluster method that has been used.
`dist.method`	the distance that has been used to create `d` (only returned if the distance object has a `"method"` attribute).

Author(s)

The `hclust` function is based on Fortran code contributed to STATLIB by F. Murtagh.

References

Everitt, B. (1974). *Cluster Analysis*. London: Heinemann Educ. Books.

Hartigan, J. A. (1975). *Clustering Algorithms*. New York: Wiley.

Sneath, P. H. A. and R. R. Sokal (1973). *Numerical Taxonomy*. San Francisco: Freeman.

Anderberg, M. R. (1973). *Cluster Analysis for Applications*. Academic Press: New York.

Gordon, A. D. (1981). *Classification*. London: Chapman and Hall.

Murtagh, F. (1985). "Multidimensional Clustering Algorithms", in *COMPSTAT Lectures 4*. Wuerzburg: Physica-Verlag (for algorithmic details of algorithms used).

See Also

`kmeans`.

Examples

```
library(mva)
data(USArrests)
hc <- hclust(dist(USArrests), "ave")
plot(hc, hang=-1)
plot(hc)
```

`identify.hclust` *Identify Clusters in a Dendrogram* `identify.hclust`

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

```
identify.hclust(HCOBJ, FUN=NULL, N=20, MAXCLUSTER=20, DEV.FUN=NULL, ...)
```

Arguments

`HCOBJ`	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 amde 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. If 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, or by clicking outside the graphics window.

Value

Either a list of data point index vectors or a list of return values of `FUN`.

Examples

```
library(mva)
data(USArrests)
hca <- hclust(dist(USArrests))
plot(hca)
x <- identify.hclust(hca)
x

data(iris)
hci <- hclust(dist(iris[,1:4]))
plot(hci)
identify.hclust(hci, function(k) print(table(iris[k,5])))

x11()
dev.set(2)
plot(hci)
identify.hclust(hci, function(k) barplot(table(iris[k,5])), DEV.FUN=3)
```

kmeans	*K-Means Clustering*	kmeans

Description

Perform k-means clustering on a data matrix.

Usage

```
kmeans(x, centers, iter.max = 10)
```

Arguments

x	A data frame or matrix of data.
centers	Either the number of clusters or a set of initial cluster centers. If the first, a random set of rows in x are chosen as the initial centers.
iter.max	The maximum number of iterations allowed.

Details

The data given by x is clustered by the k-Means algorithm. When this terminates, all cluster centres are at to 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.

Value

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.

References

Hartigan, J.A. and Wong, M.A. (1979). A K-means clustering algorithm. *Applied Statistics* **28**, 100–108.

Examples

```
# a 2-dimensional example
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
cl <- kmeans(x, 2, 20)
plot(x, col = cl$cluster)
points(cl$centers, col = 1:2, pch = 8)
```

`prcomp` *Principal Components Analysis* `prcomp`

Description

Performs a principal components analysis on the given data matrix and returns the results as an object of class `prcomp`.

Usage

```
prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE, tol = NULL)
```

Arguments

`x`	a 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. Alternately, 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. With the default null setting, no components are omitted. Other settings for tol could be `tol = 0` or `tol = sqrt(.Machine$double.eps)`.

Details

The calculation is done by a singular value decomposition of the 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 an list with class `"prcomp"` containing the following components:

`sdev`	the standard deviation of the principal components (i.e., the eigenvalues of the cov 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 olumns contain the eigenvectors). The function `princomp` returns this in the element `loadings`.
`x`	if `retx` is true the value of the rotated data (the data multiplied by the `rotation` matrix) is returned.

References

Mardia, K. V., J. T. Kent, J and M. Bibby (1979), *Multivariate Analysis*, London: Academic Press.

Venables, W. N. and B. D. Ripley (1997), *Modern Applied Statistics with S-PLUS*, Springer-Verlag.

See Also

princomp, cor, cov, svd, eigen.

Examples

```
## the variances of the variables in the
## USArrests data vary by orders of magnitude
data(USArrests)
prcomp(USArrests)
prcomp(USArrests, scale = TRUE)
plot(prcomp(USArrests))
summary(prcomp(USArrests))
```

---

princomp *Principal Components Analysis* princomp

Description

`princomp` performs a principal components analysis on the given data matrix and returns the results as an object of class `princomp`.

`loadings` extracts the loadings.

`screeplot` plots the variances against the number of the principal component. This is also the `plot` method.

Usage

```
princomp(x, cor = FALSE, scores = TRUE, covmat = NULL,
         subset = rep(TRUE, nrow(as.matrix(x))))
loadings(x)
plot(x, npcs = min(10, length(x$sdev)),
          type = c("barplot", "lines"), ...)
screeplot(x, npcs = min(10, length(x$sdev)),
          type = c("barplot", "lines"), ...)

print(x, ...)  summary(object)  predict(object, ...)
```

Arguments

`x`	a 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`, `cov.mve` or `cov.mcd`. If supplied, this is used rather than the covariance matrix of `x`.
`subset`	a vector used to select rows (observations) of the data matrix `x`.
`x, object`	an object of class `"princomp"`, as from `princomp()`.
`npcs`	the number of principal components to be plotted.
`type`	the type of plot.
`...`	graphics parameters.

Details

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.

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).
`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.
`call`	the matched call.

References

Mardia, K. V., J. T. Kent and J. M. Bibby (1979). *Multivariate Analysis*, London: Academic Press.

Venables, W. N. and B. D. Ripley (1997, 9). *Modern Applied Statistics with S-PLUS*, Springer-Verlag.

See Also

`prcomp`, `cor`, `cov`, `eigen`.

Examples

```
## the variances of the variables in the
## USArrests data vary by orders of magnitude
data(USArrests)
(pc.cr <- princomp(USArrests))
princomp(USArrests, cor = TRUE)
```

```
princomp(scale(USArrests, scale = TRUE, center = TRUE), cor = FALSE)

summary(pc.cr <- princomp(USArrests))
loadings(pc.cr)
plot(pc.cr) # does a screeplot.
biplot(pc.cr)
```

`rect.hclust` *Draw Rectangles Around Hierarchical Clusters* `rect.hclust`

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

```
rect.hclust(hclust.obj, k=NULL, which=NULL, x=NULL, y=NULL,
            border=2, cluster=NULL)
```

Arguments

`hclust.obj`	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` seleccts 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

```
library(mva)
data(USArrests)
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
```

Chapter 7

The nls package

asOneSidedFormula	*Convert to One-Sided Formula*	asOneSidedFormula

Description

Names, expressions, numeric values, and strings are converted to one-sided formulas. If `object` is a formula, it must be one-sided, in which case it is returned unaltered.

Usage

```
asOneSidedFormula(object)
```

Arguments

`object`	a one-sided formula, an expression, a numeric value, or a character string.

Value

a one-sided formula representing `object`

Author(s)

Jose Pinheiro and Douglas Bates

See Also

formula

Examples

```
library(nls)
asOneSidedFormula("age")
asOneSidedFormula(~ age)
```

BOD *Biochemical Oxygen Demand* BOD

Description

The BOD data frame has 6 rows and 2 columns giving the biochemical oxygen demand versus time in an evaluation of water quality.

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

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley, Appendix A1.4.

Originally from Marske (1967), *Biochemical Oxygen Demand Data Interpretation Using Sum of Squares Surface* M.Sc. Thesis, University of Wisconsin – Madison.

Examples

```
library(nls)
data(BOD)
# 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 )
```

ChickWeight *Weight versus age of chicks on different diets* ChickWeight

Description

The ChickWeight data frame has 578 rows and 4 columns from an experiment on the effect of diet on early growth of chicks.

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` $< \ldots <$ `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

Crowder, M. and Hand, D. (1990), *Analysis of Repeated Measures*, Chapman and Hall (example 5.3)

Hand, D. and Crowder, M. (1996), *Practical Longitudinal Data Analysis*, Chapman and Hall (table A.2)

Pinheiro, J. C. and Bates, D. M. (2000) *Mixed-effects Models in S and S-PLUS*, Springer.

Examples

```
library(nls)
data(ChickWeight)
coplot(weight ~ Time | Chick, data = ChickWeight,
    type = "b", show = FALSE)
## fit a representative chick
fm1 <- nls(weight ~ SSlogis( Time, Asym, xmid, scal ),
    data = ChickWeight, subset = Chick == 1)
summary( fm1 )
```

`clearNames` *Remove the Names from an Object* `clearNames`

Description

This function sets the `names` attribute of `object` to `NULL` and returns the object.

Usage

```
clearNames(object)
```

Arguments

`object`	an object that may have a `names` attribute

Value

An object similar to `object` but without names.

Author(s)

Douglas Bates and Saikat DebRoy

See Also

`setNames`

Examples

```
library( nls )
data( women )
lapply( women, mean )               # has a names attribute
clearNames( lapply( women, mean ) ) # removes the names
```

CO2 — *Carbon Dioxide uptake in grass plants* — CO2

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*

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/m^2 sec).

Details

The CO_2 uptake of six plants from Quebec and six plants from Mississippi was measured at several levels of ambient CO_2 concentration. Half the plants of each type were chilled overnight before the experiment was conducted.

Source

Potvin, C., Lechowicz, M. J. and Tardif, S. (1990) "The statistical analysis of ecophysiological response curves obtained from experiments involving repeated measures", *Ecology*, **71**, 1389–1400.

Pinheiro, J. C. and Bates, D. M. (2000) *Mixed-effects Models in S and S-PLUS*, Springer.

Examples

```
library(nls)
data(CO2)
coplot(uptake ~ conc | Plant, data = CO2, show = FALSE, type = "b")
## fit the data for the first plant
fm1 <- nls(uptake ~ SSasymp(conc, Asym, lrc, c0),
   data = CO2, subset = Plant == 'Qn1')
summary(fm1)
## 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)
```

`DNase` *Elisa assay of DNase* `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.

Format

This data frame contains the following columns:

Run an ordered factor with levels `10` $< \ldots <$ `3` indicating the assay run.

conc a numeric vector giving the known concentration of the protein.

density a numeric vector giving the measured optical density (dimensionless) in the assay. Duplicate optical density measurements were obtained.

Source

Davidian, M. and Giltinan, D. M. (1995) *Nonlinear Models for Repeated Measurement Data*, Chapman & Hall (section 5.2.4, p. 134)

Pinheiro, J. C. and Bates, D. M. (2000) *Mixed-effects Models in S and S-PLUS*, Springer.

Examples

```
library(nls)
data(DNase)
coplot(density ~ conc | Run, data = DNase,
    show = FALSE, type = "b")
coplot(density ~ log(conc) | Run, data = DNase,
    show = FALSE, type = "b")
## fit a representative run
fm1 <- nls(density ~ SSlogis( log(conc), Asym, xmid, scal ),
    data = DNase, subset = Run == 1)
## compare with a four-parameter logistic
fm2 <- nls(density ~ SSfpl( log(conc), A, B, xmid, scal ),
```

```
    data = DNase, subset = Run == 1)
summary(fm2)
anova(fm1, fm2)
```

`formula.nls` *Extract Model Formula from nls Object* `formula.nls`

Description

Returns the model used to fit `object`.

Usage

```
formula(object)
```

Arguments

`object`	an object inheriting from class `nls`, representing a non-linear least squares fit.

Value

a formula representing the model used to obtain `object`.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

`nls`, `formula`

Examples

```
library(nls)
data(Orange)
fm1 <- nls(circumference ~ A/(1+exp((B-age)/C)), Orange,
  start = list(A=160, B=700, C = 350))
formula(fm1)
```

`getInitial` *Get Initial Parameter Estimates* `getInitial`

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

```
getInitial(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

```
library(nls)
data(Puromycin)
PurTrt <- Puromycin[ Puromycin$state == "treated", ]
getInitial( rate ~ SSmicmen( conc, Vm, K ), PurTrt )
```

`Indometh` *Pharmacokinetics of Indomethicin* `Indometh`

Description

The `Indometh` data frame has 66 rows and 3 columns of data on the pharmacokinetics of indomethicin.

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

Kwan, Breault, Umbenhauer, McMahon and Duggan (1976), "Kinetics of Indomethicin absorption, elimination, and enterohepatic circulation in man", *Journal of Pharmacokinetics and Biopharmaceutics*, **4**, 255–280.

Davidian, M. and Giltinan, D. M. (1995) *Nonlinear Models for Repeated Measurement Data*, Chapman & Hall (section 5.2.4, p. 134)

Pinheiro, J. C. and Bates, D. M. (2000) *Mixed-effects Models in S and S-PLUS*, Springer.

Examples

```
library(nls)
data(Indometh)
fm1 <- nls(conc ~ SSbiexp(time, A1, lrc1, A2, lrc2),
           data = Indometh, subset = Subject == 1)
summary(fm1)
```

`Loblolly` *Growth of Loblolly pine trees* `Loblolly`

Description

The `Loblolly` data frame has 84 rows and 3 columns of records of the growth of Loblolly pine trees.

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

Kung, F. H. (1986), "Fitting logistic growth curve with predetermined carrying capacity", *Proceedings of the Statistical Computing Section, American Statistical Association*, 340–343.

Pinheiro, J. C. and Bates, D. M. (2000) *Mixed-effects Models in S and S-PLUS*, Springer.

Examples

```
library(nls)
data(Loblolly)
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)))
```

`logLik` *Extract Log-Likelihood* `logLik`

Description

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `corStruct`, `gls`, `lm`, `lme`, `lmList`, `lmeStruct`, `reStruct`, and `varFunc`.

Usage

```
logLik(object, ...)
```

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.

Value

will depend on the method function used; see the appropriate documentation.

Author(s)

Jose Pinheiro and Douglas Bates

Examples

```
## see the method function documentation
```

`nls` *Nonlinear Least Squares* `nls`

Description

Determine the nonlinear least squares estimates of the parameters.

Usage

```
nls(formula, data, start, control=nls.control(),
    algorithm="default", trace=F, subset, na.action)
```

Arguments

`formula`	a nonlinear model formula including variables and parameters
`data`	an optional data frame in which to evaluate the variables in `formula`
`start`	a named list or named numeric vector of starting estimates
`control`	an optional list of control settings. See `nlsControl` for the names of the settable control values and their effect.
`algorithm`	character string specifying the algorithm to use. The default algorithm is a Gauss-Newton algorithm. The other alternative is "plinear", the Golub-Pereyra algorithm for partially linear least-squares models.
`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 `NA`s.

Details

An `nls` object is a type of fitted model object. It has methods for the generic functions `coef`, `formula`, `resid`, `print`, `summary`, and `fitted`.

Value

A list of

`m`	an `nlsModel` object incorporating the model
`data`	the expression that was passed to `nls` as the data argument. The actual data values are present in the environment of the `m` component.

Author(s)

Douglas M. Bates and Saikat DebRoy

References

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley

See Also

`nlsModel`

Examples

```
library( nls )
data( DNase )
DNase1 <- DNase[ DNase$Run == 1, ]
## using a selfStart model
fm1DNase1 <- nls( density ~ SSlogis( log(conc), Asym, xmid, scal ), DNase1 )
summary( 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 )
```

```
summary( 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 )
summary( fm3DNase1 )
```

---

nls.control *Control the Iterations in nls* nls.control

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)
```

Value

A `list` with exactly three components:

`maxiter`

`tol`

`minFactor`

Author(s)

Douglas Bates and Saikat DebRoy

References

Bates and Watts (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley.

See Also

`nls`

Examples

```
nls.control(minFactor = 1/2048)
```

`nlsModel` *Create an nlsModel Object* `nlsModel`

Description

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.

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 parmeter 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

`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 response for `newdata`.

Author(s)

Douglas M. Bates and Saikat DebRoy

References

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley

See Also

`nls`

Examples

```
library( nls )
data( DNase )
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* `NLSstAsymptotic`

Description

Fits the asymptotic regression model, in the form `b0 + b1*exp(-exp(lrc) * x)` to the `xy` data. This can be used as a building block in determining starting estimates for more complicated models.

Usage

```
NLSstAsymptotic(xy, yval)
```

Arguments

`xy`	a `sortedXyData` object

Value

A numeric value of length 3 with components labelled `b0`, `b1`, and `lrc`

Author(s)

Jose Pinheiro and Douglas Bates

See Also

SSasymp

Examples

```
library( nls )
data( Loblolly )
Lob.329 <- Loblolly[ Loblolly$Seed == "329", ]
NLSstAsymptotic(sortedXyData(expression(age), expression(height), Lob.329 ))
```

`NLSstClosestX` *Inverse Interpolation* `NLSstClosestX`

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

```
library( nls )
data( DNase )
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* `NLSstLfAsymptote`

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

```
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

Examples

```
library( nls )
data( DNase )
DNase.2 <- DNase[ DNase$Run == "2", ]
DN.srt <- sortedXyData( expression(log(conc)), expression(density), DNase.2 )
NLSstLfAsymptote( DN.srt )
```

`NLSstRtAsymptote` *Horizontal Asymptote on the Right Side* `NLSstRtAsymptote`

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

```
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

```
library( nls )
data( DNase )
DNase.2 <- DNase[ DNase$Run == "2", ]
DN.srt <- sortedXyData( expression(log(conc)), expression(density), DNase.2 )
NLSstRtAsymptote( DN.srt )
```

`Orange` *Growth of orange trees* `Orange`

Description

The `Orange` data frame has 35 rows and 3 columns of records of the growth of orange trees.

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

Draper, N. R. and Smith, H. (1998), *Applied Regression Analysis (3rd ed)*, Wiley (exercise 24.N).

Pinheiro, J. C. and Bates, D. M. (2000) *Mixed-effects Models in S and S-PLUS*, Springer.

Examples

```
library(nls)
data(Orange)
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)))
```

`plot.profile.nls` *Plot a profile.nls Object* `plot.profile.nls`

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

```
plot.profile.nls(x, levels, conf, nseg, absVal, ...)
```

Author(s)

Douglas M. Bates and Saikat DebRoy

References

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley (chapter 6)

See Also

`nls`, `profile`, `profile.nls`

Examples

```
library( nls )
data( BOD )
# 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)
```

`predict.nls` *Predicting from Nonlinear Least Squares Fits* `predict.nls`

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

```
predict[.nls](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 with values of the input variables for the model in `object`. 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 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 predictions

`residual.scale`
: residual standard deviations

`df`
: degrees of freedom for residual

See Also

The model fitting function `nls`, `predict`.

Examples

```
library( nls )
data( BOD )
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)
```

`profile.nls` *Method for Profiling nls Objects* `profile.nls`

Description

Investigates behavior of the log-likelihood function near the solution represented by `fitted`.

Usage

```
profile.nls(fitted, which, 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.

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

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley (chapter 6)

See Also

`nls`, `profile`, `profiler.nls`, `plot.profile.nls`

Examples

```
library( nls )
data( BOD )
# 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 Non-Linear Models* `profiler`

Description

Create a profiler object for the model object `fitted`.

Usage

```
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* `profiler.nls`

Description

Create a profiler object for the model object `fitted` of class `nls`.

Usage

```
profiler.nls(fitted)
```

Arguments

`fitted`	the original fitted model object of class `nls`.

