

prada

March 24, 2012

`analysePlate` *Apply a statistic to the data from each well in a plate*

Description

Apply a statistic to the data from each well in a plate

Usage

```
analysePlate(x, wellcol="well", wellrange, statfun, platenname, plotdir=".", ...)
```

Arguments

<code>x</code>	data frame. It must contain a column whose name is the value of <code>wellcol</code> , and further columns that are needed by the function named by <code>stat</code> .
<code>wellcol</code>	character of length 1. Name of a column in <code>x</code> that contains the well ID.
<code>wellrange</code>	vector of the same type as <code>x[, wellcol]</code> . All values <code>x[, wellcol]</code> must be contained in <code>wellrange</code> .
<code>statfun</code>	character of length 1. Name of a function that can calculate a statistic from selected rows of <code>x</code> .
<code>platenname</code>	character of length 1. The name or ID of this plate, which will be used for graphics output filenames and as the value of the column <code>platenname</code> of the return value.
<code>plotdir</code>	character of length 1. The name of directory where diagnostic plots will be saved.
<code>...</code>	further arguments that are passed on to <code>statfun</code> .

Details

The semantics of this function are similar to `tapply`, but some additional checking and reporting is performed, and the return value is a data frame.

Value

A data frame with number of rows equal to `length(wellrange)`. Rows (wells) for which there is no data contains `NA`s. The columns comprise `platenname`, `well-ID` (from `x[, wellcol]`), and the results from `statfun`.

Author(s)

Wolfgang Huber

Examples

```
##see vignette
```

as.all

Coercion without introduction of NAs

Description

Coercion without introduction of NAs

Usage

```
as.all(x, what)
```

Arguments

x	an object.
what	character of length 1.

Details

The function calls `do.call(paste("as.", what, sep=""), list(x))`, and checks whether any NAs were introduced.

Value

A vector of type what

Author(s)

Wolfgang Huber

See Also

[as](#)

Examples

```
as.all(runif(5)*10, "integer")
```

barploterrbar *Barplot with error bars.*

Description

Barplot with error bars.

Usage

```
barploterrbar(y, yl, yh, barcol="orange", errcol="black", horiz=FALSE,
w=0.2, ylim=c(0, max(yh)*1.05), ...)
```

Arguments

y	Numeric vector.
yl	Numeric vector of same length as y.
yh	Numeric vector of same length as y.
barcol	Color of the bars.
errcol	Color of the error bars.
horiz	Logical. As in barplot .
w	The plot limits. The default value wil cause the error bars to fit nicely on the plotting device.
ylim	Size of the error bar ticks.
...	Further arguments that get passed on to barplot .

Details

The function calls [barplot](#) with y and decorates it with error bars according to yl and yh.

Value

The function is called for its side effect, producing a plot.

Author(s)

Wolfgang Huber <http://www.dkfz.de/abt0840/whuber>

See Also

[barplot](#)

Examples

```
y <- matrix(runif(80), ncol=5)
ym <- apply(y, 2, mean)
dy <- apply(y, 2, sd)*2/sqrt(nrow(y))
barploterrbar(ym, ym-dy, ym+dy, barcol="#0000c0", errcol="orange",
ylim=c(0, max(ym+dy)))
```

combineFrames	<i>Combine the cytoFrames within a cytoSet according to some grouping factor</i>
---------------	--

Description

Combine the cytoFrames within a cytoSet according to some grouping factor.

Usage

```
combineFrames(x, by)
```

Arguments

x	cytoSet.
by	factor. Length must be same as that of x.

Value

cytoSet.

Author(s)

Wolfgang Huber <huber@ebi.ac.uk>

Examples

```
cset <- readCytoSet(path=system.file("extdata", package="prada"),
                    pattern="[A-Z][0-9][0-9]$")
nr1 <- csApply(cset, nrow)
sm1 <- csApply(cset, sum)

fac <- factor(c(1,1,2,2,2,2))
cc <- combineFrames(cset, fac)
nr2 <- csApply(cc, nrow)
sm2 <- csApply(cc, sum)

stopifnot(all(nr2==tapply(nr1, fac, sum)))
stopifnot(all(sm2==tapply(sm1, fac, sum)))
```

csApply	<i>Apply a function over the intensities in a cytoSet</i>
---------	---

Description

This is a wrapper for [sapply](#) for objects of class [cytoSet](#).

