Selected Functions from R Documentation

Table of Contents

names {base}

The Names of an Object

Description

Functions to get or set the names of an object.

Usage

```
names(x)
names(x) <- value
```

Arguments

x

an **R** object.

value

a character vector of up to the same length as x, or NULL.

Details

names is a generic accessor function, and names<- is a generic replacement function. The default methods get and set the "names" attribute of a vector (including a list) or pairlist.

If value is shorter than x, it is extended by character NAs to the length of x.

It is possible to update just part of the names attribute via the general rules: see the examples. This works because the expression there is evaluated as z <- "names<-"(z, "[<-"(names(z), 3, "c2")).

The name "" is special: it is used to indicate that there is no name associated with an element of a (atomic or generic) vector. Subscripting by "" will match nothing (not even elements which have no name).

A name can be character NA, but such a name will never be matched and is likely to lead to confusion.

Both are primitive functions.

Value

For names, NULL or a character vector of the same length as x. (NULL is given if the object has no names, including for objects of types which cannot have names.)

For names<-, the updated object. (Note that the value of names(x) <- value is that of the assignment, value, not the return value from the left-hand side.)

Note

For vectors, the names are one of the attributes with restrictions on the possible values. For pairlists, the names are the tags and converted to and from a character vector.

For a one-dimensional array the names attribute really is dimnames[[1]].

Formally classed aka “S4” objects typically have slotNames() (and no names()).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

slotNames, dimnames.

Examples

```
# print the names attribute of the islands data set
names(islands)

# remove the names attribute
names(islands) <- NULL
islands
rm(islands) # remove the copy made

z <- list(a = 1, b = "c", c = 1:3)
names(z)
# change just the name of the third element.
names(z)[3] <- "c2"
z

z <- 1:3
```

```
names(z)
## assign just one name
names(z)[2] <- "b"
z
```

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x

An **R** object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.

trim

the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

na.rm

a logical value indicating whether NA values should be stripped before the computation proceeds.

...

further arguments passed to or from other methods.

Value

If trim is zero (the default), the arithmetic mean of the values in x is computed, as a numeric or complex vector of length one. If x is not logical (coerced to numeric), numeric (including integer) or complex, NA_real_ is returned, with a warning.

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.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

weighted.mean, mean.POSIXct, colMeans for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Extremes {base}

Maxima and Minima

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)

pmax.int(..., na.rm = FALSE)
pmin.int(..., na.rm = FALSE)

Arguments

...
numeric or character arguments (see Note).

na.rm
a logical indicating whether missing values should be removed.

Details

max and min return the maximum or minimum of *all* the values present in their arguments, as integer if all are logical or integer, as double if all are numeric, and character otherwise.

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.

The minimum and maximum of a numeric empty set are +Inf and -Inf (in this order!) which ensures *transitivity*, e.g., min(x1, min(x2)) == min(x1, x2). For numeric x max(x) == -Inf and min(x) == +Inf whenever length(x) == 0 (after removing missing values if requested). However, pmax and pmin return NA if all the parallel elements are NA even for na.rm = TRUE.

pmax and pmin take one or more vectors (or matrices) as arguments and return a single vector giving the 'parallel' maxima (or minima) of the vectors. The first element of the result is the maximum (minimum) of the first elements of all the arguments, the second element of the result is the maximum (minimum) of the second elements of all the arguments and so on. Shorter inputs (of non-zero length) are recycled if necessary. Attributes (see attributes: such as names or dim) are copied from the first argument (if applicable).

pmax.int and pmin.int are faster internal versions only used when all arguments are atomic vectors and there are no classes: they drop all attributes. (Note that all versions fail for raw and complex vectors since these have no ordering.)

max and min are generic functions: methods can be defined for them individually or via the Summary group generic. For this to work properly, the arguments ... should be unnamed, and dispatch is on the first argument.

By definition the min/max of a numeric vector containing an NaN is NaN, except that the min/max of any vector containing an NA is NA even if it also contains an NaN. Note that max(NA, Inf) == NA even though the maximum would be Inf whatever the missing value actually is.

Character versions are sorted lexicographically, and this depends on the collating sequence of the locale in use: the help for 'Comparison' gives details. The max/min of an empty character vector is defined to be character NA. (One could argue that as "" is the smallest character element, the maximum should be "", but there is no obvious candidate for the minimum.)

Value

For min or max, a length-one vector. For pmin or pmax, a vector of length the longest of the input vectors, or length zero if one of the inputs had zero length.

The type of the result will be that of the highest of the inputs in the hierarchy integer < double < character.

For min and max if there are only numeric inputs and all are empty (after possible removal of NAs), the result is double (Inf or -Inf).

S4 methods

max and min are part of the S4 Summary group generic. Methods for them must use the signature x, ..., na.rm.

Note

'Numeric' arguments are vectors of type integer and numeric, and logical (coerced to integer). For historical reasons, NULL is accepted as equivalent to integer(0).

pmax and pmin will also work on classed objects with appropriate methods for comparison, is.na and rep (if recycling of arguments is needed).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

range (*both* min and max) and which.min (which.max) for the *arg min*, i.e., the location where an extreme value occurs.

'plotmath' for the use of min in plot annotation.

Examples

```
require(stats); require(graphics)
 min(5:1, pi) #-> one number
pmin(5:1, pi) #->  5  numbers

x <- sort(rnorm(100));  cH <- 1.35
pmin(cH, quantile(x)) # no names
pmin(quantile(x), cH) # has names
plot(x, pmin(cH, pmax(-cH, x)), type = "b", main =  "Huber's function")

cut01 <- function(x) pmax(pmin(x, 1), 0)
curve(      x^2 - 1/4, -1.4, 1.5, col = 2)
curve(cut01(x^2 - 1/4), col = "blue", add = TRUE, n = 500)
## pmax(), pmin() preserve attributes of *first* argument
D <- diag(x = (3:1)/4) ; n0 <- numeric()
stopifnot(identical(D,  cut01(D) ),
          identical(n0, cut01(n0)),
          identical(n0, cut01(NULL)),
          identical(n0, pmax(3:1, n0, 2)),
          identical(n0, pmax(n0, 4)))
```

range {base}

Range of Values

Description

range returns a vector containing the minimum and maximum of all the given arguments.

Usage

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

## Default S3 method:
range(..., na.rm = FALSE, finite = FALSE)
```

Arguments

...
any numeric or character objects.

na.rm
logical, indicating if NA's should be omitted.

finite
logical, indicating if all non-finite elements should be omitted.

Details

range is a generic function: methods can be defined for it directly or via the Summary group generic. For this to work properly, the arguments ... should be unnamed, and dispatch is on the first argument.

If na.rm is FALSE, NA and NaN values in any of the arguments will cause NA values to be returned, otherwise NA values are ignored.

If finite is TRUE, the minimum and maximum of all finite values is computed, i.e., finite = TRUE *includes* na.rm = TRUE.

A special situation occurs when there is no (after omission of NAs) nonempty argument left, see min.

S4 methods

This is part of the S4 Summary group generic. Methods for it must use the signature x, ..., na.rm.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also
min, max.
The extendrange() utility in package **grDevices**.
Examples
(r.x <- range(stats::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)

table {base}

Cross Tabulation and Table Creation

Description
table uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.
Usage
table(..., exclude = if (useNA == "no") c(NA, NaN), useNA = c("no",
  "ifany", "always"), dnn = list.names(...), deparse.level = 1)

as.table(x, ...)
is.table(x)

S3 method for class 'table'
as.data.frame(x, row.names = NULL, ...,
        responseName = "Freq", stringsAsFactors = TRUE,
        sep = "", base = list(LETTERS))
Arguments
...
one or more objects which can be interpreted as factors (including character strings), or a list (or data frame) whose components can be so interpreted. (For as.table, arguments passed to specific methods; for as.data.frame, unused.)
exclude
levels to remove for all factors in .... If set to NULL, it implies useNA = "always". See ‘Details’ for its interpretation for non-factor arguments.
useNA
whether to include NA values in the table. See ‘Details’.
dnn
the names to be given to the dimensions in the result (the *dimnames names*).
deparse.level
controls how the default dnn is constructed. See ‘Details’.
x
an arbitrary **R** object, or an object inheriting from class "table" for the as.data.frame method. Note that as.data.frame.table(x, *) may be called explicitly for non-table x for “reshaping” arrays.
row.names
a character vector giving the row names for the data frame.
responseName
The name to be used for the column of table entries, usually counts.
stringsAsFactors
logical: should the classifying factors be returned as factors (the default) or character vectors?

sep, base
passed to provideDimnames.

Details

If the argument dnn is not supplied, the internal function list.names is called to compute the 'dimname names'. If the arguments in ... are named, those names are used. For the remaining arguments, deparse.level = 0 gives an empty name, deparse.level = 1 uses the supplied argument if it is a symbol, and deparse.level = 2 will deparse the argument.

Only when exclude is specified and non-NULL (i.e., not by default), will table potentially drop levels of factor arguments.

useNA controls if the table includes counts of NA values: the allowed values correspond to never, only if the count is positive and even for zero counts. This is overridden by specifying exclude = NULL. Note that levels specified in exclude are mapped to NA and so included in NA counts.

Both exclude and useNA operate on an "all or none" basis. If you want to control the dimensions of a multiway table separately, modify each argument using factor or addNA.

It is best to supply factors rather than rely on coercion. In particular, exclude will be used in coercion to a factor, and so values (not levels) which appear in exclude before coercion will be mapped to NA rather than be discarded.

The summary method for class "table" (used for objects created by table or xtabs) which gives basic information and performs a chi-squared test for independence of factors (note that the function chisq.test currently only handles 2-d tables).

Value

table() returns a *contingency table*, an object of class "table", an array of integer values. Note that unlike S the result is always an array, a 1D array if one factor is given.

as.table and is.table coerce to and test for contingency table, respectively.

The as.data.frame method for objects inheriting from class "table" can be used to convert the array-based representation of a contingency table to a data frame containing the classifying factors and the corresponding entries (the latter as component named by responseName). This is the inverse of xtabs.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

tabulate is the underlying function and allows finer control.

Use ftable for printing (and more) of multidimensional tables. margin.table, prop.table, addmargins.

xtabs for cross tabulation of data frames with a formula interface.

Examples

```
require(stats) # for rpois and xtabs
## Simple frequency distribution
table(rpois(100, 5))
## Check the design:
with(warpbreaks, table(wool, tension))
table(state.division, state.region)

# simple two-way contingency table
with(airquality, table(cut(Temp, quantile(Temp)), Month))

a <- letters[1:3]
table(a, sample(a))                 # dnn is c("a", "")
table(a, sample(a), deparse.level = 0) # dnn is c("", "")
table(a, sample(a), deparse.level = 2) # dnn is c("a", "sample(a)")

## xtabs() <-> as.data.frame.table() :
UCBAdmissions ## already a contingency table
DF <- as.data.frame(UCBAdmissions)
class(tab <- xtabs(Freq ~ ., DF)) # xtabs & table
## tab *is* "the same" as the original table:
all(tab == UCBAdmissions)
all.equal(dimnames(tab), dimnames(UCBAdmissions))
```

```
a <- rep(c(NA, 1/0:3), 10)
table(a)
table(a, exclude = NULL)
b <- factor(rep(c("A","B","C"), 10))
table(b)
table(b, exclude = "B")
d <- factor(rep(c("A","B","C"), 10), levels = c("A","B","C","D","E"))
table(d, exclude = "B")
print(table(b, d), zero.print = ".")

## NA counting:
is.na(d) <- 3:4
d. <- addNA(d)
d.[1:7]
table(d.) # ", exclude = NULL" is not needed
## i.e., if you want to count the NA's of 'd', use
table(d, useNA = "ifany")

## Two-way tables with NA counts. The 3rd variant is absurd, but shows
## something that cannot be done using exclude or useNA.
with(airquality,
  table(OzHi = Ozone > 80, Month, useNA = "ifany"))
with(airquality,
  table(OzHi = Ozone > 80, Month, useNA = "always"))
with(airquality,
  table(OzHi = Ozone > 80, addNA(Month)))
```

row+colnames {base}

Row and Column Names

Description

Retrieve or set the row or column names of a matrix-like object.

Usage

```
rownames(x, do.NULL = TRUE, prefix = "row")
rownames(x) <- value

colnames(x, do.NULL = TRUE, prefix = "col")
colnames(x) <- value
```

Arguments

x
a matrix-like **R** object, with at least two dimensions for colnames.
do.NULL
logical. If FALSE and names are NULL, names are created.
prefix
for created names.
value
a valid value for that component of dimnames(x). For a matrix or array this is either NULL or a character vector of non-zero length equal to the appropriate dimension.

Details

The extractor functions try to do something sensible for any matrix-like object x. If the object has dimnames the first component is used as the row names, and the second component (if any) is used for the column names. For a data frame, rownames and colnames eventually call row.names and names respectively, but the latter are preferred.
If do.NULL is FALSE, a character vector (of length NROW(x) or NCOL(x)) is returned in any case, prepending prefix to simple numbers, if there are no dimnames or the corresponding component of the dimnames is NULL.
The replacement methods for arrays/matrices coerce vector and factor values of value to character, but do not dispatch methods for as.character.
For a data frame, value for rownames should be a character vector of non-duplicated and non-missing names (this is enforced), and for colnames a character vector of (preferably) unique syntactically-valid names. In both cases, value will be coerced by as.character, and setting colnames will convert the row names to character.

Note

If the replacement versions are called on a matrix without any existing dimnames, they will add suitable dimnames. But constructions such as

```
rownames(x)[3] <- "c"
```

may not work unless x already has dimnames, since this will create a length-3 value from the NULL value of rownames(x).

See Also

dimnames, case.names, variable.names.

Examples

```
m0 <- matrix(NA, 4, 0)
rownames(m0)

m2 <- cbind(1, 1:4)
colnames(m2, do.NULL = FALSE)
colnames(m2) <- c("x","Y")
rownames(m2) <- rownames(m2, do.NULL = FALSE, prefix = "Obs.")
m2
```

as.data.frame {base}

Coerce to a Data Frame

Description

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

Usage

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

## S3 method for class 'character'
as.data.frame(x, ...,
              stringsAsFactors = default.stringsAsFactors())

## S3 method for class 'matrix'
as.data.frame(x, row.names = NULL, optional = FALSE, ...,
              stringsAsFactors = default.stringsAsFactors())

is.data.frame(x)
```

Arguments

x
any R object.
row.names
NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional

logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional.

...

additional arguments to be passed to or from methods.

stringsAsFactors

logical: should the character vector be converted to a factor?

Details

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

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

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

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

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

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

Value

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

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

References

Chambers, J. M. (1992) *Data for models.* Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

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

vector {base}

Vectors

Description

vector produces a vector of the given length and mode.

as.vector, a generic, attempts to coerce its argument into a vector of mode mode (the default is to coerce to whichever vector mode is most convenient): if the result is atomic all attributes are removed.

is.vector returns TRUE if x is a vector of the specified mode having no attributes *other than names*. It returns FALSE otherwise.

Usage

```
vector(mode = "logical", length = 0)
as.vector(x, mode = "any")
is.vector(x, mode = "any")
```

Arguments

mode

character string naming an atomic mode or "list" or "expression" or (except for vector) "any".

length

a non-negative integer specifying the desired length. For a long vector, i.e., length > .Machine$integer.max, it has to be of type "double". Supplying an argument of length other than one is an error.

x

an R object.

Details

The atomic modes are "logical", "integer", "numeric" (synonym "double"), "complex", "character" and "raw".

If mode = "any", is.vector may return TRUE for the atomic modes, list and expression. For any mode, it will return FALSE if x has any attributes except names. (This is incompatible with S.) On the other hand, as.vector removes *all* attributes including names for results of atomic mode (but not those of mode "list" nor "expression").

Note that factors are *not* vectors; is.vector returns FALSE and as.vector converts a factor to a character vector for mode = "any".

Value

For vector, a vector of the given length and mode. Logical vector elements are initialized to FALSE, numeric vector elements to 0, character vector elements to "", raw vector elements to nul bytes and list/expression elements to NULL.

For as.vector, a vector (atomic or of type list or expression). All attributes are removed from the result if it is of an atomic mode, but not in general for a list result. The default method handles 24 input types and 12 values of type: the details of most coercions are undocumented and subject to change.

For is.vector, TRUE or FALSE. is.vector(x, mode = "numeric") can be true for vectors of types "integer" or "double" whereas is.vector(x, mode = "double") can only be true for those of type "double".

Methods for as.vector()

Writers of methods for as.vector need to take care to follow the conventions of the default method. In particular

- Argument mode can be "any", any of the atomic modes, "list", "expression", "symbol", "pairlist" or one of the aliases "double" and "name".
- The return value should be of the appropriate mode. For mode = "any" this means an atomic vector or list.
- Attributes should be treated appropriately: in particular when the result is an atomic vector there should be no attributes, not even names.
- is.vector(as.vector(x, m), m) should be true for any mode m, including the default "any".

Note

as.vector and is.vector are quite distinct from the meaning of the formal class "vector" in the **methods** package, and hence as(x, "vector") and is(x, "vector").

Note that as.vector(x) is not necessarily a null operation if is.vector(x) is true: any names will be removed from an atomic vector.

Non-vector modes "symbol" (synonym "name") and "pairlist" are accepted but have long been undocumented: they are used to implement as.name and as.pairlist, and those functions should preferably be used directly. None of the description here applies to those modes: see the help for the preferred forms.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

c, is.numeric, is.list, etc.

Examples

```
df <- data.frame(x = 1:3, y = 5:7)
## Error:
try(as.vector(data.frame(x = 1:3, y = 5:7), mode = "numeric"))

x <- c(a = 1, b = 2)
is.vector(x)
as.vector(x)
all.equal(x, as.vector(x)) ## FALSE


###-- All the following are TRUE:
is.list(df)
! is.vector(df)
! is.vector(df, mode = "list")

is.vector(list(), mode = “list")
```

factor {base}

Factors

Description

The function factor is used to encode a vector as a factor (the terms ‘category’ and ‘enumerated type’ are also used for factors). If argument 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 = character(), levels, labels = levels,
       exclude = NA, ordered = is.ordered(x), nmax = NA)

ordered(x, ...)

is.factor(x)
is.ordered(x)

as.factor(x)
as.ordered(x)

addNA(x, ifany = FALSE)
```

Arguments

x

a vector of data, usually taking a small number of distinct values.

levels

an optional vector of the values (as character strings) that x might have taken. The default is the unique set of values taken by as.character(x), sorted into increasing order *of x*. Note that this set can be specified as smaller than sort(unique(x)).

labels

either an optional character vector of labels for the levels (in the same order as levels after removing those in exclude), *or* a character string of length 1.

exclude

a vector of values to be excluded when forming the set of levels. This should be of the same type as x, and will be coerced if necessary.

ordered

logical flag to determine if the levels should be regarded as ordered (in the order given).

nmax

an upper bound on the number of levels; see ‘Details’.

...

(in ordered(.)): any of the above, apart from ordered itself.

ifany

(only add an NA level if it is used, i.e. if any(is.na(x)).

Details

The type of the vector x is not restricted; it only must have an as.character method and be sortable (by sort.list).

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 (which will happen for excluded values) then the i-th element of the result is set to NA.

Normally the ‘levels’ used as an attribute of the result are the reduced set of levels after removing those in exclude, but this can be altered by supplying labels. This should either be a set of new labels for the levels, or a character string, in which case the levels are that character string with a sequence number appended.

factor(x, exclude = NULL) applied to a factor is a no-operation unless there are unused levels: in that case, a factor with the reduced level set is returned. If exclude is used it should also be a factor with the same level set as x or a set of codes for the levels to be excluded.
The codes of a factor may contain NA. For a numeric x, set exclude = NULL to make NA an extra level (prints as <NA>); by default, this is the last level.
If NA is a level, the way to set a code to be missing (as opposed to the code of the missing level) is to use is.na on the left-hand-side of an assignment (as in is.na(f)[i] <- TRUE; indexing inside is.na does not work). Under those circumstances missing values are currently printed as <NA>, i.e., identical to entries of level NA.
is.factor is generic: you can write methods to handle specific classes of objects, see InternalMethods.
Where levels is not supplied, unique is called. Since factors typically have quite a small number of levels, for large vectors x it is helpful to supply nmax as an upper bound on the number of unique values.

Value

factor returns an object of class "factor" which has a set of integer codes the length of x with a "levels" attribute of mode character and unique (!anyDuplicated(.)) entries. If argument ordered is true (or ordered() is used) the result has class c("ordered", "factor").
Applying factor to an ordered or unordered factor returns a factor (of the same type) with just the levels which occur: see also [.factor for a more transparent way to achieve this.
is.factor returns TRUE or FALSE depending on whether its argument is of type factor or not. Correspondingly, is.ordered returns TRUE when its argument is an ordered factor 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.
addNA modifies a factor by turning NA into an extra level (so that NA values are counted in tables, for instance).

Warning

The interpretation of a factor depends on both the codes and the "levels" attribute. Be careful only to compare factors with the same set of levels (in the same order). In particular, as.numeric applied to a factor is meaningless, and may happen by implicit coercion. To transform a factor f to approximately its original numeric values, as.numeric(levels(f))[f] is recommended and slightly more efficient than as.numeric(as.character(f)).
The levels of a factor are by default sorted, but the sort order may well depend on the locale at the time of creation, and should not be assumed to be ASCII.
There are some anomalies associated with factors that have NA as a level. It is suggested to use them sparingly, e.g., only for tabulation purposes.

Comparison operators and group generic methods

There are "factor" and "ordered" methods for the group generic Ops which provide methods for the Comparison operators, and for the min, max, and range generics in Summary of "ordered". (The rest of the groups and the Math group generate an error as they are not meaningful for factors.)
Only == and != can be used for factors: a factor can only be compared to another factor with an identical set of levels (not necessarily in the same ordering) or to a character vector. Ordered factors are compared in the same way, but the general dispatch mechanism precludes comparing ordered and unordered factors.
All the comparison operators are available for ordered factors. Collation is done by the levels of the operands: if both operands are ordered factors they must have the same level set.

Note

In earlier versions of R, storing character data as a factor was more space efficient if there is even a small proportion of repeats. However, identical character strings now share storage, so the difference is small in most cases. (Integer values are stored in 4 bytes whereas each reference to a character string needs a pointer of 4 or 8 bytes.)

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

See Also

[.factor for subsetting of factors.
gl for construction of balanced factors and C for factors with specified contrasts. levels and nlevels for accessing the levels, and unclass to get integer codes.

Examples