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

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley

See Also

`nls`, `nlsModel`, `profiler`, `profile.nls`

Examples

```
library( nls )
data( BOD )
# 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$getProfile(c(F,T), 16.0)
# vary lrc
prof1$setDefault(varying = c(F, T))
# 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()
```

`Puromycin`	*Reaction velocity of an enzymatic reaction*	`Puromycin`

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.

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

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley, Appendix A1.3.

Treloar, M. A. (1974), *Effects of Puromycin on Galactosyltransferase in Golgi Membranes*, M.Sc. Thesis, U. of Toronto.

Examples

```
library(nls)
data(Puromycin)
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)
```

`selfStart` *Construct Self-starting Nonlinear Models* `selfStart`

Description

This function is generic; methods functions can be written to handle specific classes of objects. Available methods include `selfStart.default` and `selfStart.formula`. See the documentation on the appropriate method function.

Usage

```
selfStart(model, initial, parameters, template)
```

Value

a function object of the `selfStart` class.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

`selfStart.default`, `selfStart.formula`

Examples

```
## see documentation for the methods
```

`selfStart.default` *Construct Self-starting Nonlinear Models* `selfStart.default`

Description

A method for the generic function 'selfStart' for formula objects.

Usage

```
selfStart(model, initial, parameters, template)
```

Arguments

`model`	a function object defining a nonlinear model.
`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, template`	these arguments are included for consistency with the generic function, but are not used in the `default` method. See the documentation on `selfStart.formula`.

Value

a function object of class `selfStart`, corresponding to a self-starting nonlinear model function. 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

`selfStart.formula`

Examples

```
library(nls)
# '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'

# self-starting first order compartment model

SSfol <- selfStart(first.order.log.model, first.order.log.initial)
```

`selfStart.formula` *Construct Self-starting Nonlinear Models* `selfStart.formula`

Description

A method for the generic function 'selfStart' for formula objects.

Usage

```
selfStart(model, initial, parameters, template)
```

Arguments

`model`	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.

Value

a function object of class `selfStart`, 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

`selfStart.default`, `deriv`

Examples

```
library( nls )

## 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"))
```

`setNames` *Set the Names in an Object* `setNames`

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

```
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.

Author(s)

Douglas M. Bates and Saikat DebRoy

See Also

`clearNames`

Examples

```
library( nls )
setNames( 1:3, c("foo", "bar", "baz") )
# this is just a short form of
tmp <- 1:3
names(tmp) <-  c("foo", "bar", "baz")
tmp
```

sortedXyData	*Create a sortedXyData object*	sortedXyData

Description

This is a constructor function for the class of `sortedXyData` objects. These objects are mostly used in the `initial` function for a self-starting nonlinear regression model, which will be of the `selfStart` class.

Usage

```
sortedXyData(x, y, data)
```

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

```
library( nls )
data( DNase )
DNase.2 <- DNase[ DNase$Run == "2", ]
sortedXyData( expression(log(conc)), expression(density), DNase.2 )
```

SSasymp *Asymptotic Regression Model* SSasymp

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)*exp(-exp(lrc)*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

```
library( nls )
data( Loblolly )
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)
```

SSasympOff *Asymptotic Regression Model with an Offset* SSasympOff

Description

This `selfStart` model evaluates an alternative parameterization of the asymptotic regression function and the gradient with respect to those parameters. It has an `initial` attribute that creates initial estimates of the parameters `Asym`, `lrc`, and `c0`.

Usage

```
SSasympOff(input, Asym, lrc, c0)
```

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`).
`lrc`	a numeric parameter representing the natural logarithm of the rate constant.
`c0`	a numeric parameter representing the `input` for which the response is zero.

Value

a numeric vector of the same length as `input`. It is the value of the expression `Asym*(1 - exp(-exp(lrc)*(input - c0)))`. If all of the arguments `Asym`, `lrc`, and `c0` 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

```
library( nls )
data( CO2 )
CO2.Qn1 <- CO2[CO2$Plant == "Qn1", ]
SSasympOff( CO2.Qn1$conc, 32, -4, 43 )  # response only
Asym <- 32; lrc <- -4; c0 <- 43
SSasympOff( CO2.Qn1$conc, Asym, lrc, c0 ) # response and gradient
getInitial(uptake ~ SSasymp( conc, Asym, lrc, c0), data = CO2.Qn1)
## Initial values are in fact the converged values
fm1 <- nls(uptake ~ SSasymp( conc, Asym, lrc, c0), data = CO2.Qn1)
summary(fm1)
```

`SSasympOrig` *Asymptotic Regression Model through the Origin* `SSasympOrig`

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 `Asym*(1 - exp(-exp(lrc)*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

```
library( nls )
data( Loblolly )
Lob.329 <- Loblolly[ Loblolly$Seed == "329", ]
SSasympOrig( Lob.329$age, 100, -3.2 )  # response only
Asym <- 100; lrc <- -3.2
SSasympOrig( Lob.329$age, Asym, lrc ) # response and gradient
getInitial(height ~ SSasympOrig(age, Asym, lrc), data = Lob.329)
## Initial values are in fact the converged values
fm1 <- nls(height ~ SSasympOrig( age, Asym, lrc), data = Lob.329)
summary(fm1)
```

SSbiexp *Biexponential model* SSbiexp

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 `A1*exp(-exp(lrc1)*input)+A2*exp(-exp(lrc2)*input)`. If all of the arguments `A1`, `lrc1`, `A2`, and `lrc2` 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

```
library( nls )
data( Indometh )
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
fm1 <- nls(conc ~ SSbiexp(time, A1, lrc1, A2, lrc2), data = Indo.1)
summary(fm1)
```

SSfol *First-order Compartment Model* SSfol

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.
`lKe`	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 * exp(lKe+lKa-lCl) * (exp(-exp(lKe)*input)-exp(-exp(lKa)*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

```
library( nls )
data( Theoph )
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
fm1 <- nls(conc ~ SSfol(Dose, Time, lKe, lKa, lCl), data = Theoph.1)
summary(fm1)
```

SSfpl *Four-parameter Logistic Model* SSfpl

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)
```

Arguments

`input`	a numeric vector of values at which to evaluate the model.
`A`	a numeric parameter representing the horizontal asymptote on the left side (very small values of `input`).
`B`	a numeric parameter representing the horizontal asymptote on the right side (very large values of `input`).
`xmid`	a numeric parameter representing the `input` value at the inflection point of the curve. The value of `SSfpl` will be midway between `A` and `B` at `xmid`.
`scal`	a numeric scale parameter on the `input` axis.

Value

a numeric vector of the same length as `input`. It is the value of the expression `A+(B-A)/(1+exp((xmid-input)/scal))`. If all of the arguments `A`, `B`, `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

```
library(nls)
data( ChickWeight )
Chick.1 <- ChickWeight[ChickWeight$Chick == 1, ]
SSfpl( Chick.1$Time, 13, 368, 14, 6 )  # response only
A <- 13; B <- 368; xmid <- 14; scal <- 6
SSfpl( Chick.1$Time, A, B, xmid, scal ) # response and gradient
getInitial(weight ~ SSfpl(Time, A, B, xmid, scal), data = Chick.1)
## Initial values are in fact the converged values
fm1 <- nls(weight ~ SSfpl(Time, A, B, xmid, scal), data = Chick.1)
summary(fm1)
```

SSlogis	*Logistic Model*	SSlogis

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

```
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.

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

```
library(nls)
data( ChickWeight )
Chick.1 <- ChickWeight[ChickWeight$Chick == 1, ]
SSlogis( Chick.1$Time, 368, 14, 6 )  # response only
Asym <- 368; xmid <- 14; scal <- 6
SSlogis( Chick.1$Time, Asym, xmid, scal ) # response and gradient
getInitial(weight ~ SSlogis(Time, Asym, xmid, scal), data = Chick.1)
## Initial values are in fact the converged values
fm1 <- nls(weight ~ SSlogis(Time, Asym, xmid, scal), data = Chick.1)
summary(fm1)
```

SSmicmen *Michaelis-Menten Model* SSmicmen

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

```
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`.

Author(s)

Jose Pinheiro and Douglas Bates

See Also

`nls`, `selfStart`

Examples

```
library( nls )
data( Puromycin )
PurTrt <- Puromycin[ Puromycin$state == "treated", ]
SSmicmen( PurTrt$conc, 200, 0.05 )  # response only
Vm <- 200; K <- 0.05
SSmicmen( PurTrt$conc, Vm, K ) # response and gradient
getInitial(rate ~ SSmicmen(conc, Vm, K), data = PurTrt)
## Initial values are in fact the converged values
fm1 <- nls(rate ~ SSmicmen(conc, Vm, K), data = PurTrt)
summary( fm1 )
## Alternative call using the subset argument
fm2 <- nls(rate ~ SSmicmen(conc, Vm, K), data = Puromycin,
           subset = state == "treated")
summary(fm2)
```

Chapter 8

The splines package

---

asVector — *Coerce an Object to a Vector* — asVector

Description

This is a generic function. Methods for this function coerce objects of given classes to vectors.

Usage

```
asVector(object)
```

Arguments

object	An object.

Value

a vector

Author(s)

Douglas Bates and Bill Venables

See Also

xyVector

Examples

```
library( splines )
data( women )
ispl <- interpSpline( weight ~ height,  women )
pred <- predict(ispl)
class(pred)
str(pred)
asVector(pred)
```

backSpline *Monotone Inverse Spline* backSpline

Description

Create a monotone inverse of a monotone natural spline.

Usage

```
backSpline(object)
```

Arguments

object	An object that inherits from class nbSpline or npolySpline. That is, the object must represent a natural interpolation spline but it can be either in the B-spline representation or the piecewise polynomial representation. The spline is checked to see if represents a monotone function.

Value

An object of class polySpline that contains the piecewise polynomial representation of a function that has the appropriate values and derivatives at the knot positions to be an inverse of the spline represented by object. Technically this object is not a spline because the second derivative is not constrained to be continuous at the knot positions. However, it is often a much better approximation to the inverse than fitting an interpolation spline to the y/x pairs.

Author(s)

Douglas Bates and Bill Venables

See Also

interpSpline

Examples

```
library( splines )
data( women )
ispl <- interpSpline( women$height, women$weight )
bspl <- backSpline( ispl )
plot( bspl )                    # plots over the range of the knots
points( women$weight, women$height )
```

bs *Generate a Basis for Polynomial Splines* bs

Description

Generate the B-spline basis matrix for a cubic spline.

Usage

```
bs(x, df, knots, degree=3, intercept=FALSE, Boundary.knots)
```

Arguments

`x`	the predictor variable.
`df`	degrees of freedom; one can specify `df` rather than `knots`; `bs()` then chooses `df-degree-1` knots at suitable quantiles of `x`.
`knots`	the *internal* breakpoints that define the spline. The default is `NULL`, which results in a basis for ordinary polynomial regression. Typical values are the mean or median for one knot, quantiles for more knots. See also `Boundary.knots`.
`degree`	degree of the piecewise polynomial—default is 3 for cubic splines.
`intercept`	if `TRUE`, an intercept is included in the basis; default is `FALSE`.
`Boundary.knots`	boundary points at which to anchor the B-spline basis (default the range of the data). If both `knots` and `Boundary.knots` are supplied, the basis parameters do not depend on `x`. Data can extend beyond `Boundary.knots`

Value

A matrix of dimension `length(x) * df`, where either `df` was supplied or if `knots` were supplied, `df = length(knots) + 3 + intercept`. Attributes are returned that correspond to the arguments to `bs`, and explicitly give the `knots`, `Boundary.knots` etc for use by `predict.bs()`.

`bs()` is based on the function `spline.des()` written by Douglas Bates. It generates a basis matrix for representing the family of piecewise polynomials with the specified interior knots and degree, evaluated at the values of `x`. A primary use is in modeling formulas to directly specify a piecewise polynomial term in a model.

Beware of making predictions with new `x` values when `df` is used as an argument. Either use `safe.predict.gam()`, or else specify `knots` and `Boundary.knots`.

See Also

`ns`, `poly`, `smooth.spline`, `predict.bs`.

Examples

```
library(splines)
data(women)
bs(women$height, df = 5)
summary(fm1 <- lm(weight ~ bs(height, df = 5), data = women))
```

`interpSpline` *Create an Interpolation Spline* `interpSpline`

Description

Create an interpolation spline, either from `x` and `y` vectors, or from a formula/data.frame combination.

Usage

```
interpSpline(obj1, obj2, bSpline, period, na.action)
```

Arguments

`obj1`	Either a numeric vector of `x` values or a formula.
`obj2`	If `obj1` is numeric this should be a numeric vector of the same length. If `obj1` is a formula this can be an optional data frame in which to evaluate the names in the formula.
`bSpline`	If `TRUE` the b-spline representation is returned, otherwise the piecewise polynomial representation is returned. Defaults to `FALSE`.
`period`	An optional positive numeric value giving a period for a periodic interpolation spline.
`na.action`	a optional function which indicates what should happen when the data contain `NA`s. The default action (`na.omit`) is to omit any incomplete observations. The alternative action `na.fail` causes `interpSpline` to print an error message and terminate if there are any incomplete observations.

Value

An object that inherits from class `spline`. The object can be in the B-spline representation, in which case it will be of class `nbSpline` for natural B-spline, or in the piecewise polynomial representation, in which case it will be of class `npolySpline`.

Author(s)

Douglas Bates and Bill Venables

See Also

`splineKnots`, `splineOrder`, `periodicSpline`

Examples

```
library( splines )
data( women )
ispl <- interpSpline( women$height, women$weight )
ispl2 <- interpSpline( weight ~ height,  women )
# ispl and ispl2 should be the same
plot( predict( ispl, seq( 55, 75, len = 51 ) ), type = "l" )
points( women$height, women$weight )
plot( ispl )    # plots over the range of the knots
points( women$height, women$weight )
splineKnots( ispl )
```

ns *Generate a Basis Matrix for Natural Cubic Splines* ns

Description

Generate the B-spline basis matrix for a natural cubic spline.

Usage

```
ns(x, df, knots, intercept=F, Boundary.knots)
```

Arguments

`x`	the predictor variable.
`df`	degrees of freedom. One can supply `df` rather than knots; `ns()` then chooses `df - 1 - intercept` knots at suitably chosen quantiles of `x`.
`knots`	breakpoints that define the spline. The default is no knots; together with the natural boundary conditions this results in a basis for linear regression on `x`. Typical values are the mean or median for one knot, quantiles for more knots. See also `Boundary.knots`.
`intercept`	if `TRUE`, an intercept is included in the basis; default is `FALSE`.
`Boundary.knots`	boundary points at which to impose the natural boundary conditions and anchor the B-spline basis (default the range of the data). If both `knots` and `Boundary.knots` are supplied, the basis parameters do not depend on `x`. Data can extend beyond `Boundary.knots`

Value

A matrix of dimension `length(x) * df` where either `df` was supplied or if `knots` were supplied, `df = length(knots) + 1 + intercept`. Attributes are returned that correspond to the arguments to `ns`, and explicitly give the `knots`, `Boundary.knots` etc for use by `predict.ns()`.

`ns()` is based on the function `spline.des()`. It generates a basis matrix for representing the family of piecewise-cubic splines with the specified sequence of interior knots, and the natural boundary conditions. These enforce the constraint that the function is linear beyond the boundary knots, which can either be supplied, else default to the extremes of the data. A primary use is in modeling formula to directly specify a natural spline term in a model.

Beware of making predictions with new `x` values when `df` is used as an argument. Either use `safe.predict.gam()`, or else specify `knots` and `Boundary.knots`.

See Also

`bs`, `poly`, `predict.ns`

Examples

```
library(splines)
data(women)
ns(women$height, df = 5)
summary(fm1 <- lm(weight ~ ns(height, df = 5), data = women))
```

periodicSpline *Create a Periodic Interpolation Spline* periodicSpline

Description

Create a periodic interpolation spline, either from `x` and `y` vectors, or from a formula/data.frame combination.

Usage

```
periodicSpline(obj1, obj2, knots, period, ord)
```

Arguments

`obj1`	Either a numeric vector of `x` values or a formula.
`obj2`	If `obj1` is numeric this should be a numeric vector of the same length. If `obj1` is a formula this can be an optional data frame in which to evaluate the names in the formula.
`knots`	An optional numeric vector of knot positions.
`period`	A positive numeric value giving the period for the periodic spline. Defaults to `2 * pi`.
`ord`	A positive integer giving the order of the spline. Defaults to 4. See `?splineOrder` for a definition of the order of a spline.

Value

An object that inherits from class `spline`. The object can be in the B-spline representation, in which case it will be a `pbSpline` object, or in the piecewise polynomial representation (a `ppolySpline` object).

Author(s)

Douglas Bates and Bill Venables

See Also

splineKnots, interpSpline

Examples

```
library( splines )
xx <- seq( -pi, pi, len = 16 )[-1]
yy <- sin( xx )
frm <- data.frame( xx, yy )
print( pispl <- periodicSpline( xx, yy, period = 2 * pi ) )
print( pispl2 <- periodicSpline( yy ~ xx, frm, period = 2 * pi ) )
# pispl and pispl2 should be the same
plot( predict( pispl, seq(-3*pi, 3*pi, len = 101) ), type = "l" )
plot( pispl )          # displays over one period
```

`polySpline` *Piecewise Polynomial Spline Representation* `polySpline`

Description

Create the piecewise polynomial representation of a spline object.

Usage

```
polySpline(object, ...)
as.polySpline(object, ...)
```

Arguments

`object`	An object that inherits from class `spline`.
`...`	Optional additional arguments. At present no additional arguments are used.

Value

An object that inherits from class `polySpline`. This is the piecewise polynomial representation of a univariate spline function. It is defined by a set of distinct numeric values called knots. The spline function is a polynomial function between each successive pair of knots. At each interior knot the polynomial segments on each side are constrained to have the same value of the function and some of its derivatives.

Author(s)

Douglas Bates and Bill Venables

See Also

`interpSpline`, `periodicSpline`, `splineKnots`, `splineOrder`

Examples

```
library( splines )
data( women )
ispl <- polySpline( interpSpline( weight ~ height,  women ) )
print( ispl )   # print the piecewise polynomial representation
plot( ispl )    # plots over the range of the knots
points( women$weight, women$height )
```

predict.bs *Evaluate a Spline Basis* predict.bs

Description

Evaluate a predefined spline basis at given values.

Usage

```
predict.bs(object, newx, ...)
predict.ns(object, newx, ...)
```

Arguments

object	the result of a call to bs() or ns() having attributes describing knots, degree, etc.
newx	the x values at which evaluations are required.
...	Optional additional arguments. Presently no additional arguments are used.

Value

An object just like basis, except evaluated at the new values of x.

These are methods for the generic function predict() for objects inheriting from classes bs or ns. See predict for the general behavior of this function.

See Also

bs, ns, poly, lo, s

Examples

```
library(splines)
data(women)
basis <- ns(women$height, df = 5)
newX <- seq(58, 72, len = 51)
# evaluate the basis at the new data
predict(basis, newX)
```

predict.bSpline *Evaluate a spline at new values of x* predict.bSpline

Description

The predict methods for the classes that inherit from the virtual classes bSpline and polySpline are used to evaluate the spline or its derivatives. The plot method for a spline object first evaluates predict with the x argument missing, then plots the resulting xyVector with type = "l".