Usage

```
csApply(X, FUN, ..., simplify = TRUE)
```

Arguments

X	cytoSet.
FUN	the function to be applied.
...	optional arguments to FUN.
simplify	logical; should the result be simplified to a vector or matrix if possible? Gets passed on the <code>sapply</code> .

Details

A wrapper for `sapply`.

Value

Like `sapply`: If FUN always returns a scalar, then the value of this function is a named vector. If FUN always returns a vector of length n, then the value of this function is an $n \times \text{length}(X)$ matrix with dimnames. Else, the value of this function is a named list whose values are the return values of the individual calls to FUN.

Author(s)

Wolfgang Huber <http://www.ebi.ac.uk/huber>

See Also

`sapply`

Examples

```
cset=readCytoSet(path=system.file("extdata", package="prada"),
  pattern="[A-Z][0-9][0-9]$")
csApply(cset, nrow)
csApply(cset, colMeans)
```

cytoFrame-class	<i>'cytoFrame': a class for storing observed quantitative properties from a population of cells, most likely from a FACS run or, alternatively, from automated microscopy</i>
-----------------	---

Description

This class represents the data contained in a FCS 3.0 file or similar data structures.

Details

Although objects of class `cytoFrame` can be used to hold arbitrary data of cell populations, the main focus lies on flow-cytometry data.

FCS 3.0 is the Data File Standard for Flow Cytometry, Version FCS3.0. See the vignette of this package for additional information on using the object system for handling of flow-cytometry data.

Creating Objects

Objects can be created using

```
new('cytoFrame',
    exprs = ....., # Object of class matrix
    description = .... # Object of class character
)
```

or the function [readFCS](#).

Slots

exprs: Object of class `matrix` containing the measured intensities. Rows correspond to cells, columns to the different channels. The `colnames` attribute of the matrix is supposed to hold the names or identifiers for the channels. The `rownames` attribute would usually not be set.

description: A named character vector containing the experiment description as key-value pairs.

well: A single integer vector giving the position of the well on a microtitre plate. This only applies when using the object within a `cytoSet` collection and will usually be filled in by the function [readCytoSet](#).

gate: An object of class `gateSet`. This object can be used to select defined subsets of the data, a process referred to as `gating` in the analysis of flow-cytometry data.

Methods

[subsetting. Returns an object of class `cytoFrame`. The subsetting is applied to the `exprs` slot, while the `description` slot is unchanged.

exprs, exprs<- extract or replace the intensities.

description, description<- extract or replace the description.

show display summary.

plot scatterplot for `cytoFrame` objects. The additional argument `gate` can be used to plot subsets of the data defined by either an object of class `gate` or by a character vector giving the name of one of the gates in the list.

gate, gate<- extract or replace the list of gates.

ncol, nrow extract the dimensions of the data matrix.

appendGate Append a gate or `gateSet` to the gate slot.

drawGate Create an object of class `gate` or `gateSet` based on a selection made from the data.

hist Draw a histogram of the data

Author(s)

Florian Hahne, Wolfgang Huber

See Also

[readFCS](#), [cytoSet](#), [gate](#), [gateSet](#)

Examples

```

intens <- matrix(runif(100), ncol=4)
colnames(intens) <- c("FL1-H", "FL2-H", "FL3-H", "FL4-H")

a <- new("cytoFrame",
        exprs=intens,
        description=c(name="example data", date=date()))

description(a)
dim(exprs(a))

a[1:3, -4]

plot(a)
## Not run:
g1 <- drawGate(a, name="Gate1")

## End(Not run)

```

cytoSet-class	<i>'cytoSet': a class for storing raw data from a quantitative cell-based assay</i>
---------------	---

Description

This class is a container for a set of [cytoFrame](#) objects

Creating Objects

Objects can be created using the function [readCytoSet](#) or via

```

new('cytoSet',
  frames = ..., # environment with cytoFrames
  phenoData = ... # object of class phenoData
  colnames = ... # object of class character
)

```

Slots

frames: An [environment](#) containing one or more [cytoFrame](#) objects.

phenoData: A [phenoData](#). Each row corresponds to one of the cytoFrames in the frames slot. It is mandatory that the pData has column named name

colnames: A [character](#) object with the (common) column names of all the data matrices in the cytoFrames.