```
(ff <- factor(substring("statistics", 1:10, 1:10), levels = letters))
as.integer(ff)      # the internal codes
(f. <- factor(ff))  # drops the levels that do not occur
ff[, drop = TRUE]   # the same, more transparently
```

```
factor(letters[1:20], labels = "letter")

class(ordered(4:1)) # "ordered", inheriting from "factor"
z <- factor(LETTERS[3:1], ordered = TRUE)
## and "relational" methods work:
stopifnot(sort(z)[c(1,3)] == range(z), min(z) < max(z))


## suppose you want "NA" as a level, and to allow missing values.
(x <- factor(c(1, 2, NA), exclude = NULL))
is.na(x)[2] <- TRUE
x  # [1] 1    <NA> <NA>
is.na(x)
# [1] FALSE  TRUE FALSE

## Using addNA()
Month <- airquality$Month
table(addNA(Month))
table(addNA(Month, ifany = TRUE))
```

numeric {base}

Numeric Vectors

Description

Creates or coerces objects of type "numeric". is.numeric is a more general test of an object being interpretable as numbers.

Usage

```
numeric(length = 0)
as.numeric(x, ...)
is.numeric(x)
```

Arguments

length

A non-negative integer specifying the desired length. Double values will be coerced to integer: supplying an argument of length other than one is an error.

x

object to be coerced or tested.

...

further arguments passed to or from other methods.

Details

numeric is identical to double (and real). It creates a double-precision vector of the specified length with each element equal to 0.

as.numeric is a generic function, but S3 methods must be written for as.double. It is identical to as.double.

is.numeric is an internal generic primitive function: you can write methods to handle specific classes of objects, see InternalMethods. It is **not** the same as is.double. Factors are handled by the default method, and there are methods for classes "Date", "POSIXt" and "difftime" (all of which return false). Methods for is.numeric should only return true if the base type of the class is double or integer *and* values can reasonably be regarded as numeric (e.g. arithmetic on them makes sense, and comparison should be done via the base type).

Value

for numeric and as.numeric see double.

The default method for is.numeric returns TRUE if its argument is of mode "numeric" (type "double" or type "integer") and not a factor, and FALSE otherwise. That is, is.integer(x) || is.double(x), or (mode(x) == "numeric") && !is.factor(x).

S4 methods

as.numeric and is.numeric are internally S4 generic and so methods can be set for them *via* setMethod.
To ensure that as.numeric and as.double remain identical, S4 methods can only be set for as.numeric.

Note on names

It is a historical anomaly that R has two names for its floating-point vectors, double and numeric (and formerly had real).
double is the name of the type. numeric is the name of the mode and also of the implicit class. As an S4 formal class, use "numeric".
The potential confusion is that R has used *mode* "numeric" to mean 'double or integer', which conflicts with the S4 usage. Thus is.numeric tests the mode, not the class, but as.numeric (which is identical to as.double) coerces to the class.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

double, integer, storage.mode.

Examples

```
as.numeric(c("-.1"," 2.7 ","B")) # (-0.1, 2.7, NA)  +  warning
as.numeric(factor(5:10))
```

summary {base}

Object Summaries

Description

summary is a generic function used to produce result summaries of the results of various model fitting functions.
The function invokes particular methods which depend on the class of the first argument.

Usage

```
summary(object, ...)

## Default S3 method:
summary(object, ..., digits = max(3, getOption("digits")-3))
## S3 method for class 'data.frame'
summary(object, maxsum = 7,
      digits = max(3, getOption("digits")-3), ...)

## S3 method for class 'factor'
summary(object, maxsum = 100, ...)

## S3 method for class 'matrix'
summary(object, ...)
```

Arguments

object
an object for which a summary is desired.
maxsum
integer, indicating how many levels should be shown for factors.
digits
integer, used for number formatting with signif() (for summary.default) or format() (for summary.data.frame).
...
additional arguments affecting the summary produced.

Details

For factors, the frequency of the first maxsum - 1 most frequent levels is shown, and the less frequent levels are summarized in "(Others)" (resulting in at most maxsum frequencies).
The functions summary.lm and summary.glm are examples of particular methods which summarize 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.
The default method returns an object of class c("summaryDefault", "table") which has a specialized print method.
The factor method returns an integer vector.
The matrix and data frame methods return a matrix of class "table", obtained by applying summary to each column and collating the results.

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

See Also

anova, summary.glm, summary.lm.

Examples

```
summary(attenu, digits = 4) #-> summary.data.frame(...), default precision
summary(attenu $ station, maxsum = 20) #-> summary.factor(...)

lst <- unclass(attenu$station) > 20 # logical with NAs
## summary.default() for logicals -- different from *.factor:
summary(lst)
summary(as.factor(lst))
```

rep {base}

Replicate Elements of Vectors and Lists

Description

rep replicates the values in x. It is a generic function, and the (internal) default method is described here.
rep.int and rep_len are faster simplified versions for two common cases. They are not generic.

Usage

```
rep(x, ...)

rep.int(x, times)

rep_len(x, length.out)
```

Arguments

x
a vector (of any mode including a list) or a factor or (for rep only) a POSIXct or POSIXlt or Date object; or an S4 object containing such an object.
...
further arguments to be passed to or from other methods. For the internal default method these can include:
times
A integer vector giving the (non-negative) number of times to repeat each element if of length length(x), or to repeat the whole vector if of length 1. Negative or NA values are an error.
length.out
non-negative integer. The desired length of the output vector. Other inputs will be coerced to an integer vector and the first element taken. Ignored if NA or invalid.
each
non-negative integer. Each element of x is repeated each times. Other inputs will be coerced to an integer vector and the first element taken. Treated as 1 if NA or invalid.
times
see ....
length.out
non-negative integer: the desired length of the output vector.

Details

The default behaviour is as if the call was
rep(x, times = 1, length.out = NA, each = 1)
. Normally just one of the additional arguments is specified, but if each is specified with either of the other two, its replication is performed first, and then that implied by times or length.out. If times consists of a single integer, the result consists of the whole input repeated this many times. If times is a vector of the same length as x (after replication by each), 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. If both are given, length.out takes priority and times is ignored. Non-integer values of times will be truncated towards zero. If times is a computed quantity it is prudent to add a small fuzz. If x has length zero and length.out is supplied and is positive, the values are filled in using the extraction rules, that is by an NA of the appropriate class for an atomic vector (0 for raw vectors) and NULL for a list.

Value

An object of the same type as x.

rep.int and rep_len return no attributes (except the class if returning a factor).

The default method of rep gives the result names (which will almost always contain duplicates) if x had names, but retains no other attributes.

Note

Function rep.int is a simple case which was provided as a separate function partly for S compatibility and partly for speed (especially when names can be dropped). The performance of rep has been improved since, but rep.int is still at least twice as fast when x has names.

The name rep.int long precedes making rep generic.

Function rep is a primitive, but (partial) matching of argument names is performed as for normal functions.

For historical reasons rep (only) works on NULL: the result is always NULL even when length.out is positive.

Although it has never been documented, these functions have always worked on expression vectors. **R** 2.x.y accepted pairlists and some other objects (although the results were rarely what their users intended).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

seq, sequence, replicate.

Examples

```
rep(1:4, 2)
rep(1:4, each = 2)       # not the same.
rep(1:4, c(2,2,2,2))     # same as second.
rep(1:4, c(2,1,2,1))
rep(1:4, each = 2, len = 4)    # first 4 only.
rep(1:4, each = 2, len = 10)   # 8 integers plus two recycled 1's.
rep(1:4, each = 2, times = 3)  # length 24, 3 complete replications

rep(1, 40*(1-.8)) # length 7 on most platforms
rep(1, 40*(1-.8)+1e-7) # better

## replicate a list
fred <- list(happy = 1:10, name = "squash")
rep(fred, 5)

# date-time objects
x <- .leap.seconds[1:3]
rep(x, 2)
rep(as.POSIXlt(x), rep(2, 3))

## named factor
x <- factor(LETTERS[1:4]); names(x) <- letters[1:4]
x
rep(x, 2)
rep(x, each = 2)
```

```
rep.int(x, 2)  # no names
rep_len(x, 10)
```

sum {base}

Sum of Vector Elements

Description

sum returns the sum of all the values present in its arguments.

Usage

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

Arguments

...
numeric or complex or logical vectors.

na.rm
logical. Should missing values (including NaN) be removed?

Details

This is a generic function: methods can be defined for it directly or via the Summary group generic. For this to work properly, the arguments ... should be unnamed, and dispatch is on the first argument.

If na.rm is FALSE an NA or NaN value in any of the arguments will cause a value of NA or NaN to be returned, otherwise NA and NaN values are ignored.

Logical true values are regarded as one, false values as zero. For historical reasons, NULL is accepted and treated as if it were integer(0).

Loss of accuracy can occur when summing values of different signs: this can even occur for sufficiently long integer inputs if the partial sums would cause integer overflow. Where possible extended-precision accumulators are used, but this is platform-dependent.

Value

The sum. If all of ... are of type integer or logical, then the sum is integer, and in that case the result will be NA (with a warning) if integer overflow occurs. Otherwise it is a length-one numeric or complex vector.

NB: the sum of an empty set is zero, by definition.

S4 methods

This is part of the S4 Summary group generic. Methods for it must use the signature x, ..., na.rm.

'plotmath' for the use of sum in plot annotation.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

colSums for row and column sums.

read.table {utils}

Data Input

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"",
        dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
        row.names, col.names, as.is = !stringsAsFactors,
        na.strings = "NA", colClasses = NA, nrows = -1,
        skip = 0, check.names = TRUE, fill = !blank.lines.skip,
```

```
        strip.white = FALSE, blank.lines.skip = TRUE,
        comment.char = "#",
        allowEscapes = FALSE, flush = FALSE,
        stringsAsFactors = default.stringsAsFactors(),
        fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

read.csv(file, header = TRUE, sep = ",", quote = "\"",
      dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
       dec = ",", fill = TRUE, comment.char = "", ...)

read.delim(file, header = TRUE, sep = "\t", quote = "\"",
       dec = ".", fill = TRUE, comment.char = "", ...)

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
        dec = ",", fill = TRUE, comment.char = "", ...)
```

Arguments

file

the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an *absolute* path, the file name is *relative* to the current working directory, getwd(). Tilde-expansion is performed where supported. This can be a compressed file (see file).

Alternatively, file can be a readable text-mode connection (which will be opened for reading if necessary, and if so closed (and hence destroyed) at the end of the function call). (If stdin() is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, Ctrl-D on Unix and Ctrl-Z on Windows. Any pushback on stdin() will be cleared before return.)

file can also be a complete URL. (For the supported URL schemes, see the 'URLs' section of the help for url.)

header

a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if and only if the first row contains one fewer field than the number of columns.

sep

the field separator character. Values on each line of the file are separated by this character. If sep = "" (the default for read.table) the separator is 'white space', that is one or more spaces, tabs, newlines or carriage returns.

quote

the set of quoting characters. To disable quoting altogether, use quote = "". See scan for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless colClasses is specified.

dec

the character used in the file for decimal points.

numerals

string indicating how to convert numbers whose conversion to double precision would lose accuracy, see type.convert.

row.names

a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names.

If there is a header and the first row contains one fewer field than the number of columns, the first column in the input is used for the row names. Otherwise if row.names is missing, the rows are numbered.

Using row.names = NULL forces row numbering. Missing or NULL row.names generate row names that are considered to be 'automatic' (and not preserved by as.matrix).

col.names

a vector of optional names for the variables. The default is to use "V" followed by the column number.

as.is

the default behavior of read.table is to convert character variables (which are not converted to logical, numeric or complex) to factors. The variable as.is controls the conversion of columns not otherwise specified by colClasses. Its value is either a vector of logicals (values are recycled if necessary), or a vector of numeric or character indices which specify which columns should not be converted to factors.
Note: to suppress all conversions including those of numeric columns, set colClasses = "character".
Note that as.is is specified per column (not per variable) and so includes the column of row names (if any) and any columns to be skipped.
na.strings
a character vector of strings which are to be interpreted as NA values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields.
colClasses
character. A vector of classes to be assumed for the columns. Recycled as necessary, or if the character vector is named, unspecified values are taken to be NA.
Possible values are NA (the default, when type.convert is used), "NULL" (when the column is skipped), one of the atomic vector classes (logical, integer, numeric, complex, character, raw), or "factor", "Date" or "POSIXct". Otherwise there needs to be an as method (from package **methods**) for conversion from "character" to the specified formal class.
Note that colClasses is specified per column (not per variable) and so includes the column of row names (if any).
nrows
integer: the maximum number of rows to read in. Negative and other invalid values are ignored.
skip
integer: the number of lines of the data file to skip before beginning to read data.
check.names
logical. If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.
fill
logical. If TRUE then in case the rows have unequal length, blank fields are implicitly added. See ‘Details’.
strip.white
logical. Used only when sep has been specified, and allows the stripping of leading and trailing white space from unquoted character fields (numeric fields are always stripped). See scan for further details (including the exact meaning of ‘white space’), remembering that the columns may include the row names.
blank.lines.skip
logical: if TRUE blank lines in the input are ignored.
comment.char
character: a character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.
allowEscapes
logical. Should C-style escapes such as \n be processed or read verbatim (the default)? Note that if not within quotes these could be interpreted as a delimiter (but not as a comment character). For more details see scan.
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.
stringsAsFactors
logical: should character vectors be converted to factors? Note that this is overridden by as.is and colClasses, both of which allow finer control.
fileEncoding
character string: if non-empty declares the encoding used on a file (not a connection) so the character data can be re-encoded. See the ‘Encoding’ section of the help for file, the ‘R Data Import/Export Manual’ and ‘Note’.
encoding
encoding to be assumed for input strings. It is used to mark character strings as known to be in Latin-1 or UTF-8 (see Encoding): it is not used to re-encode the input, but allows **R** to handle encoded strings in their native encoding (if one of those two). See ‘Value’ and ‘Note’.
text

character string: if file is not supplied and this is, then data are read from the value of text via a text connection. Notice that a literal string can be used to include (small) data sets within R code.
skipNul
logical: should nuls be skipped?
...
Further arguments to be passed to read.table.

Details

This function is the principal means of reading tabular data into R.
Unless colClasses is specified, all columns are read as character columns and then converted using type.convert to logical, integer, numeric, complex or (depending on as.is) factor as appropriate. Quotes are (by default) interpreted in all fields, so a column of values like "42" will result in an integer column.
A field or line is 'blank' if it contains nothing (except whitespace if no separator is specified) before a comment character or the end of the field or line.
If row.names is not specified and the header line has one less entry than the number of columns, the first column is taken to be the row names. This allows data frames to be read in from the format in which they are printed. If row.names is specified and does not refer to the first column, that column is discarded from such files.
The number of data columns is determined by looking at the first five lines of input (or the whole file if it has less than five lines), or from the length of col.names if it is specified and is longer. This could conceivably be wrong if fill or blank.lines.skip are true, so specify col.names if necessary (as in the 'Examples').
read.csv and read.csv2 are identical to read.table except for the defaults. They are intended for reading 'comma separated value' files ('.csv') or (read.csv2) the variant used in countries that use a comma as decimal point and a semicolon as field separator. Similarly, read.delim and read.delim2 are for reading delimited files, defaulting to the TAB character for the delimiter. Notice that header = TRUE and fill = TRUE in these variants, and that the comment character is disabled.
The rest of the line after a comment character is skipped; quotes are not processed in comments. Complete comment lines are allowed provided blank.lines.skip = TRUE; however, comment lines prior to the header must have the comment character in the first non-blank column.
Quoted fields with embedded newlines are supported except after a comment character. Embedded nuls are unsupported: skipping them (with skipNul = TRUE) may work.

Value

A data frame (data.frame) containing a representation of the data in the file.
Empty input is an error unless col.names is specified, when a 0-row data frame is returned: similarly giving just a header line if header = TRUE results in a 0-row data frame. Note that in either case the columns will be logical unless colClasses was supplied.
Character strings in the result (including factor levels) will have a declared encoding if encoding is "latin1" or "UTF-8".

Memory usage

These functions can use a surprising amount of memory when reading large files. There is extensive discussion in the 'R Data Import/Export' manual, supplementing the notes here.
Less memory will be used if colClasses is specified as one of the six atomic vector classes. This can be particularly so when reading a column that takes many distinct numeric values, as storing each distinct value as a character string can take up to 14 times as much memory as storing it as an integer.
Using nrows, even as a mild over-estimate, will help memory usage.
Using comment.char = "" will be appreciably faster than the read.table default.
read.table is not the right tool for reading large matrices, especially those with many columns: it is designed to read *data frames* which may have columns of very different classes. Use scan instead for matrices.

Note

The columns referred to in as.is and colClasses include the column of row names (if any).
There are two approaches for reading input that is not in the local encoding. If the input is known to be UTF-8 or Latin1, use the encoding argument to declare that. If the input is in some other encoding, then it may be translated on input. The fileEncoding argument achieves this by setting up a connection to do the re-encoding into the current locale. Note that on Windows or other systems not running in a UTF-8 locale, this may not be possible.

References

Chambers, J. M. (1992) *Data for models.* Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

The 'R Data Import/Export' manual.
scan, type.convert, read.fwf for reading *f*ixed *w*idth *f*ormatted input; write.table; data.frame.
count.fields can be useful to determine problems with reading files which result in reports of incorrect record lengths (see the 'Examples' below).
http://tools.ietf.org/html/rfc4180 for the IANA definition of CSV files (which requires comma as separator and CRLF line endings).

Examples

```
## using count.fields to handle unknown maximum number of fields
## when fill = TRUE
test1 <- c(1:5, "6,7", "8,9,10")
tf <- tempfile()
writeLines(test1, tf)

read.csv(tf, fill = TRUE) # 1 column
ncol <- max(count.fields(tf, sep = ","))
read.csv(tf, fill = TRUE, header = FALSE,
         col.names = paste0("V", seq_len(ncol)))
unlink(tf)

## "Inline" data set, using text=
## Notice that leading and trailing empty lines are auto-trimmed

read.table(header = TRUE, text = "
a b
1 2
3 4
")
```

stack {utils}

Stack or Unstack Vectors from a Data Frame or List

Description

Stacking vectors concatenates multiple vectors into a single vector along with a factor indicating where each observation originated. Unstacking reverses this operation.

Usage

```
stack(x, ...)
## Default S3 method:
stack(x, ...)
## S3 method for class 'data.frame'
stack(x, select, ...)

unstack(x, ...)
## Default S3 method:
unstack(x, form, ...)
## S3 method for class 'data.frame'
unstack(x, form, ...)
```

Arguments

x
a list or data frame to be stacked or unstacked.
select
an expression, indicating which variable(s) to select from a data frame.

form
a two-sided formula whose left side evaluates to the vector to be unstacked and whose right side evaluates to the indicator of the groups to create. Defaults to formula(x) in the data frame method for unstack.
...
further arguments passed to or from other methods.