Usage

```
predict.bSpline(object, x, nseg=50, deriv=0)
predict.nbSpline(object, x, nseg=50, deriv=0)
predict.pbSpline(object, x, nseg=50, deriv=0)
predict.npolySpline(object, x, nseg=50, deriv=0)
predict.ppolySpline(object, x, nseg=50, deriv=0)
plot.spline(x, ...)
```

Arguments

`object`	An object that inherits from the `bSpline` or the `polySpline` class. For `plot.spline` this argument is called `x`.
`x`	A numeric vector of `x` values at which to evaluate the spline. If this argument is missing a suitable set of `x` values is generated as a sequence of `nseq` segments spanning the range of the knots. For `plot.spline` the `x` argument is as described under `object` above.
`nseg`	A positive integer giving the number of segments in a set of equally-spaced `x` values spanning the range of the knots in `object`. This value is only used if `x` is missing.
`deriv`	An integer between 0 and `splineOrder(object) - 1` specifying the derivative to evaluate.
`...`	additional graphical parameters (see `link{par}`).

Value

an `xyVector` with components

`x`	the supplied or inferred numeric vector of `x` values
`y`	the value of the spline (or its `deriv`'th derivative) at the `x` vector

Author(s)

Douglas Bates and Bill Venables

See Also

`xyVector`, `interpSpline`, `periodicSpline`

Examples

```
library( splines )
data( women )
ispl <- interpSpline( weight ~ height,  women )
opar <- par(mfrow = c(2, 2), las = 1)
plot(predict(ispl, nseg = 201),     # plots over the range of the knots
     main = "Original data with interpolating spline", type = "l",
     xlab = "height", ylab = "weight")
points(women$height, women$weight, col = 4)
plot(predict(ispl, nseg = 201, deriv = 1),
     main = "First derivative of interpolating spline", type = "l",
     xlab = "height", ylab = "weight")
plot(predict(ispl, nseg = 201, deriv = 2),
     main = "Second derivative of interpolating spline", type = "l",
```

```
     xlab = "height", ylab = "weight")
plot(predict(ispl, nseg = 401, deriv = 3),
     main = "Third derivative of interpolating spline", type = "l",
     xlab = "height", ylab = "weight")
par(opar)
```

`splineDesign` *Design Matrix for B-splines* `splineDesign`

Description

Evaluate the design matrix for the B-splines defined by `knots` at the values in `x`.

Usage

```
splineDesign(knots, x, ord, derivs)
spline.des(knots, x, ord, derivs)
```

Arguments

`knots`	a numeric vector of knot positions with non-decreasing values.
`x`	a numeric vector of values at which to evaluate the B-spline functions or derivatives. The values in `x` must be between `knots[ord]` and `knots[ length(knots) + 1 - ord ]`.
`ord`	a positive integer giving the order of the spline function. This is the number of coefficients in each piecewise polynomial segment, thus a cubic spline has order 4. Defaults to 4.
`derivs`	an integer vector of the same length as `x` and with values between `0` and `ord - 1`. The derivative of the given order is evaluated at the `x` positions. Defaults to a vector of zeroes of the same length as `x`.

Value

A matrix with `length( x )` rows and `length( knots ) - ord` columns. The i'th row of the matrix contains the coefficients of the B-splines (or the indicated derivative of the B-splines) defined by the `knot` vector and evaluated at the i'th value of `x`. Each B-spline is defined by a set of `ord` successive knots so the total number of B-splines is `length(knots)-ord`.

Note

The older `spline.des` function takes the same arguments but returns a list with several components including `knots`, `ord`, `derivs`, and `design`. The `design` component is the same as the value of the `splineDesign` function.

Author(s)

Douglas Bates and Bill Venables

Examples

```
library( splines )
splineDesign(knots = 1:10, x = 4:7)
```

splineKnots *Knot Vector from a Spline* splineKnots

Description

Return the knot vector corresponding to a spline object.

Usage

```
splineKnots(object)
```

Arguments

object	an object that inherits from class spline.

Value

A non-decreasing numeric vector of knot positions.

Author(s)

Douglas Bates and Bill Venables

Examples

```
library( splines )
data( women )
ispl <- interpSpline( weight ~ height, women )
splineKnots( ispl )
```

splineOrder *Determine the Order of a Spline* splineOrder

Description

Return the order of a spline object.

Usage

```
splineOrder(object)
```

Arguments

object	An object that inherits from class spline.

Details

The order of a spline is the number of coefficients in each piece of the piecewise polynomial representation. Thus a cubic spline has order 4.

Value

An integer vector of length 1.

Author(s)

Douglas Bates and Bill Venables

See Also

`splineKnots`, `interpSpline`, `periodicSpline`

Examples

```
library( splines )
data( women )
splineOrder( interpSpline( weight ~ height, women ) )
```

`xyVector` *Construct an xyVector Object* `xyVector`

Description

Create an object to represent a set of x-y pairs. The resulting object can be treated as a matrix or as a data frame or as a vector. When treated as a vector it reduces to the `y` component only.

The result of functions such as `predict.spline` is returned as an `xyVector` object so the x-values used to generate the y-positions are retained, say for purposes of generating plots.

Usage

```
xyVector(x, y)
```

Arguments

`x`	a numeric vector
`y`	a numeric vector of the same length as `x`

Value

An object of class `xyVector` with components

`x`	a numeric vector
`y`	a numeric vector of the same length as `x`

Author(s)

Douglas Bates and Bill Venables

Examples

```
library( splines )
data( women )
ispl <- interpSpline( weight ~ height, women )
weights <- predict( ispl, seq( 55, 75, len = 51 ))
class( weights )
plot( weights, type = "l", xlab = "height", ylab = "weight" )
points( women$height, women$weight )
weights
```

Chapter 9

The stepfun package

---

ecdf *Empirical Cumulative Distribution Function* ecdf

Description

Compute an empirical cumulative distribution function.

Usage

```
ecdf(x)
plot(ecdf(x), verticals = FALSE, col.01line = "gray70", ...)
```

Arguments

x	numeric vector with the "observations".

Details

The e.c.d.f. (empirical cumulative distribution function) F_n is a step function with jump $1/n$ at each observation (possibly with multiple jumps at one place if there are ties).

For observations x= $(x_1, x_2, \ldots x_n)$, F_n is the fraction of observations less or equal to t, i.e.,

$$F_n(t) = \#\{x_i \le t\} \ /n = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{[x_i \le t]}.$$

The function plot.ecdf which implements the plot method for ecdf objects, is implemented via a call to plot.stepfun.

Value

A function of class "ecdf", inheriting from the "stepfun" class.

Author(s)

Martin Maechler, ⟨maechler@stat.math.ethz.ch⟩.

See Also

`stepfun`, the more general class of step functions, `approxfun` and `splinefun`.

Examples

```
##-- 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 where 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))

F10 <- ecdf(rnorm(10))
summary(F10)

plot(F10)
plot(F10, verticals= TRUE, do.p = F)

plot(Fn12)# , lwd=2) dis-regarded
xx <- unique(sort(c(seq(-3,2, length=201), knots(Fn12))))
lines(xx, Fn12(xx), col='blue')
abline(v=knots(Fn12),lty=2,col='gray70')

plot(xx, Fn12(xx), type='b', cex=.1)#- plot.default
plot(Fn12, col.h='red', add= TRUE)  #- plot method
abline(v=knots(Fn12),lty=2,col='gray70')
plot(Fn12, verticals=T, col.p='blue', col.h='red',col.v='bisque')
par(op)

##-- this works too (automatic call to  ecdf(.)):
plot.ecdf(rnorm(24))
```

`plot.stepfun` *Plot Step Functions* `plot.stepfun`

Description

Method of the generic `plot` for `stepfun` objects and utility for plotting piecewise constant functions.

Usage

```
plot.stepfun(Fn, xval, xlim, 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"), ...)
```

Arguments

`Fn`	an R object inheriting from `"stepfun"`.
`xval`	numeric vector of abscissa values at which to evaluate `Fn`. Defaults to `knots(Fn)` restricted to `xlim`.
`xlim`	numeric(2); range of `x` values to use.
`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)

Martin Maechler ⟨maechler@stat.math.ethz.ch⟩, 1990, 1993; ported to R, 1997.

See Also

`ecdf` for empirical distribution functions as special step functions, `approxfun` and `splinefun`.

Examples

```
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)

tt <- seq(0,3, by=0.1)
```

```
op <- par(mfrow=c(2,2))
plot(sfun0); plot(sfun0, xval=tt, add=T, col.h="bisque")
plot(sfun.2);plot(sfun.2,xval=tt, add=T, col.h="orange")
plot(sfun1); plot(sfun1, xval=tt, add=T, 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)
  plot(list(sfun0,sfun.2,sfun1)[[i]], add=TRUE, 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)

##-- this works too (automatic call to  ecdf(.)):
plot.stepfun(rt(50, df=3), col.vert = "gray20")
```

`stepfun` *Step Functions* `stepfun`

Description

Given the vectors $(x_1, \ldots, x_n)$ and $(y_0, y_1, \ldots, y_n)$ (one value more!), `stepfun(x,y,...)` returns an interpolating "step" function, say `fn`. I.e., $fn(t) = c_i$ (constant) for $t \in (x_i, x_{i+1})$ and $fn(x_i) = y_i$ for $i = 1, \ldots, n$.

The value of the constant c_i above depends on the "continuity" parameter `f`. For the default, `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.

Usage

```
fn <- stepfun(x, y, f=0)
is.stepfun(fn)
knots(fn)
plot(fn, ...)
print(fn, ...)
summary(fn)
```

Arguments

`x`	numeric vector giving the "knots" or jump locations of the step function.
`y`	numeric vector one longer than `x`, giving the heights of the function values *between* the x values.
`f`	a number between 0 and 1, indicating how interpolation outside the given x values should happen. See `approxfun`.
`fn`	an R object inheriting from `"stepfun"`.

Value

A function of class `"stepfun"`, say `fn`. There are methods available for summarizing (`"summary(.)"`), representing (`"print(.)"`) and plotting (`"plot(.)"`, see `plot.stepfun`) `"stepfun"` objects.

The `environment` of `fn` contains all the information needed; `"x","y"`: the original arguments; `"n"`: number of knots (x values); `"f"`: continuity parameter; `"yleft", "yright"` the function values *outside* the knots; `"method"` (always `== "constant"`; not used, from `approxfun(.)`). The knots are also available by `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 `plot.stepfun` for *plotting* step functions.

`approxfun` and `splinefun`.

Examples

```
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)
sfun0
summary(sfun0)
summary(sfun.2)

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))
```

Chapter 10

The ts package

---

acf — *Autocovariance and Autocorrelation Function Estimation* — acf

---

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.

The generic function plot has a method for acf objects.

Usage

```
acf(x, lag.max = NULL,
    type = c("correlation", "covariance", "partial"),
    plot = TRUE, na.action, demean = TRUE, ...)
pacf(x, lag.max = NULL, plot = TRUE, na.action, ...)
ccf(x, y, lag.max = NULL, type = c("correlation", "covariance"),
    plot = TRUE,na.action, ...)

plot.acf(acf.obj, ci=0.95, ci.col="blue", ci.type=c("white", "ma"), ...)
```

Arguments

x, y	a univariate or multivariate (not ccf) time series object or a numeric vector or matrix.
lag.max	maximum lag at which to calculate the acf. Default is $10\log_{10}(N)$ where N is the number of observations.
plot	logical. If TRUE the acf is plotted.
type	character string giving the type of acf to be computed. Allowed values are "correlation" (the default), "covariance" or "partial".
na.action	function to be called to handle missing values.
demean	logical. Should the covariances be about the sample means?
acf.obj	an object of class acf.

ci	coverage probability for confidence interval. Plotting of the confidence interval is suppressed if ci is zero or negative.
ci.col	colour to plot the confidence interval lines.
ci.type	should the confidence limits assume a white noise input or for lag k an MA(k-1) input?
...	graphical parameters.

Details

For type = "correlation" and "covariance", the estimates are based on the sample covariance.

The partial correlation coefficient is estimated by fitting autoregressive models of successively higher orders up to lag.max.

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.

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.

Author(s)

Original: Paul Gilbert, Martyn Plummer. Extensive modifications and univariate case of pacf by B.D. Ripley.

Examples

```
## Examples from Venables & Ripley
data(lh)
acf(lh)
acf(lh, type="covariance")
pacf(lh)

data(UKLungDeaths)
acf(ldeaths)
acf(ldeaths, ci.type="ma")
acf(ts.union(mdeaths, fdeaths))
ccf(mdeaths, fdeaths) # just the cross-correlations.
```

`ar` *Fit Autoregressive Models to Time Series* `ar`

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, aic = TRUE, order.max = NULL, na.action, demean = TRUE, series,
        var.method = 1)
ar.yw(x, aic = TRUE, order.max = NULL, na.action, demean = TRUE, series)
ar.mle(x, aic = TRUE, order.max = NULL, na.action, demean = TRUE, series)

predict(ar.obj, 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.
`ar.obj`	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.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.
asy.var.coef	(univariate case.) 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

Brockwell, P. J. and Davis, R. A. (1991) *Time Series and Forecasting Methods.* Second edition. Springer, New York. Section 11.4.

Brockwell, P. J. and Davis, R. A. (1996) *Introduction to Time Series and Forecasting.* Springer, New York. Sections 5.1 and 7.6.

Percival, D. P. and Walden, A. T. (1998) *Spectral Analysis for Physical Applications.* Cambridge University Press.

Whittle, P. (1963) On the fitting of multivariate autoregressions and the approximate canonical factorization of a spectral density matrix. *Biometrika* **40**, 129–134.

See Also

`ar.ols`, `arima0` for ARMA models.

Examples

```
data(lh)
ar(lh)
ar(lh, method="burg")
ar(lh, method="ols")
ar(lh, F, 4) # fit ar(4)

data(LakeHuron)
ar(LakeHuron)
ar(LakeHuron, method="burg")
ar(LakeHuron, method="ols")

data(sunspot)
sunspot.ar <- ar(sunspot.year)
sunspot.ar
ar(x = sunspot.year, method = "burg")
ar(x = sunspot.year, method = "ols")
## next is slow and may have convergence problems,
## as it cares about invertibility
ar(x = sunspot.year, method = "mle")

predict(sunspot.ar, n.ahead=25)

data(BJsales)
ar(ts.union(BJsales, BJsales.lead))
## 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* ar.ols

Description

Fit an autoregressive time series model to the data by ordinary least squares, by default selecting the complexity by AIC.

Usage

```
ar.ols(x, aic = TRUE, order.max = NULL, na.action, 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)).

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

$$x_t - \mu = a_0 + a_1(x_{t-1} - \mu) + \cdots + a_p(x_{t-p} - \mu) + e_t$$

where a[0] is zero unless intercept is true, and m 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 is the series are roughly centred on zero. Otherwise the computations may be inaccurate or fail entirely.

Value

`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`	`"Unconstrained LS"`.
`series`	The name(s) of the time series.
`asy.se.coef`	The asymptotic-theory standard errors of the coefficient estimates.

Author(s)

Adrian Trapletti, Brian Ripley.

References

Luetkepohl, H. (1991): *Introduction to Multiple Time Series Analysis.* Springer Verlag, NY, pp. 368–370.

See Also

ar

Examples

```
data(lh)
ar(lh, method="burg")
ar.ols(lh)
ar.ols(lh, F, 4) # fit ar(4)

data(BJsales)
ar.ols(ts.union(BJsales, BJsales.lead))

data(EuStockMarkets)
x <- diff(log(EuStockMarkets))
ar.ols(x, order.max=6, demean=FALSE, intercept=TRUE)
```

arima0	*ARIMA Modelling of Time Series – Preliminary Version*	arima0

Description

Fit an ARIMA model to a univariate time series by exact maximum likelihood, 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, na.action = na.fail,
       delta = 0.01, transform.pars = 2)

predict(arima0.obj, n.ahead = 1, newxreg, se.fit = TRUE)

arima0.diag(fit, gof.lag = 10)
```

Arguments

`x`	a univariate time series
`order`	A specification of the non-seasonal part of the ARIMA model: the three components are (p, d, q), 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)`).
`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).
`na.action`	Function to be applied to remove missing values.
`delta`	A value to indicate at which point 'fast recursions' should be used. See the Details section.
`transform.pars`	If greater than 0, the ARMA parameters are transformed to ensure that they remain in the region of invertibility. If equal to 2, the optimization is rerun on the original scale to find the Hessian.
`arima0.obj, fit`	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?
`gof.lag`	Number of lags to be used in goodness-of-fit test.

Details

Different definitions of ARIMA models have different signs for the AR and/or MA coefficients. The definition here has

$$X_t = a_1 X_{t-1} + \cdots + a_p X_{t-p} + e_t + b_1 e_{t-1} + \ldots + b_q e_{t-q}$$

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.

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 using the Fortran code of Gardener *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.

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 (currently) done by `nlm`. It will work best if the columns in `xreg` are roughly scaled to zero mean and unit variance.

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`. If `transform.pars = 1`, only the portion corresponding to the untransformed parameters is returned.
`loglik`	the maximized log-likelihood (of the differenced data).
`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.
`resid`	the residuals.
`call`	the matched call.
`series`	the name of the series `x`.
`convergence`	the value returned by `optim`.

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.

Note

This is a preliminary version, and will be replaced in due course.

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.

Author(s)

B.D. Ripley

References

Brockwell, P. J. and Davis, R. A. (1996) *Introduction to Time Series and Forecasting.* Springer, New York. Sections 3.3 and 8.3.

Gardener, G, Harvey, A. C. and Phillips, G. D. A. (1980) Algorithm AS154. An algorithm for exact maximum likelihood estimation of autoregressive-moving average models by means of Kalman filtering. *Applied Statistics* **29**, 311–322.

Harvey, A. C. (1993) *Time Series Models*, 2nd Edition, Harvester Wheatsheaf, section 4.4.

Harvey, A. C. and McKenzie, C. R. (1982) Algorithm AS182. An algorithm for finite sample prediction from ARIMA processes. *Applied Statistics* **31**, 180–187.

See Also

`ar`

Examples

```
data(lh)
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)

data(USAccDeaths)
fit <- arima0(USAccDeaths, order=c(0,1,1), seasonal=list(order=c(0,1,1)))
fit
predict(fit, n.ahead=6)

data(LakeHuron)
arima0(LakeHuron, order=c(2,0,0), xreg=1:98)
```

`austres` *Quarterly Time Series of the Number of Australian Residents* `austres`

Description

Numbers (in thousands) of Australian residents measured quarterly from March 1971 to March 1994. The object is of class `"ts"`.

Source

P. J. Brockwell and R. A. Davis (1996) *Introduction to Time Series and Forecasting.* Springer

---

`beavers` *Body Temperature Series of Two Beavers* `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

```
data(beavers)
```

Arguments

`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

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.

Note

The observation at 22:20 is missing in `beaver1`.

Source

P. S. Reynolds (1994) Time-series analyses of beaver body temperatures. Chapter 11 of Lange, N., Ryan, L., Billard, L., Brillinger, D., Conquest, L. and Greenhouse, J. eds (1994) *Case Studies in Biometry.* New York: John Wiley and Sons.

`BJsales` *Sales Data with Leading Indicator.* `BJsales`

Description

The sales time series `BJsales` and leading indicator `BJsales.lead` each contain 150 observations. The objects are of class `"ts"`.

Usage

```
data(BJsales)
```

Source

The data are given in Box & Jenkins (1976). Obtained from the Time Series Data Library at http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/

References

G. E. P. Box and G. M. Jenkins (1976): *Time Series Analysis, Forecasting and Control*, Holden-Day, San Francisco, p. 537.

P. J. Brockwell and R. A. Davis (1991): *Time Series: Theory and Methods*, Second edition, Springer Verlag, NY, pp. 414.

`Box.test` *Box–Pierce and Ljung–Box Tests* `Box.test`

Description

Compute the Box–Pierce or Ljung–Box test statistic for examining the null hypothesis of independence in the time series `x` is computed.

Usage

```
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-square distribution of the test statistic.
`p.value`	the p-value of the test.
`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

Box, G. E. P. and Pierce, D. A. (1970) Distribution of residual correlations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association* **65**, 1509–1526.