Methods

[, [[subsetting. If *x* is [cytoSet](#), then *x*[*i*] returns a [cytoSet](#) object, and *x*[[*i*]] a [cytoFrame](#) object. The semantics is similar to the behavior of the subsetting operators for lists.

colnames, colnames<- extract or replace the `colnames` slot.

phenoData, phenoData<- extract or replace the `phenoData` slot.

show display summary.

plot Scatterplot of one or all (consecutively) `cytoFrame` objects. The additional argument `gate` can be used to plot subsets of the data defined by an object of class `gate` or `gateSet`.

hist Draw histogram of the data. The additional argument `variable` can be used to subset to one variable prior to plotting.

Important note on storage and performance

The bulk of the data in a `cytoSet` object is stored in an `environment`, and is therefore not automatically copied when the `cytoSet` object is copied. If `x` is an object of class `cytoSet`, then the code

```
y <- x
```

will create a an object `y` that contains copies of the `phenoData` and administrative data in `x`, but refers to the *same* environment with the actual fluorescence data. See below for how to create proper copies.

The reason for this is performance. The pass-by-value semantics of function calls in R can result in numerous copies of the same data object being made in the course of a series of nested function calls. If the data object is large, this can result in a considerable cost of memory and performance. `cytoSet` objects are intended to contain experimental data in the order of hundreds of Megabytes, which can effectively be treated as read-only: typical tasks are the extraction of subsets and the calculation of summary statistics. This is afforded by the design of the `cytoSet` class: an object of that class contains a `phenoData` slot, some administrative information, and a *reference* to an environment with the fluorescence data; when it is copied, only the reference is copied, but not the potentially large set of fluorescence data themselves.

However, note that subsetting operations, such as

```
y <- x[i]
```

do create proper copies, including a copy of the appropriate part of the fluorescence data, as it should be expected. Thus, to make a proper copy of a `cytoSet` `x`, use

```
y <- x[seq(along=x)]
```

Author(s)

Florian Hahne, Wolfgang Huber <http://www.ebi.ac.uk/huber>

See Also

[readCytoSet](#), [cytoFrame](#), [gate](#), [gateSet](#)

Examples

```
cset<-readCytoSet(path=system.file("extdata", package="prada"),
  pattern="[A-Z][0-9][0-9]$")
cset
pData(cset)
cset[[1]]
cset[["fas-Bcl2-plate323-04-04.A02"]]
cset["fas-Bcl2-plate323-04-04.A02"]
cset[1:3]
```



```

cset[[1]] <- exprs(cset[[1]])[1:100, ]

## Not run:
plot(cset[2])

## End(Not run)

```

cframe	<i>A sample cytoFrame object - German Cancer Research Center Heidelberg -</i>
--------	---

Description

Archived `cytoFrame` object from a MAP kinase screen conducted at the German Cancer Research Center Heidelberg. In the fluorescence channel 3 the expression of a YFP tag and in channel 7 the activation state of ERK2 was measured.

Usage

```
##cytoFrame object, see examples for details
```

Format

cytoFrame object

Source

German Cancer Research Center Heidelberg, Germany

Examples

```
data(cytoFrame)
```

cset	<i>A sample cytoSet object - German Cancer Research Center Heidelberg -</i>
------	---

Description

Archived `cytoSet` object from a MAP kinase screen conducted at the German Cancer Research Center Heidelberg. In the fluorescence channel 3 the expression of a YFP tag and in channel 7 the activation state of ERK2 was measured. The set contains measurements from 5 wells of a 96 well plate

Usage

```
##cytoSet object, see examples for details
```

Format

cytoSet object

Source

German Cancer Research Center Heidelberg, Germany

Examples

```
data(cytoSet)
```

devDims

Device Dimensions for plate plots

Description

Calculate device dimensions for plate plots

Usage

```
devDims(width, height, ncol=12, nrow=8, res=72)
```

Arguments

width	Device width in inches.
height	Device width in inches.
ncol	Number of columns for plate plot.
nrow	Number of rows for plate plot.
res	The resolution of the graphic device used for plotting.

Details

The function computes the device dimensions needed to create plate plots that fit perfectly in the device. This is necessary to retain the aspect ratio of the plots.

One of width or height need to be specified, the missing value will be computed.

Value

A list with items `width`, `height`, `pwidth` and `pheight`. These are the width and height values in inches and pixels respectively.