Details

The stack function is used to transform data available as separate columns in a data frame or list into a single column that can be used in an analysis of variance model or other linear model. The unstack function reverses this operation.
Note that stack applies to *vectors* (as determined by is.vector): non-vector columns (e.g., factors) will be ignored (with a warning as from R 2.15.0). Where vectors of different types are selected they are concatenated by unlist whose help page explains how the type of the result is chosen.
These functions are generic: the supplied methods handle data frames and objects coercible to lists by as.list.

Value

unstack produces a list of columns according to the formula form. If all the columns have the same length, the resulting list is coerced to a data frame.
stack produces a data frame with two columns:
values
the result of concatenating the selected vectors in x.
ind
a factor indicating from which vector in x the observation originated.

Author(s)

Douglas Bates

See Also

lm, reshape

Examples

```
require(stats)
formula(PlantGrowth)        # check the default formula
pg <- unstack(PlantGrowth)   # unstack according to this formula
pg
stack(pg)                   # now put it back together
stack(pg, select = -ctrl)    # omitting one vector
```

Normal {stats}

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Arguments

x, q
vector of quantiles.
p
vector of probabilities.
n
number of observations. If length(n) > 1, the length is taken to be the number required.
mean
vector of means.

sd
vector of standard deviations.
log, log.p
logical; if TRUE, probabilities p are given as log(p).
lower.tail
logical; if TRUE (default), probabilities are $P[X \le x]$ 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

$$f(x) = 1/(\sqrt{(2\pi)}\sigma)\, e^{-((x-\mu)^2/(2\sigma^2))}$$

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.
The length of the result is determined by n for rnorm, and is the maximum of the lengths of the numerical arguments for the other functions.
The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.
For sd = 0 this gives the limit as sd decreases to 0, a point mass at mu. sd < 0 is an error and returns NaN.

Source

For pnorm, based on
Cody, W. D. (1993) Algorithm 715: SPECFUN – A portable FORTRAN package of special function routines and test drivers. *ACM Transactions on Mathematical Software* **19**, 22–32.
For qnorm, the code is a C translation of
Wichura, M. J. (1988) Algorithm AS 241: The percentage points of the normal distribution. *Applied Statistics*, **37**, 477–484.
which provides precise results up to about 16 digits.
For rnorm, see RNG for how to select the algorithm and for references to the supplied methods.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 13. Wiley, New York.

See Also

Distributions for other standard distributions, including dlnorm for the *Log*normal distribution.

Examples

```
require(graphics)

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.p = 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)
```

```
## if you want the so-called 'error function'
erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1
## (see Abramowitz and Stegun 29.2.29)
## and the so-called 'complementary error function'
erfc <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)
## and the inverses
erfinv <- function (x) qnorm((1 + x)/2)/sqrt(2)
erfcinv <- function (x) qnorm(x/2, lower = FALSE)/sqrt(2)
```

hist {graphics}

Histograms

Description

The generic function hist computes a histogram of the given data values. If plot = TRUE, the resulting object of class "histogram" is plotted by plot.histogram, before it is returned.

Usage

```
hist(x, ...)

## Default S3 method:
hist(x, breaks = "Sturges",
    freq = NULL, probability = !freq,
    include.lowest = TRUE, right = TRUE,
    density = NULL, angle = 45, col = NULL, border = NULL,
    main = paste("Histogram of" , xname),
    xlim = range(breaks), ylim = NULL,
    xlab = xname, ylab,
    axes = TRUE, plot = TRUE, labels = FALSE,
    nclass = NULL, warn.unused = TRUE, ...)
```

Arguments

x

a vector of values for which the histogram is desired.

breaks

one of:

- a vector giving the breakpoints between histogram cells,
- a function to compute the vector of breakpoints,
- a single number giving the number of cells for the histogram,
- a character string naming an algorithm to compute the number of cells (see ‘Details’),
- a function to compute the number of cells.

In the last three cases the number is a suggestion only; the breakpoints will be set to pretty values. If breaks is a function, the x vector is supplied to it as the only argument.

freq

logical; if TRUE, the histogram graphic is a representation of frequencies, the counts component of the result; if FALSE, probability densities, component density, are plotted (so that the histogram has a total area of one). Defaults to TRUE *if and only if* breaks are equidistant (and probability is not specified).

probability

an *alias* for !freq, for S compatibility.

include.lowest

logical; if TRUE, an x[i] equal to the breaks value will be included in the first (or last, for right = FALSE) bar. This will be ignored (with a warning) unless breaks is a vector.

right

logical; if TRUE, the histogram cells are right-closed (left open) intervals.
density
the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.
angle
the slope of shading lines, given as an angle in degrees (counter-clockwise).
col
a colour to be used to fill the bars. The default of NULL yields unfilled bars.
border
the color of the border around the bars. The default is to use the standard foreground color.
main, xlab, ylab
these arguments to title have useful defaults here.
xlim, ylim
the range of x and y values with sensible defaults. Note that xlim is *not* used to define the histogram (breaks), but only for plotting (when plot = TRUE).
axes
logical. If TRUE (default), axes are draw if the plot is drawn.
plot
logical. If TRUE (default), a histogram is plotted, otherwise a list of breaks and counts is returned. In the latter case, a warning is used if (typically graphical) arguments are specified that only apply to the plot = TRUE case.
labels
logical or character. Additionally draw labels on top of bars, if not FALSE; see plot.histogram.
nclass
numeric (integer). For S(-PLUS) compatibility only, nclass is equivalent to breaks for a scalar or character argument.
warn.unused
logical. If plot = FALSE and warn.unused = TRUE, a warning will be issued when graphical parameters are passed to hist.default().
...
further arguments and graphical parameters passed to plot.histogram and thence to title and axis (if plot = TRUE).

Details

The definition of *histogram* differs by source (with country-specific biases). **R**'s default with equi-spaced breaks (also the default) is to plot the counts in the cells defined by breaks. Thus the height of a rectangle is proportional to the number of points falling into the cell, as is the area *provided* the breaks are equally-spaced.
The default with non-equi-spaced breaks is to give a plot of area one, in which the *area* of the rectangles is the fraction of the data points falling in the cells.
If right = TRUE (default), the histogram cells are intervals of the form (a, b], i.e., they include their right-hand endpoint, but not their left one, with the exception of the first cell when include.lowest is TRUE.
For right = FALSE, the intervals are of the form [a, b), and include.lowest means '*include highest*'.
A numerical tolerance of *1e-7* times the median bin size is applied when counting entries on the edges of bins. This is not included in the reported breaks nor (as from **R** 2.11.0) in the calculation of density.
The default for breaks is "Sturges": see nclass.Sturges. Other names for which algorithms are supplied are "Scott" and "FD" / "Freedman-Diaconis" (with corresponding functions nclass.scott and nclass.FD). Case is ignored and partial matching is used. Alternatively, a function can be supplied which will compute the intended number of breaks or the actual breakpoints as a function of x.

Value

an object of class "histogram" which is a list with components:
breaks
the *n+1* cell boundaries (= breaks if that was a vector). These are the nominal breaks, not with the boundary fuzz.
counts
n
integers; for each cell, the number of x[] inside.
density
values *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; f^(x[i]) (b[i+1]-b[i])] = 1*, where *b[i]* = breaks[i].
mids

the *n* cell midpoints.
xname
a character string with the actual x argument name.
equidist
logical, indicating if the distances between breaks are all the same.
Prior to **R** 3.0.0 there was a component intensities, the same as density, for long-term back compatibility.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Venables, W. N. and Ripley. B. D. (2002) *Modern Applied Statistics with S*. Springer.

See Also

nclass.Sturges, stem, density, truehist in package **MASS**.
Typical plots with vertical bars are *not* histograms. Consider barplot or plot(*, type = "h") for such bar plots.

Examples

```
op <- par(mfrow = c(2, 2))
hist(islands)
utils::str(hist(islands, col = "gray", labels = TRUE))

hist(sqrt(islands), breaks = 12, col = "lightblue", border = "pink")
##-- For non-equidistant breaks, counts should NOT be graphed unscaled:
r <- hist(sqrt(islands), breaks = c(4*0:5, 10*3:5, 70, 100, 140),
        col = "blue1")
text(r$mids, r$density, r$counts, adj = c(.5, -.5), col = "blue3")
sapply(r[2:3], sum)
sum(r$density * diff(r$breaks)) # == 1
lines(r, lty = 3, border = "purple") # -> lines.histogram(*)
par(op)

require(utils) # for str
str(hist(islands, breaks = 12, plot =  FALSE)) #-> 10 (~= 12) breaks
str(hist(islands, breaks = c(12,20,36,80,200,1000,17000), plot = FALSE))

hist(islands, breaks = c(12,20,36,80,200,1000,17000), freq = TRUE,
    main = "WRONG histogram") # and warning

require(stats)
set.seed(14)
x <- rchisq(100, df = 4)

## Comparing data with a model distribution should be done with qqplot()!
qqplot(x, qchisq(ppoints(x), df = 4)); abline(0, 1, col = 2, lty = 2)

## if you really insist on using hist() ... :
hist(x, freq = FALSE, ylim = c(0, 0.2))
curve(dchisq(x, df = 4), col = 2, lty = 2, lwd = 2, add = TRUE)
```

curve {graphics}

Draw Function Plots

Description

Draws a curve corresponding to a function over the interval [from, to]. curve can plot also an expression in the variable xname, default x.

Usage

```
curve(expr, from = NULL, to = NULL, n = 101, add = FALSE,
      type = "l", xname = "x", xlab = xname, ylab = NULL,
      log = NULL, xlim = NULL, ...)

## S3 method for class 'function'
plot(x, y = 0, to = 1, from = y, xlim = NULL, ylab = NULL, ...)
```

Arguments

expr
The name of a function, or a call or an expression written as a function of x which will evaluate to an object of the same length as x.
x
a 'vectorizing' numeric **R** function.
y
alias for from for compatibility with plot
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 an already existing plot; if NA start a new plot taking the defaults for the limits and log-scaling of the x-axis from the previous plot. Taken as FALSE (with a warning if a different value is supplied) if no graphics device is open.
xlim
NULL or a numeric vector of length 2; if non-NULL it provides the defaults for c(from, to) and, unless add = TRUE, selects the x-limits of the plot – see plot.window.
type
plot type: see plot.default.
xname
character string giving the name to be used for the x axis.
xlab, ylab, log, ...
labels and graphical parameters can also be specified as arguments. See 'Details' for the interpretation of the default for log.
For the "function" method of plot, ... can include any of the other arguments of curve, except expr.

Details

The function or expression expr (for curve) or function x (for plot) is evaluated at n points equally spaced over the range [from, to]. The points determined in this way are then plotted.
If either from or to is NULL, it defaults to the corresponding element of xlim if that is not NULL.
What happens when neither from/to nor xlim specifies both x-limits is a complex story. For plot(<function>) and for curve(add = FALSE) the defaults are *(0, 1)*. For curve(add = NA) and curve(add = TRUE) the defaults are taken from the x-limits used for the previous plot. (This differs from versions of **R** prior to 2.14.0.)
The value of log is used both to specify the plot axes (unless add = TRUE) and how 'equally spaced' is interpreted: if the x component indicates log-scaling, the points at which the expression or function is plotted are equally spaced on log scale.
The default value of log is taken from the current plot when add = TRUE, whereas if add = NA the x component is taken from the existing plot (if any) and the y component defaults to linear. For add = FALSE the default is ""
This used to be a quick hack which now seems to serve a useful purpose, but can give bad results for functions which are not smooth.
For expensive-to-compute expressions, you should use smarter tools.
The way curve handles expr has caused confusion. It first looks to see if expr is a name (also known as a symbol), in which case it is taken to be the name of a function, and expr is replaced by a call to expr with a single argument with name given by xname. Otherwise it checks that expr is either a call or an expression, and that it contains a reference to the variable given by xname (using all.vars): anything else is an error. Then expr is evaluated in an environment which supplies a vector of name given by xname of length n, and should evaluate to an object of length n. Note that

this means that curve(x, ...) is taken as a request to plot a function named x (and it is used as such in the function method for plot).
The plot method can be called directly as plot.function.

Value

A list with components x and y of the points that were drawn is returned invisibly.

Warning

For historical reasons, add is allowed as an argument to the "function" method of plot, but its behaviour may surprise you. It is recommended to use add only with curve.

See Also

splinefun for spline interpolation, lines.

Examples

```
plot(qnorm) # default range c(0, 1) is appropriate here,
          # but end values are -/+Inf and so are omitted.
plot(qlogis, main = "The Inverse Logit : qlogis()")
abline(h = 0, v = 0:2/2, lty = 3, col = "gray")

curve(sin, -2*pi, 2*pi, xname = "t")
curve(tan, xname = "t", add = NA,
      main = "curve(tan)  --> same x-scale as previous plot")

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

## simple and advanced versions, quite similar:
plot(cos, -pi,  3*pi)
curve(cos, xlim = c(-pi, 3*pi), n = 1001, col = "blue", add = TRUE)

chippy <- function(x) sin(cos(x)*exp(-x/2))
curve(chippy, -8, 7, n = 2001)
plot (chippy, -8, -5)

for(ll in c("", "x", "y", "xy"))
  curve(log(1+x), 1, 100, log = ll, sub = paste0("log = '", ll, "'"))
par(op)
```

barplot {graphics}

Bar Plots

Description

Creates a bar plot with vertical or horizontal bars.

Usage

```
barplot(height, ...)

## Default S3 method:
barplot(height, width = 1, space = NULL,
        names.arg = NULL, legend.text = NULL, beside = FALSE,
        horiz = FALSE, density = NULL, angle = 45,
        col = NULL, border = par("fg"),
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        xlim = NULL, ylim = NULL, xpd = TRUE, log = "",
```

axes = TRUE, axisnames = TRUE,
cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
add = FALSE, args.legend = NULL, ...)

Arguments

height
either a vector or matrix of values describing the bars which make up the plot. If height is a vector, the plot consists of a sequence of rectangular bars with heights given by the values in the vector. If height is a matrix and beside is FALSE then each bar of the plot corresponds to a column of height, with the values in the column giving the heights of stacked sub-bars making up the bar. If height is a matrix and beside is TRUE, then the values in each column are juxtaposed rather than stacked.

width
optional vector of bar widths. Re-cycled to length the number of bars drawn. Specifying a single value will have no visible effect unless xlim is specified.

space
the amount of space (as a fraction of the average bar width) left before each bar. May be given as a single number or one number per bar. If height is a matrix and beside is TRUE, space may be specified by two numbers, where the first is the space between bars in the same group, and the second the space between the groups. If not given explicitly, it defaults to c(0,1) if height is a matrix and beside is TRUE, and to 0.2 otherwise.

names.arg
a vector of names to be plotted below each bar or group of bars. If this argument is omitted, then the names are taken from the names attribute of height if this is a vector, or the column names if it is a matrix.

legend.text
a vector of text used to construct a legend for the plot, or a logical indicating whether a legend should be included. This is only useful when height is a matrix. In that case given legend labels should correspond to the rows of height; if legend.text is true, the row names of height will be used as labels if they are non-null.

beside
a logical value. If FALSE, the columns of height are portrayed as stacked bars, and if TRUE the columns are portrayed as juxtaposed bars.

horiz
a logical value. If FALSE, the bars are drawn vertically with the first bar to the left. If TRUE, the bars are drawn horizontally with the first at the bottom.

density
a vector giving the density of shading lines, in lines per inch, for the bars or bar components. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.

angle
the slope of shading lines, given as an angle in degrees (counter-clockwise), for the bars or bar components.

col
a vector of colors for the bars or bar components. By default, grey is used if height is a vector, and a gamma-corrected grey palette if height is a matrix.

border
the color to be used for the border of the bars. Use border = NA to omit borders. If there are shading lines, border = TRUE means use the same colour for the border as for the shading lines.

main,sub
overall and sub title for the plot.

xlab
a label for the x axis.

ylab
a label for the y axis.

xlim
limits for the x axis.

ylim
limits for the y axis.

xpd

logical. Should bars be allowed to go outside region?
log
string specifying if axis scales should be logarithmic; see plot.default.
axes
logical. If TRUE, a vertical (or horizontal, if horiz is true) axis is drawn.
axisnames
logical. If TRUE, and if there are names.arg (see above), the other axis is drawn (with lty = 0) and labeled.
cex.axis
expansion factor for numeric axis labels.
cex.names
expansion factor for axis names (bar labels).
inside
logical. If TRUE, the lines which divide adjacent (non-stacked!) bars will be drawn. Only applies when space = 0 (which it partly is when beside = TRUE).
plot
logical. If FALSE, nothing is plotted.
axis.lty
the graphics parameter lty applied to the axis and tick marks of the categorical (default horizontal) axis. Note that by default the axis is suppressed.
offset
a vector indicating how much the bars should be shifted relative to the x axis.
add
logical specifying if bars should be added to an already existing plot; defaults to FALSE.
args.legend
list of additional arguments to pass to legend(); names of the list are used as argument names. Only used if legend.text is supplied.
...
arguments to be passed to/from other methods. For the default method these can include further arguments (such as axes, asp and main) and graphical parameters (see par) which are passed to plot.window(), title() and axis.

Details

This is a generic function, it currently only has a default method. A formula interface may be added eventually.

Value

A numeric vector (or matrix, when beside = TRUE), say mp, giving the coordinates of *all* the bar midpoints drawn, useful for adding to the graph.
If beside is true, use colMeans(mp) for the midpoints of each *group* of bars, see example.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

plot(..., type = "h"), dotchart, hist.

Examples

```
require(grDevices) # for colours
tN <- table(Ni <- stats::rpois(100, lambda = 5))
r <- barplot(tN, col = rainbow(20))
#- type = "h" plotting *is* 'bar'plot
lines(r, tN, type = "h", col = "red", lwd = 2)

barplot(tN, space = 1.5, axisnames = FALSE,
        sub = "barplot(..., space= 1.5, axisnames = FALSE)")

barplot(VADeaths, plot = FALSE)
barplot(VADeaths, plot = FALSE, beside = TRUE)

mp <- barplot(VADeaths) # default
```

```
tot <- colMeans(VADeaths)
text(mp, tot + 3, format(tot), xpd = TRUE, col = "blue")
barplot(VADeaths, beside = TRUE,
        col = c("lightblue", "mistyrose", "lightcyan",
                "lavender", "cornsilk"),
        legend = rownames(VADeaths), ylim = c(0, 100))
title(main = "Death Rates in Virginia", font.main = 4)

hh <- t(VADeaths)[, 5:1]
mybarcol <- "gray20"
mp <- barplot(hh, beside = TRUE,
        col = c("lightblue", "mistyrose",
                "lightcyan", "lavender"),
        legend = colnames(VADeaths), ylim = c(0,100),
        main = "Death Rates in Virginia", font.main = 4,
        sub = "Faked upper 2*sigma error bars", col.sub = mybarcol,
        cex.names = 1.5)
segments(mp, hh, mp, hh + 2*sqrt(1000*hh/100), col = mybarcol, lwd = 1.5)
stopifnot(dim(mp) == dim(hh))  # corresponding matrices
mtext(side = 1, at = colMeans(mp), line = -2,
      text = paste("Mean", formatC(colMeans(hh))), col = "red")

# Bar shading example
barplot(VADeaths, angle = 15+10*1:5, density = 20, col = "black",
        legend = rownames(VADeaths))
title(main = list("Death Rates in Virginia", font = 4))

# border :
barplot(VADeaths, border = "dark blue")


# log scales (not much sense here):
barplot(tN, col = heat.colors(12), log = "y")
barplot(tN, col = gray.colors(20), log = "xy")

# args.legend
barplot(height = cbind(x = c(465, 91) / 465 * 100,
                       y = c(840, 200) / 840 * 100,
                       z = c(37, 17) / 37 * 100),
        beside = FALSE,
        width = c(465, 840, 37),
        col = c(1, 2),
        legend.text = c("A", "B"),
        args.legend = list(x = “topleft"))
```

plot {graphics}

Generic X-Y Plotting

Description

Generic function for plotting of **R** objects. For more details about the graphical parameter arguments, see par.

For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.

Usage

plot(x, y, ...)

Arguments

x

the coordinates of points in the plot. Alternatively, a single plotting structure, function 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.

...

Arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

type

what type of plot should be drawn. Possible types are

- "p" for **p**oints,
- "l" for **l**ines,
- "b" for **b**oth,
- "c" for the lines part alone of "b",
- "o" for both ‘**o**verplotted’,
- "h" for ‘**h**istogram’ like (or ‘high-density’) vertical lines,
- "s" for stair **s**teps,
- "S" for other **s**teps, see ‘Details’ below,
- "n" for no plotting.

All other types give a warning or an error; using, e.g., type = "punkte" being equivalent to type = "p" for S compatibility. Note that some methods, e.g. plot.factor, do not accept this.

main

an overall title for the plot: see title.

sub

a sub title for the plot: see title.

xlab

a title for the x axis: see title.

ylab

a title for the y axis: see title.

asp

the *y/x* aspect ratio, see plot.window.