Ljung, G. M. and Box, G. E. P. (1978) On a measure of lack of fit in time series models. *Biometrika* **65**, 553–564.

Harvey, A. C. (1993) *Time Series Models*, 2nd Edition, Harvester Wheatsheaf, NY, pp. 44, 45.

Examples

```
x <- rnorm (100)
Box.test (x, lag = 1)
Box.test (x, lag = 1, type="Ljung")
```

`cpgram` *Plot Cumulative Periodogram* `cpgram`

Description

Plots a cumulative periodogram

Usage

```
cpgram(ts, taper=0.1, main=
         paste("Series: ", deparse(substitute(ts))))
```

Arguments

`ts`	a univariate time series
`taper`	proportion tapered in forming the periodogram
`main`	main title

Value

None.

Side Effects

Plots the cumulative periodogram in a square plot.

Note

From package 'MASS'.

Author(s)

B.D. Ripley

Examples

```
par(pty = "s", mfrow = c(1,2))
data(lh)
cpgram(lh)
lh.ar <- ar(lh, order.max = 9)
cpgram(lh.ar$resid, main = "AR(3) fit to lh")

data(UKLungDeaths)
cpgram(ldeaths)
```

---

diff.ts — *diff Method for ts Objects* — diff.ts

Description

`diff` method for `ts` objects.

Usage

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

See Also

`diff`

---

diffinv — *Discrete Integrals: Inverse of Differencing* — diffinv

Description

Computes the inverse function of the lagged differences function `diff`.

Usage

```
diffinv(x, lag = 1, differences = 1,
        xi = rep(0.0, lag*differences*NCOL(x)))
```

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.

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 representing the discrete integral of `x`.

Author(s)

A. Trapletti

See Also

`diff`

Examples

```
s <- 1:10
d <- diff(s)
diffinv(d, xi = 1)
```

`embed` *Embedding a Time Series* `embed`

Description

Embeds the time series `x` into a low-dimensional Euclidean space.

Usage

```
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]`, ..., `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

```
x <- 1:10
embed (x, 3)
```

`EuStockMarkets` *Daily Closing Prices of Major European Stock Indices, 1991-1998.* `EuStockMarkets`

Description

Contains the daily closing prices of major European stock indices: Germany DAX (Ibis), Switzerland SMI, France CAC, and UK FTSE. The data are sampled in business time, i.e., weekends and holidays are omitted.

Usage

```
data(EuStockMarkets)
```

Format

A multivariate time series with 1860 observations on 4 variables. The object is of class `"mts"`.

Source

The data were kindly provided by Erste Bank AG, Vienna, Austria.

`filter` *Linear Filtering on a Time Series* `filter`

Description

Applies linear filtering to a univariate time series or to each series separately of a multivariate time series.

Usage

```
filter(x, filter, method="convolution", 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.

Author(s)

B.D. Ripley

See Also

`convolve`

Examples

```
x <- 1:100
filter(x, rep(1, 3))
filter(x, rep(1, 3), sides = 1)
filter(x, rep(1, 3), sides = 1, circular = TRUE)
data(presidents)
filter(presidents, rep(1,3))

## A simple simulation function for ARMA processes
arma.sim <- function(n, ar = NULL, ma = NULL, sigma = 1.0)
{
    x <- ts(rnorm(n+100, 0, sigma), start = -99)
    if(length(ma)) x <- filter(x, c(1, ma), sides=1)
    if(length(ar)) x <- filter(x, ar, method = "recursive")
    as.ts(x[-(1:100)])
}
arma.sim(63, c(0.8897,-0.4858), c(-0.2279, 0.2488), sigma=sqrt(0.1796))
```

`kernapply` *Apply Smoothing Kernel* `kernapply`

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.

Value

A smoothed version of the input sequence.

Author(s)

A. Trapletti

See Also

`kernel`, `convolve`, `filter`, `spectrum`

Examples

```
## see 'kernel' for examples
```

`kernel` *Smoothing Kernel Objects* `kernel`

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.

Usage

```
kernel(coef, m, r, name)

df.kernel(k)
bandwidth.kernel(k)
is.tskernel(k)

print(k, digits = max(3,getOption("digits")-3))
plot(k)
```

Arguments

`coef`	the upper half of the smoothing kernel coefficients (inclusive of coefficient zero) *or* the name of a kernel (currently `"daniell"`, `"dirichlet"`, `"fejer"` or `"modified.daniell"`.
`m`	the kernel dimension. The number of kernel coefficients is `2*m+1`.
`name`	the name of the kernel.
`r`	the kernel order for a Fejer kernel.
`digits`	the number of digits to format real numbers.

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).

`df.kernel` returns the "equivalent degrees of freedom" of a smoothing kernel as defined in Brockwell and Davies (1991), p. 362, and `bandwidth.kernel` returns the equivalent bandwidth as defined in Bloomfield (1991), p. 201, with a continuity correction.

Value

`kernel` returns a list with class `"tskernel"`, and components the coefficients `coef` and the kernel dimension `m`. An additional attribute is `"name"`.

Author(s)

A. Trapletti; modifications by B.D. Ripley

References

Bloomfield, P. (1976) *Fourier Analysis of Time Series: An Introduction.* Wiley.

Brockwell, P.J. and Davis, R.A. (1991) *Time Series: Theory and Methods.* Second edition. Springer, pp. 350–365.

See Also

`kernapply`

Examples

```
data(EuStockMarkets)    # Demonstrate a simple trading strategy for the
x <- EuStockMarkets[,1]  # financial time series German stock index DAX.
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

data(sunspot)     # Reproduce example 10.4.3 from Brockwell and Davies (1991)
spectrum(sunspot.year, kernel=kernel("daniell", c(11,7,3)), log="no")
```

`lag` *Lag a Time Series* `lag`

Description

Computed a lagged version of a time series, shifting the time base back by `k` observations.

Usage

```
lag(x, ...)
lag.default(x, k=1)
```

Arguments

`x`	A vector or matrix or univariate or multivariate time series
`k`	The number of lags (in units of observations).
`...`	Arguments for future methods.

Details

Vector or matrix arguments `x` are coerced to time series.

Value

A time series object.

Note

Note the sign of `k`: a series lagged by a positive `k` starts *earlier*.

Author(s)

B.D. Ripley

See Also

`diff`, `deltat`

Examples

```
data(UKLungDeaths)
lag(ldeaths, 12) # starts one year earlier
```

LakeHuron	*Level of Lake Huron 1875–1972*	LakeHuron

Description

Annual measurements of the level, in feet, of Lake Huron 1875–1972.

Usage

```
data(LakeHuron)
```

Format

A time series of length 98.

Source

Brockwell, P. J. & Davis, R. A. (1991). *Time Series and Forecasting Methods*. Second edition. Springer, New York. Series A, page 555.

Brockwell, P. J. & Davis, R. A. (1996). *Introduction to Time Series and Forecasting*. Springer, New York. Sections 5.1 and 7.6.

lh	*Luteinizing Hormone in Blood Samples*	lh

Description

A regular time series giving the luteinizing hormone in blood samples at 10 mins intervals from a human female, 48 samples.

Source

P.J. Diggle (1990) *Time Series: A Biostatistical Introduction.* Oxford, table A.1, series 3

lynx	*Annual Canadian Lynx trappings 1821–1934*	lynx

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).

Source

Brockwell, P. J. and Davis, R. A. (1991) *Time Series and Forecasting Methods.* Second edition. Springer. Series G (page 557).

References

Campbell, M. J.and A. M. Walker (1977). A Survey of statistical work on the Mackenzie River series of annual Canadian lynx trappings for the years 1821–1934 and a new analysis. *Journal of the Royal Statistical Society series A*, **140**, 411–431.

`na.omit.ts` *NA Handling Routines for Time Series* `na.omit.ts`

Description

For `na.omit.ts`, initial and final segments with missing values in one or more of the series are omitted. 'Internal' missing values will lead to failure.

For `na.contiguous` the longest consecutive stretch of non-missing values is used. (In the event of a tie, the first such stretch.)

Usage

```
na.contiguous(frame)
na.omit.ts(frame)
```

Arguments

`frame`	a univariate or multivariate time series.

Value

A time series without missing values. The class of `frame` will be preserved.

Author(s)

B. D. Ripley

See Also

`na.omit`, `na.fail`

Examples

```
data(BJsales)
sales1 <- ts.union(BJsales, lead3 = lag(BJsales.lead, -3))
na.omit.ts(sales1)

data(presidents)
na.contiguous(presidents)
```

`nottem` *Average Monthly Temperatures at Nottingham, 1920–1939* `nottem`

Description

A time series object containing average air temperatures at Nottingham Castle in degrees Fahrenheit for 20 years.

Source

Anderson, O.D. (1976) *Time Series Analysis and Forecasting: The Box-Jenkins approach.* Butterworths. Series R.

`PP.test` *Phillips–Perron Unit Root Test* `PP.test`

Description

Computes the Phillips-Perron test for the null hypothesis that `x` has a unit root.

Usage

```
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, p. 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

A. Banerjee, J. J. Dolado, J. W. Galbraith, and D. F. Hendry (1993) *Cointegration, Error Correction, and the Econometric Analysis of Non-Stationary Data*, Oxford University Press, Oxford.

P. Perron (1988) Trends and random walks in macroeconomic time series. *Journal of Economic Dynamics and Control* **12**, 297–332.

Examples

```
x <- rnorm(1000)
PP.test(x)
y <- cumsum(x) # has unit root
PP.test(y)
```

`spec.ar` *Estimate Spectral Density of a Time Series from AR Fit* `spec.ar`

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

```
spec.ar(x, n.freq, order = NULL, plot = TRUE, na.action,
        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.

Author(s)

B.D. Ripley

References

Thompson, D.J. (1990) Time series analysis of Holocene climate data. *Phil. Trans. Roy. Soc. A* **330**, 601–616.

Venables, W.N. and Ripley, B.D. (1997) *Modern Applied Statistics with S-PLUS.* Second edition. Springer. (Especially p. 448.)

See Also

ar, spectrum.

Examples

```
data(lh)
spec.ar(lh)

data(UKLungDeaths)
spec.ar(ldeaths)
spec.ar(ldeaths, method="burg")
```

`spec.pgram` *Estimate Spectral Density of a Time Series from Smoothed Periodogram* `spec.pgram`

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

```
spec.pgram(x, spans = NULL, kernel, taper = 0.1,
           demean = FALSE, detrend =TRUE,
           pad = 0, fast = TRUE, plot = FALSE, na.action, ...)
```

Arguments

`x`	A 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 `"kernel"`.
`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.
`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.
`pad`	Proportion of data to pad. Zeros are added to the end of the series to increase its length by the proportion `pad`.
`plot`	Plot the periodogram?
`fast`	If `TRUE`, pad the series to a highly composite length.
`...`	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` with the following elements.

`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, p.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

Bloomfield, P. (1976) *Fourier Analysis of Time Series: An Introduction.* Wiley.

Brockwell, P.J. and Davis, R.A. (1991) *Time Series: Theory and Methods.* Second edition. Springer.

Venables, W.N. and Ripley, B.D. (1997) *Modern Applied Statistics with S-PLUS.* Second edition. Springer. (Especially pp. 437–442.)

See Also

`spectrum`, `spec.taper`, `plot.spec`, `fft`

Examples

```
## Examples from Venables & Ripley
data(UKLungDeaths)
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")

data(EuStockMarkets)
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")

data(BJsales)
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* `spec.taper`

Description

Apply a cosine-bell taper to a time series.

Usage

```
spec.taper(x, p=0.1)
```

Arguments

`x`	A univariate or multivariate time series
`p`	The total proportion to be tapered, either a scalar or a vector of the length of the number of series.

Details

The cosine-bell taper is applied to the first and last `p[i]/2` observations of time series `x[, i]`.

Value

A new time series object.

Note

From package 'MASS'.

Author(s)

Kurt Hornik, B.D. Ripley

See Also

spec.pgram, cpgram

---

spectrum *Spectral Density Estimation* spectrum

Description

The spectrum function estimates the spectral density of a time series. This is a wrapper function which calls the methods spec.pgram and spec.ar.

The generic function plot has a method for spec objects: 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

```
spectrum(x, method = c("pgram","ar"), plot = TRUE, ...)
plot.spec(spec.obj, add=FALSE, ci=0.95,
          log = c("yes", "dB", "no"), ci.col = "blue", ci.lty = 3,
          plot.type = c("marginal", "coherency", "phase"), ...)
```

Arguments

x	A univariate or multivariate time series.
method	String specifying the method used to estimate the spectral density. Allowed methods are "pgram" (the default) and "ar".
plot	logical. If TRUE then the spectral density is plotted.
...	Further arguments to specific spec methods or plot.spec.
spec.obj	An object of class spec.
add	logical. If TRUE then lines are added to the existing plot.
ci	Coverage probability for confidence interval. Plotting of the confidence bar 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".
ci.col, ci.lty	Colour for plotting confidence bar, colour and line type for confidence intervals for coherency and phase.
plot.type	For multivariate time series, the type of plot required. Only the first character is needed.
...	Further graphical parameters.

Details

The spectrum here is defined with scaling `1/frequency(x)`, following S-PLUS. This makes the spectral density a density over the range `(-frequency(x)/2, +frequency(x)/2]`, whereas a more common scaling is 2π and range $(-0.5, 0.5]$ (e.g. Bloomfield) or 1 and range $(-\pi, \pi]$.

If available, a confidence interval will be plotted by `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 elements:

`freq`	vector of frequencies at which the spectral density is estimated. (Possibly approximate Fourier frequencies.)
`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 $i + (j-1) * (j-2)/2$ 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 `spec` objects 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

Bloomfield, P. (1976) *Fourier Analysis of Time Series: An Introduction.* Wiley.

Brockwell, P. J. and Davis, R. A. (1991) *Time Series: Theory and Methods.* Second edition. Springer.

Venables, W. N. and Ripley, B. D. (1997) *Modern Applied Statistics with S-PLUS.* Second edition. Springer. (Especially pp. 437–442.)

See Also

`spec.pgram`

Examples

```
## Examples from Venables & Ripley
## spec.pgram
par(mfrow=c(2,2))
data(lh)
spectrum(lh)
spectrum(lh, spans=3)
spectrum(lh, spans=c(3,3))
spectrum(lh, spans=c(3,5))

data(UKLungDeaths)
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")
```

---

stl *Seasonal Decomposition of Time Series by Loess* stl

Description

Decompose a time series into seasonal, trend and irregular components.

Usage

```
stl(x, s.window = NULL, s.degree = 0, t.window = NULL, t.degree = 1,
    robust = FALSE, na.action = na.fail)
```

Arguments

`x`	A univariate time series to be decomposed. This should be an object of class `"ts"` with a frequency greater than one.
`s.window`	Either the 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. There is a reasonable default.
`t.degree`	Degree of locally-fitted polynomial in trend extraction. Should be zero or one.
`robust`	Should robust fitting be used in the `loess` procedure?
`na.action`	Action on missing values.

Details

The seasonal component is found by *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.

Value

An object of class `"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.

Note

This is similar to but not identical to the `stl` function in S-PLUS. The `remainder` component given by S-PLUS is the sum of the `trend` and `remainder` series from this function.

Author(s)

B.D. Ripley; Fortran code by Cleveland *et al.* (1990) from 'netlib'.

References

R. B. Cleveland, W. S. Cleveland, J.E. McRae, and I. Terpenning (1990). STL: A Seasonal-Trend Decomposition Procedure Based on Loess. *Journal of Official Statistics*, **6**, 3–73.

See Also

`loess` in package 'modreg' (which is not actually used in `stl`).

Examples

```
data(nottem)
plot(stl(nottem, "per"))
data(co2)
plot(stl(log(co2), s.window=21))
## linear trend, strict period.
plot(stl(log(co2), s.window="per", t.window=1000))
```

`sunspot` — *Yearly Sunspot Data, 1700–1988. Monthly Sunspot Data, 1749–1997.* — `sunspot`

Description

Monthly and yearly number of sunspots.

Usage

```
data(sunspot)
```

Format

The univariate time series `sunspot.year` and `sunspot.month` contain 289 and 2988 observations, respectively. The objects are of class `"ts"`.

Source

Monthly data: Sunspot Index Data Center, 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\protect\T1\textunderscoretxt.html

Yearly data: H. Tong (1996) *Non-Linear Time Series*. Clarendon Press, Oxford, p. 471.

See Also

`sunspot.month` is a longer version of `sunspots` in base R, that runs until 1988.

`toeplitz` *Form Symmetric Toeplitz Matrix* `toeplitz`

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)
```

`treering` *Yearly Treering Data, -6000–1979.* `treering`

Description

Contains normalized tree-ring widths in dimensionless units. Each tree ring corresponds to one year. Tree: Methuselah Walk, Pilo; Location: California, Gt Basin B C pine 2805M, 3726-11810; Author: Donald A. Graybill, 1980.

Usage

```
data(treering)
```

Format

A univariate time series with 7981 observations. The object is of class `"ts"`.

Source

Time Series Data Library: http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/

`ts-internal` *Internal ts functions* `ts-internal`

Description

Internal ts functions

Usage

```
Ops.ts(e1, e2)
arma0f(p)
is.mts(x)
```

Details

These are not to be called by the user.

ts.plot *Plot Multiple Time Series* ts.plot

Description

Plot several time series on a common plot. Unlike plot.ts the series can have a different time bases, but they should have the same frequency.

Usage

```
ts.plot(..., gpars = list())
```

Arguments

...	one or more univariate or multivariate time series
gpars	list of named graphics parameters to be passed to the plotting functions

Value

None.

Note

Although this can be used for a single time series, plot is easier to use and is preferred.