Author(s)

Florian Hahne

See Also

[plotPlate](#)

Examples

```
devDims (width=10)
```

`devRes`*Resolution of current plotting device*

Description

Calculates what R thinks to be the resolution of the current graphic device.

Usage

```
devRes ()
```

Details

This function may be used to get the resolution of the current graphics device. This can be important when calculating pixel coordinates for the output graphic.

Value

A vector with items `xres` and `yres`, the resolution in `x` and `y` direction respectively.

Author(s)

Florian Hahne

See Also

[plotPlate](#)

Examples

```
devRes ()
```

`fitNorm2`*Fit bivariate normal distribution.*

Description

Fits a bivariate normal distribution into a data set of paired values and selects data points according to their standard deviation from the fitted distribution.

Usage

```
fitNorm2(x, y=NA, scalefac=1, method="covMcd", noise, gateName = "fitNorm")
```

Arguments

<code>x</code>	Numeric vector containing x-value or n by 2 matrix containing x and y values or object of class <code>cytoFrame</code> .
<code>y</code>	Numeric vector containing y-value (optional). The length of <code>x</code> must be the same as that of <code>y</code> .
<code>scalefac</code>	Numeric vector giving factor of standard deviations used for data selection (all points within <code>scalefac</code> standard deviations are selected).
<code>method</code>	One of <code>covMcd</code> or <code>cov.rob</code> defining method used for computation of covariance matrix.
<code>noise</code>	Numeric or logical index vector defining value pairs in <code>x</code> that are not used for fitting of distributions. Can be used to deal with noisy data.
<code>gateName</code>	Character giving the name of the gate object.

Details

Computes the densities of a bivariate normal distribution from the covariance matrix of the paired data. Covariance matrices are acquired either by function `covMcd` (considerably faster) or by function `cov.rob`.

Value

A list containing items `mu` (midpoint of distribution), `S` (covariance matrix), `p` (density values for each data pair), `sel` (selection of data points), `scalefac` (factor of standard deviations used for data selection), `data` (x and y values of data points) and `gate`, an object of class `gate` containing the selection.

Author(s)

Florian Hahne

See Also

`cov.rob`, `covMcd`, `plotNorm2`

Examples

```
sampdat <- readFCS(system.file("extdata",
  "fas-Bcl2-plate323-04-04.A01", package="prada"))
nfit <- fitNorm2(exprs(sampdat[,1:2]), scalefac=2)
plotNorm2(nfit, selection=TRUE, ellipse=TRUE)
```

gate-class

'gate': a class for subsetting flow-cytometry data by defining regions in two-dimensional projections of the data

Description

In flow-cytometry analysis, regions in two-dimensional projections of the data space often have to be selected. Objects of this class can store the properties of these selections.

Creating Objects

```
Objects can be created using methods of the generic function drawGate or via
new("gate",
  gateFun = ....., # function returning logical vector
  colnames = ..... # object of class character and length 2
  logic = ..... # object of class character
)
```

Slots

name: A character vector for the name of the `gate` object. You can reference the object by its name for subsequent operations (e.g. plotting).

gateFun: A function call together with necessary arguments to produce a logical vector when applied on the data.

colnames: The colnames of the data matrix to which the gating function is to be applied.

logic: A character object, either `&` or `|`. This specifies the logical operation that will be applied when combining the selection from the `gate` with other object of that class. See `link{gateSet}` for additional information on combining gates.

type: A character giving the type of the object. This is currently not used but might become important in the future.

boundaries: A matrix with two columns giving the boundaries of the gate in two dimensional space. Can be used to superimpose the gate boundaries on a plot using `lines()`.

Methods

applyGate: `applyGate(x, data)` applies the gating of object `x` on data objects of class `cytoFrame` or `matrix`. In the former case `x` may be of class `gate`, `gateSet`, `character`, `numeric` or `logical`. See vignette for details.

show display summary.

names, names<- access and replace slot name.

as.gateSet Convert `gate` object to `gateSet` object

combineGates Combine multiple `gate` objects to one `gateSet` object

lines Draw the boundaries of the gate.