Details

The two step types differ in their x-y preference: Going from *(x1,y1)* to *(x2,y2)* with $x1 < x2$, type = "s" moves first horizontal, then vertical, whereas type = "S" moves the other way around.

See Also

plot.default, plot.formula and other methods; points, lines, par.

For X-Y-Z plotting see contour, persp and image.

Examples

```
require(stats)
plot(cars)
lines(lowess(cars))

plot(sin, -pi, 2*pi) # see ?plot.function

## Discrete Distribution Plot:
plot(table(rpois(100, 5)), type = "h", col = "red", lwd = 10,
    main = "rpois(100, lambda = 5)")

## Simple quantiles/ECDF, see ecdf() {library(stats)} for a better one:
```

```
plot(x <- sort(rnorm(47)), type = "s", main = "plot(x, type = \"s\")")
points(x, cex = .5, col = "dark red")
```

lines {graphics}

Add Connected Line Segments to a Plot

Description

A generic function taking coordinates given in various ways and joining the corresponding points with line segments.

Usage

```
lines(x, ...)

## Default S3 method:
lines(x, y = NULL, type = "l", ...)
```

Arguments

x, y
coordinate vectors of points to join.

type
character indicating the type of plotting; actually any of the types as in plot.default.

...
Further graphical parameters (see par) may also be supplied as arguments, particularly, line type, lty, line width, lwd, color, col and for type = "b", pch. Also the line characteristics lend, ljoin and lmitre.

Details

The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, a time series, ... See xy.coords. If supplied separately, they must be of the same length.

The coordinates can contain NA values. If a point contains NA in either its x or y value, it is omitted from the plot, and lines are not drawn to or from such points. Thus missing values can be used to achieve breaks in lines.

For type = "h", col can be a vector and will be recycled as needed.

lwd can be a vector: its first element will apply to lines but the whole vector to symbols (recycled as necessary).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

lines.formula for the formula method; points, particularly for type %in% c("p","b","o"), plot, and the workhorse function plot.xy.

abline for drawing (single) straight lines.

par for line type (lty) specification and how to specify colors.

Examples

```
# draw a smooth line through a scatter plot
plot(cars, main = "Stopping Distance versus Speed")
lines(stats::lowess(cars))
```

points {graphics}

Add Points to a Plot

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, ...)
```

Default S3 method:
points(x, y = NULL, type = "p", ...)

Arguments

x, y
coordinate vectors of points to plot.

type
character indicating the type of plotting; actually any of the types as in plot.default.

...
Further graphical parameters may also be supplied as arguments. See 'Details'.

Details

The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, a time series, ... See xy.coords. If supplied separately, they must be of the same length.

Graphical parameters commonly used are

pch
plotting 'character', i.e., symbol to use. This can either be a single character or an integer code for one of a set of graphics symbols. The full set of S symbols is available with pch = 0:18, see the examples below. (NB: R uses circles instead of the octagons used in S.)
Value pch = "." (equivalently pch = 46) is handled specially. It is a rectangle of side 0.01 inch (scaled by cex). In addition, if cex = 1 (the default), each side is at least one pixel (1/72 inch on the pdf, postscript and xfig devices).
For other text symbols, cex = 1 corresponds to the default fontsize of the device, often specified by an argument pointsize. For pch in 0:25 the default size is about 75% of the character height (see par("cin")).

col
color code or name, see par.

bg
background (fill) color for the open plot symbols given by pch = 21:25.

cex
character (or symbol) expansion: a numerical vector. This works as a multiple of par("cex").

lwd
line width for drawing symbols see par.

Others less commonly used are lty and lwd for types such as "b" and "l".

The graphical parameters pch, col, bg, cex and lwd can be vectors (which will be recycled as needed) giving a value for each point plotted. If lines are to be plotted (e.g. for type = "b") the first element of lwd is used.

Points whose x, y, pch, col or cex value is NA are omitted from the plot.

'pch' values

Values of pch are stored internally as integers. The interpretation is

- NA_integer_: no symbol.
- 0:18: S-compatible vector symbols.
- 19:25: further R vector symbols.
- 26:31: unused (and ignored).
- 32:127: ASCII characters.
- 128:255 native characters *only in a single-byte locale and for the symbol font*. (128:159 are only used on Windows.)
- -32 ... Unicode code point (where supported).

Note that unlike S (which uses octagons), symbols 1, 10, 13 and 16 use circles. The filled shapes 15:18 do not include a border.

The following R plotting symbols are can be obtained with pch = 19:25: those with 21:25 can be colored and filled with different colors: col gives the border color and bg the background color (which is "grey" in the figure)

- pch = 19: solid circle,
- pch = 20: bullet (smaller solid circle, 2/3 the size of 19),

- pch = 21: filled circle,
- pch = 22: filled square,
- pch = 23: filled diamond,
- pch = 24: filled triangle point-up,
- pch = 25: filled triangle point down.

Note that all of these both fill the shape and draw a border. Some care in interpretation is needed when semi-transparent colours are used for both fill and border (and the result might be device-specific and even viewer-specific for pdf).
The difference between pch = 16 and pch = 19 is that the latter uses a border and so is perceptibly larger when lwd is large relative to cex.
Values pch = 26:31 are currently unused and pch = 32:127 give the ASCII characters. In a single-byte locale pch = 128:255 give the corresponding character (if any) in the locale's character set. Where supported by the OS, negative values specify a Unicode code point, so e.g. -0x2642L is a 'male sign' and -0x20ACL is the Euro.
A character string consisting of a single character is converted to an integer: 32:127 for ASCII characters, and usually to the Unicode code point otherwise. (In non-Latin-1 single-byte locales, 128:255 will be used for 8-bit characters.)
If pch supplied is a logical, integer or character NA or an empty character string the point is omitted from the plot.
If pch is NULL or otherwise of length 0, par("pch") is used.
If the symbol font (par(font = 5)) is used, numerical values should be used for pch: the range is c(32:126, 160:254) in all locales (but 240 is not defined (used for 'apple' on OS X) and 160, Euro, may not be present).

Note

A single-byte encoding may include the characters in pch = 128:255, and if it does, a font may not include all (or even any) of them.
Not all negative numbers are valid as Unicode code points, and no check is done. A display device is likely to use a rectangle for (or omit) Unicode code points which are invalid or for which it does not have a glyph in the font used.
What happens for very small or zero values of cex is device-dependent: symbols or characters may become invisible or they may be plotted at a fixed minimum size. As from **R** 2.15.0, circles of zero radius will not be plotted.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

points.formula for the formula method; plot, lines, and the underlying workhorse function plot.xy.

Examples

```
require(stats) # for rnorm
plot(-4:4, -4:4, type = "n")  # setting up coord. system
points(rnorm(200), rnorm(200), col = "red")
points(rnorm(100)/2, rnorm(100)/2, col = "blue", cex = 1.5)

op <- par(bg = "light blue")
x <- seq(0, 2*pi, len = 51)
## something "between type='b' and type='o'":
plot(x, sin(x), type = "o", pch = 21, bg = par("bg"), col = "blue", cex = .6,
 main = 'plot(..., type="o", pch=21, bg=par("bg"))')
par(op)

## Not run:
## The figure was produced by calls like
png("pch.png", height = 0.7, width = 7, res = 100, units = "in")
par(mar = rep(0,4))
plot(c(-1, 26), 0:1, type = "n", axes = FALSE)
text(0:25, 0.6, 0:25, cex = 0.5)
points(0:25, rep(0.3, 26), pch = 0:25, bg = "grey")

## End(Not run)

##-------- Showing all the extra & some char graphics symbols ---------
```

```
pchShow <-
  function(extras = c("*",".", "o","O","0","+","-","|","%","#"),
           cex = 3, ## good for both .Device=="postscript" and "x11"
           col = "red3", bg = "gold", coltext = "brown", cextext = 1.2,
           main = paste("plot symbols :  points (...  pch = *, cex =",
                        cex,")"))
  {
    nex <- length(extras)
    np  <- 26 + nex
    ipch <- 0:(np-1)
    k <- floor(sqrt(np))
    dd <- c(-1,1)/2
    rx <- dd + range(ix <- ipch %/% k)
    ry <- dd + range(iy <- 3 + (k-1)- ipch %% k)
    pch <- as.list(ipch) # list with integers & strings
    if(nex > 0) pch[26+ 1:nex] <- as.list(extras)
    plot(rx, ry, type = "n", axes  =  FALSE, xlab = "", ylab = "", main = main)
    abline(v = ix, h = iy, col = "lightgray", lty = "dotted")
    for(i in 1:np) {
      pc <- pch[[i]]
      ## 'col' symbols with a 'bg'-colored interior (where available) :
      points(ix[i], iy[i], pch = pc, col = col, bg = bg, cex = cex)
      if(cextext > 0)
         text(ix[i] - 0.3, iy[i], pc, col = coltext, cex = cextext)
    }
  }

pchShow()
pchShow(c("o","O","0"), cex = 2.5)
pchShow(NULL, cex = 4, cextext = 0, main = NULL)


## ------------ test code for various pch specifications -------------
# Try this in various font families (including Hershey)
# and locales.  Use sign = -1 asserts we want Latin-1.
# Standard cases in a MBCS locale will not plot the top half.
TestChars <- function(sign = 1, font = 1, ...)
{
  MB <- l10n_info()$MBCS
  r <- if(font == 5) { sign <- 1; c(32:126, 160:254)
     } else if(MB) 32:126 else 32:255
  if (sign == -1) r <- c(32:126, 160:255)
  par(pty = "s")
  plot(c(-1,16), c(-1,16), type = "n", xlab = "", ylab = "",
       xaxs = "i", yaxs = "i",
       main = sprintf("sign = %d, font = %d", sign, font))
  grid(17, 17, lty = 1) ; mtext(paste("MBCS:", MB))
  for(i in r) try(points(i%%16, i%/%16, pch = sign*i, font = font,...))
}
TestChars()
try(TestChars(sign = -1))
TestChars(font = 5)  # Euro might be at 160 (0+10*16).
                     # Mac OS has apple at 240 (0+15*16).
try(TestChars(-1, font = 2))  # bold
```

legend {graphics}

Add Legends to Plots

Description

This function can be used to add legends to plots. Note that a call to the function locator(1) can be used in place of the x and y arguments.

Usage

```
legend(x, y = NULL, legend, fill = NULL, col = par("col"),
       border = "black", lty, lwd, pch,
       angle = 45, density = NULL, bty = "o", bg = par("bg"),
       box.lwd = par("lwd"), box.lty = par("lty"), box.col = par("fg"),
       pt.bg = NA, cex = 1, pt.cex = cex, pt.lwd = lwd,
       xjust = 0, yjust = 1, x.intersp = 1, y.intersp = 1,
       adj = c(0, 0.5), text.width = NULL, text.col = par("col"),
       text.font = NULL, merge = do.lines && has.pch, trace = FALSE,
       plot = TRUE, ncol = 1, horiz = FALSE, title = NULL,
       inset = 0, xpd, title.col = text.col, title.adj = 0.5,
       seg.len = 2)
```

Arguments

x, y

the x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by xy.coords: See ‘Details’.

legend

a character or expression vector of length ≥ *1* to appear in the legend. Other objects will be coerced by as.graphicsAnnot.

fill

if specified, this argument will cause boxes filled with the specified colors (or shaded in the specified colors) to appear beside the legend text.

col

the color of points or lines appearing in the legend.

border

the border color for the boxes (used only if fill is specified).

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, as numeric vector or a vector of 1-character strings (see points). Unlike points, this can all be specified as a single multi-character string. *Must* be specified for symbol drawing.

angle

angle of shading lines.

density

the density of shading lines, if numeric and positive. If NULL or negative or NA color filling is assumed.

bty

the type of box to be drawn around the legend. The allowed values are "o" (the default) and "n".

bg

the background color for the legend box. (Note that this is only used if bty != "n".)

box.lty, box.lwd, box.col

the line type, width and color for the legend box (if bty = "o").

pt.bg

the background color for the points, corresponding to its argument bg.

cex

character expansion factor **relative** to current par("cex"). Used for text, and provides the default for pt.cex and title.cex.
pt.cex
expansion factor(s) for the points.
pt.lwd
line width for the points, defaults to the one for lines, or if that is not set, to par("lwd").
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. (Should be positive even for a reversed x axis.) Defaults to the proper value computed by strwidth(legend).
text.col
the color used for the legend text.
text.font
the font used for the legend text, see text.
merge
logical; if TRUE, merge points and lines but not filled boxes. Defaults to TRUE if there are points and lines.
trace
logical; if TRUE, shows how legend does all its magical computations.
plot
logical. If FALSE, nothing is plotted but the sizes are returned.
ncol
the number of columns in which to set the legend items (default is 1, a vertical legend).
horiz
logical; if TRUE, set the legend horizontally rather than vertically (specifying horiz overrides the ncol specification).
title
a character string or length-one expression giving a title to be placed at the top of the legend. Other objects will be coerced by as.graphicsAnnot.
inset
inset distance(s) from the margins as a fraction of the plot region when legend is placed by keyword.
xpd
if supplied, a value of the graphical parameter xpd to be used while the legend is being drawn.
title.col
color for title.
title.adj
horizontal adjustment for title: see the help for par("adj").
seg.len
the length of lines drawn to illustrate lty and/or lwd (in units of character widths).

Details

Arguments x, y, legend are interpreted in a non-standard way to allow the coordinates to be specified *via* one or two arguments. If legend is missing and y is not numeric, it is assumed that the second argument is intended to be legend and that the first argument specifies the coordinates.

The coordinates can be specified in any way which is accepted by xy.coords. If this gives the coordinates of one point, it is used as the top-left coordinate of the rectangle containing the legend. If it gives the coordinates of two points, these specify opposite corners of the rectangle (either pair of corners, in any order).
The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location. Partial argument matching is used. The optional inset argument specifies how far the legend is inset from the plot margins. If a single value is given, it is used for both margins; if two values are given, the first is used for x- distance, the second for y-distance.
Attribute arguments such as col, pch, lty, etc, are recycled if necessary: merge is not. Set entries of lty to 0 or set entries of lwd to NA to suppress lines in corresponding legend entries; set pch values to NA to suppress points.
Points are drawn *after* lines in order that they can cover the line with their background color pt.bg, if applicable.
See the examples for how to right-justify labels.
Since they are not used for Unicode code points, values -31:-1 are silently omitted, as are NA and "" values.

Value

A list with list components
rect
a list with components
w, h
positive numbers giving **w**idth and **h**eight of the legend's box.
left, top
x and y coordinates of upper left corner of the box.
text
a list with components
x, y
numeric vectors of length length(legend), giving the x and y coordinates of the legend's text(s).
returned invisibly.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

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  (1, y.leg[i] - 0.1, paste("cex=", formatC(cexv[i])), cex = 0.8, adj = 0)
  legend(3, y.leg[i], leg.txt, pch = "sSvV", col = c(1, 3), cex = cexv[i])
}

## 'merge = TRUE' for merging lines & points:
x <- seq(-pi, pi, len = 65)
plot(x, sin(x), type = "l", ylim = c(-1.2, 1.8), col = 3, lty = 2)
points(x, cos(x), pch = 3, col = 4)
lines(x, tan(x), type = "b", lty = 1, pch = 4, col = 6)
title("legend(..., lty = c(2, -1, 1), pch = c(NA, 3, 4), merge = TRUE)",
      cex.main = 1.1)
legend(-1, 1.9, c("sin", "cos", "tan"), col = c(3, 4, 6),
       text.col = "green4", lty = c(2, -1, 1), pch = c(NA, 3, 4),
       merge = TRUE, bg = "gray90")
```

```
## right-justifying a set of labels: thanks to Uwe Ligges
x <- 1:5; y1 <- 1/x; y2 <- 2/x
plot(rep(x, 2), c(y1, y2), type = "n", xlab = "x", ylab = "y")
lines(x, y1); lines(x, y2, lty = 2)
temp <- legend("topright", legend = c(" ", " "),
               text.width = strwidth("1,000,000"),
               lty = 1:2, xjust = 1, yjust = 1,
               title = "Line Types")
text(temp$rect$left + temp$rect$w, temp$text$y,
     c("1,000", "1,000,000"), pos = 2)


##--- 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 = paste0("log = '", ll, "'"))
  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, 0.3))
  legend(list(x = 2,y = 8), legend = leg.txt, col = 2:3, pch = 1:2,
         lty = 1, merge = TRUE)   #, trace = TRUE)
}
par(mfrow = c(1,1))

##-- Math expressions:  ------------------------------
x <- seq(-pi, pi, len = 65)
plot(x, sin(x), type = "l", col = 2, xlab = expression(phi),
     ylab = expression(f(phi)))
abline(h = -1:1, v = pi/2*(-6:6), col = "gray90")
lines(x, cos(x), col = 3, lty = 2)
ex.cs1 <- expression(plain(sin) * phi,  paste("cos", phi))  # 2 ways
utils::str(legend(-3, .9, ex.cs1, lty = 1:2, plot = FALSE,
           adj = c(0, 0.6)))  # adj y !
legend(-3, 0.9, ex.cs1, lty = 1:2, col = 2:3,  adj = c(0, 0.6))

require(stats)
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))
utils::str(legend(4.1, 30, ex12, col = 2:3, lty = 2:3, lwd = 2))

## 'Filled' boxes -- for more, see example(plot.factor)
op <- par(bg = "white") # to get an opaque box for the legend
plot(cut(weight, 3) ~ group, data = PlantGrowth, col = NULL,
     density = 16*(1:3))
```

```
par(op)

## Using 'ncol' :
x <- 0:64/64
matplot(x, outer(x, 1:7, function(x, k) sin(k * pi * x)),
        type = "o", col = 1:7, ylim = c(-1, 1.5), pch = "*")
op <- par(bg = "antiquewhite1")
legend(0, 1.5, paste("sin(", 1:7, "pi * x)"), col = 1:7, lty = 1:7,
       pch = "*", ncol = 4, cex = 0.8)
legend(.8,1.2, paste("sin(", 1:7, "pi * x)"), col = 1:7, lty = 1:7,
       pch = "*", cex = 0.8)
legend(0, -.1, paste("sin(", 1:4, "pi * x)"), col = 1:4, lty = 1:4,
       ncol = 2, cex = 0.8)
legend(0, -.4, paste("sin(", 5:7, "pi * x)"), col = 4:6,  pch = 24,
       ncol = 2, cex = 1.5, lwd = 2, pt.bg = "pink", pt.cex = 1:3)
par(op)

## point covering line :
y <- sin(3*pi*x)
plot(x, y, type = "l", col = "blue",
    main = "points with bg & legend(*, pt.bg)")
points(x, y, pch = 21, bg = "white")
legend(.4,1, "sin(c x)", pch = 21, pt.bg = "white", lty = 1, col = "blue")

## legends with titles at different locations
plot(x, y, type = "n")
legend("bottomright", "(x,y)", pch = 1, title = "bottomright")
legend("bottom", "(x,y)", pch = 1, title = "bottom")
legend("bottomleft", "(x,y)", pch = 1, title = "bottomleft")
legend("left", "(x,y)", pch = 1, title = "left")
legend("topleft", "(x,y)", pch = 1, title = "topleft, inset = .05",
       inset = .05)
legend("top", "(x,y)", pch = 1, title = "top")
legend("topright", "(x,y)", pch = 1, title = "topright, inset = .02",
       inset = .02)
legend("right", "(x,y)", pch = 1, title = "right")
legend("center", "(x,y)", pch = 1, title = "center")

# using text.font (and text.col):
op <- par(mfrow = c(2, 2), mar = rep(2.1, 4))
c6 <- terrain.colors(10)[1:6]
for(i in 1:4) {
  plot(1, type = "n", axes = FALSE, ann = FALSE); title(paste("text.font =",i))
  legend("top", legend = LETTERS[1:6], col = c6,
         ncol = 2, cex = 2, lwd = 3, text.font = i, text.col = c6)
}
par(op)
```

axis {graphics}

Add an Axis to a Plot

Description

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

Usage

```
axis(side, at = NULL, labels = TRUE, tick = TRUE, line = NA,
    pos = NA, outer = FALSE, font = NA, lty = "solid",
    lwd = 1, lwd.ticks = lwd, col = NULL, col.ticks = NULL,
    hadj = NA, padj = NA, ...)
```

Arguments

side

an integer specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.

at

the points at which tick-marks are to be drawn. Non-finite (infinite, NaN or NA) values are omitted. By default (when NULL) tickmark locations are computed, see 'Details' below.

labels

this can either be a logical value specifying whether (numerical) annotations are to be made at the tickmarks, or a character or expression vector of labels to be placed at the tickpoints. (Other objects are coerced by as.graphicsAnnot.) If this is not logical, at should also be supplied and of the same length. If labels is of length zero after coercion, it has the same effect as supplying TRUE.

tick

a logical value specifying whether tickmarks and an axis line should be drawn.

line

the number of lines into the margin at which the axis line will be drawn, if not NA.

pos

the coordinate at which the axis line is to be drawn: if not NA this overrides the value of line.

outer

a logical value indicating whether the axis should be drawn in the outer plot margin, rather than the standard plot margin.

font

font for text. Defaults to par("font").

lty

line type for both the axis line and the tick marks.

lwd, lwd.ticks

line widths for the axis line and the tick marks. Zero or negative values will suppress the line or ticks.

col, col.ticks

colors for the axis line and the tick marks respectively. col = NULL means to use par("fg"), possibly specified inline, and col.ticks = NULL means to use whatever color col resolved to.

hadj

adjustment (see par("adj")) for all labels *parallel* ('horizontal') to the reading direction. If this is not a finite value, the default is used (centring for strings parallel to the axis, justification of the end nearest the axis otherwise).

padj

adjustment for each tick label *perpendicular* to the reading direction. For labels parallel to the axes, padj = 0 means right or top alignment, and padj = 1 means left or bottom alignment. This can be a vector given a value for each string, and will be recycled as necessary.