Author(s)

B.D. Ripley

See Also

plot.ts

Examples

```
data(UKLungDeaths)
ts.plot(ldeaths, mdeaths, fdeaths,
        gpars=list(xlab="year", ylab="deaths", lty=c(1:3)))

data(nottem)
nott <- window(nottem, end=c(1936,12))
fit <- arima0(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)))
```

ts.union *Bind Two or More Time Series* ts.union

Description

Bind time series which have a common frequency. `ts.union` pads with `NA`s to the total time coverage, `ts.intersect` restricts to the time covered by all the series.

Usage

```
ts.intersect(..., dframe = FALSE)
ts.union(..., dframe = FALSE)
cbind.ts(..., dframe = FALSE, union = TRUE)
```

Arguments

`...`	Two or more univariate or multivariate time series, or objects which can coerced to time series.
`dframe`	If `TRUE` return the result as a data frame.
`union`	Act as `ts.union` or `ts.intersect`.

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.

Author(s)

B. D. Ripley

Examples

```
data(UKLungDeaths)
ts.union(mdeaths, fdeaths)
cbind(mdeaths, fdeaths) # same as the previous line
ts.intersect(window(mdeaths, 1976), window(fdeaths, 1974, 1978))
data(BJsales)
sales1 <- ts.union(BJsales, lead = BJsales.lead)
ts.intersect(sales1, lead3 = lag(BJsales.lead, -3))
```

UKDriverDeaths	*Deaths of Car Drivers in Great Britain 1969–84*	UKDriverDeaths

Description

A regular 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.

Source

Harvey, A.C. (1989) *Forecasting, Structural Time Series Models and the Kalman Filter.* Cambridge University Press, pp. 519–523.

UKLungDeaths	*Monthly Deaths from Lung Diseases in the UK*	UKLungDeaths

Description

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

```
data(UKLungDeaths)
```

Source

P. J. Diggle (1990) *Time Series: A Biostatistical Introduction.* Oxford, table A.3

USAccDeaths	*Accidental Deaths in the US 1973–1978*	USAccDeaths

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.

Source

P. J. Brockwell and R. A. Davis (1991) *Time Series: Theory and Methods.* Springer, New York.

Index

! *(Logic)*, 267
!= *(Comparison)*, 71
∗Topic **NA**
  complete.cases, 72
  factor, 143
  NA, 310
  na.action, 311
  na.fail, 311
∗Topic **algebra**
  backsolve, 34
  chol, 62
  chol2inv, 63
  crossprod, 87
  eigen, 129
  matrix, 292
  qr, 398
  QR.Auxiliaries, 399
  solve, 444
  svd, 469
∗Topic **aplot**
  abline, 3
  arrows, 24
  axis, 33
  box, 42
  bxp, 49
  contour, 75
  coplot, 82
  filled.contour, 154
  frame, 167
  grid, 184
  image, 206
  legend, 244
  lines, 257
  matplot, 290
  mtext, 308
  persp, 350
  plot.window, 363
  plot.xy, 364
  plotmath, 365
  points, 372
  polygon, 375
  rect, 416
  rect.hclust, 602
  screen, 433
  segments, 437
  text, 486
  title, 489
∗Topic **arith**
  all.equal, 11
  approxfun, 19
  Arithmetic, 22
  cumsum, 87
  diff, 113
  diff.ts, 670
  Extremes, 142
  gl, 177
  matmult, 289
  ppoints, 380
  prod, 393
  range, 409
  Round, 425
  sign, 441
  sort, 445
  sum, 466
  tabulate, 481
∗Topic **array**
  aggregate, 6
  aperm, 16
  apply, 17
  array, 23
  backsolve, 34
  cbind, 56
  chol, 62
  chol2inv, 63
  col, 68
  contrast, 77
  cor, 84
  crossprod, 87
  data.matrix, 94
  diag, 113
  dim, 114
  dimnames, 115
  drop, 120
  eigen, 129
  Extract, 140
  kronecker, 239

lower.tri, 272
margin.table, 285
mat.or.vec, 285
matmult, 289
matplot, 290
matrix, 292
nrow, 321
outer, 333
prop.table, 396
qr, 398
row, 426
row/colnames, 426
scale, 430
svd, 469
sweep, 470
t, 479
∗Topic **attribute**
attr, 30
attributes, 31
call, 52
comment, 70
length, 246
mode, 301
name, 312
names, 312
NULL, 322
numeric, 322
structure, 461
typeof, 498
which, 519
∗Topic **category**
aggregate, 6
by, 50
codes, 66
cut, 89
factor, 143
ftable, 169
ftable.formula, 170
gl, 177
interaction, 213
levels, 247
levels.factor, 247
loglin, 269
nlevels, 316
split, 448
table, 480
tapply, 481
∗Topic **character**
abbreviate, 2
char.expand, 57
character, 58
charmatch, 58
format, 162
format.info, 163
formatC, 164
grep, 183
make.names, 282
nchar, 313
paste, 349
pmatch, 371
strsplit, 460
strwidth, 461
substr, 465
symnum, 473
∗Topic **classes**
character, 58
class, 64
codes, 66
data.class, 92
data.frame, 93
double, 118
integer, 213
is.object, 221
is.recursive, 222
is.single, 222
logical, 267
numeric, 322
real, 415
vector, 513
∗Topic **cluster**
as.hclust, 587
cutree, 592
dist, 593
hclust, 594
identify.hclust, 596
kmeans, 598
rect.hclust, 602
∗Topic **color**
colors, 69
gray, 182
hsv, 201
palette, 339
Palettes, 340
rgb, 423
∗Topic **complex**
complex, 72
∗Topic **datasets**
airmiles, 7
airquality, 8
anscombe, 13
attenu, 28
attitude, 29
austres, 666
beavers, 667

BJsales, 668
BOD, 604
cars, 53
ChickWeight, 604
chickwts, 60
CO2, 606
co2, 66
data, 91
discoveries, 115
DNase, 607
esoph, 131
euro, 133
eurodist, 134
EuStockMarkets, 672
faithful, 146
Formaldehyde, 161
freeny, 168
HairEyeColor, 185
Indometh, 609
infert, 208
InsectSprays, 210
iris, 216
islands, 223
LakeHuron, 677
lh, 677
LifeCycleSavings, 256
Loblolly, 610
longley, 272
lynx, 677
mtcars, 308
nhtemp, 316
nottem, 678
Orange, 618
OrchardSprays, 331
phones, 352
PlantGrowth, 355
precip, 380
presidents, 385
pressure, 386
Puromycin, 625
quakes, 401
randu, 408
rivers, 423
rock, 581
sleep, 443
stackloss, 449
state, 454
sunspot, 687
sunspots, 469
swiss, 471
Titanic, 488
ToothGrowth, 490
treering, 689
trees, 493
UCBAdmissions, 498
UKDriverDeaths, 692
UKLungDeaths, 692
USAccDeaths, 692
USArrests, 508
USJudgeRatings, 509
USPersonalExpenditure, 510
uspop, 510
VADeaths, 511
volcano, 514
warpbreaks, 516
women, 522
∗Topic **data**
apropos, 20
assign, 26
attach, 27
autoload, 31
clearNames, 605
delay, 99
deparse, 104
detach, 107
environment, 130
eval, 134
exists, 137
get, 175
getInitial, 608
library, 248
library.dynam, 250
NLSstAsymptotic, 615
NLSstClosestX, 616
NLSstLfAsymptote, 617
NLSstRtAsymptote, 617
search, 437
setNames, 629
sortedXyData, 630
substitute, 463
sys.parent, 475
∗Topic **density**
sunflowerplot, 467
∗Topic **design**
contrast, 77
contrasts, 78
∗Topic **device**
dev.xxx, 107
dev2, 108
Devices, 112
Gnome, 181
pictex, 353
postscript, 377
screen, 433

x11, 524
∗Topic **distribution**
Beta, 39
Binomial, 40
Cauchy, 55
chisq.test, 533
Chisquare, 61
density, 101
Exponential, 138
FDist, 148
fivenum, 156
GammaDist, 172
Geometric, 174
hist, 199
Hypergeometric, 203
IQR, 215
Logistic, 268
Lognormal, 271
NegBinomial, 314
Normal, 319
Poisson, 373
ppoints, 380
qqnorm, 397
Random, 404
sample, 429
SignRank, 442
stem, 455
TDist, 483
Tukey, 496
Uniform, 500
Weibull, 516
Wilcoxon, 520
∗Topic **documentation**
apropos, 20
args, 21
data, 91
Defunct, 98
demo, 101
Deprecated, 105
example, 136
help, 186
help.search, 188
help.start, 189
NotYet, 321
prompt, 395
str, 458
undoc, 499
∗Topic **dplot**
approxfun, 19
as.char.or.expr, 25
boxplot.stats, 45
colors, 69
convolve, 81
density, 101
expression, 139
fft, 150
hsv, 201
jitter, 237
layout, 242
Palettes, 340
panel.smooth, 341
par, 342
ppoints, 380
pretty, 386
screen, 433
splinefun, 447
strwidth, 461
units, 502
xy.coords, 525
∗Topic **environment**
apropos, 20
browser, 46
commandArgs, 69
debug, 96
gc, 173
gctorture, 174
getenv, 176
interactive, 214
is.R, 221
layout, 242
ls, 274
machine, 279
Memory, 294
options, 328
par, 342
quit, 402
R.Version, 403
remove, 418
Startup, 452
stop, 457
∗Topic **error**
bug.report, 46
warning, 514
warnings, 515
∗Topic **file**
cat, 54
count.fields, 85
dataentry, 95
dput, 119
dump, 122
file, 151
file.choose, 152
file.path, 153
file.show, 153

httpclient, 202
list.files, 259
load, 264
parse, 348
Platform, 356
read.fwf, 410
read.table, 412
read.table.url, 414
save, 430
scan, 431
sink, 443
source, 445
sys.source, 476
system, 477
system.file, 478
tempfile, 484
unlink, 503
write, 523
write.table, 524
zip.file.extract, 526

∗Topic **hplot**
barplot, 35
biplot, 588
biplot.princomp, 589
boxplot, 43
chull, 63
contour, 75
coplot, 82
cpgram, 669
curve, 88
dotplot, 117
ecdf, 651
filled.contour, 154
hist, 199
image, 206
matplot, 290
mosaicplot, 306
pairs, 336
pairs.formula, 338
panel.smooth, 341
persp, 350
piechart, 354
plot, 357
plot.default, 358
plot.factor, 360
plot.formula, 360
plot.lm, 361
plot.ppr, 575
plot.stepfun, 652
qqnorm, 397
rug, 428
stars, 450
stripplot, 459
sunflowerplot, 467

∗Topic **htest**
ansari.test, 529
bartlett.test, 530
binom.test, 531
chisq.test, 533
cor.test, 534
fisher.test, 536
fligner.test, 538
friedman.test, 539
kruskal.test, 541
ks.test, 542
mantelhaen.test, 543
mcnemar.test, 544
mood.test, 545
p.adjust, 334
pairwise.prop.test, 547
pairwise.t.test, 547
pairwise.table, 548
pairwise.wilcox.test, 549
power.prop.test, 549
power.t.test, 550
print.pairwise.htest, 551
print.power.htest, 552
prop.test, 553
shapiro.test, 554
t.test, 555
var.test, 557
wilcox.test, 558

∗Topic **interface**
dyn.load, 123
Internal, 214
Primitive, 388
system, 477

∗Topic **iplot**
dev.xxx, 107
ecdf, 651
frame, 167
hist, 199
identify, 204
identify.hclust, 596
layout, 242
locator, 265
par, 342
stepfun, 654

∗Topic **iteration**
apply, 17
by, 50
Control, 80
lapply, 241
merge, 296

sweep, 470
tapply, 481
∗Topic **list**
clearNames, 605
Extract, 140
lapply, 241
list, 257
NULL, 322
setNames, 629
unlist, 504
∗Topic **logic**
all, 10
all.equal, 11
any, 14
Comparison, 71
complete.cases, 72
Control, 80
duplicated, 122
ifelse, 205
Logic, 267
logical, 267
match, 286
NA, 310
unique, 501
which, 519
∗Topic **manip**
append, 17
c, 52
cbind, 56
deparse, 104
dimnames, 115
duplicated, 122
list, 257
match, 286
model.extract, 302
NA, 310
NLSstAsymptotic, 615
NLSstClosestX, 616
NLSstLfAsymptote, 617
NLSstRtAsymptote, 617
NULL, 322
order, 332
rep, 419
replace, 419
rev, 422
rle, 424
row/colnames, 426
rowsum, 427
seq, 438
sequence, 439
sort, 445
structure, 461
subset, 462
transform, 492
unique, 501
unlist, 504
∗Topic **math**
abs, 4
Bessel, 37
convolve, 81
deriv, 105
fft, 150
Hyperbolic, 202
is.finite, 217
kappa, 238
log, 266
Machine, 277
nextn, 315
poly, 374
polyroot, 376
Special, 446
splinefun, 447
Trig, 493
∗Topic **methods**
class, 64
data.class, 92
data.frame, 93
is.object, 221
Methods, 297
methods, 299
na.action, 311
noquote, 318
predict, 381
summary, 466
∗Topic **misc**
close.socket, 65
contributors, 79
copyright, 84
license, 251
make.socket, 283
read.socket, 411
read.table.url, 414
sets, 440
∗Topic **models**
add1, 4
alias, 9
anova, 12
aov, 15
asOneSidedFormula, 603
asVector, 639
backSpline, 640
C, 51
case/variable.names, 54
coefficients, 67

deviance, 111
df.residual, 112
dummy.coef, 120
eff.aovlist, 127
effects, 128
expand.grid, 138
extractAIC, 141
factor.scope, 145
family, 147
fitted.values, 156
formula, 166
formula.nls, 608
glm, 178
glm.summaries, 180
interpSpline, 642
is.empty.model, 217
labels, 240
logLik, 611
loglin, 269
lqs, 567
make.link, 282
make.tables, 284
model.extract, 302
model.frame, 303
model.matrix, 304
model.tables, 305
nls, 611
nls.control, 613
nlsModel, 614
offset, 323
periodicSpline, 644
plot.profile.nls, 619
polySpline, 645
power, 379
predict.bSpline, 646
predict.glm, 382
predict.lqs, 569
predict.nls, 620
preplot, 384
profile, 393
profile.nls, 621
profiler, 622
profiler.nls, 623
proj, 394
relevel, 417
replications, 420
residuals, 421
se.contrast, 436
selfStart, 626
selfStart.default, 627
selfStart.formula, 628
splineDesign, 648
splineKnots, 649
splineOrder, 649
SSasymp, 631
SSasympOff, 632
SSasympOrig, 633
SSbiexp, 634
SSfol, 635
SSfpl, 636
SSlogis, 637
SSmicmen, 638
stat.anova, 453
step, 455
terms, 485
update, 505
update.formula, 506
xyVector, 650

∗Topic **multivariate**
as.hclust, 587
biplot, 588
biplot.princomp, 589
cancor, 590
cmdscale, 591
cor, 84
cov.rob, 565
cov.wt, 86
cutree, 592
dist, 593
hclust, 594
kmeans, 598
mahalanobis, 281
prcomp, 599
princomp, 600
var, 512

∗Topic **nonlinear**
CO2, 606
deriv, 105
getInitial, 608
nlm, 317
nls, 611
nls.control, 613
nlsModel, 614
optim, 324
plot.profile.nls, 619
predict.nls, 620
profile.nls, 621
profiler, 622
profiler.nls, 623

∗Topic **nonparametric**
sunflowerplot, 467

∗Topic **optimize**
nlm, 317
optim, 324

optimize, 327
uniroot, 501
∗Topic **print**
cat, 54
format, 162
format.info, 163
formatC, 164
labels, 240
noquote, 318
options, 328
print, 388
print.coefmat, 389
print.default, 390
print.matrix, 391
str, 458
write.table, 524
∗Topic **programming**
.Alias, 1
all.names, 12
as.function, 25
autoload, 31
body, 41
browser, 46
call, 52
check.options, 59
commandArgs, 69
Control, 80
debug, 96
delay, 99
delete.response, 100
deparse, 104
do.call, 116
dput, 119
environment, 130
eval, 134
expression, 139
Foreign, 158
formals, 162
format.info, 163
function, 171
ifelse, 205
interactive, 214
invisible, 215
is.finite, 217
is.function, 220
is.language, 220
is.recursive, 222
is.symbol, 223
Last.value, 241
Machine, 277
machine, 279
match.arg, 286
match.call, 287
match.fun, 288
menu, 296
missing, 300
model.extract, 302
name, 312
nargs, 313
on.exit, 323
Paren, 347
parse, 348
R.Version, 403
Recall, 416
restart, 421
source, 445
stop, 457
substitute, 463
switch, 472
sys.parent, 475
trace, 491
traceback, 491
warning, 514
warnings, 515
∗Topic **regression**
anova, 12
aov, 15
case/variable.names, 54
coefficients, 67
contrast, 77
contrasts, 78
df.residual, 112
effects, 128
fitted.values, 156
glm, 178
glm.summaries, 180
influence.measures, 209
line, 561
lm, 260
lm.influence, 261
lm.summaries, 263
ls.diag, 275
ls.print, 276
lsfit, 276
nls, 611
nls.control, 613
plot.lm, 361
plot.profile.nls, 619
ppr, 576
predict.glm, 382
predict.lm, 383
predict.nls, 620
profile.nls, 621
profiler.nls, 623

qr, 398
residuals, 421
stat.anova, 453
weighted.residuals, 518
∗Topic **robust**
cov.rob, 565
fivenum, 156
IQR, 215
line, 561
lqs, 567
mad, 280
median, 293
medpolish, 562
smooth, 563
∗Topic **smooth**
bs, 641
density, 101
ksmooth, 571
loess, 572
loess.control, 574
lowess, 273
ns, 643
predict.bs, 646
predict.loess, 579
predict.smooth.spline, 580
scatter.smooth, 582
smooth, 563
smooth.spline, 583
supsmu, 584
∗Topic **sysdata**
colors, 69
commandArgs, 69
Constants, 74
Machine, 277
NULL, 322
palette, 339
R.Version, 403
Random, 404
∗Topic **ts**
acf, 657
ar, 659
ar.ols, 662
arima0, 664
Box.test, 668
cpgram, 669
diff, 113
diff.ts, 670
diffinv, 670
embed, 671
filter, 672
kernapply, 674
kernel, 674
lag, 676
na.omit.ts, 678
PP.test, 679
spec.ar, 680
spec.pgram, 681
spec.taper, 683
spectrum, 684
start, 452
stl, 686
time, 487
toeplitz, 688
ts, 494
ts-internal, 689
ts.plot, 690
ts.union, 691
tsp, 496
window, 521
∗Topic **univar**
ave, 32
Extremes, 142
fivenum, 156
IQR, 215
mad, 280
mean, 293
median, 293
order, 332
quantile, 401
range, 409
rank, 410
sd, 435
sort, 445
stem, 455
var, 512
weighted.mean, 517
∗Topic **utilities**
all.equal, 11
as.char.or.expr, 25
BATCH, 37
bug.report, 46
builtins, 49
check.options, 59
COMPILE, 71
conflicts, 73
dataentry, 95
date, 95
debugger, 97
Defunct, 98
demo, 101
Deprecated, 105
dev2bitmap, 110
download.file, 118
edit, 125