Author(s)

Florian Hahne

See Also

`cytoFrame`, `gateSet`

Examples

```
sampdat <- readFCS(system.file("extdata", "fas-Bcl2-plate323-04-04.A01",
                             package="prada"))
g1 <- new("gate", name="test1", gateFun=function(x)x[,"FSC-H"]<500, logic="&",
         colnames="FSC-H", type="misc")
g1
```

```

g2 <- new("gate", name="test2", gateFun=function(x)x[, "SSC-H"]>800, logic("&",
        colnames="SSC-H", type="misc")
gs1 <- combineGates(g1,g2)
gs2 <- as.gateSet(g2)
names(g1)
names(g1) <- "testName"
applyGate(sampdat, g1)
applyGate(exprs(sampdat), g2)
gate(sampdat) <- g1
applyGate(sampdat, 1)
applyGate(sampdat, "testName")
applyGate(sampdat, TRUE)

```

gateSet-class	<i>'gateSet': a class for subsetting flow-cytometry data by defining multiple regions in two-dimensional projections of the data</i>
---------------	--

Description

In flow-cytometry analysis, regions in two-dimensional projections of the data space often have to be selected. Objects of this class can store the properties for several of these selections

Creating Objects

Objects can be created using methods of the generic function `drawGate` or via

```

new("gateSet",
  glist = ..., # object of class list
)

```

Slots

name: Object of class `character` giving the name of the object. You can reference the object by its name for subsequent operations (e.g. plotting).

glist: Object of class `"list"` with items of class `gate`. The individual `gate` objects will be combined according to the value of their slot `logic`.

Methods

applyGate: `applyGate(x, data)` applies the gating of object `x` on data objects of class `cytoFrame` or `matrix`

length length of slot `glist`

show display summary

names, names<- extract or replace the names of the individual `gate` objects.

[subset to `gateSet` objects.

[[subset to individual `gate` objects.

appendGates append a `gate` or `gateSet` to a `cytoFrame`

Author(s)

Florian Hahne

See Also

[cytoFrame](#), [gate](#)

Examples

```
sampdat <- readFCS(system.file("extdata", "fas-Bcl2-plate323-04-04.A01",
                             package="prada"))
g1 <- new("gate", name="G1", gateFun=function(x)x[, "FSC-H"]<500, logic="&",
         colnames="FSC-H")
g2 <- new("gate", name="G2", gateFun=function(x)x[, "SSC-H"]>800, logic="&",
         colnames="SSC-H")
g3 <- new("gate", name="G3", gateFun=function(x)x[, "FL1-H"]>800, logic="&",
         colnames="FL1-H")
gs <- new("gateSet", name="Set1", glist=list(G1=g1, G2=g2))
length(gs)
gs[[1]]
gs[1]
gsnames <- names(gs)
names(gs) <- gsnames
applyGate(sampdat, gs)
applyGate(exprs(sampdat), gs)
gate(sampdat) <- gs
applyGate(sampdat, 1)
applyGate(sampdat, "G1")
applyGate(sampdat, TRUE)
appendGates(sampdat, g3)
```

getPradaPar

Set and query global parameters for functions in the prada package

Description

Set and query global parameters for functions in the prada package

Usage

```
setPradaPars(pars)
getPradaPar(parname)
```

Arguments

pars	Named list
parname	Character string of length 1

Details

TBA

Value

For `getPradaPar`, the value of the list element with name `parname` of the global parameters list. The function `setPradaPars` is invoked for its side effect, which is assigning a value to the global parameters list. It returns the value `invisible(NULL)`.

Author(s)

Wolfgang Huber <http://www.ebi.ac.uk/huber>

Examples

```
setPradaPars(list(tomato=1, apple="two", pear=as.integer(3)))
getPradaPar("pear")
```

plotNorm2

Plot fitted bivariate normal distribution.

Description

Plots objects derived from function `fitNorm2` in false color representation.

Usage

```
plotNorm2(fn, colrange=c("gray82", "blue"), center=TRUE, selection=FALSE,
          ellipse=FALSE, pch=20, cex=1, col="dens", ...)
```

Arguments

<code>fn</code>	List. Object derived from function <code>fitNorm2</code>
<code>colrange</code>	Character vector with valid color identifiers (eg name or RGB values) from which a smooth color palette is derived.
<code>center</code>	Logical. Assign center of distribution.
<code>selection</code>	Logical. Mark all points beyond selection.
<code>ellipse</code>	Logical. Plot area and borders of selection as ellipse.
<code>pch</code>	see <code>par</code>
<code>cex</code>	see <code>par</code>
<code>col</code>	see <code>par</code> or special cases <code>dens</code> for coloring according to density and <code>prob</code> for coloring according to probability.
<code>...</code>	further arguments that are passed on to <code>plot</code> .