If padj is not a finite value (the default), the value of par("las") determines the adjustment. For strings plotted perpendicular to the axis the default is to centre the string.

...

other graphical parameters may also be passed as arguments to this function, particularly, cex.axis, col.axis and font.axis for axis annotation, mgp and xaxp or yaxp for positioning, tck or tcl for tick mark length and direction, las for vertical/horizontal label orientation, or fg instead of col, and xpd for clipping. See par on these.

Parameters xaxt (sides 1 and 3) and yaxt (sides 2 and 4) control if the axis is plotted at all.

Note that lab will partial match to argument labels unless the latter is also supplied. (Since the default axes have already been set up by plot.window, lab will not be acted on by axis.)

Details

The axis line is drawn from the lowest to the highest value of at, but will be clipped at the plot region. By default, 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. Use xpd = TRUE or xpd = NA to allow axes to extend further.
When at = NULL, pretty tick mark locations are computed internally (the same way axTicks(side) would) from par("xaxp") or "yaxp" and par("xlog") (or "ylog"). Note that these locations may change if an on-screen plot is resized (for example, if the plot argument asp (see plot.window) is set.)
If labels is not specified, the numeric values supplied or calculated for at are converted to character strings as if they were a numeric vector printed by print.default(digits = 7).
The code tries hard not to draw overlapping tick labels, and so will omit labels where they would abut or overlap previously drawn labels. This can result in, for example, every other tick being labelled. (The ticks are drawn left to right or bottom to top, and space at least the size of an 'm' is left between labels.)
If either line or pos is set, they (rather than par("mgp")[3]) determine the position of the axis line and tick marks, and the tick labels are placed par("mgp")[2] further lines into (or towards for pos) the margin.
Several of the graphics parameters affect the way axes are drawn. The vertical (for sides 1 and 3) positions of the axis and the tick labels are controlled by mgp[2:3] and mex, the size and direction of the ticks is controlled by tck and tcl and the appearance of the tick labels by cex.axis, col.axis and font.axis with orientation controlled by las (but not srt, unlike S which uses srt if at is supplied and las if it is not). Note that adj is not supported and labels are always centered. See par for details.

Value

The numeric locations on the axis scale at which tick marks were drawn when the plot was first drawn (see 'Details').
This function is usually invoked for its side effect, which is to add an axis to an already existing plot.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

Axis for a generic interface.
axTicks returns the axis tick locations corresponding to at = NULL; pretty is more flexible for computing pretty tick coordinates and does *not* depend on (nor adapt to) the coordinate system in use.
Several graphics parameters affecting the appearance are documented in par.

Examples

```
require(stats) # for rnorm
plot(1:4, rnorm(4), axes = FALSE)
axis(1, 1:4, LETTERS[1:4])
axis(2)
box() #- to make it look "as usual"

plot(1:7, rnorm(7), main = "axis() examples",
    type = "s", xaxt = "n", frame = FALSE, col = "red")
axis(1, 1:7, LETTERS[1:7], col.axis = "blue")
# unusual options:
axis(4, col = "violet", col.axis = "dark violet", lwd = 2)
axis(3, col = "gold", lty = 2, lwd = 0.5)

# one way to have a custom x axis
plot(1:10, xaxt = "n")
axis(1, xaxp = c(2, 9, 7))
```

box {graphics}

Draw a Box around a Plot

Description

This function draws a box around the current plot in the given color and linetype. The bty parameter determines the type of box drawn. See par for details.

Usage

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. Note that xpd is not accepted as clipping is always to the device region.

Details

The choice of colour is complicated. If col was supplied and is not NA, it is used. Otherwise, if fg was supplied and is not NA, it is used. The final default is par("col").

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

rect for drawing of arbitrary rectangles.

Examples

```
plot(1:7, abs(stats::rnorm(7)), type = "h", axes = FALSE)
axis(1, at = 1:7, labels = letters[1:7])
box(lty = '1373', col = 'red')
```

par {graphics}

Set or Query Graphical Parameters

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> (\dots, <tag> = <value>)
```

Arguments

...
arguments in tag = value form, or a list of tagged values. The tags must come from the names of graphical parameters described in the 'Graphical Parameters' section.
no.readonly
logical; if TRUE and there are no other arguments, only parameters are returned which can be set by a subsequent par() call *on the same device*.

Details

Each device has its own set of graphical parameters. If the current device is the null device, par will open a new device before querying/setting parameters. (What device is controlled by options("device").)

Parameters are queried by giving one or more character vectors of parameter names to par.

par() (no arguments) or par(no.readonly = TRUE) is used to get *all* the graphical parameters (as a named list). Their names are currently taken from the unexported variable graphics:::.Pars.

R.O. indicates ***read-only arguments***: These may only be used in queries and cannot be set. ("cin", "cra", "csi", "cxy", "din" and "page" are always read-only.)

Several parameters can only be set by a call to par():

- "ask",
- "fig", "fin",

- "lheight",
- "mai", "mar", "mex", "mfcol", "mfrow", "mfg",
- "new",
- "oma", "omd", "omi",
- "pin", "plt", "ps", "pty",
- "usr",
- "xlog", "ylog",
- "ylbias"

The remaining parameters can also be set as arguments (often via ...) to high-level plot functions such as plot.default, plot.window, points, lines, abline, axis, title, text, mtext, segments, symbols, arrows, polygon, rect, box, contour, filled.contour and image. Such settings will be active during the execution of the function, only. However, see the comments on bg, cex, col, lty, lwd and pch which may be taken as *arguments* to certain plot functions rather than as graphical parameters.

The meaning of 'character size' is not well-defined: this is set up for the device taking pointsize into account but often not the actual font family in use. Internally the corresponding pars (cra, cin, cxy and csi) are used only to set the inter-line spacing used to convert mar and oma to physical margins. (The same inter-line spacing multiplied by lheight is used for multi-line strings in text and strheight.)

Note that graphical parameters are suggestions: plotting functions and devices need not make use of them (and this is particularly true of non-default methods for e.g. plot).

Value

When parameters are set, their previous 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. However, restoring all of these is not wise: see the 'Note' section.

When just one parameter is queried, the value of that parameter is returned as (atomic) vector. When two or more parameters are queried, their values are returned in a list, with the list names giving the parameters.

Note the inconsistency: setting one parameter returns a list, but querying one parameter returns a vector.

Graphical Parameters

adj

The value of adj determines the way in which text strings are justified in text, mtext and title. A value of 0 produces left-justified text, 0.5 (the default) centered text and 1 right-justified text. (Any value in *[0, 1]* is allowed, and on most devices values outside that interval will also work.)

Note that the adj *argument* of text also allows adj = c(x, y) for different adjustment in x- and y- directions. Note that whereas for text it refers to positioning of text about a point, for mtext and title it controls placement within the plot or device region.

ann

If set to FALSE, high-level plotting functions calling plot.default do not annotate the plots they produce with axis titles and overall titles. The default is to do annotation.

ask

logical. If TRUE (and the R session is interactive) the user is asked for input, before a new figure is drawn. As this applies to the device, it also affects output by packages **grid** and **lattice**. It can be set even on non-screen devices but may have no effect there.

This not really a graphics parameter, and its use is deprecated in favour of devAskNewPage.

bg

The color to be used for the background of the device region. When called from par() it also sets new = FALSE. See section 'Color Specification' for suitable values. For many devices the initial value is set from the bg argument of the device, and for the rest it is normally "white".

Note that some graphics functions such as plot.default and points have an *argument* of this name with a different meaning.

bty

A character string which determined the type of box which is drawn about plots. If bty is one of "o" (the default), "l", "7", "c", "u", or "]" the resulting box resembles the corresponding upper case letter. A value of "n" suppresses the box.

cex

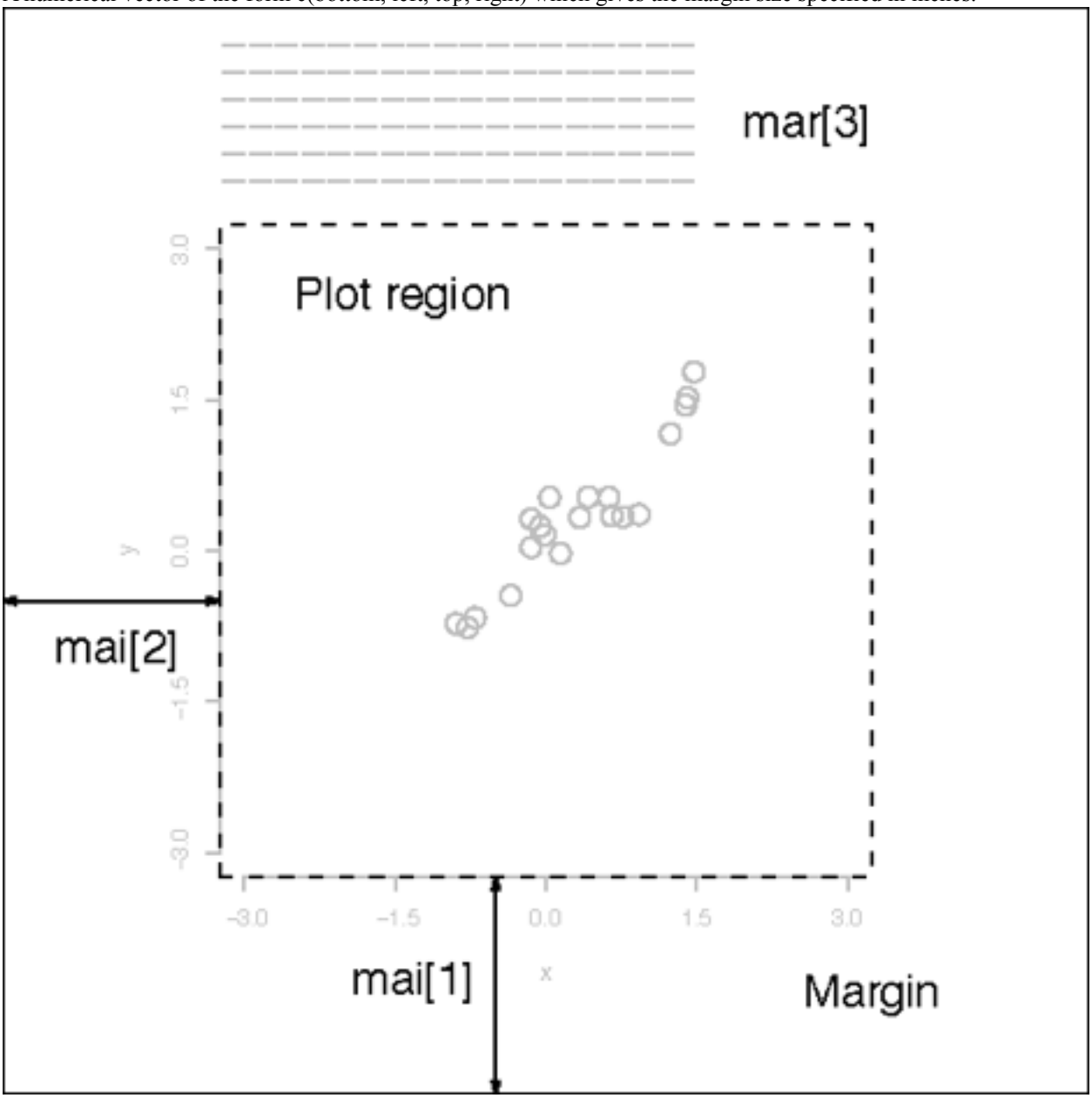
A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. This starts as 1 when a device is opened, and is reset when the layout is changed, e.g. by setting mfrow.

Note that some graphics functions such as plot.default have an *argument* of this name which *multiplies* this graphical parameter, and some functions such as points and text accept a vector of values which are recycled.
cex.axis
The magnification to be used for axis annotation relative to the current setting of cex.
cex.lab
The magnification to be used for x and y labels relative to the current setting of cex.
cex.main
The magnification to be used for main titles relative to the current setting of cex.
cex.sub
The magnification to be used for sub-titles relative to the current setting of cex.
cin
R.O.; character size (width, height) in inches. These are the same measurements as cra, expressed in different units.
col
A specification for the default plotting color. See section 'Color Specification'.
Some functions such as lines and text accept a vector of values which are recycled and may be interpreted slightly differently.
col.axis
The color to be used for axis annotation. Defaults to "black".
col.lab
The color to be used for x and y labels. Defaults to "black".
col.main
The color to be used for plot main titles. Defaults to "black".
col.sub
The color to be used for plot sub-titles. Defaults to "black".
cra
R.O.; size of default character (width, height) in 'rasters' (pixels). Some devices have no concept of pixels and so assume an arbitrary pixel size, usually 1/72 inch. These are the same measurements as cin, expressed in different units.
crt
A numerical value specifying (in degrees) how single characters should be rotated. It is unwise to expect values other than multiples of 90 to work. Compare with srt which does string rotation.
csi
R.O.; height of (default-sized) characters in inches. The same as par("cin")[2].
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), strheight(ch)) for a given string ch is usually much more precise.
din
R.O.; the device dimensions, (width, height), in inches. See also dev.size, which is updated immediately when an on-screen device windows is re-sized.
err
(*Unimplemented*; **R** is silent when points outside the plot region are *not* plotted.) The degree of error reporting desired.
family
The name of a font family for drawing text. The maximum allowed length is 200 bytes. This name gets mapped by each graphics device to a device-specific font description. The default value is "" which means that the default device fonts will be used (and what those are should be listed on the help page for the device). Standard values are "serif", "sans" and "mono", and the Hershey font families are also available. (Different devices may define others, and some devices will ignore this setting completely.) This can be specified inline for text.
fg
The color to be used for the foreground of plots. This is the default color used for things like axes and boxes around plots. When called from par() this also sets parameter col to the same value. See section 'Color Specification'. A few devices have an argument to set the initial value, which is otherwise "black".
fig

A numerical vector of the form c(x1, x2, y1, y2) which gives the (NDC) coordinates of the figure region in the display region of the device. If you set this, unlike S, you start a new plot, so to add to an existing plot use new = TRUE as well.
fin
The figure region dimensions, (width, height), in inches. If you set this, unlike S, you start a new plot.
font
An integer which specifies which font to use for text. If possible, device drivers arrange so that 1 corresponds to plain text (the default), 2 to bold face, 3 to italic and 4 to bold italic. Also, font 5 is expected to be the symbol font, in Adobe symbol encoding. On some devices font families can be selected by family to choose different sets of 5 fonts.
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.
lab
A numerical vector of the form c(x, y, len) which modifies the default 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 length. The default is c(5, 5, 7). Note that this only affects the way the parameters xaxp and yaxp are set when the user coordinate system is set up, and is not consulted when axes are drawn. len *is unimplemented* in **R**.
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.
Also supported by mtext. Note that string/character rotation *via* argument srt to par does *not* affect the axis labels.
lend
The line end style. This can be specified as an integer or string:
0
and "round" mean rounded line caps [*default*];
1
and "butt" mean butt line caps;
2
and "square" mean square line caps.
lheight
The line height multiplier. The height of a line of text (used to vertically space multi-line text) is found by multiplying the character height both by the current character expansion and by the line height multiplier. Default value is 1. Used in text and strheight.
ljoin
The line join style. This can be specified as an integer or string:
0
and "round" mean rounded line joins [*default*];
1
and "mitre" mean mitred line joins;
2
and "bevel" mean bevelled line joins.

lmitre
The line mitre limit. This controls when mitred line joins are automatically converted into bevelled line joins. The value must be larger than 1 and the default is 10. Not all devices will honour this setting.
lty
The line type. Line types can either be specified as an integer (0=blank, 1=solid (default), 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., does not draw them).
Alternatively, a string of up to 8 characters (from c(1:9, "A":"F")) may be given, giving the length of line segments which are alternatively drawn and skipped. See section ‘Line Type Specification’.
Functions such as lines and segments accept a vector of values which are recycled.
lwd
The line width, a *positive* number, defaulting to 1. The interpretation is device-specific, and some devices do not implement line widths less than one. (See the help on the device for details of the interpretation.)
Functions such as lines and segments accept a vector of values which are recycled: in such uses lines corresponding to values NA or NaN are omitted. The interpretation of 0 is device-specific.
mai

A numerical vector of the form c(bottom, left, top, right) which gives the margin size specified in inches.



mar

A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot. The default is c(5, 4, 4, 2) + 0.1.

mex

mex is a character size expansion factor which is used to describe coordinates in the margins of plots. Note that this does not change the font size, rather specifies the size of font (as a multiple of csi) used to convert between mar and mai, and between oma and omi.

This starts as 1 when the device is opened, and is reset when the layout is changed (alongside resetting cex).

mfcol, mfrow

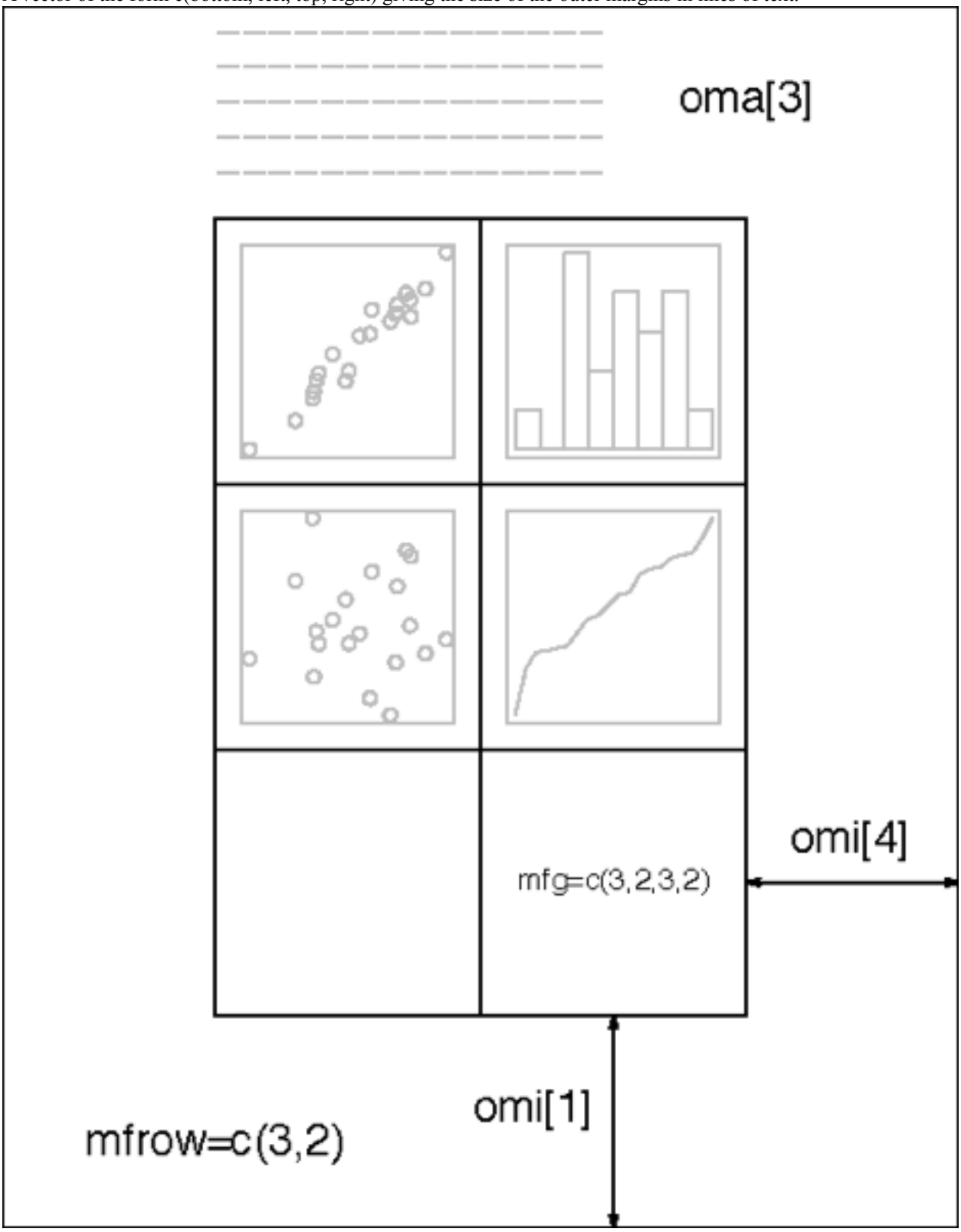
A vector of the form c(nr, nc). Subsequent figures will be drawn in an nr-by-nc array on the device by *columns* (mfcol), or *rows* (mfrow), respectively.

In a layout with exactly two rows and columns the base value of "cex" is reduced by a factor of 0.83: if there are three or more of either rows or columns, the reduction factor is 0.66.

Setting a layout resets the base value of cex and that of mex to 1.

If either of these is queried it will give the current layout, so querying cannot tell you the order in which the array will be filled.
Consider the alternatives, layout and split.screen.
mfg
A numerical vector of the form c(i, j) where i and j indicate which figure in an array of figures is to be drawn next (if setting) or is being drawn (if enquiring). The array must already have been set by mfcol or mfrow.
For compatibility with S, the form c(i, j, nr, nc) is also accepted, when nr and nc should be the current number of rows and number of columns. Mismatches will be ignored, with a warning.
mgp
The margin line (in mex units) for the axis title, axis labels and axis line. Note that mgp[1] affects title whereas mgp[2:3] affect axis. 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 in **R***.
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 were on a **new** device*. It is an error (ignored with a warning) to try to use new = TRUE on a device that does not currently contain a high-level plot.
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 region *inside* outer margins in NDC (= normalized device coordinates), i.e., as a 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.
page
R.O.; A boolean value indicating whether the next call to plot.new is going to start a new page. This value may be FALSE if there are multiple figures on the page.
pch
Either an integer specifying a symbol or a single character to be used as the default in plotting points. See points for possible values and their interpretation. Note that only integers and single-character strings can be set as a graphics parameter (and not NA nor NULL).
Some functions such as points accept a vector of values which are recycled.
pin
The current plot dimensions, (width, height), in inches.
plt
A vector of the form c(x1, x2, y1, y2) giving the coordinates of the plot region as fractions of the current figure region.
ps
integer; the point size of text (but not symbols). Unlike the pointsize argument of most devices, this does not change the relationship between mar and mai (nor oma and omi).
What is meant by 'point size' is device-specific, but most devices mean a multiple of 1bp, that is 1/72 of an inch.
pty
A character specifying the type of plot region to be used; "s" generates a square plotting region and "m" generates the maximal plotting region.
smo
(*Unimplemented*) a value which indicates how smooth circles and circular arcs should be.
srt
The string rotation in degrees. See the comment about crt. Only supported by text.
tck
The length of tick marks as a fraction of the smaller of the width or height of the plotting region. If tck >= 0.5 it is interpreted as a fraction of the relevant side, so if tck = 1 grid lines are drawn. The default setting (tck = NA) is to use tcl = -0.5.
tcl
The length of tick marks as a fraction of the height of a line of text. The default value is -0.5; setting tcl = NA sets tck = -0.01 which is S' default.
usr
A vector of the form c(x1, x2, y1, y2) giving the extremes of the user coordinates of the plotting region. When a logarithmic scale is in use (i.e., par("xlog") is true, see below), then the x-limits will be 10 ^ par("usr")[1:2]. Similarly for the y-axis.
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 when par("xlog") is false. Otherwise, when *log* coordinates are active, the three values have a different meaning: For a small range, n is *negative*, and the ticks are as in the linear case, otherwise, n is in 1:3, specifying a case number, and x1 and x2 are the lowest and highest power of 10 inside the user coordinates, 10 ^ par("usr")[1:2]. (The "usr" coordinates are log10-transformed here!)
n = 1
will produce tick marks at *10^j* for integer *j*,
n = 2
gives marks *k 10^j* with *k in {1,5}*,
n = 3
gives marks *k 10^j* with *k in {1,2,5}*.
See axTicks() for a pure **R** implementation of this.
This parameter is reset when a user coordinate system is set up, for example by starting a new page or by calling plot.window or setting par("usr"): n is taken from par("lab"). It affects the default behaviour of subsequent calls to axis for sides 1 or 3.
It is only relevant to default numeric axis systems, and not for example to dates.
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 at each end and then finds an axis with pretty labels that fits within the extended 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 ensures 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 have been implemented in R.*)

xaxt

A character which specifies the x axis type. Specifying "n" suppresses plotting of the axis. The standard value is "s": for compatibility with S values "l" and "t" are accepted but are equivalent to "s": any value other than "n" implies plotting.

xlog

A logical value (see log in plot.default). If TRUE, a logarithmic scale is in use (e.g., after plot(*, log = "x")). For a new device, it defaults to FALSE, i.e., linear scale.

xpd

A logical value or NA. If FALSE, all plotting is clipped to the plot region, if TRUE, all plotting is clipped to the figure region, and if NA, all plotting is clipped to the device region. See also clip.

yaxp

A vector of the form c(y1, y2, n) giving the coordinates of the extreme tick marks and the number of intervals between tick-marks unless for log coordinates, see xaxp above.

yaxs

The style of axis interval calculation to be used for the y-axis. See xaxs above.

yaxt

A character which specifies the y axis type. Specifying "n" suppresses plotting.

ylbias

A positive real value used in the positioning of text in the margins by axis and mtext. The default is in principle device-specific, but currently 0.2 for all of R's own devices. Set this to 0.2 for compatibility with R < 2.14.0 on x11 and windows() devices.

ylog

A logical value; see xlog above.

Color Specification

Colors can be specified in several different ways. The simplest way is with a character string giving the color name (e.g., "red"). A list of the possible colors can be obtained with the function colors. Alternatively, colors can be specified directly in terms of their RGB components with a string of the form "#RRGGBB" where each of the pairs RR, GG, BB consist of two hexadecimal digits giving a value in the range 00 to FF. Colors can also be specified by giving an index into a small table of colors, the palette: indices wrap round so with the default palette of size 8, 10 is the same as 2. This provides compatibility with S. Index 0 corresponds to the background color. Note that the palette (apart from 0 which is per-device) is a per-session setting.
Negative integer colours were accepted prior to R 3.0.0, but treated inconsistently. They are now errors.
Additionally, "transparent" is *transparent*, useful for filled areas (such as the background!), and just invisible for things like lines or text. In most circumstances (integer) NA is equivalent to "transparent" (but not for text and mtext).
Semi-transparent colors are available for use on devices that support them.
The functions rgb, hsv, hcl, gray and rainbow provide additional ways of generating colors.
It was possible to specify color numbers as strings prior to R 3.0.0.

Line Type Specification

Line types can either be specified by giving an index into a small built-in table of line types (1 = solid, 2 = dashed, etc, see lty above) or directly as the lengths of on/off stretches of line. This is done with a string of an even number (up to eight) of characters, namely *non-zero* (hexadecimal) digits which give the lengths in consecutive positions in the string. For example, the string "33" specifies three units on followed by three off and "3313" specifies three units on followed by three off followed by one on and finally three off. The 'units' here are (on most devices) proportional to lwd, and with lwd = 1 are in pixels or points or 1/96 inch.
The five standard dash-dot line types (lty = 2:6) correspond to c("44", "13", "1343", "73", "2262").

Note that NA is not a valid value for lty.

Note

The effect of restoring all the (settable) graphics parameters as in the examples is hard to predict if the device has been resized. Several of them are attempting to set the same things in different ways, and those last in the alphabet will win. In particular, the settings of mai, mar, pin, plt and pty interact, as do the outer margin settings, the figure layout and figure region size.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

plot.default for some high-level plotting parameters; colors; clip; 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)
## Note this is not in general good practice

par("ylog") # FALSE
plot(1 : 12, log = "y")
par("ylog") # TRUE

plot(1:2, xaxs = "i") # 'inner axis' w/o extra space
par(c("usr", "xaxp"))

( 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

require(grDevices) # for gray
## 'fg' use:
plot(1:12, type = "b", main = "'fg' : axes, ticks and box in gray",
     fg = gray(0.7), bty = "7" , sub = R.version.string)

ex <- function() {
  old.par <- par(no.readonly = TRUE) # all par settings which
                                     # could be changed.
  on.exit(par(old.par))
  ## ...
  ## ... do lots of par() settings and plots
  ## ...
  invisible() #-- now,  par(old.par)  will be executed
```

```
}
ex()

## Line types
showLty <- function(ltys, xoff = 0, ...) {
  stopifnot((n <- length(ltys)) >= 1)
  op <- par(mar = rep(.5,4)); on.exit(par(op))
  plot(0:1, 0:1, type = "n", axes = FALSE, ann = FALSE)
  y <- (n:1)/(n+1)
  clty <- as.character(ltys)
  mytext <- function(x, y, txt)
    text(x, y, txt, adj = c(0, -.3), cex = 0.8, ...)
  abline(h = y, lty = ltys, ...); mytext(xoff, y, clty)
  y <- y - 1/(3*(n+1))
  abline(h = y, lty = ltys, lwd = 2, ...)
  mytext(1/8+xoff, y, paste(clty," lwd = 2"))
}
showLty(c("solid", "dashed", "dotted", "dotdash", "longdash", "twodash"))
par(new = TRUE)  # the same:
showLty(c("solid", "44", "13", "1343", "73", "2262"), xoff = .2, col = 2)
showLty(c("11", "22", "33", "44",   "12", "13", "14",   "21", "31"))
```