edit.data.frame, 126
example, 136
fix, 157
getenv, 176
getwd, 177
grep, 183
index.search, 207
INSTALL, 212
is.R, 221
jitter, 237
memory.profile, 295
menu, 296
modreg-internal, 575
noquote, 318
NotYet, 321
package.contents, 335
package.dependencies, 335
page, 336
parse.dcf, 348
Platform, 356
pos.to.env, 377
proc.time, 392
R.home, 403
readline, 415
relevel, 417
REMOVE, 418
RHOME, 423
se.aov, 435
SHLIB, 441
str, 458
symnum, 473
sys.source, 476
system, 477
system.file, 478
system.time, 478
unname, 504
update.packages, 507
E *(Arithmetic)*, 22
E *(formula)*, 166
* *(Arithmetic)*, 22
+ *(Arithmetic)*, 22
- *(Arithmetic)*, 22
-> *(assign)*, 26
->> *(assign)*, 26
.Alias, 1, *67*, *239*, *403*, *479*
.AutoloadEnv *(autoload)*, 31
.Autoloaded *(library)*, 248
.C, *118*, *124*, *214*
.C *(Foreign)*, 158
.Call, *124*
.Call *(Foreign)*, 158
.Class *(Methods)*, 297
.Defunct *(Defunct)*, 98
.Deprecated *(Deprecated)*, 105
.Device *(dev.xxx)*, 107
.Devices *(dev.xxx)*, 107
.Dyn.libs *(library.dynam)*, 250
.External, *124*
.External *(Foreign)*, 158
.First, *214*, *403*
.First *(Startup)*, 452
.First.lib, *124*, *250*
.First.lib *(library)*, 248
.Fortran, *118*, *124*, *214*
.Fortran *(Foreign)*, 158
.Generic *(Methods)*, 297
.GlobalEnv, *437*, *464*, *475*
.GlobalEnv *(environment)*, 130
.Group *(Methods)*, 297
.Internal, *49*, *298*, *388*
.Internal *(Internal)*, 214
.Last, *452*, *453*
.Last *(quit)*, 402
.Last.value *(Last.value)*, 241
.Library *(library)*, 248
.Machine *(Machine)*, 277
.Method *(Methods)*, 297
.NotYetImplemented *(NotYet)*, 321
.NotYetUsed *(NotYet)*, 321
.Options *(options)*, 328
.Pars *(par)*, 342
.Platform, *477*
.Platform *(Platform)*, 356
.PostScript.Options *(postscript)*, 377
.Primitive, *214*, *298*, *347*
.Primitive *(Primitive)*, 388
.Provided *(library)*, 248
.Random.seed, *320*, *501*
.Random.seed *(Random)*, 404
.Rprofile *(Startup)*, 452
.Traceback *(traceback)*, 491
.lib.loc, *250*, *478*
.lib.loc *(library)*, 248
.packages, *250*, *508*
.packages *(library)*, 248
.path.package *(library)*, 248
/ *(Arithmetic)*, 22
: *(seq)*, 438
< *(Comparison)*, 71
<-, *1*
<- *(assign)*, 26
<= *(Comparison)*, 71
<<- *(assign)*, 26
==, *11*

== *(Comparison)*, 71
> *(Comparison)*, 71
>= *(Comparison)*, 71
? *(help)*, 186
[, *120*, *298*, *463*
[*(Extract)*, 140
[.tskernel *(kernel)*, 674
[<- *(Extract)*, 140
[[, *298*
[[*(Extract)*, 140
[[<- *(Extract)*, 140
$ *(Extract)*, 140
$<- *(Extract)*, 140
%*%, *87*
%*% *(matmult)*, 289
%/% *(Arithmetic)*, 22
%% *(Arithmetic)*, 22
%in%, *440*
%in% *(match)*, 286
%o%, *87*
%o% *(outer)*, 333
%x% *(kronecker)*, 239
& *(Logic)*, 267
&& *(Logic)*, 267
| *(Logic)*, 267

abbreviate, 2
abline, 3, *184*, *185*, *375*
abs, 4, *441*
acf, 657
acos, *203*
acos *(Trig)*, 493
acosh *(Hyperbolic)*, 202
add.scope *(factor.scope)*, 145
add1, 4, *142*, *146*, *456*
aggregate, 6, *18*, *482*
agnes, *587*
airmiles, 7
airquality, 8
alias, 9, *16*
alist, *26*, *42*, *162*
alist *(list)*, 257
all, 10, *11*
all.equal, 11
all.names, 12
all.vars *(all.names)*, 12
anova, 12, *179*, *181*, *261*, *264*, *275*, *467*, *542*
anova.glm, *179*, *453*
anova.glm *(glm.summaries)*, 180
anova.glmlist *(glm.summaries)*, 180
anova.lm, *261*
anova.lm *(lm.summaries)*, 263
anova.loess *(loess)*, 572
anovalist.lm *(lm.summaries)*, 263
ansari.test, 529, *531*, *539*, *546*, *557*
anscombe, 13
any, 14
aov, *5*, *6*, 15, *78*, *79*, *121*, *127*, *128*, *141*, *146*, *261*, *306*, *329*, *394*, *395*, *455*, *563*
aperm, 16, *23*
append, 17
apply, *7*, 17, *241*, *288*, *471*, *482*
approx, *448*
approx *(approxfun)*, 19
approxfun, 19, *448*, *652–655*
apropos, 20, *184*, *189*, *274*
ar, 659, *663*, *666*, *680*, *681*
ar.burg *(ar)*, 659
ar.mle *(ar)*, 659
ar.ols, *660*, *661*, 662
ar.yw *(ar)*, 659
Arg *(complex)*, 72
args, 21, *42*, *162*, *172*, *313*, *458*, *459*
arima0, *661*, 664
Arithmetic, *4*, 22, *218*, *266*, *289*, *447*
arma0f *(ts-internal)*, 689
array, 23, *115*, *120*, *140*, *322*, 482, *519*
arrows, 24, *438*
as.array *(array)*, 23
as.call *(call)*, 52
as.char.or.expr, 25
as.character, *163*, *349*, *474*
as.character *(character)*, 58
as.complex *(complex)*, 72
as.data.frame *(data.frame)*, 93
as.data.frame.xyVector *(xyVector)*, 650
as.dist *(dist)*, 593
as.double *(double)*, 118
as.expression *(expression)*, 139
as.factor *(factor)*, 143
as.formula *(formula)*, 166
as.function, 25
as.hclust, 587
as.integer, *425*
as.integer *(integer)*, 213
as.list, *504*
as.list *(list)*, 257
as.logical *(logical)*, 267
as.matrix, *94*
as.matrix *(matrix)*, 292
as.matrix.dist *(dist)*, 593
as.matrix.noquote *(noquote)*, 318
as.name *(name)*, 312

as.null *(NULL)*, 322
as.numeric, *67*
as.numeric *(numeric)*, 322
as.ordered *(factor)*, 143
as.pairlist *(list)*, 257
as.polySpline *(polySpline)*, 645
as.qr *(qr)*, 398
as.real *(real)*, 415
as.single, *160*
as.single *(double)*, 118
as.ts *(ts)*, 494
as.vector, *52*, *299*
as.vector *(vector)*, 513
asin, *203*
asin *(Trig)*, 493
asinh *(Hyperbolic)*, 202
asOneSidedFormula, 603
assign, 26, *59*
asVector, 639
atan, *203*
atan *(Trig)*, 493
atan2 *(Trig)*, 493
atanh *(Hyperbolic)*, 202
attach, 27, *107*, *249*, *437*
attenu, 28
attitude, 29
attr, 30, *31*, *70*
attr.all.equal *(all.equal)*, 11
attr<- *(attr)*, 30
attributes, *11*, *30*, 31, *59*, *70*, 115, *143*, *301*
attributes<- *(attributes)*, 31
austres, 666
autoload, 31, *249*
autoloader *(autoload)*, 31
ave, 32
axis, 33, *163*, *345*, *365*, *367*

backsolve, 34, *62*, *444*
backSpline, 640
bandwidth.kernel *(kernel)*, 674
barplot, 35, *245*, *360*
bartlett.test, 530, *530*, *539*, *546*, *557*
basename *(file)*, 151
BATCH, 37, *70*
beaver1 *(beavers)*, 667
beaver2 *(beavers)*, 667
beavers, 667
Bessel, 37, *447*
bessel *(Bessel)*, 37
besselI *(Bessel)*, 37
besselJ *(Bessel)*, 37
besselK *(Bessel)*, 37
besselY *(Bessel)*, 37
Beta, 39
beta, *38*, *40*
beta *(Special)*, 446
binom.test, 531
Binomial, 40
binomial *(family)*, 147
biplot, 588, *590*
biplot.default, *589*
biplot.princomp, 589, *589*
BJsales, 668
BOD, 604
body, 41, *105*, *162*, *172*
body<- *(body)*, 41
box, 42, *375*, *417*
Box.test, 668
boxplot, 43, *45*, *49*, *360*, *459*
boxplot.stats, *44*, 45, *157*
break *(Control)*, 80
browser, 46, *96*
bs, 641, *646*
bug.report, 46, *329*
builtins, 49
bxp, *44*, *45*, 49
by, 50, *297*

C, 51, *78*, *79*, *145*
c, 52, *57*, *258*, *299*, *504*, *513*
call, *25*, 52, *106*, *116*, 140, *140*, *220*, *288*, *301*, *312*, *416*
cancor, 590
cars, 53
case.names, *427*
case.names *(case/variable.names)*, 54
case/variable.names, 54
cat, 54, *515*
category *(Defunct)*, 98
Cauchy, 55
cbind, 56, *297*
cbind.ts *(ts.union)*, 691
ccf *(acf)*, 657
ceiling *(Round)*, 425
char.expand, 57
character, 58, *144*, *283*, *319*, *372*, *390*, *391*, *403*, *477*, *486*, *489*
charmatch, 58, *58*, *184*, *286*, *371*, *372*
check.options, 59, *378*
ChickWeight, 604
chickwts, 60
chisq.test, 533, *537*
Chisquare, 61
chol, *34*, 62, *63*, *130*
chol2inv, *62*, 63

choose *(Special)*, 446
chull, 63
class, *9*, 64, *92*, *221*, *261*, *274*, *298*, *300*, *318*, *319*, *381*, *388*, *466*
class<- *(class)*, 64
clearNames, 605, *629*
close.screen *(screen)*, 433
close.socket, 65, *284*, *412*
cm *(units)*, 502
cm.colors *(Palettes)*, 340
cmdscale, 591
co.intervals *(coplot)*, 82
CO2, 606
co2, 66
codes, 66, *94*, *145*, *268*
codes<- *(codes)*, 66
coef, *129*
coef *(coefficients)*, 67
coef.nls *(nls)*, 611
coef.tukeyline *(line)*, 561
coefficients, *13*, 67, *156*, *179*, *181*, *261*, *264*, *421*
coefficients.glm *(glm.summaries)*, 180
coefficients.lm *(lm.summaries)*, 263
coefficients.tukeyline *(line)*, 561
col, 68, *426*, *439*
colnames, *115*
colnames *(row/colnames)*, 426
colnames<- *(row/colnames)*, 426
colors, 69, *339*, *340*, *346*, *364*, *372*
colours *(colors)*, 69
commandArgs, 69
comment, 70
comment<- *(comment)*, 70
Comparison, 71
COMPILE, 71, *441*
complete.cases, 72, *310*
complex, 72, *376*
conflicts, 73, *248*
Conj *(complex)*, 72
Constants, 74
contour, 75, *155*, *190*, *192*, *207*, *224*, *351*
contr.helmert, *79*
contr.helmert *(contrast)*, 77
contr.poly, *79*, *375*
contr.poly *(contrast)*, 77
contr.sum, *51*, *79*
contr.sum *(contrast)*, 77
contr.treatment, *79*, *417*
contr.treatment *(contrast)*, 77
contrast, 77
contrasts, *51*, 78, *78*, *304*, *329*
contrasts<- *(contrasts)*, 78
contrib.url *(update.packages)*, 507
contributors, 79, *84*
Control, 80
convolve, 81, *150*, *315*, *673*, *674*
cooks.distance, *262*, *363*
cooks.distance *(influence.measures)*, 209
coplot, 82, *341*
copyright, 84
copyrights *(copyright)*, 84
cor, 84, *600*, *601*
cor.test, 534
cos, *203*
cos *(Trig)*, 493
cosh *(Hyperbolic)*, 202
count.fields, 85
cov, *87*, *281*, *512*, *600*, *601*
cov *(cor)*, 84
cov.mcd, *601*
cov.mcd *(cov.rob)*, 565
cov.mve, *601*
cov.mve *(cov.rob)*, 565
cov.rob, 565
cov.wt, *85*, 86, *601*
covratio, *262*
covratio *(influence.measures)*, 209
cpgram, 669, *684*
CRAN.packages, *335*
CRAN.packages *(update.packages)*, 507
crossprod, 87
cummax *(cumsum)*, 87
cummin *(cumsum)*, 87
cumprod, *393*
cumprod *(cumsum)*, 87
cumsum, 87, *393*
curve, 88
cut, 89, *449*
cutree, 592
cycle *(time)*, 487

D *(deriv)*, 105
data, *75*, 91, *187*, *249*
data.class, 92
data.entry, *125*, *126*
data.entry *(dataentry)*, 95
data.frame, *43*, *70*, 93, *94*, *107*, *115*, *283*, *292*, *297*, *303*, *357*, *410*, *411*, *413*, *414*, *437*, *479*, *492*, *504*
data.matrix, 94, *292*
dataentry, 95
date, 95, *488*
dbeta, *173*

dbeta *(Beta)*, 39
dbinom, *315*, *374*
dbinom *(Binomial)*, 40
dcauchy *(Cauchy)*, 55
dchisq, *173*
dchisq *(Chisquare)*, 61
de *(dataentry)*, 95
debug, *46*, 96, *172*, *491*
debugger, 97
Defunct, 98, *105*, *321*
delay, *32*, 99, *464*
delete.response, 100
deltat, *676*
deltat *(time)*, 487
demo, 101, *137*, *446*
density, 101, *200*, *357*, *468*
deparse, 104, *119*, *348*, *464*
Deprecated, *99*, 105, *321*
deriv, 105, *317*, *318*, *628*
detach, *28*, 107, *249*, *437*
dev.control *(dev2)*, 108
dev.copy *(dev2)*, 108
dev.cur, *109*, *112*
dev.cur *(dev.xxx)*, 107
dev.list *(dev.xxx)*, 107
dev.next *(dev.xxx)*, 107
dev.off *(dev.xxx)*, 107
dev.prev *(dev.xxx)*, 107
dev.print, *112*
dev.print *(dev2)*, 108
dev.set *(dev.xxx)*, 107
dev.xxx, 107
dev2, 108
dev2bitmap, 110, *112*
deviance, 111, *112*, *141*, *142*, *181*, *264*
device *(Devices)*, 112
Devices, *108*, 112, *182*, *354*, *378*, *434*, *525*
dexp, *517*
dexp *(Exponential)*, 138
df, *484*
df *(FDist)*, 148
df.kernel *(kernel)*, 674
df.residual, *111*, 112
df.residual.glm *(glm.summaries)*, 180
df.residual.lm *(lm.summaries)*, 263
dfbetas, *262*
dfbetas *(influence.measures)*, 209
dffits, *262*
dffits *(influence.measures)*, 209
dgamma, *40*, *61*, *139*
dgamma *(GammaDist)*, 172
dgeom, *315*
dgeom *(Geometric)*, 174
dget, *122*
dget *(dput)*, 119
dhyper *(Hypergeometric)*, 203
diag, 113, *273*, *289*
diag<- *(diag)*, 113
diana, *587*
diff, 113, *670*, *671*, *676*
diff.ts, 670
diffinv, 670
digamma *(Special)*, 446
dim, *23*, *31*, 114, *143*, *292*, *299*, *322*, *482*, *519*
dim<-, *299*
dim<- *(dim)*, 114
dimnames, *23*, *31*, *114*, 115, *292*, *299*, *391*, *426*, *427*, *504*, *505*
dimnames<-, *299*
dimnames<- *(dimnames)*, 115
dir *(list.files)*, 259
dir.create *(file)*, 151
dirname *(file)*, 151
discoveries, 115
dist, *592*, 593
dlnorm, *320*
dlnorm *(Lognormal)*, 271
dlogis *(Logistic)*, 268
DNase, 607
dnbinom, *41*, *175*, *374*
dnbinom *(NegBinomial)*, 314
dnchisq *(Defunct)*, 98
dnorm, *271*
dnorm *(Normal)*, 319
do.call, *53*, 116, *416*
dotplot, *36*, 117, *355*
double, 118, *143*
download.file, 118, *202*, *414*, *507*, *508*
download.packages *(update.packages)*, 507
dpois, *41*, *315*
dpois *(Poisson)*, 373
dput, 119, *122*, *430*, *458*
drop, 120, *289*
drop.scope *(factor.scope)*, 145
drop.terms *(delete.response)*, 100
drop1, *120*, *142*, *146*, *456*
drop1 *(add1)*, 4
dsignrank, *521*
dsignrank *(SignRank)*, 442
dt, *56*, *149*
dt *(TDist)*, 483
dummy.coef, 120

dump, *119*, 122, *430*
dump.frames, *329*
dump.frames *(debugger)*, 97
dunif *(Uniform)*, 500
duplicated, 122, *501*
dweibull, *139*
dweibull *(Weibull)*, 516
dwilcox, *442*
dwilcox *(Wilcoxon)*, 520
dyn.load, *71*, 123, *158*, *161*, *250*, *441*
dyn.unload *(dyn.load)*, 123

ecdf, 651, *653*, *655*
edit, *95*, 125, *126*, *157*, *329*, *336*
edit.data.frame, *125*, 126, *157*
eff.aovlist, 127
effects, *13*, 128, *179*, *181*, *261*, *264*
eigen, 129, *399*, *470*, *600*, *601*
else *(Control)*, 80
emacs *(edit)*, 125
embed, 671
end, *495*
end *(start)*, 452
environment, *26–28*, *43*, *59*, 130, *134*, *135*, *475*, *655*
environment<- *(environment)*, 130
erase.screen *(screen)*, 433
esoph, 131
euro, 133
eurodist, 134
EuStockMarkets, 672
eval, *131*, 134, *140*, *242*, *348*, *446*, *464*, *476*
evalq *(eval)*, 134
example, 136, *208*
exists, *27*, 137, *176*
exp, *139*
exp *(log)*, 266
expand.grid, 138
Exponential, 138
expression, *53*, *104–106*, *134*, *135*, 139, *220*, *244*, *461*, *464*, *486*, *489*
Extract, 140
extractAIC, *6*, *111*, 141, *456*
extractAIC.glm, *456*
Extremes, 142

F *(logical)*, 267
factor, *43*, *67*, *90*, *99*, 143, *178*, *179*, *214*, *247*, *268*, *304*, *316*, *360*, *414*, *417*, *466*, *481*
factor.scope, 145
faithful, 146