Details

Produces a scatterplot of paired data showing the densities of the fitted bivariate distribution from function `fitNorm2` in false color representation. Additionally a selection of data points can be highlighted either by marking outliers or by showing its area.

Value

A list containing items `p`, `cov`, `mu`, `S` (density values for each data pair, resulting object from call to `cov.rob`, midpoint of distribution, covariance matrix).

Author(s)

Florian Hahne

See Also[fitNorm2](#)**Examples**

```
sampdat <- readFCS(system.file("extdata",
  "fas-Bcl2-plate323-04-04.A01", package="prada"))
nfit <- fitNorm2(exprs(sampdat[,1:2]), scalefac=2)
plotNorm2(nfit, selection=TRUE, ellipse=TRUE)
```

plotPlate

*Plot a well statistic for microtiter plates.***Description**

Plot a well statistic in false color representation or using a self-defined grid plotting function. The plot is supposed to resemble the physical geometry of a microtitre plate.

Usage

```
plotPlate(x, nrow = 8, ncol = 12, col=c("red", "blue"),
  ind = 1:(ncol*nrow), xrange=function(y) range(y, na.rm=TRUE), na.action = "zero"
  main, char, desc = character(2), add=FALSE, gridFun="default",
  funArgs=NULL, ...)
```

Arguments

<code>x</code>	Numeric vector of length <code>ncol*nrow</code> or matrix with <code>ncol*nrow</code> rows (except if argument <code>ind</code> is specified). If of class <code>matrix</code> , the use of argument <code>gridFun</code> is expected.
<code>nrow</code>	Numeric of length 1. The number of rows of the plate.
<code>ncol</code>	Numeric of length 1. The number of columns of the plate.
<code>col</code>	Character vector. Usually the names of two or three colors between which the color map is interpolated, using the function colorRampPalette .
<code>ind</code>	Optional integer vector of equal length as <code>x</code> . It indicates the position of the respective value of <code>x</code> on the plate. Can be used to address the problem of missing values. Each well that is not allocated a value of <code>x</code> by <code>ind</code> will not be plotted.
<code>xrange</code>	Numeric vector of length two giving the range of <code>x</code> that is mapped into the color scale. Alternatively, this can be a function which takes the values of <code>x</code> as input and creates such a vector.
<code>na.action</code>	Character. One of "zero" "omit" or "xout". How should the wells for which <code>x</code> is NA be treated? For "zero", they are plotted as if the value were 0. For "omit", they are omitted. For "xout", they are crossed out. When <code>x</code> is a matrix, <code>na.action</code> is only applied to rows containing nothing but NAs. Further special treatment of NA values in matrices need to be implemented in <code>gridFun</code> .
<code>main</code>	Character of length 1. Plot title.

char	An optional character vector of equal length as <code>x</code> (except if argument <code>ind</code> is specified) to be used for well annotation. Each element of the vector may contain a string to be superimposed on the respective well or <code>NA</code> for no plotting.
desc	Character of length 2. Legend for the two extremes of the colorbar, e.g. <code>'act'</code> and <code>'inh'</code> .
add	Logical. If <code>TRUE</code> add plate plot to current plot. May be used when plotting in grid layout panels.
gridFun	Character. The name of the plotting function to create individual graphs for each well. See functions <code>.drawCircle</code> and <code>.drawPie</code> for examples.
funArgs	Dataframe with argument values to be passed to <code>gridCall</code> . For each argument specified in <code>gridCall</code> there must be one column with the argument name as col-name and the argument values for every well.
...	Further graphical parameters that can be used to control the output of <code>plotPlate</code> . cex.main: expansion factor for title. cex.lab: expansion factor for label. cex.char: expansion factor for well annotation. cex.legend: expansion factor for well legend labels. cex.desc: expansion factor for well legend description.

Details

You may use this function either to create plots showing a single-value per well statistic for microtiter plates, or you can use a self-made plotting function using a combination of any valid grid commands to produce arbitrary plots in a plate array format. These plots may also show multifactorial data. Self-defined plotting functions need to have `data` as first argument. `plotPlate` passes all data values for the respective well to the plotting function. Any further arguments may be passed on using argument `funArgs`. See `.drawCircle` and `.drawPie` for examples of valid plotting functions and the vignette for detailed information. Note that using `funCall` overrides some of the default functionalities, e.g. plotting of legends and alters the treatment of `NA` values.