pie {graphics}

Pie Charts

Description

Draw a pie chart.

Usage

```
pie(x, labels = names(x), edges = 200, radius = 0.8,
    clockwise = FALSE, init.angle = if(clockwise) 90 else 0,
    density = NULL, angle = 45, col = NULL, border = NULL,
    lty = NULL, main = NULL, ...)
```

Arguments

x

a vector of non-negative numerical quantities. The values in x are displayed as the areas of pie slices.

labels

one or more expressions or character strings giving names for the slices. Other objects are coerced by as.graphicsAnnot. For empty or NA (after coercion to character) labels, no label nor pointing line is drawn.

edges

the circular outline of the pie is approximated by a polygon with this many edges.

radius

the pie is drawn centered in a square box whose sides range from *-1* to *1*. If the character strings labeling the slices are long it may be necessary to use a smaller radius.

clockwise

logical indicating if slices are drawn clockwise or counter clockwise (i.e., mathematically positive direction), the latter is default.

init.angle

number specifying the *starting angle* (in degrees) for the slices. Defaults to 0 (i.e., '3 o'clock') unless clockwise is true where init.angle defaults to 90 (degrees), (i.e., '12 o'clock').

density

the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.
angle
the slope of shading lines, given as an angle in degrees (counter-clockwise).
col
a vector of colors to be used in filling or shading the slices. If missing a set of 6 pastel colours is used, unless density is specified when par("fg") is used.
border, lty
(possibly vectors) arguments passed to polygon which draws each slice.
main
an overall title for the plot.
...
graphical parameters can be given as arguments to pie. They will affect the main title and labels only.

Note

Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.
Cleveland (1985), page 264: "Data that can be shown by pie charts always can be shown by a dot chart. This means that judgements of position along a common scale can be made instead of the less accurate angle judgements." This statement is based on the empirical investigations of Cleveland and McGill as well as investigations by perceptual psychologists.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Cleveland, W. S. (1985) *The Elements of Graphing Data*. Wadsworth: Monterey, CA, USA.

See Also

dotchart.

Examples

```
require(grDevices)
pie(rep(1, 24), col = rainbow(24), radius = 0.9)

pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
names(pie.sales) <- c("Blueberry", "Cherry",
   "Apple", "Boston Cream", "Other", "Vanilla Cream")
pie(pie.sales) # default colours
pie(pie.sales, col = c("purple", "violetred1", "green3",
                       "cornsilk", "cyan", "white"))
pie(pie.sales, col = gray(seq(0.4, 1.0, length = 6)))
pie(pie.sales, density = 10, angle = 15 + 10 * 1:6)
pie(pie.sales, clockwise = TRUE, main = "pie(*, clockwise = TRUE)")
segments(0, 0, 0, 1, col = "red", lwd = 2)
text(0, 1, "init.angle = 90", col = "red")

n <- 200
pie(rep(1, n), labels = "", col = rainbow(n), border = NA,
   main = "pie(*, labels=\"\", col=rainbow(n), border=NA,..")
```

median {stats}

Median Value

Description

Compute the sample median.

Usage

```
median(x, na.rm = FALSE)
```

Arguments

x

an object for which a method has been defined, or 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.

Details

This is a generic function for which methods can be written. However, the default method makes use of is.na, sort and mean from package **base** all of which are generic, and so the default method will work for most classes (e.g. "Date") for which a median is a reasonable concept.

Value

The default method returns a length-one object of the same type as x, except when x is integer of even length, when the result will be double.

If there are no values or if na.rm = FALSE and there are NA values the result is NA of the same type as x (or more generally the result of x[FALSE][NA]).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

quantile for general quantiles.

Examples

```
median(1:4)              # = 2.5 [even number]
median(c(1:3, 100, 1000))  # = 3 [odd, robust]
```

cor {stats}

Correlation, Variance and Covariance (Matrices)

Description

var, cov and cor compute the variance of x and the covariance or correlation of x and y if these are vectors. If x and y are matrices then the covariances (or correlations) between the columns of x and the columns of y are computed.

cov2cor scales a covariance matrix into the corresponding correlation matrix *efficiently*.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)

cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cov2cor(V)
```

Arguments

x

a numeric vector, matrix or data frame.

y

NULL (default) or a vector, matrix or data frame with compatible dimensions to x. The default is equivalent to y = x (but more efficient).

na.rm

logical. Should missing values be removed?

use

an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".

method

a character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson" (default), "kendall", or "spearman", can be abbreviated.

V

symmetric numeric matrix, usually positive definite such as a covariance matrix.

Details

For cov and cor one must *either* give a matrix or data frame for x *or* give both x and y.

The inputs must be numeric (as determined by is.numeric: logical values are also allowed for historical compatibility): the "kendall" and "spearman" methods make sense for ordered inputs but xtfrm can be used to find a suitable prior transformation to numbers.

var is just another interface to cov, where na.rm is used to determine the default for use when that is unspecified. If na.rm is TRUE then the complete observations (rows) are used (use = "na.or.complete") to compute the variance. Otherwise, by default use = "everything".

If use is "everything", NAs will propagate conceptually, i.e., a resulting value will be NA whenever one of its contributing observations is NA.

If use is "all.obs", then the presence of missing observations will produce an error. If use is "complete.obs" then missing values are handled by casewise deletion (and if there are no complete cases, that gives an error). "na.or.complete" is the same unless there are no complete cases, that gives NA. Finally, if use has the value "pairwise.complete.obs" then the correlation or covariance 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 semi-definite, as well as NA entries if there are no complete pairs for that pair of variables. For cov and var, "pairwise.complete.obs" only works with the "pearson" method. Note that (the equivalent of) var(double(0), use = *) gives NA for use = "everything" and "na.or.complete", and gives an error in the other cases.

The denominator *n - 1* is used which gives an unbiased estimator of the (co)variance for i.i.d. observations. These functions return NA when there is only one observation (whereas S-PLUS has been returning NaN), and fail if x has length zero.

For cor(), if method is "kendall" or "spearman", Kendall's *tau* or Spearman's *rho* statistic is used to estimate a rank-based measure of association. These are more robust and have been recommended if the data do not necessarily come from a bivariate normal distribution.

For cov(), a non-Pearson method is unusual but available for the sake of completeness. Note that "spearman" basically computes cor(R(x), R(y)) (or cov(., .)) where R(u) := rank(u, na.last = "keep"). In the case of missing values, the ranks are calculated depending on the value of use, either based on complete observations, or based on pairwise completeness with reranking for each pair.

Scaling a covariance matrix into a correlation one can be achieved in many ways, mathematically most appealing by multiplication with a diagonal matrix from left and right, or more efficiently by using sweep(.., FUN = "/") twice. The cov2cor function is even a bit more efficient, and provided mostly for didactical reasons.

Value

For r <- cor(*, use = "all.obs"), it is now guaranteed that all(r <= 1).

Note

Some people have noted that the code for Kendall's tau is slow for very large datasets (many more than 1000 cases). It rarely makes sense to do such a computation, but see function cor.fk in package **pcaPP**.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

cor.test for confidence intervals (and tests).

cov.wt for *weighted* covariance computation.

sd for standard deviation (vectors).

Examples

```
var(1:10)  # 9.166667

var(1:5, 1:5) # 2.5

## Two simple vectors
```

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

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

## Spearman's rho  and  Kendall's tau
symnum(clS <- cor(longley, method = "spearman"))
symnum(clK <- cor(longley, method = "kendall"))
## How much do they differ?
i <- lower.tri(Cl)
cor(cbind(P = Cl[i], S = clS[i], K = clK[i]))


## cov2cor() scales a covariance matrix by its diagonal
##           to become the correlation matrix.
cov2cor # see the function definition {and learn ..}
stopifnot(all.equal(Cl, cov2cor(cov(longley))),
          all.equal(cor(longley, method = "kendall"),
            cov2cor(cov(longley, method = "kendall"))))

##--- Missing value treatment:

C1 <- cov(swiss)
range(eigen(C1, only.values = TRUE)$values) # 6.19        1921

## swM := "swiss" with  3 "missing"s :
swM <- swiss
colnames(swM) <- abbreviate(colnames(swiss), min=6)
swM[1,2] <- swM[7,3] <- swM[25,5] <- NA # create 3 "missing"

## Consider all 5 "use" cases :
(C. <- cov(swM)) # use="everything"  quite a few NA's in cov.matrix
try(cov(swM, use = "all")) # Error: missing obs...
C2 <- cov(swM, use = "complete")
stopifnot(identical(C2, cov(swM, use = "na.or.complete")))
range(eigen(C2, only.values = TRUE)$values) # 6.46   1930
C3 <- cov(swM, use = "pairwise")
range(eigen(C3, only.values = TRUE)$values) # 6.19   1938

## Kendall's tau doesn't change much:
symnum(Rc <- cor(swM, method = "kendall", use = "complete"))
symnum(Rp <- cor(swM, method = "kendall", use = "pairwise"))
symnum(R. <- cor(swiss, method = "kendall"))

## "pairwise" is closer componentwise,
summary(abs(c(1 - Rp/R.)))
summary(abs(c(1 - Rc/R.)))

## but "complete" is closer in Eigen space:
EV <- function(m) eigen(m, only.values=TRUE)$values
summary(abs(1 - EV(Rp)/EV(R.)) / abs(1 - EV(Rc)/EV(R.)))
```

sd {stats}

Standard Deviation

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.

Usage

sd(x, na.rm = FALSE)

Arguments

x

a numeric vector or an **R** object which is coercible to one by as.vector(x, "numeric").

na.rm

logical. Should missing values be removed?

Details

Like var this uses denominator *n - 1*.

The standard deviation of a zero-length vector (after removal of NAs if na.rm = TRUE) is not defined and gives an error. The standard deviation of a length-one vector is NA.

See Also

var for its square, and mad, the most robust alternative.

Examples

sd(1:2) ^ 2

Normal {stats}

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

Usage

dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)

Arguments

x, q

vector of quantiles.

p

vector of probabilities.

n

number of observations. If length(n) > 1, the length is taken to be the number required.

mean

vector of means.

sd

vector of standard deviations.

log, log.p

logical; if TRUE, probabilities p are given as log(p).

lower.tail

logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise, $P[X > x]$.

Details

If mean or sd are not specified they assume the default values of 0 and 1, respectively.
The normal distribution has density

$$f(x) = 1/(\sqrt{(2\pi)}\sigma) e^{-((x-\mu)^2/(2\sigma^2))}$$

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.
The length of the result is determined by n for rnorm, and is the maximum of the lengths of the numerical arguments for the other functions.
The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.
For sd = 0 this gives the limit as sd decreases to 0, a point mass at mu. sd < 0 is an error and returns NaN.

Source

For pnorm, based on
Cody, W. D. (1993) Algorithm 715: SPECFUN – A portable FORTRAN package of special function routines and test drivers. *ACM Transactions on Mathematical Software* **19**, 22–32.
For qnorm, the code is a C translation of
Wichura, M. J. (1988) Algorithm AS 241: The percentage points of the normal distribution. *Applied Statistics*, **37**, 477–484.
which provides precise results up to about 16 digits.
For rnorm, see RNG for how to select the algorithm and for references to the supplied methods.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 13. Wiley, New York.

See Also

Distributions for other standard distributions, including dlnorm for the *Log*normal distribution.