FALSE *(logical)*, 267
family, 147, *178*, *282*, *379*
family.glm *(glm.summaries)*, 180
family.lm *(lm.summaries)*, 263
fdeaths *(UKLungDeaths)*, 692
FDist, 148
fft, *81*, *102*, 150, *315*, *682*
file, 151, *154*
file.choose, 152
file.path, 153
file.show, *152*, 153, *336*, *414*
filled.contour, *76*, 154, *514*
filter, 672, *674*
find, *274*
find *(apropos)*, 20
fisher.test, 536
fitted *(fitted.values)*, 156
fitted.nls *(nls)*, 611
fitted.tukeyline *(line)*, 561
fitted.values, *13*, *68*, 156, *179*, *181*, *261*, *264*, *421*
fitted.values.glm *(glm.summaries)*, 180
fitted.values.lm *(lm.summaries)*, 263
fitted.values.tukeyline *(line)*, 561
fivenum, *45*, 156, *216*
fix, *125*, 157, *336*
fligner.test, *530*, *531*, 538, *546*
floor *(Round)*, 425
for *(Control)*, 80
Foreign, 158
Formaldehyde, 161
formals, *22*, 162, *258*, *313*
formals<- *(formals)*, 162
format, *55*, 162, *163–166*, *390*
format.char *(formatC)*, 164
format.info, 163
format.pval, *389*, *390*
formatC, *163*, 164, *164*
formula, *106*, 166, *361*, *485*, *603*, *608*
formula.lm *(lm.summaries)*, 263
formula.nls, 608
forwardsolve *(backsolve)*, 34
frame, 167
freeny, 168
frequency, *495*
frequency *(time)*, 487
friedman.test, 539
ftable, 169, *171*
ftable.default, *170*, *171*
ftable.formula, *169*, 170, *170*
ftable2table *(ftable)*, 169

function, *26*, *42*, *53*, *83*, *131*, *140*, *162*, 171, *215*, *357*

Gamma *(family)*, 147
gamma, *38*, *173*
gamma *(Special)*, 446
gammaCody *(Bessel)*, 37
GammaDist, 172
gaussian *(family)*, 147
gc, 173, *294*, *295*
gcinfo, *295*
gcinfo *(gc)*, 173
gctorture, *173*, 174
Geometric, 174
get, *27*, *59*, *137*, 175, *289*, *377*
getenv, 176
geterrmessage, *97*, *422*
geterrmessage *(stop)*, 457
getInitial, 608
getOption *(options)*, 328
getwd, *176*, 177, *413*, *432*
gl, *145*, 177, *439*
glm, *5*, *68*, *78*, *79*, *111*, *112*, *141*, *147*, *148*, *156*, *166*, *167*, 178, *180*, *181*, *217*, *261*, *264*, *282*, *311*, *323*, *362*, *382*, *421*, *455*, *456*, *466*, *485*, *518*
glm.control, *179*
glm.fit.null *(glm)*, 178
glm.summaries, 180
globalenv *(environment)*, 130
GNOME *(Gnome)*, 181
Gnome, 181
gnome *(Gnome)*, 181
graphics.off, *108*, *112*
graphics.off *(dev.xxx)*, 107
gray, *69*, 182, *201*, *339*, *340*, *346*, *423*
grep, 183, *274*, *460*
grey *(gray)*, 182
grid, 184
gsub *(grep)*, 183

HairEyeColor, 185
hasTsp *(tsp)*, 496
hat, *262*, *275*, *362*
hat *(influence.measures)*, 209
hclust, *587*, *592*, *593*, 594, *594*, *602*
heat.colors, *69*, *206*, *207*
heat.colors *(Palettes)*, 340
help, *22*, *92*, *136*, *154*, 186, *189*, *190*, *208*, *396*
help.search, *187*, 188
help.start, *186*, *187*, 189, *189*, *329*
Hershey, *76*, 190, *224*, *487*
hist, *36*, *103*, 199, *364*
hsv, *69*, *183*, 201, *207*, *339*, *340*, *343*, *423*
httpclient, 202
Hyperbolic, 202
Hypergeometric, 203

I, *93*
I *(formula)*, 166
identify, 204
identify.hclust, 596, *602*
if, *205*, *347*
if *(Control)*, 80
ifelse, *80*, 205
Im *(complex)*, 72
image, *76*, *112*, *155*, 206, *351*, *364*
index.search, 207
Indometh, 609
Inf, *22*, *156*
Inf *(is.finite)*, 217
infert, 208
influence.measures, 209, *262*
inherits *(class)*, 64
InsectSprays, 210
INSTALL, 212, *249*, *418*, *508*
install.packages, *249*
install.packages *(update.packages)*, 507
installed.packages *(update.packages)*, 507
integer, *118*, *164*, 213, *406*
interaction, 213
interactive, 214
Internal, 214
interpSpline, *640*, 642, *644*, *645*, *647*, *650*
intersect *(sets)*, 440
inverse.gaussian *(family)*, 147
invisible, *172*, 215, *250*, *339*, *388*, *597*
IQR, *157*, 215, *280*
iris, 216
iris3 *(iris)*, 216
is.array *(array)*, 23
is.atomic *(is.recursive)*, 222
is.call *(call)*, 52
is.character *(character)*, 58
is.complex *(complex)*, 72
is.data.frame *(data.frame)*, 93
is.double *(double)*, 118
is.element *(sets)*, 440
is.empty.model, 217
is.environment *(environment)*, 130
is.expression *(expression)*, 139
is.factor *(factor)*, 143

is.finite, 217
is.function, 220
is.infinite *(is.finite)*, 217
is.integer *(integer)*, 213
is.language, *53*, 220, *222*, *312*
is.list, *222*, *513*
is.list *(list)*, 257
is.loaded *(dyn.load)*, 123
is.logical *(logical)*, 267
is.matrix *(matrix)*, 292
is.mts *(ts-internal)*, 689
is.na, *72*, *299*
is.na *(NA)*, 310
is.name *(name)*, 312
is.nan, *299*, *310*
is.nan *(is.finite)*, 217
is.null *(NULL)*, 322
is.numeric, *513*
is.numeric *(numeric)*, 322
is.object, 221
is.ordered *(factor)*, 143
is.pairlist *(list)*, 257
is.qr *(qr)*, 398
is.R, 221
is.real *(real)*, 415
is.recursive, 222
is.single, 222
is.stepfun *(stepfun)*, 654
is.symbol, 223
is.ts *(ts)*, 494
is.tskernel *(kernel)*, 674
is.vector *(vector)*, 513
islands, 223
isoMDS, *592*

Japanese, *192*, 224
jitter, 237, *428*, *468*

kappa, 238
kernapply, 674, *675*
kernel, 674, *674*
kmeans, *596*, 598
knots, *653*, *655*
knots *(stepfun)*, 654
kronecker, 239, *333*
kruskal.test, 541, *559*
ks.test, 542
ksmooth, 571

labels, 240
lag, 676
LakeHuron, 677
lapply, *7*, *18*, 241, *288*, *482*
Last.value, 241
layout, *108*, 242, *344*, *346*, *434*
lbeta *(Special)*, 446
lchoose *(Special)*, 446
lcm *(layout)*, 242
ldeaths *(UKLungDeaths)*, 692
legend, *140*, 244
length, 246
length<- *(length)*, 246
LETTERS *(Constants)*, 74
letters *(Constants)*, 74
levels, *67*, *145*, 247, *247*, *268*, *316*
levels.factor, 247
levels<-, *247*
levels<- *(levels)*, 247
levels<-.factor, *247*
levels<-.factor *(levels.factor)*, 247
lgamma *(Special)*, 446
lh, 677
library, *28*, *32*, *107*, *187*, *213*, 248, *250*, *349*, *437*, *508*
library.dynam, *124*, *249*, 250
licence *(license)*, 251
license, *84*, 251
LifeCycleSavings, 256
line, 561
lines, *3*, *89*, *185*, 257, *290*, *291*, *341*, *345*, *357*, *364*, *365*, *373*, *375*, *438*
lines.formula *(plot.formula)*, 360
lines.ts *(ts)*, 494
list, *43*, *107*, *140*, 257, *310*, *403*, *482*, *492*
list.files, *152*, *154*, 259, *478*
lm, *5*, *6*, *16*, *54*, *68*, *78*, *79*, *111*, *112*, *128*, *141*, *146*, *156*, *166*, *167*, *179*, *209*, *217*, 260, *262–264*, *276*, *277*, *311*, *329*, *362*, *381*, *384*, *395*, *421*, *455*, *466*, *485*, *518*, *542*, *561*
lm.fit.null *(lm)*, 260
lm.influence, *209*, *210*, 261, *261*, *264*, *275*, *276*, *362*, *363*, *518*, *519*
lm.summaries, 263
lm.wfit *(lm)*, 260
lmsreg *(lqs)*, 567
load, *91*, 264, *430*
loadings *(princomp)*, 600
Loblolly, 610
local *(eval)*, 134
locator, *244*, 265
loess, *564*, 572, *575*, *580*, *583*, *687*
loess.control, 573, *573*, 574
loess.smooth *(scatter.smooth)*, 582
log, *4*, 266

log10 *(log)*, 266
log1p *(log)*, 266
log2 *(log)*, 266
Logic, 267, *519*
logical, 267, *267*, *519*
Logistic, 268
logLik, 611
logLik.nls *(nls)*, 611
loglin, 269
Lognormal, 271
longley, 272
lower.tri, 272
lowess, 273, *341*, *525*, *564*
lqs, 567, *567*, *570*
ls, 274, *377*, *418*, *458*
ls.diag, 275, *276*, *277*
ls.print, *275*, 276, *277*
ls.str, *458*
ls.str *(str)*, 458
lsf.str, *458*
lsf.str *(str)*, 458
lsfit, *275*, 276, *276*, *399*, *400*
ltsreg *(lqs)*, 567
lynx, 677

Machine, 277, *279*
machine, 279, *279*, *404*
mad, *216*, 280, *435*
mahalanobis, 281
make.link, 282, *379*
make.names, *93*, *94*, 282
make.socket, *65*, 283, *412*, *414*
make.tables, 284
mantelhaen.test, 543
margin.table, 285, *396*
mat.or.vec, 285
match, *59*, *184*, 286, *372*
match.arg, 286, *286*, *288*, *289*, *372*
match.call, 287, *287*, *372*
match.fun, *287*, 288, *288*, *372*
Math, *23*, *266*, *447*
Math *(Methods)*, 297
Math.data.frame, *94*
matlines *(matplot)*, 290
matmult, *240*, 289, *333*
matplot, *216*, 290
matpoints *(matplot)*, 290
matrix, *23*, *83*, *94*, *113*, *115*, *140*, *273*, *289*, *291*, 292, *322*
max, *409*
max *(Extremes)*, 142
mcnemar.test, 544
mdeaths *(UKLungDeaths)*, 692
mean, *11*, *33*, 293, *483*, *518*
median, *33*, *157*, *280*, 293, *563*
medpolish, 562
Memory, *173*, 294, *414*
memory.profile, 295
menu, 296
merge, 296
Methods, 297, *409*
methods, *11*, *93*, *180*, *187*, *221*, *263*, *274*, *298*, 299, *299*, *319*, *388*, *466*
min, *409*
min *(Extremes)*, 142
missing, 300, *464*
Mod, *11*
Mod *(complex)*, 72
mode, *11*, *20*, 301, *391*, *440*, *464*, *498*
mode<- *(mode)*, 301
model.extract, 302, *304*
model.frame, *302*, 303, *304*, *323*
model.matrix, *93*, *303*, 304, *506*
model.matrix.glm.null *(model.matrix)*, 304
model.matrix.lm *(model.matrix)*, 304
model.offset, *323*
model.offset *(model.extract)*, 302
model.response *(model.extract)*, 302
model.tables, *16*, *121*, *284*, 305, *395*, *420*, *435*
model.tables.aovlist, *127*
model.weights *(model.extract)*, 302
modreg-internal, 575
month.abb *(Constants)*, 74
month.name *(Constants)*, 74
mood.test, *530*, *531*, *539*, 545, *557*
mosaicplot, 306
mosaicplot.default, *307*
mosaicplot.formula, *307*
mostattributes<- *(attributes)*, 31
mtcars, 308
mtext, 308, *342*, *345*, *365*, *367*, *487*, *489*
mvfft *(fft)*, 150

NA, *23*, *43*, *45*, *144*, *156*, *218*, *246*, *300*, 310, *311*, *329*, *332*, *389*, *390*, *409*, *413*, *432*, *465*, *473*, *519*, *526*, *577*
na.action, *310*, 311, *311*
na.contiguous *(na.omit.ts)*, 678
na.fail, *72*, *178*, *260*, *303*, *310*, 311, *311*, *678*
na.omit, *72*, *178*, *260*, *303*, *310*, *311*, *577*, *678*
na.omit *(na.fail)*, 311
na.omit.ts, 678

name, *220*, *248*, 312
names, *23*, *31*, *113*, *115*, *143*, *241*, *283*, 312, *402*, *504*, *505*
names.dist *(dist)*, 593
names<- *(names)*, 312
names<-.dist *(dist)*, 593
NaN, *23*, *45*, *156*, *310*
NaN *(is.finite)*, 217
nargs, 313
nchar, 313, *349*, *460*, *461*, *465*
NCOL, *427*
NCOL *(nrow)*, 321
ncol, *114*
ncol *(nrow)*, 321
NegBinomial, 314
new.env *(environment)*, 130
next *(Control)*, 80
NextMethod, *65*
NextMethod *(methods)*, 299
nextn, *81*, *150*, 315
nhtemp, 316
nlevels, *67*, *145*, *247*, 316
nlm, *106*, 317, *327*, *328*, *502*, *665*
nls, *608*, *609*, 611, *613*, *615*, *619*, *621–624*, *631–638*
nls.control, 613
nlsModel, *612*, 614, *624*
NLSstAsymptotic, 615
NLSstClosestX, 616, *618*, *630*
NLSstLfAsymptote, *616*, 617, *630*
NLSstRtAsymptote, *616*, 617, *618*, *630*
noquote, 318, *388*, *389*, *391*, *474*
Normal, 319
nottem, 678
NotYet, 321
NotYetImplemented *(NotYet)*, 321
NotYetUsed *(NotYet)*, 321
NROW, *427*
NROW *(nrow)*, 321
nrow, *114*, 321
ns, 643, *646*
NULL, 322
numeric, *107*, *163*, 322, *409*

objects, *21*, *28*, *107*, *249*, *418*, *437*
objects *(ls)*, 274
offset, *260*, *302*, 323
old.packages *(update.packages)*, 507
on.exit, *283*, 323, *475*
Ops *(Methods)*, 297
Ops.ts *(ts-internal)*, 689
optim, 324, *665*
optimise *(optimize)*, 327
optimize, *318*, 327, *327*, *502*
options, *47*, *78*, *97*, *124*, *157*, *164*, *178*, *260*, *303*, *311*, 328, *346*, *389*, *391*, *422*, *446*, *457*, *458*, *476*, *515*
Orange, 618
OrchardSprays, 331
order, 332, *410*, *445*
ordered *(factor)*, 143
outer, *240*, *288*, 333

p.adjust, 334, *547–549*
pacf *(acf)*, 657
package.contents, 335, *349*
package.dependencies, 335
package.description *(package.contents)*, 335
page, 336
pairlist *(list)*, 257
pairs, *83*, *94*, 336, *338*, *341*
pairs.default, *338*
pairs.formula, 338
pairwise.prop.test, 547
pairwise.t.test, *335*, 547, *548*, *552*
pairwise.table, 548
pairwise.wilcox.test, 549
palette, *69*, *155*, 339, *340*, *364*, *372*
Palettes, 340
panel.smooth, *83*, 341, *362*
par, *3*, *24*, *36*, *42*, *44*, *76*, *182*, *206*, *207*, *243*, *257*, *290*, *291*, *309*, *338*, *341*, 342, *351*, *357*, *359–363*, *372*, *375*, *434*, *437*, *468*, *486*, *489*, *502*, *576*
Paren, 347
parent.frame *(sys.parent)*, 475
parse, *104*, *105*, 348, *445*, *446*
parse.dcf, *335*, 348, *508*
paste, *163*, *313*, 349, *460*, *465*
pbeta *(Beta)*, 39
pbinom *(Binomial)*, 40
pcauchy *(Cauchy)*, 55
pchisq, *483*, *496*
pchisq *(Chisquare)*, 61
pentagamma *(Special)*, 446
periodicSpline, *642*, 644, *645*, *647*, *650*
persp, 350
pexp *(Exponential)*, 138
pf *(FDist)*, 148
pgamma, *314*
pgamma *(GammaDist)*, 172
pgeom *(Geometric)*, 174
phones, 352
phyper *(Hypergeometric)*, 203
pi *(Constants)*, 74

pico *(edit)*, 125
pictex, 112, 353
piechart, 354
PlantGrowth, 355
Platform, 356
plnorm *(Lognormal)*, 271
plogis *(Logistic)*, 268
plot, *36*, *94*, *185*, *206*, *245*, *257*, *290*, *291*, *309*, *345*, 357, *359*, *360*, *364*, *365*, *372*, *373*, *468*, *526*, *562*, *601*, *651–653*
plot.acf *(acf)*, 657
plot.data.frame *(data.frame)*, 93
plot.default, *49*, *94*, *168*, *345*, *346*, *357*, 358, *360*, *361*, *363–365*, *467*, *525*
plot.density *(density)*, 101
plot.ecdf *(ecdf)*, 651
plot.factor, 360, *361*
plot.formula, *357*, 360, *360*
plot.function *(curve)*, 88
plot.hclust *(hclust)*, 594
plot.lm, 361
plot.medpolish *(medpolish)*, 562
plot.mlm *(lm.summaries)*, 263
plot.mts *(ts)*, 494
plot.new, *344*, *363*
plot.new *(frame)*, 167
plot.polySpline *(polySpline)*, 645
plot.ppr, 575, *578*
plot.prcomp *(prcomp)*, 599
plot.princomp *(princomp)*, 600
plot.profile.nls, 619, *622*
plot.spec, *680*, *682*
plot.spec *(spectrum)*, 684
plot.spline *(predict.bSpline)*, 646
plot.stepfun, *651*, 652, *654*, *655*
plot.stl *(stl)*, 686
plot.ts, *690*
plot.ts *(ts)*, 494
plot.tskernel *(kernel)*, 674
plot.window, *36*, *155*, *168*, *359*, 363
plot.xy, *257*, 364, *364*, *372*, *373*
plot.xyVector *(xyVector)*, 650
plotmath, *25*, 245, *309*, 365, *487*, *489*
pmatch, *58*, *59*, *184*, *286–288*, 371
pmax *(Extremes)*, 142
pmin *(Extremes)*, 142
pnbinom *(NegBinomial)*, 314
pnchisq *(Defunct)*, 98
pnorm, *497*
pnorm *(Normal)*, 319
points, *83*, *185*, *257*, *290*, *291*, *345*, *357*, *362*, *365*, 372
points.default, *364*
points.formula *(plot.formula)*, 360
pointwise *(modreg-internal)*, 575
Poisson, 373
poisson *(family)*, 147
poly, 374
polygon, *64*, 375, *417*, *438*
polyroot, 376, *502*
polySpline, 645
pos.to.env, 377
postscript, *108–111*, 112, *346*, *354*, 377
power, *147*, *148*, 379
power.prop.test, 549, *552*
power.t.test, 550, *552*
PP.test, 679
ppoints, 380, *397*
ppois *(Poisson)*, 373
ppr, 576, *576*, *585*
prcomp, 599, *601*
precip, 380
predict, 381, *384*, *621*, *646*
predict.ar *(ar)*, 659
predict.arima0 *(arima0)*, 664
predict.bs, 646
predict.bSpline, 646
predict.glm, *179*, 382
predict.lm, *261*, *381*, 383
predict.loess, *573*, 579
predict.lqs, 569, *569*
predict.mlm *(predict.lm)*, 383
predict.nbSpline *(predict.bSpline)*, 646
predict.nls, 620
predict.npolySpline *(predict.bSpline)*, 646
predict.ns *(predict.bs)*, 646
predict.pbSpline *(predict.bSpline)*, 646
predict.polySpline *(polySpline)*, 645
predict.ppolySpline *(predict.bSpline)*, 646
predict.ppr *(ppr)*, 576
predict.princomp *(princomp)*, 600
predict.smooth.spline, 580, *584*
predict.smooth.spline.fit *(smooth.spline)*, 583
predLoess *(modreg-internal)*, 575
preplot, 384
presidents, 385
pressure, 386