Argument `ind` allows the user to indicate the position (well number) for each element of vector `x` on the plate. This can be used either to change the order in which elements of `x` are to be plotted or to deal with the problem of missing data for some of the wells on a plate.

To further increase the amount of information of the `platePlot` one may decorate wells with short annotations using argument `char`. Each element of `char` `!= NA` will be superimposed on the respective well (see examples).

Value

The function produces a plot in the active graphics device.

It returns a list with four elements. The element `which` is a vector with the indices of those elements in `x` that were plotted (see argument `na.action`). The element `coord` is a length (`which`) by 4 matrix in which each row specifies the corners of a rectangle that contains a well. It is intended to be used as an argument to a subsequent call to `imageMap`. Elements `width` and `height` may be used to open a graphic devices that can hold the plate plot with the correct aspect ratio.

Author(s)

Florian Hahne, Wolfgang Huber <http://www.ebi.ac.uk/huber>

See Also[imageMap](#)**Examples**

```

plotPlate(runif(96), main="example 1", col=c("#0000e0", "#e00000"), desc=c("act", "inh"))
plotPlate(runif(384), nrow=16, ncol=24, main="example 2", col=c("#0000e0", "white", "#e00000"))
plotPlate(runif(48), main="example 3", col=c("#0000e0", "#e00000"), ind=c(1:24, 73:96))
x <- runif(96)
x[sample(96, 10)] <- NA
plotPlate(x, main="example 4", col=c("#0000e0", "#e00000"),
char=c(rep(NA, 72), LETTERS[1:24]), na.action="xout")
plotPlate(runif(96, min=0.1, max=0.5), gridFun=".drawCircle")
plotPlate(matrix(runif(288), ncol=3), gridFun=".drawPie",
funArgs=as.data.frame(matrix(2:4, ncol=3, nrow=96, byrow=TRUE)))

```

readCytoSet

*Create a cytoSet object from one or more FCS 3.0 files***Description**

Create a cytoSet object from one or more FCS 3.0 files

Usage

```
readCytoSet(files=NULL, path=".", pattern=NULL, phenoData, sep="\t", ...)
```

Arguments

files	Optional character vector with filenames
path	Directory where to look for the files
pattern	This argument is passed on to dir (see details).
phenoData	Either an object of class <code>phenoData</code> or character.
sep	Separator character that gets passed on to read.AnnotatedDataFrame .
...	Further arguments that get passed on to read.AnnotatedDataFrame , see details.

Details

There are three different ways to specify the file names:

First, if the argument `phenoData` is present and is of class `AnnotatedDataFrame`, then it is obtained from its column name. The column is mandatory, and an error will be generated if it is not there. Alternatively, the argument `phenoData` can be of class `character`, in which case this function tries to read a `AnnotatedDataFrame` object from the file with that name by calling [read.AnnotatedDataFrame](#) with arguments `file.path(path, phenoData)`, ...

Second, if the argument `phenoData` is not present and the argument `files` is not `NULL`, then `files` is expected to be a character vector with the file names.

Third, if neither the argument `phenoData` is present nor `files` is not `NULL`, then the file names are obtained by calling `dir(path, pattern)`.

Value

An object of class `cytoSet`.

Author(s)

Wolfgang Huber <http://www.ebi.ac.uk/huber>

See Also

`readFCSdata`

Examples

```
## Please see man page for cytoSet-class
```

`readFCS`

Read an FCS file

Description

Read one or several FCS files: Data File Standard for Flow Cytometry

Usage

```
read.fcs(filename=NULL, objectModel="prada", ...)
readFCS(filename)
```

Arguments

<code>filename</code>	Character of length 1: filename
<code>objectModel</code>	Character of length 1: the object model to use for the output. Currently only 'prada' for <code>cytoFrame</code> objects is supported.
<code>...</code>	Arguments that get passed on to higher-level import functions.

Details

The function `readFCS` works with the output of the FACS machine software from a number of vendors. However, the FCS 3.0 standard includes some options that are not yet implemented in this function. If you need extensions, please let me know. The output of the function is an object of class `cytoFrame`.

`read.fcs` is a wrapper function that allows the user to specify the class of the output. The purpose of the function is to standardize the way flow cytometry data is imported into R using the `prada` package. If the `filename` argument to `read.fcs` is a character vector of length > 1 , multiple FCS files can be imported. Please see the documentation for `readCytoSet` for alternatives ways to import multiple FCS files and for more details on the higher-level import function.