Examples

```
require(graphics)

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.p = 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)

## if you want the so-called 'error function'
erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1
## (see Abramowitz and Stegun 29.2.29)
## and the so-called 'complementary error function'
erfc <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)
## and the inverses
```

```
erfinv <- function (x) qnorm((1 + x)/2)/sqrt(2)
erfcinv <- function (x) qnorm(x/2, lower = FALSE)/sqrt(2)
```

TDist {stats}

The Student t Distribution

Description

Density, distribution function, quantile function and random generation for the t distribution with df degrees of freedom (and optional non-centrality parameter ncp).

Usage

```
dt(x, df, ncp, log = FALSE)
pt(q, df, ncp, lower.tail = TRUE, log.p = FALSE)
qt(p, df, ncp, lower.tail = TRUE, log.p = FALSE)
rt(n, df, ncp)
```

Arguments

x, q
vector of quantiles.

p
vector of probabilities.

n
number of observations. If length(n) > 1, the length is taken to be the number required.

df
degrees of freedom (> *0*, maybe non-integer). df = Inf is allowed.

ncp
non-centrality parameter *delta*; currently except for rt(), only for abs(ncp) <= 37.62. If omitted, use the central t distribution.

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 = *n* degrees of freedom has density

$$f(x) = \Gamma((n+1)/2) / (\sqrt{(n\,\pi)}\,\Gamma(n/2))\,(1 + x^2/n)^{-((n+1)/2)}$$

for all real *x*. It has mean *0* (for $n > 1$) and variance $n/(n-2)$ (for $n > 2$).

The general *non-central t* with parameters *(df, Del)* = (df, ncp) is defined as the distribution of $T(df, Del) := (U + Del) / \sqrt{(V/df)}$ where *U* and *V* are independent random variables, $U \sim N(0,1)$ and $V \sim \chi^2(df)$ (see Chisquare).

The most used applications are power calculations for *t*-tests:

Let $T = (mX - m0) / (S/\sqrt{n})$ where *mX* 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-central *t* with df= $n - 1$ degrees of freedom and **n**on-**c**entrality **p**arameter ncp = $(\mu - m0) * \sqrt{n}/\sigma$.

Value

dt gives the density, pt gives the distribution function, qt gives the quantile function, and rt generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rt, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

Note

Supplying ncp = 0 uses the algorithm for the non-central distribution, which is not the same algorithm used if ncp is omitted. This is to give consistent behaviour in extreme cases with values of ncp very near zero.

The code for non-zero ncp is principally intended to be used for moderate values of ncp: it will not be highly accurate, especially in the tails, for large values.

Source

The central dt is computed via an accurate formula provided by Catherine Loader (see the reference in dbinom).
For the non-central case of dt, C code contributed by Claus Ekstrøm based on the relationship (for *x != 0*) to the cumulative distribution.
For the central case of pt, a normal approximation in the tails, otherwise via pbeta.
For the non-central case of pt based on a C translation of
Lenth, R. V. (1989). *Algorithm AS 243* — Cumulative distribution function of the non-central *t* distribution, *Applied Statistics* **38**, 185–189.
This computes the lower tail only, so the upper tail suffers from cancellation and a warning will be given when this is likely to be significant.
For central qt, a C translation of
Hill, G. W. (1970) Algorithm 396: Student's t-quantiles. *Communications of the ACM*, **13(10)**, 619–620.
altered to take account of
Hill, G. W. (1981) Remark on Algorithm 396, *ACM Transactions on Mathematical Software*, **7**, 250–1.
The non-central case is done by inversion.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole. (Except non-central versions.)
Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 2, chapters 28 and 31. Wiley, New York.

See Also

Distributions for other standard distributions, including df for the F distribution.

Examples

```
require(graphics)

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))
t.tit <- "Non-central t - Probabilities"
image(tt, ncp, ptn, zlim = c(0,1), main = t.tit)
persp(tt, ncp, ptn, zlim = 0:1, r = 2, phi = 20, theta = 200, main = t.tit,
      xlab = "t", ylab = "non-centrality parameter",
      zlab = "Pr(T <= t)")

plot(function(x) dt(x, df = 3, ncp = 2), -3, 11, ylim = c(0, 0.32),
     main = "Non-central t - Density", yaxs = "i")
```

Chisquare {stats}

The (non-central) Chi-Squared Distribution

Description

Density, distribution function, quantile function and random generation for the chi-squared (*chi^2*) distribution with df degrees of freedom and optional non-centrality parameter ncp.

Usage

```
dchisq(x, df, ncp = 0, log = FALSE)
pchisq(q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qchisq(p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
```

rchisq(n, df, ncp = 0)

Arguments

x, q
vector of quantiles.
p
vector of probabilities.
n
number of observations. If length(n) > 1, the length is taken to be the number required.
df
degrees of freedom (non-negative, but can be non-integer).
ncp
non-centrality parameter (non-negative).
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 chi-squared distribution with df= $n \ge 0$ degrees of freedom has density

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

for $x > 0$. The mean and variance are n and $2n$.

The non-central chi-squared distribution with df= n degrees of freedom and non-centrality parameter ncp = λ has density

$$f(x) = exp(-\lambda/2) SUM_{r=0}^{\infty} ((\lambda/2)^r / r!) dchisq(x, df + 2r)$$

for $x \ge 0$. For integer n, this is the distribution of the sum of squares of n normals each with variance one, λ being the sum of squares of the normal means; further,
$E(X) = n + \lambda$, $Var(X) = 2(n + 2*\lambda)$, and $E((X - E(X))^3) = 8(n + 3*\lambda)$.

Note that the degrees of freedom df= n, can be non-integer, and also $n = 0$ which is relevant for non-centrality $\lambda > 0$, see Johnson *et al* (1995, chapter 29).

Note that ncp values larger than about 1e5 may give inaccurate results with many warnings for pchisq and qchisq.

Value

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

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rchisq, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

Note

Supplying ncp = 0 uses the algorithm for the non-central distribution, which is not the same algorithm used if ncp is omitted. This is to give consistent behaviour in extreme cases with values of ncp very near zero.

The code for non-zero ncp is principally intended to be used for moderate values of ncp: it will not be highly accurate, especially in the tails, for large values.

Source

The central cases are computed via the gamma distribution.

The non-central dchisq and rchisq are computed as a Poisson mixture central of chi-squares (Johnson *et al*, 1995, p. 436).

The non-central pchisq is for ncp < 80 computed from the Poisson mixture of central chi-squares and for larger ncp *via* a C translation of

Ding, C. G. (1992) Algorithm AS275: Computing the non-central chi-squared distribution function. *Appl.Statist.*, **41** 478–482.

which computes the lower tail only (so the upper tail suffers from cancellation and a warning will be given when this is likely to be significant).

The non-central qchisq is based on inversion of pchisq.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, chapters 18 (volume 1) and 29 (volume 2). Wiley, New York.

See Also

Distributions for other standard distributions.

A central chi-squared distribution with *n* degrees of freedom is the same as a Gamma distribution with shape *a = n/2* and scale *s = 2*. Hence, see dgamma for the Gamma distribution.

Examples

```
require(graphics)

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

x <- 1:10
## Chi-squared(df = 2) is a special exponential distribution
all.equal(dchisq(x, df = 2), dexp(x, 1/2))
all.equal(pchisq(x, df = 2), pexp(x, 1/2))

## non-central RNG -- df = 0 with ncp > 0:  Z0 has point mass at 0!
Z0 <- rchisq(100, df = 0, ncp = 2.)
graphics::stem(Z0)

## visual testing
## do P-P plots for 1000 points at various degrees of freedom
L <- 1.2; n <- 1000; pp <- ppoints(n)
op <- par(mfrow = c(3,3), mar = c(3,3,1,1)+.1, mgp = c(1.5,.6,0),
          oma = c(0,0,3,0))
for(df in 2^(4*rnorm(9))) {
  plot(pp, sort(pchisq(rr <- rchisq(n, df = df, ncp = L), df = df, ncp = L)),
       ylab = "pchisq(rchisq(.),.)", pch = ".")
  mtext(paste("df = ", formatC(df, digits = 4)), line =  -2, adj = 0.05)
  abline(0, 1, col = 2)
}
mtext(expression("P-P plots : Noncentral  "*
                 chi^2 *"(n=1000, df=X, ncp= 1.2)"),
      cex = 1.5, font = 2, outer = TRUE)
par(op)

## "analytical" test
lam <- seq(0, 100, by = .25)
p00 <- pchisq(0,      df = 0, ncp = lam)
p.0 <- pchisq(1e-300, df = 0, ncp = lam)
stopifnot(all.equal(p00, exp(-lam/2)),
          all.equal(p.0, exp(-lam/2)))
```

lm {stats}

Fitting Linear Models

Description

lm is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although aov may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

formula

an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.

data

an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called.

subset

an optional vector specifying a subset of observations to be used in the fitting process.

weights

an optional vector of weights to be used in the fitting process. Should be NULL or a numeric vector. If non-NULL, weighted least squares is used with weights weights (that is, minimizing sum(w*e^2)); otherwise ordinary least squares is used. See also 'Details',

na.action

a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The 'factory-fresh' default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.

method

the method to be used; for fitting, currently only method = "qr" is supported; method = "model.frame" returns the model frame (the same as with model = TRUE, see below).

model, x, y, qr

logicals. If TRUE the corresponding components of the fit (the model frame, the model matrix, the response, the QR decomposition) are returned.

singular.ok

logical. If FALSE (the default in S but not in **R**) a singular fit is an error.

contrasts

an optional list. See the contrasts.arg of model.matrix.default.

offset

this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one are specified their sum is used. See model.offset.

...

additional arguments to be passed to the low level regression fitting functions (see below).

Details

Models for lm are specified symbolically. A typical model has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with duplicates removed. A specification of the form first:second indicates the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the *cross* of first and second. This is the same as first + second + first:second.

If the formula includes an offset, this is evaluated and subtracted from the response.

If response is a matrix a linear model is fitted separately by least-squares to each column of the matrix.

See model.matrix for some further details. The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a terms object as the formula (see aov and demo(glm.vr) for an example).

A formula has an implied intercept term. To remove this use either y ~ x - 1 or y ~ 0 + x. See formula for more details of allowed formulae.

Non-NULL weights can be used to indicate that different observations have different variances (with the values in weights being inversely proportional to the variances); or equivalently, when the elements of weights are positive

integers *w_i*, that each response *y_i* is the mean of *w_i* unit-weight observations (including the case that there are *w_i* observations equal to *y_i* and the data have been summarized).
lm calls the lower level functions lm.fit, etc, see below, for the actual numerical computations. For programming only, you may consider doing likewise.
All of weights, subset and offset are evaluated in the same way as variables in formula, that is first in data and then in the environment of formula.

Value

lm returns an object of class "lm" or for multiple responses of class c("mlm", "lm").
The functions summary and anova are used to obtain and print a summary and analysis of variance table of the results. The generic accessor functions coefficients, effects, fitted.values and residuals extract various useful features of the value returned by lm.
An object of class "lm" is a list containing at least the following components:
coefficients
a named vector of coefficients
residuals
the residuals, that is response minus fitted values.
fitted.values
the fitted mean values.
rank
the numeric rank of the fitted linear model.
weights
(only for weighted fits) the specified weights.
df.residual
the residual degrees of freedom.
call
the matched call.
terms
the terms object used.
contrasts
(only where relevant) the contrasts used.
xlevels
(only where relevant) a record of the levels of the factors used in fitting.
offset
the offset used (missing if none were used).
y
if requested, the response used.
x
if requested, the model matrix used.
model
if requested (the default), the model frame used.
na.action
(where relevant) information returned by model.frame on the special handling of NAs.
In addition, non-null fits will have components assign, effects and (unless not requested) qr relating to the linear fit, for use by extractor functions such as summary and effects.

Using time series

Considerable care is needed when using lm with time series.
Unless na.action = NULL, the time series attributes are stripped from the variables before the regression is done. (This is necessary as omitting NAs would invalidate the time series attributes, and if NAs are omitted in the middle of the series the result would no longer be a regular time series.)
Even if the time series attributes are retained, they are not used to line up series, so that the time shift of a lagged or differenced regressor would be ignored. It is good practice to prepare a data argument by ts.intersect(..., dframe = TRUE), then apply a suitable na.action to that data frame and call lm with na.action = NULL so that residuals and fitted values are time series.

Note

Offsets specified by offset will not be included in predictions by predict.lm, whereas those specified by an offset term in the formula will be.

Author(s)

The design was inspired by the S function of the same name described in Chambers (1992). The implementation of model formula by Ross Ihaka was based on Wilkinson & Rogers (1973).

References

Chambers, J. M. (1992) *Linear models.* Chapter 4 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Wilkinson, G. N. and Rogers, C. E. (1973) Symbolic descriptions of factorial models for analysis of variance. *Applied Statistics*, **22**, 392–9.

See Also

summary.lm for summaries and anova.lm for the ANOVA table; aov for a different interface.

The generic functions coef, effects, residuals, fitted, vcov.

predict.lm (via predict) for prediction, including confidence and prediction intervals; confint for confidence intervals of *parameters*.

lm.influence for regression diagnostics, and glm for **generalized** linear models.

The underlying low level functions, lm.fit for plain, and lm.wfit for weighted regression fitting.

More lm() examples are available e.g., in anscombe, attitude, freeny, LifeCycleSavings, longley, stackloss, swiss.

biglm in package **biglm** for an alternative way to fit linear models to large datasets (especially those with many cases).

Examples

```
require(graphics)

## 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.D90 <- lm(weight ~ group - 1) # omitting intercept

anova(lm.D9)
summary(lm.D90)

opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(lm.D9, las = 1)      # Residuals, Fitted, ...
par(opar)

### less simple examples in "See Also" above
```

aov {stats}

Fit an Analysis of Variance Model

Description

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

Usage

```
aov(formula, data = NULL, projections = FALSE, qr = TRUE,
   contrasts = NULL, ...)
```

Arguments

formula

A formula specifying the model.

data
A data frame in which the variables specified in the formula will be found. If missing, the variables are searched for in the standard way.
projections
Logical flag: should the projections be returned?
qr
Logical flag: should the QR decomposition be returned?
contrasts
A list of contrasts to be used for some of the factors in the formula. These are not used for any Error term, and supplying contrasts for factors only in the Error term will give a warning.
...
Arguments to be passed to lm, such as subset or na.action. See 'Details' about weights.

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 that 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.
Weights can be specified by a weights argument, but should not be used with an Error term, and are incompletely supported (e.g., not by model.tables).

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.

Note

aov is designed for balanced designs, and the results can be hard to interpret without balance: beware that missing values in the response(s) will likely lose the balance. If there are two or more error strata, the methods used are statistically inefficient without balance, and it may be better to use lme in package **nlme**.
Balance can be checked with the replications function.
The default 'contrasts' in **R** are not orthogonal contrasts, and aov and its helper functions will work better with such contrasts: see the examples for how to select these.

Author(s)

The design was inspired by the S function of the same name described in Chambers *et al* (1992).

References

Chambers, J. M., Freeny, A and Heiberger, R. M. (1992) *Analysis of variance; designed experiments.* Chapter 5 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

lm, summary.aov, replications, alias, proj, model.tables, TukeyHSD

Examples

```
## From Venables and Ripley (2002) p.165.

## Set orthogonal contrasts.
op <- options(contrasts = c("contr.helmert", "contr.poly"))
( npk.aov <- aov(yield ~ block + N*P*K, npk) )
summary(npk.aov)
coefficients(npk.aov)

## to show the effects of re-ordering terms contrast the two fits
aov(yield ~ block + N * P + K, npk)
aov(terms(yield ~ block + N * P + K, keep.order = TRUE), npk)


## as a test, not particularly sensible statistically
npk.aovE <- aov(yield ~  N*P*K + Error(block), npk)
```

```
npk.aovE
summary(npk.aovE)
options(op)  # reset to previous
```

binom.test {stats}

Exact Binomial Test

Description

Performs an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment.

Usage

```
binom.test(x, n, p = 0.5,
        alternative = c("two.sided", "less", "greater"),
        conf.level = 0.95)
```

Arguments

x
number of successes, or a vector of length 2 giving the numbers of successes and failures, respectively.
n
number of trials; ignored if x has length 2.
p
hypothesized probability of success.
alternative
indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter.
conf.level
confidence level for the returned confidence interval.

Details

Confidence intervals are obtained by a procedure first given in Clopper and Pearson (1934). This guarantees that the confidence level is at least conf.level, but in general does not give the shortest-length confidence intervals.

Value

A list with class "htest" containing the following components:
statistic
the number of successes.
parameter
the number of trials.
p.value
the p-value of the test.
conf.int
a confidence interval for the probability of success.
estimate
the estimated probability of success.
null.value
the probability of success under the null, p.
alternative
a character string describing the alternative hypothesis.
method
the character string "Exact binomial test".
data.name
a character string giving the names of the data.

References

Clopper, C. J. & Pearson, E. S. (1934). The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, **26**, 404–413.

William J. Conover (1971), *Practical nonparametric statistics*. New York: John Wiley & Sons. Pages 97–104.
Myles Hollander & Douglas A. Wolfe (1973), *Nonparametric Statistical Methods.* 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(c(682, 243), p = 3/4)
binom.test(682, 682 + 243, p = 3/4)   # The same.
## => Data are in agreement with the null hypothesis.
```

wilcox.test {stats}

Wilcoxon Rank Sum and Signed Rank Tests

Description

Performs one- and two-sample Wilcoxon tests on vectors of data; the latter is also known as ‘Mann-Whitney’ test.

Usage

```
wilcox.test(x, ...)

## Default S3 method:
wilcox.test(x, y = NULL,
            alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, exact = NULL, correct = TRUE,
            conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'formula'
wilcox.test(formula, data, subset, na.action, ...)
```

Arguments

x
numeric vector of data values. Non-finite (e.g. infinite or missing) values will be omitted.
y
an optional numeric vector of data values: as with x non-finite values will be omitted.
alternative
a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
mu
a number specifying an optional parameter used to form the null hypothesis. See ‘Details’.
paired
a logical indicating whether you want a paired test.
exact
a logical indicating whether an exact p-value should be computed.
correct
a logical indicating whether to apply continuity correction in the normal approximation for the p-value.
conf.int

a logical indicating whether a confidence interval should be computed.
conf.level
confidence level of the interval.
formula
a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups.
data
an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset
an optional vector specifying a subset of observations to be used.
na.action
a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...
further arguments to be passed to or from methods.

Details

The formula interface is only applicable for the 2-sample tests.
If 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 distribution of x (in the one sample case) or of x - y (in the paired two sample case) is symmetric about mu is performed.
Otherwise, if both x and y are given and paired is FALSE, a Wilcoxon rank sum test (equivalent to the Mann-Whitney test: see the Note) is carried out. In this case, the null hypothesis is that the distributions of x and y differ by a location shift of mu and the alternative is that they differ by some other location shift (and the one-sided alternative "greater" is that x is shifted to the right of y).
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.
Optionally (if argument conf.int is true), a nonparametric confidence interval and an estimator for the pseudomedian (one-sample case) or for the difference of the location parameters x-y is computed. (The pseudomedian of a distribution F is the median of the distribution of $(u+v)/2$, where u and v are independent, each with distribution F. If F is symmetric, then the pseudomedian and median coincide. See Hollander & Wolfe (1973), page 34.) Note that in the two-sample case the estimator for the difference in location parameters does **not** estimate the difference in medians (a common misconception) but rather the median of the difference between a sample from x and a sample from y.
If exact p-values are available, an exact confidence interval is obtained by the algorithm described in Bauer (1972), and the Hodges-Lehmann estimator is employed. Otherwise, the returned confidence interval and point estimate are based on normal approximations. These are continuity-corrected for the interval but *not* the estimate (as the correction depends on the alternative).
With small samples it may not be possible to achieve very high confidence interval coverages. If this happens a warning will be given and an interval with lower coverage will be substituted.

Value

A list with class "htest" containing the following components:
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.

conf.int
a confidence interval for the location parameter. (Only present if argument conf.int = TRUE.)
estimate
an estimate of the location parameter. (Only present if argument conf.int = TRUE.)

Warning

This function can use large amounts of memory and stack (and even crash **R** if the stack limit is exceeded) if exact = TRUE and one sample is large (several thousands or more).

Note

The literature is not unanimous about the definitions of the Wilcoxon rank sum and Mann-Whitney tests. The two most common definitions correspond to the sum of the ranks of the first sample with the minimum value subtracted or not: **R** subtracts and S-PLUS does not, giving a value which is larger by $m(m+1)/2$ for a first sample of size m. (It seems Wilcoxon's original paper used the unadjusted sum of the ranks but subsequent tables subtracted the minimum.)

R's value can also be computed as the number of all pairs (x[i], y[j]) for which y[j] is not greater than x[i], the most common definition of the Mann-Whitney test.

References

David F. Bauer (1972), Constructing confidence sets using rank statistics. *Journal of the American Statistical Association* **67**, 687–690.

Myles Hollander and Douglas A. Wolfe (1973), *Nonparametric Statistical Methods*. New York: John Wiley & Sons. Pages 27–33 (one-sample), 68–75 (two-sample).
Or second edition (1999).

See Also

psignrank, pwilcox.
wilcox_test in package **coin** for exact, asymptotic and Monte Carlo *conditional* p-values, including in the presence of ties.
kruskal.test for testing homogeneity in location parameters in the case of two or more samples; t.test for an alternative under normality assumptions [or large samples]

Examples