pretty, 386
Primitive, 388
princomp, *589, 590*, 600, *600*
print, *9*, *15*, *55*, *70*, *94*, *163*, *319*, 388, *389–391*, *458*, *562*, *601*
print.anova, *389*
print.anova *(anova)*, 12
print.anova.glm *(Defunct)*, 98
print.anova.lm *(Defunct)*, 98
print.aov *(aov)*, 15
print.aovlist *(aov)*, 15
print.ar *(ar)*, 659
print.arima0 *(arima0)*, 664
print.atomic *(print.default)*, 390
print.bSpline *(interpSpline)*, 642
print.by *(by)*, 50
print.coefmat, *329*, 389
print.data.frame *(data.frame)*, 93
print.default, *70*, *388*, *389*, 390, *391*
print.density *(density)*, 101
print.dist *(dist)*, 593
print.dummy.coef *(dummy.coef)*, 120
print.ecdf *(ecdf)*, 651
print.family *(family)*, 147
print.formula *(formula)*, 166
print.ftable *(ftable)*, 169
print.glm *(glm.summaries)*, 180
print.hclust *(hclust)*, 594
print.infl *(influence.measures)*, 209
print.lm *(lm.summaries)*, 263
print.loess *(loess)*, 572
print.lqs *(lqs)*, 567
print.matrix, *94*, 391
print.medpolish *(medpolish)*, 562
print.mtable *(alias)*, 9
print.nls *(nls)*, 611
print.noquote *(noquote)*, 318
print.pairwise.htest, 551
print.plot *(Defunct)*, 98
print.polySpline *(polySpline)*, 645
print.power.htest, 552
print.ppolySpline *(interpSpline)*, 642
print.ppr *(ppr)*, 576
print.prcomp *(prcomp)*, 599
print.princomp *(princomp)*, 600
print.simple.list *(print)*, 388
print.smooth.spline *(smooth.spline)*, 583
print.socket *(make.socket)*, 283
print.stepfun *(stepfun)*, 654
print.stl *(stl)*, 686
print.summary.aov *(aov)*, 15
print.summary.aovlist *(aov)*, 15
print.summary.glm, *389*
print.summary.glm *(glm.summaries)*, 180
print.summary.lm, *163*, *389*, *390*
print.summary.lm *(lm.summaries)*, 263
print.summary.loess *(loess)*, 572
print.summary.nls *(nls)*, 611
print.summary.ppr *(ppr)*, 576
print.summary.prcomp *(prcomp)*, 599
print.table *(print)*, 388
print.tables.aov *(model.tables)*, 305
print.tabular *(Defunct)*, 98
print.terms *(terms)*, 485
print.ts *(ts)*, 494
print.tskernel *(kernel)*, 674
print.tukeyline *(line)*, 561
print.xyVector *(xyVector)*, 650
prmatrix *(print.matrix)*, 391
proc.time, 392, *479*
prod, 393
profile, 393, *619*, *622*, *623*
profile.nls, *619*, 621, *624*
profiler, 622, *624*
profiler.nls, *622*, 623, *623*
proj, *16*, 394
prompt, *187*, 395
prop.table, 396
prop.test, *532*, *547*, *550*, 553
provide *(library)*, 248
ps.options, *60*
ps.options *(postscript)*, 377
psignrank *(SignRank)*, 442
pt *(TDist)*, 483
ptukey *(Tukey)*, 496
punif *(Uniform)*, 500
Puromycin, 625
pweibull *(Weibull)*, 516
pwilcox *(Wilcoxon)*, 520

q, *430*
q *(quit)*, 402
qbeta *(Beta)*, 39
qbinom *(Binomial)*, 40
qcauchy *(Cauchy)*, 55
qchisq *(Chisquare)*, 61
qexp *(Exponential)*, 138
qf *(FDist)*, 148
qgamma *(GammaDist)*, 172
qgeom *(Geometric)*, 174
qhyper *(Hypergeometric)*, 203
qlnorm *(Lognormal)*, 271
qlogis *(Logistic)*, 268
qnbinom *(NegBinomial)*, 314

qnchisq *(Defunct)*, 98
qnorm, *497*
qnorm *(Normal)*, 319
qpois *(Poisson)*, 373
qqline *(qqnorm)*, 397
qqnorm, *380*, 397, *555*
qqplot, *380*
qqplot *(qqnorm)*, 397
qr, *34*, *62*, *129*, *130*, *238*, *239*, *260*, 398, *400*, *470*, *591*
QR.Auxiliaries, 399
qr.Q, *399*
qr.Q *(QR.Auxiliaries)*, 399
qr.R, *399*
qr.R *(QR.Auxiliaries)*, 399
qr.solve, *444*
qr.X, *399*
qr.X *(QR.Auxiliaries)*, 399
qsignrank *(SignRank)*, 442
qt *(TDist)*, 483
qtukey *(Tukey)*, 496
quakes, 401
quantile, *157*, *215*, *216*, *294*, 401
quasi *(family)*, 147
quit, 402
qunif *(Uniform)*, 500
quote *(substitute)*, 463
qweibull *(Weibull)*, 516
qwilcox *(Wilcoxon)*, 520

R.home, 403
R.Version, 403
R.version, *221*
R.version *(R.Version)*, 403
R_HOME *(RHOME)*, 423
R_LIBS *(library)*, 248
rainbow, *69*, *183*, *201*, *206*, *207*, *339*, *346*, *423*
rainbow *(Palettes)*, 340
Random, 404
Random.user, *406*, 407
randu, 408
range, *83*, *143*, *157*, *216*, 409
range.default, *409*
rank, *332*, 410, *445*
rbeta *(Beta)*, 39
rbind *(cbind)*, 56
rbinom *(Binomial)*, 40
rcauchy *(Cauchy)*, 55
rchisq *(Chisquare)*, 61
Re *(complex)*, 72
read.csv *(read.table)*, 412
read.csv2 *(read.table)*, 412
read.fwf, 410, *414*
read.socket, *65*, *284*, 411, *414*
read.table, *85*, *86*, *91*, *94*, *295*, *411*, 412, *414*, *432*, *524*
read.table.url, *202*, 414
readline, 415
real, 415
Recall, *53*, 416
rect, *42*, *375*, 416
rect.hclust, 602
reformulate *(delete.response)*, 100
regexpr *(grep)*, 183
relevel, 417
REMOVE, *213*, *249*, 418, *508*
remove, 418
rep, 419, *439*, *526*
repeat *(Control)*, 80
replace, 419
replications, 420
require *(library)*, 248
resid *(residuals)*, 421
residuals, *13*, *68*, *128*, *156*, *179*, *181*, *261*, *264*, 421, *519*
residuals.glm, *264*
residuals.glm *(glm.summaries)*, 180
residuals.lm *(lm.summaries)*, 263
residuals.lqs *(lqs)*, 567
residuals.nls *(nls)*, 611
residuals.tukeyline *(line)*, 561
restart, 421, *457*
return, *215*, *347*
return *(function)*, 171
rev, 422
rexp *(Exponential)*, 138
rf *(FDist)*, 148
rgamma *(GammaDist)*, 172
rgb, *69*, *183*, *201*, *340*, *346*, 423
rgeom *(Geometric)*, 174
RHOME, 423
rhyper *(Hypergeometric)*, 203
rivers, 423
rle, 424
rlnorm *(Lognormal)*, 271
rlogis *(Logistic)*, 268
rm *(remove)*, 418
rnbinom *(NegBinomial)*, 314
rnchisq *(Defunct)*, 98
RNG *(Random)*, 404
RNGkind, *407*, *565*
RNGkind *(Random)*, 404
rnorm, *407*, *501*
rnorm *(Normal)*, 319

rock, 581
Round, 425
round *(Round)*, 425
row, *68*, 426, *439*
row.names *(data.frame)*, 93
row.names<- *(data.frame)*, 93
row/colnames, 426
rownames, *94*, *115*
rownames *(row/colnames)*, 426
rownames<- *(row/colnames)*, 426
rowsum, 427
rpois *(Poisson)*, 373
Rprofile *(Startup)*, 452
rsignrank *(SignRank)*, 442
rstandard *(influence.measures)*, 209
rstudent *(influence.measures)*, 209
rt *(TDist)*, 483
rug, *238*, 428
runif, *320*, *407*
runif *(Uniform)*, 500
rweibull *(Weibull)*, 516
rwilcox *(Wilcoxon)*, 520

sammon, *592*
sample, 429
sapply, *482*
sapply *(lapply)*, 241
save, *97*, *122*, *265*, 430
save.plot *(Defunct)*, 98
scale, 430, *471*
scan, *295*, *348*, *411*, *414*, 431, *446*, *523*
scan.url *(read.table.url)*, 414
scatter.smooth, 582
screen, 433
screeplot *(princomp)*, 600
sd, 435, *483*
se.aov, 435
se.aovlist *(se.aov)*, 435
se.contrast, 436
se.contrast.aovlist, *127*
search, *21*, *28*, *59*, *73*, *107*, *187*, *249*, 437
searchpaths *(search)*, 437
segments, *3*, *24*, *375*, *417*, 437, *653*
selfStart, *609*, *616*, *618*, 626, *630–638*
selfStart.default, *609*, *626*, 627, *628*
selfStart.formula, *609*, *626*, *627*, 628
seq, *419*, *422*, 438, *439*
sequence, *419*, 439, *439*
set.seed *(Random)*, 404
setdiff *(sets)*, 440
setequal *(sets)*, 440
setNames, *606*, 629
sets, 440
setwd *(getwd)*, 177
shapiro.test, *543*, 554
SHLIB, *71*, 441
show.data *(data)*, 91
sign, 441
signif *(Round)*, 425
SignRank, 442
simpleLoess *(modreg-internal)*, 575
sin, *4*, *203*
sin *(Trig)*, 493
single, *160*
single *(double)*, 118
sinh *(Hyperbolic)*, 202
sink, 443
sleep, 443
smooth, 563
smooth.spline, *564*, *578*, *580*, 583
solve, *34*, *63*, 444
solve.qr, *399*
sort, *332*, *410*, *422*, 445
sort.list *(order)*, 332
sortedXyData, *616*, *618*, 630
source, *91*, *101*, *122*, *214*, *348*, *414*, 445, *477*
source.url *(read.table.url)*, 414
spec, *680*, *682*
spec *(spectrum)*, 684
spec.ar, 680, *684*
spec.pgram, 681, *684*, *685*
spec.taper, *682*, 683
Special, *4*, *23*, 446
spectrum, *674*, *681*, *682*, 684
spline, *20*
spline *(splinefun)*, 447
spline.des *(splineDesign)*, 648
splineDesign, 648
splinefun, *20*, *89*, 447, *652*, *653*, *655*
splineKnots, *642*, *644*, *645*, 649, *650*
splineOrder, *642*, *645*, 649
split, *90*, 448
split.screen, *344*, *346*
split.screen *(screen)*, 433
sqrt *(abs)*, 4
SSasymp, *616*, 631
SSasympOff, 632
SSasympOrig, 633
SSbiexp, 634
SSfol, 635
SSfpl, 636
SSlogis, 637
SSmicmen, 638
stack.loss *(stackloss)*, 449

stack.x *(stackloss)*, 449
stackloss, 449
stars, 450
start, 452, *488*, *495*, *496*
Startup, 452
stat.anova, *181*, 453
state, 454
stem, *200*, 455
step, *6*, *141*, *142*, 455
stepfun, *651*, *652*, 654
stl, 686
stop, *329*, 457, *515*
storage.mode, *498*
storage.mode *(mode)*, 301
storage.mode<- *(mode)*, 301
str, 458
strheight *(strwidth)*, 461
stripplot, *44*, *94*, 459
strsplit, *313*, *349*, 460, *465*
structure, 461
strwidth, *245*, *313*, *343*, 461
sub, *460*
sub *(grep)*, 183
Subscript *(Extract)*, 140
subset, 462, *492*
substitute, *105*, *300*, 463
substr, *3*, *313*, *349*, *460*, 465
substring *(substr)*, 465
sum, *393*, 466
Summary *(Methods)*, 297
summary, *13*, *15*, *179*, *181*, *262*, *264*, *458*, *459*, 466
summary.aov *(aov)*, 15
summary.aovlist *(aov)*, 15
summary.ecdf *(ecdf)*, 651
summary.glm, *179*, *467*
summary.glm *(glm.summaries)*, 180
summary.infl *(influence.measures)*, 209
summary.lm, *261*, *262*, *275*, *467*
summary.lm *(lm.summaries)*, 263
summary.loess *(loess)*, 572
summary.mlm *(lm.summaries)*, 263
summary.nls *(nls)*, 611
summary.ppr *(ppr)*, 576
summary.prcomp *(prcomp)*, 599
summary.princomp *(princomp)*, 600
summary.stepfun *(stepfun)*, 654
sunflowerplot, 467
sunspot, 687
sunspots, 469, *688*
supsmu, *564*, *578*, 584
svd, *62*, *130*, *239*, *399*, 469, *591*, *600*
sweep, *18*, *288*, *431*, 470
swiss, 471
switch, *80*, 472
symbol.C *(dyn.load)*, 123
symbol.For *(dyn.load)*, 123
symbols *(NotYet)*, 321
symnum, 473
sys.call, *313*
sys.call *(sys.parent)*, 475
sys.calls *(sys.parent)*, 475
sys.frame, *135*
sys.frame *(sys.parent)*, 475
sys.frames *(sys.parent)*, 475
sys.function *(sys.parent)*, 475
sys.nframe *(sys.parent)*, 475
sys.on.exit *(sys.parent)*, 475
sys.parent, 475
sys.parents *(sys.parent)*, 475
sys.source, 476
sys.status *(sys.parent)*, 475
system, *221*, *356*, 477
system.file, 478
system.test *(Defunct)*, 98
system.time, *392*, 478, *488*

T *(logical)*, 267
t, *17*, 479
t.test, *542*, *548*, *551*, 555, *559*
table, *90*, *169–171*, *270*, 480, *481*
tabulate, *90*, 481
tan, *203*
tan *(Trig)*, 493
tanh *(Hyperbolic)*, 202
tapply, *7*, *18*, *50*, *241*, *427*, 481
TDist, 483
tempfile, 484, *503*
terms, *100*, *167*, *304*, 485, *506*
terrain.colors, *206*, *207*, *339*
terrain.colors *(Palettes)*, 340
tetragamma *(Special)*, 446
text, *76*, *140*, *190*, *192*, *224*, *245*, *309*, *342*, *345*, *365*, *367*, *461*, 486, *489*
time, *452*, *479*, 487, *495*, *496*, *522*
Titanic, 488
title, *36*, *76*, *290*, *309*, *345*, *351*, *362*, *367*, *487*, 489
toeplitz, 688
ToothGrowth, 490
topo.colors, *69*, *206*, *207*
topo.colors *(Palettes)*, 340
trace, 491
traceback, *46*, *96*, 491
transform, *463*, 492

treering, 689
trees, 493
Trig, *266*, 493
trigamma *(Special)*, 446
TRUE, *267*
TRUE *(logical)*, 267
trunc *(Round)*, 425
try, *329*
try *(restart)*, 421
ts, *329*, *452*, *487*, *488*, 494, *496*, *522*
ts-internal, 689
ts.intersect *(ts.union)*, 691
ts.plot, 690
ts.union, 691
tsp, *452*, *488*, *495*, 496
tsp<- *(tsp)*, 496
Tukey, 496
typeof, *23*, *301*, *312*, 498

UCBAdmissions, 498
UKDriverDeaths, 692
UKLungDeaths, 692
unclass, *67*
unclass *(class)*, 64
undebug *(debug)*, 96
undoc, 499
Uniform, 500
union *(sets)*, 440
unique, *122*, 501
uniroot, *318*, *328*, *376*, 501, *550*, *551*
units, 502
unix *(system)*, 477
unix.time *(system.time)*, 478
unlink, *484*, 503
unlist, *52*, *299*, 504
unname, 504
untrace *(trace)*, 491
update, 505
update.formula, *505*, 506
update.packages, *213*, *330*, *336*, 507
upper.tri *(lower.tri)*, 272
url.show *(read.table.url)*, 414
USAccDeaths, 692
USArrests, 508
UseMethod, *57*, *65*, *310*
UseMethod *(methods)*, 299
USJudgeRatings, 509
USPersonalExpenditure, 510
uspop, 510

VADeaths, 511
var, *87*, *280*, *281*, *435*, 512
var.test, *530*, *531*, *539*, *546*, 557
variable.names, *427*
variable.names *(case/variable.names)*, 54
vector, *164*, *258*, *312*, 513
Version *(Defunct)*, 98
version *(R.Version)*, 403
vi, *95*
vi *(edit)*, 125
volcano, 514

warning, *23*, *57*, *457*, 514, *515*
warnings, *330*, 515, *515*
warpbreaks, 516
Weibull, 516
weighted.mean, *293*, 517
weighted.residuals, *264*, 518
weights, *518*
weights *(lm.summaries)*, 263
weights.nls *(nls)*, 611
which, 519
while *(Control)*, 80
wilcox.test, *542*, *549*, 558
Wilcoxon, 520
window, *488*, *495*, 521
women, 522
write, *119*, *122*, *432*, 523
write.socket *(read.socket)*, 411
write.table, *414*, 524

X11, *330*
X11 *(x11)*, 524
x11, 112, *182*, *346*, 524
xedit *(edit)*, 125
xemacs *(edit)*, 125
xinch *(units)*, 502
xor *(Logic)*, 267
xpdrows.data.frame *(data.frame)*, 93
xy.coords, *64*, *358*, *359*, *364*, *467*, *486*, 525
xyinch *(units)*, 502
xyVector, *639*, *647*, 650

yinch *(units)*, 502

zapsmall, *389*
zapsmall *(Round)*, 425
zip.file.extract, 526