```
require(graphics)
## One-sample test.
## Hollander & Wolfe (1973), 29f.
## Hamilton depression scale factor measurements in 9 patients with
##  mixed anxiety and depression, taken at the first (x) and second
##  (y) visit after initiation of a therapy (administration of a
##  tranquilizer).
x <- c(1.83,  0.50,  1.62,  2.48, 1.68, 1.88, 1.55, 3.06, 1.30)
y <- c(0.878, 0.647, 0.598, 2.05, 1.06, 1.29, 1.06, 3.14, 1.29)
wilcox.test(x, y, paired = TRUE, alternative = "greater")
wilcox.test(y - x, alternative = "less")    # The same.
wilcox.test(y - x, alternative = "less",
        exact = FALSE, correct = FALSE) # H&W large sample
                                # approximation

## Two-sample test.
## Hollander & Wolfe (1973), 69f.
## Permeability constants of the human chorioamnion (a placental
##  membrane) at term (x) and between 12 to 26 weeks gestational
##  age (y).  The alternative of interest is greater permeability
##  of the human chorioamnion for the term pregnancy.
x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
wilcox.test(x, y, alternative = "g")       # greater
wilcox.test(x, y, alternative = "greater",
        exact = FALSE, correct = FALSE) # H&W large sample
                                # approximation
```

```
wilcox.test(rnorm(10), rnorm(10, 2), conf.int = TRUE)

## Formula interface.
boxplot(Ozone ~ Month, data = airquality)
wilcox.test(Ozone ~ Month, data = airquality,
        subset = Month %in% c(5, 8))
```

polygon {graphics}

Polygon Drawing

Description

polygon draws the polygons whose vertices are given in x and y.

Usage

```
polygon(x, y = NULL, density = NULL, angle = 45,
      border = NULL, col = NA, lty = par("lty"),
      ..., fillOddEven = FALSE)
```

Arguments

x, y
vectors containing the coordinates of the vertices of the polygon.

density
the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. A zero value of density means no shading nor filling whereas negative values and NA suppress shading (and so allow color filling).

angle
the slope of shading lines, given as an angle in degrees (counter-clockwise).

col
the color for filling the polygon. The default, NA, is to leave polygons unfilled, unless density is specified. (For back-compatibility, NULL is equivalent to NA.) If density is specified with a positive value this gives the color of the shading lines.

border
the color to draw the border. The default, NULL, means to use par("fg"). Use border = NA to omit borders.
For compatibility with S, border can also be logical, in which case FALSE is equivalent to NA (borders omitted) and TRUE is equivalent to NULL (use the foreground colour),

lty
the line type to be used, as in par.

...
graphical parameters such as xpd, lend, ljoin and lmitre can be given as arguments.

fillOddEven
logical controlling the polygon shading mode: see below for details. Default FALSE.

Details

The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, ... See xy.coords.
It is assumed that the polygon is to be closed by joining the last point to the first point.
The coordinates can contain missing values. The behaviour is similar to that of lines, except that instead of breaking a line into several lines, NA values break the polygon into several complete polygons (including closing the last point to the first point). See the examples below.
When multiple polygons are produced, the values of density, angle, col, border, and lty are recycled in the usual manner.
Shading of polygons is only implemented for linear plots: if either axis is on log scale then shading is omitted, with a warning.

Bugs

Self-intersecting polygons may be filled using either the “odd-even” or “non-zero” rule. These fill a region if the polygon border encircles it an odd or non-zero number of times, respectively. Shading lines are handled internally by R according to the fillOddEven argument, but device-based solid fills depend on the graphics device. The windows, pdf and postscript devices have their own fillOddEven argument to control this.

Author(s)

The code implementing polygon shading was donated by Kevin Buhr buhr@stat.wisc.edu.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

segments for even more flexibility, lines, rect, box, abline.
par for how to specify colors.

Examples

```
x <- c(1:9, 8:1)
y <- c(1, 2*(5:3), 2, -1, 17, 9, 8, 2:9)
op <- par(mfcol = c(3, 1))
for(xpd in c(FALSE, TRUE, NA)) {
  plot(1:10, main = paste("xpd =", xpd))
  box("figure", col = "pink", lwd = 3)
  polygon(x, y, xpd = xpd, col = "orange", lty = 2, lwd = 2, border = "red")
}
par(op)

n <- 100
xx <- c(0:n, n:0)
yy <- c(c(0, cumsum(stats::rnorm(n))), rev(c(0, cumsum(stats::rnorm(n)))))
plot   (xx, yy, type = "n", xlab = "Time", ylab = "Distance")
polygon(xx, yy, col = "gray", border = "red")
title("Distance Between Brownian Motions")

# Multiple polygons from NA values
# and recycling of col, border, and lty
op <- par(mfrow = c(2, 1))
plot(c(1, 9), 1:2, type = "n")
polygon(1:9, c(2,1,2,1,1,2,1,2,1),
        col = c("red", "blue"),
        border = c("green", "yellow"),
        lwd = 3, lty = c("dashed", "solid"))
plot(c(1, 9), 1:2, type = "n")
polygon(1:9, c(2,1,2,1,NA,2,1,2,1),
        col = c("red", "blue"),
        border = c("green", "yellow"),
        lwd = 3, lty = c("dashed", "solid"))
par(op)

# Line-shaded polygons
plot(c(1, 9), 1:2, type = "n")
polygon(1:9, c(2,1,2,1,NA,2,1,2,1),
        density = c(10, 20), angle = c(-45, 45))
```

text {graphics}

Add Text to a Plot

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, ...)

## Default S3 method:
 text(x, y = NULL, labels = seq_along(x), adj = NULL,
    pos = NULL, offset = 0.5, vfont = NULL,
    cex = 1, col = NULL, font = NULL, ...)
```

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

a character vector or expression specifying the *text* to be written. An attempt is made to coerce other language objects (names and calls) to expressions, and vectors and other classed objects to character vectors by as.character. If labels is longer than x and y, the coordinates are recycled to the length of labels.

adj

one or two values in *[0, 1]* which specify the x (and optionally y) adjustment of the labels. On most devices values outside that interval will also work.

pos

a position specifier for the text. If specified this overrides any adj value given. Values of 1, 2, 3 and 4, respectively indicate positions below, to the left of, above and to the right of the specified coordinates.

offset

when pos is specified, this value gives the offset of the label from the specified coordinate in fractions of a character width.

vfont

NULL for the current font family, or a character vector of length 2 for Hershey vector fonts. The first element of the vector selects a typeface and the second element selects a style. Ignored if labels is an expression.

cex

numeric **c**haracter **ex**pansion factor; multiplied by par("cex") yields the final character size. NULL and NA are equivalent to 1.0.

col, font

the color and (if vfont = NULL) font to be used, possibly vectors. These default to the values of the global graphical parameters in par().

...

further graphical parameters (from par), such as srt, family and xpd.

Details

labels must be of type character or expression (or be coercible to such a type). In the latter case, quite a bit of mathematical notation is available such as sub- and superscripts, greek letters, fractions, etc.

adj allows *adj*ustment of the text with respect to (x, y). Values of 0, 0.5, and 1 specify left/bottom, middle and right/top alignment, respectively. The default is for centered text, i.e., adj = c(0.5, NA). Accurate vertical centering needs character metric information on individual characters which is only available on some devices. Vertical alignment is done slightly differently for character strings and for expressions: adj = c(0,0) means to left-justify and to align on the baseline for strings but on the bottom of the bounding box for expressions. This also affects vertical centering: for strings the centering excludes any descenders whereas for expressions it includes them. Using NA for strings centers them, including descenders.

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). NA values of font are replaced by par("font"), and similarly for col.

Labels whose x, y or labels value is NA are omitted from the plot.
What happens when font = 5 (the symbol font) is selected can be both device- and locale-dependent. Most often labels will be interpreted in the Adobe symbol encoding, so e.g. "d" is delta, and "\300" is aleph.

Euro symbol

The Euro symbol was introduced relatively recently and may not be available in older fonts. In recent versions of Adobe symbol fonts it is character 160, so text(x, y, "\xA0", font = 5) may work. People using Western European locales on Unix-alikes can probably select ISO-8895-15 (Latin-9) which has the Euro as character 165: this can also be used for postscript and pdf. It is \u20ac in Unicode, which can be used in UTF-8 locales.
The Euro should be rendered correctly by X11 in UTF-8 locales, but the corresponding single-byte encoding in postscript and pdf will need to be selected as ISOLatin9.enc.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

text.formula for the formula method; 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 ©, but not ® ...")
mtext("«Latin-1 accented chars»: éè øØ å<Å æ<Æ", side = 3)
points(c(6,2), c(2,1), pch = 3, cex = 4, col = "red")
text(6, 2, "the text is CENTERED around (x,y) = (6,2) by default",
    cex = .8)
text(2, 1, "or Left/Bottom - JUSTIFIED at (2,1) by 'adj = c(0,0)'",
    adj = c(0,0))
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y)",
    cex = .75)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))

## Two more latin1 examples
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)")
```

readWorksheet-methods {XLConnect}

Reading data from worksheets

Description

Reads data from worksheets of a workbook.

Usage

```
## S4 method for signature 'workbook,numeric'
readWorksheet(object,sheet,startRow,startCol,endRow,endCol,autofitRow,autofitCol,
region,header,rownames,colTypes,forceConversion,dateTimeFormat,check.names,
```

useCachedValues,keep,drop, simplify, readStrategy)
S4 method for signature 'workbook,character'
readWorksheet(object,sheet,startRow,startCol,endRow,endCol,autofitRow,autofitCol,
region,header,rownames,colTypes,forceConversion,dateTimeFormat,check.names,
useCachedValues,keep,drop, simplify, readStrategy)

Arguments

object
The workbook to use
sheet
The name or index of the worksheet to read from
startRow
The index of the first row to read from. Defaults to 0 meaning that the start row is determined automatically.
startCol
The index of the first column to read from. Defaults to 0 meaning that the start column is determined automatically.
endRow
The index of the last row to read from. Defaults to 0 meaning that the end row is determined automatically.
endCol
The index of the last column to read from. Defaults to 0 meaning that the end column is determined automatically.
autofitRow
logical specifying if leading and trailing empty rows should be skipped. Defaults to TRUE.
autofitCol
logical specifying if leading and trailing empty columns should be skipped. Defaults to TRUE.
region
A range specifier in the form 'A10:B18'. This provides an alternative way to specify startRow, startCol, endRow and endCol. Range specifications take precedence over index specifications.
header
Interpret the first row of the specified area as column headers. The default is TRUE.
rownames
Index (numeric) or name (character) of column that should be used as row names. The corresponding column will be removed from the data set. Defaults to NULL which means that no row names are applied. Row names must be either integer or character. Non-numeric columns will be coerced to character.
colTypes
Column types to use when reading in the data. Specified as a character vector of the corresponding type names (see XLC; XLC$DATA_TYPE.<?>). You may also use R class names such as numeric, character, logical and POSIXt. The types are applied in the given order to the columns - elements are recycled if necessary. Defaults to character(0) meaning that column types are determined automatically (see the Note section for more information).
By default, type conversions are only applied if the specified column type is a more generic type (e.g. from Numeric to String) - otherwise NA is returned. The forceConversion flag can be set to force conversion into less generic types where possible.
forceConversion
logical specifying if conversions to less generic types should be forced. Defaults to FALSE meaning that if a column is specified to be of a certain type via the colTypes argument and a more generic type is detected in the column, then NA will be returned (example: column is specified to be DateTime but a more generic String is found). Specifying forceConversion = TRUE will try to enforce a conversion - if it succeeds the corresponding (converted) value will be returned, otherwise NA. See the Note section for some additional information.
dateTimeFormat
Date/time format used when doing date/time conversions. Defaults to getOption("XLConnect.dateTimeFormat"). This should be a POSIX format specifier according to strptime although not all specifications have been implemented yet - the most important ones however are available.
check.names
logical specifying if column names of the resulting data.frame should be checked to ensure that they are syntactically valid variable names and are not duplicated. See the check.names argument of data.frame. Defaults to TRUE.
useCachedValues

logical specifying whether to read cached formula results from the workbook instead of re-evaluating them. This is particularly helpful in cases for reading data produced by Excel features not supported in XLConnect like references to external workbooks. Defaults to FALSE, which means that formulas will be evaluated by XLConnect.

keep

Vector of column names or indices to be kept in the output data frame. It is possible to specify either keep or drop, but not both at the same time. Defaults to NULL. If a vector is passed as argument, it will be wrapped into a list. This list gets replicated to match the length of the other arguments. Example: if sheet = c("Sheet1", "Sheet2", "Sheet3") and keep = c(1,2), keep will be internally converted into list(c(1,2)) and then replicated to match the number of sheets, i.e. keep = list(c(1,2), c(1,2), c(1,2)). The result is that the first two columns of each sheet are kept. If keep = list(1,2) is specified, it will be replicated as list(1,2,1), i.e. respectively the first, second and first column of the sheets "Sheet1", "Sheet2", "Sheet3" will be kept.

drop

Vector of column names or indices to be dropped in the output data frame. It is possible to specify either keep or drop, but not both at the same time. Defaults to NULL. If a vector is passed as argument, it will be wrapped into a list. This list gets replicated to match the length of the other arguments. Example: if sheet = c("Sheet1", "Sheet2", "Sheet3") and drop = c(1,2), drop will be internally converted into list(c(1,2)) and then replicated to match the number of sheets, i.e. drop = list(c(1,2), c(1,2), c(1,2)). The result is that the first two columns of each sheet are dropped. If drop = list(1,2) is specified, it will be replicated as list(1,2,1), i.e. respectively the first, second and first column of the sheets "Sheet1", "Sheet2", "Sheet3" will be dropped.

simplify

logical specifying if the result should be simplified, e.g. in case the data.frame would only have one row or one column (and data types match). Simplifying here is identical to calling unlist on the otherwise resulting data.frame (using use.names = FALSE). The default is FALSE.

readStrategy

character specifying the reading strategy to use. Currently supported strategies are:

- "default" (default): Can handle all supported data types incl. date/time values and can deal directly with missing value identifiers (see setMissingValue)
- "fast": Increased read performance. Date/time values are read as numeric (number of days since 1900-01-01; fractional days represent hours, minutes, and seconds) and only blank cells are recognized as missing (missing value identifiers as set in setMissingValue are ignored)

Details

Reads data from the worksheet specified by sheet. Data is read starting at the top left corner specified by startRow and startCol down to the bottom right corner specified by endRow and endCol. If header = TRUE, the first row is interpreted as column names of the resulting data.frame.

If startRow <= 0 then the first available row in the sheet is assumed. If endRow <= 0 then the last available row in the sheet is assumed. If startCol <= 0 then the minimum column between startRow and endRow is assumed. If endCol <= 0 then the maximum column between startRow and endRow is assumed. In other words, if no boundaries are specified readWorksheet assumes the "bounding box" of the data as the corresponding boundaries.

The arguments autofitRow and autofitCol (both defaulting to TRUE) can be used to skip leading and trailing empty rows even in case startRow, endRow, startCol and endCol are specified to values > 0. This can be useful if data is expected within certain given boundaries but the exact location is not available.

If all four coordinate arguments are missing this behaves as above with startRow = 0, startCol = 0, endRow = 0 and endCol = 0. In this case readWorksheet assumes the "bounding box" of the data as the corresponding boundaries.

All arguments (except object) are vectorized. As such, multiple worksheets (and also multiple data regions from the same worksheet) can be read with one method call. If only one single data region is read, the return value is a data.frame. If multiple data regions are specified, the return value is a list of data.frame's returned in the order they have been specified. If worksheets have been specified by name, the list will be a named list named by the corresponding worksheets.

Note

If no specific column types (see argument colTypes) are specified, readWorksheet tries to determine the resulting column types based on the read cell types. If different cell types are found in a specific column, the most general of those is used and mapped to the corresponding R data type. The order of data types from least to most general is Boolean (logical) < DateTime (POSIXct) < Numeric (numeric) < String (character). E.g. if a column is read that contains cells of type Boolean, Numeric and String then the resulting column in R would be character since character is the most general type.

Some additional information with respect to forcing data type conversion using forceConversion = TRUE:

- Forcing conversion from String to Boolean: TRUE is returned if and only if the target string is "true" (ignoring any capitalization). Any other string will return FALSE.
- Forcing conversion from Numeric to DateTime: since Excel understands Dates/Times as Numerics with some additional formatting, a conversion from a Numeric to a DateTime is actually possible. Numerics in this case represent the number of days since 1900-01-00 (yes, day 00! - see http://www.cpearson.com/excel/datetime.htm). Note that in R 0 is represented as 1899-12-31 since there is no 1900-01-00. Fractional days represent hours, minutes, and seconds.

Author(s)

Martin Studer
Thomas Themel
Nicola Lambiase
Mirai Solutions GmbH http://www.mirai-solutions.com

See Also

workbook, writeWorksheet, readNamedRegion, writeNamedRegion, readWorksheetFromFile, readTable, onErrorCell

Examples

```
## Example 1:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read worksheet 'mtcars' (providing no specific area bounds;
# with default header = TRUE)
data <- readWorksheet(wb, sheet = "mtcars")


## Example 2:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read worksheet 'mtcars' (providing area bounds; with default header = TRUE)
data <- readWorksheet(wb, sheet = "mtcars", startRow = 1, startCol = 3,
                      endRow = 15, endCol = 8)


## Example 3:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read worksheet 'mtcars' (providing area bounds using the region argument;
# with default header = TRUE)
data <- readWorksheet(wb, sheet = "mtcars", region = "C1:H15")
```

```
## Example 4:
# conversion xlsx file from demoFiles subfolder of package XLConnect
excelFile <- system.file("demoFiles/conversion.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(excelFile)

# Read worksheet 'Conversion' with pre-specified column types
# Note: in the worksheet all data was entered as strings!
# forceConversion = TRUE is used to force conversion from String
# into the less generic data types Numeric, DateTime & Boolean
df <- readWorksheet(wb, sheet = "Conversion", header = TRUE,
                    colTypes = c(XLC$DATA_TYPE.NUMERIC,
                                 XLC$DATA_TYPE.DATETIME,
                                 XLC$DATA_TYPE.BOOLEAN),
                    forceConversion = TRUE,
                    dateTimeFormat = "%Y-%m-%d %H:%M:%S")

## Example 5:
# mtcars xlsx file from demoFiles subfolder of package XLConnect
demoExcelFile <- system.file("demoFiles/mtcars.xlsx", package = "XLConnect")

# Load workbook
wb <- loadWorkbook(demoExcelFile)

# Read the columns 1, 3 and 5 from the sheet 'mtcars' (with default header = TRUE)
data <- readWorksheet(wb, sheet = "mtcars", keep=c(1,3,5))
```

loadWorkbook {XLConnect}

Loading Microsoft Excel workbooks

Description

Loads or creates a Microsoft Excel workbook for further manipulation.

Usage

```
loadWorkbook(filename, create = FALSE)
```

Arguments

filename

Filename (absolute or relative) of Excel workbook to be loaded. Supported are Excel '97 (*.xls) and OOXML (Excel 2007+, *.xlsx) file formats. Paths are expanded using path.expand.

create

Specifies if the file should be created if it does not already exist (default is FALSE). Note that create = TRUE has no effect if the specified file exists, i.e. an existing file is loaded and not being recreated if create = TRUE.

Value

Returns a workbook object for further manipulation.

Note

loadWorkbook is basically just a shortcut form of new("workbook", filename, create) with some additional error checking. As such it is the preferred way of creating workbook instances.

Author(s)

Martin Studer

Mirai Solutions GmbH http://www.mirai-solutions.com

References

Wikipedia: Office Open XML
http://en.wikipedia.org/wiki/Office_Open_XML

See Also

workbook, saveWorkbook

Examples

```
# Load existing demo Excel file 'mtcars.xlsx' from the XLConnect package
wb.mtcars <- loadWorkbook(system.file("demoFiles/mtcars.xlsx",
                    package = "XLConnect"))

# Create new workbook
wb.new <- loadWorkbook("myNewExcelFile.xlsx", create = TRUE)

# NOTE: The above statement does not write the file to disk!
# saveWorkbook(wb.new) would need to be called in order to write/save
# the file to disk!
```