For specifications of FCS 3.0 see <http://www.isac-net.org> and the file `../doc/fcs3.html` in the `doc` directory of the package.

Value

An object of class `cytoFrame`.

Author(s)

Wolfgang Huber <http://www.ebi.ac.uk/huber>, Florian Hahne

See Also

[readCytoSet](#)

Examples

```
sampdat <- readFCS(system.file("extdata", "fas-Bcl2-plate323-04-04.A01",
                             package="prada"))
files <- dir(system.file("extdata", package="prada"),
            pattern="[A-H][0-9][0-9]")
sampdat2 <- read.fcs(system.file("extdata", "fas-Bcl2-plate323-04-04.A01",
                               package="prada"))
sampdat3 <- read.fcs(files, path=system.file("extdata", package="prada"))
sampdat
exprs(sampdat[1:3,])
description(sampdat)[3:6]
class(sampdat3)
```

readFCSaux

Auxiliary functions for readFCS

Description

Auxiliary functions for readFCS - not normally called by the user

Usage

```
readFCSgetPar(x, pnam)
readFCSheader(con)
readFCSstext(con, offsets)
readFCSdata(con, offsets, x)
```

Arguments

x	Named character vector.
pnam	Character vector, its elements must be contained in names(x).
con	Connection.
offsets	Integer vector of length 6 with byte offsets of the header, text, and data blocks.

Details

These functions are not normally called by the user. See [readFCS](#) instead.

Value

Various.

Author(s)

Wolfgang Huber <http://www.ebi.ac.uk/huber>

See Also

[readFCS](#)

removeCensored *Remove rows that contain censored data*

Description

Remove rows that contain censored data in the columns of `x` specified by `columns`.

Usage

```
## S4 method for signature 'matrix'  
removeCensored(x, values, columns, na.rm=TRUE)  
## S4 method for signature 'data.frame'  
removeCensored(x, values, columns, na.rm=TRUE)  
## S4 method for signature 'cytoFrame'  
removeCensored(x, values, columns, na.rm=TRUE)
```

Arguments

<code>x</code>	Object of class <code>matrix</code> , <code>data.frame</code> , or <code>cytoFrame</code> .
<code>values</code>	Values that correspond to censored data. If missing, <code>range(x)</code> is used.
<code>columns</code>	Numeric or character vector specifying the columns of <code>x</code> that are compared against <code>values</code> . If missing, <code>1:ncol(x)</code> is used.
<code>na.rm</code>	Logical. If <code>TRUE</code> , rows that contain NA values are also removed.

Details

The function removes all rows that contain, in the columns specified by the `columns` argument, values that are contained in the `values` argument. If `na.rm` is `TRUE`, then rows that contain NA values are also removed.

An application is with FACS data, where measurements outside of the detector's dynamic range produce minimal or maximal values. For example, if a 16-bit A/D converter was used, top-censored data would have a value of 65535.

Value

Object of the same class as `x`, with some rows removed.

Author(s)

Florian Hahne, Wolfgang Huber

Examples

```
set.seed(8215)
mat <- matrix(floor(runif(20000)*1024), ncol=4)
range(mat[,1])
mat <- removeCensored(mat, columns=1:2)
range(mat[,1])
range(mat[,3])
```

progress

A simple tcltk progress window

Description

Show progress of a task in a tcltk window as percentage

Usage

```
progress(title="processing task...", message="", sub="")
updateProgress(percentage, autoKill=FALSE, sub="")
killProgress()
```

Arguments

title	The title of the tcltk window
message	A short test message to add to the window
sub	An additional text field that can be updated viaupdateProgress
percentage	An integer giving the percentage of completion
autoKill	Logical indicating whether to kill the display after 100 is reached

Details

Function `progress` creates the progress window and sets up the necessary environment. `updateProgress` takes as argument an integer value indicating the percentage of completion and updates the display. The integer value that gets passed to `updateProgress` will usually be generated by an iterator (e.g. in a for loop). `killProgress` may be called explicitly to kill the progress window. Alternatively one can set the argument `autoKill` of `updateProgress` to `TRUE` to automatically kill the window once a value of 100 is reached.

Value

The functions are called for their side effects.

Author(s)

Florian Hahne

Examples

```

if(interactive() && capabilities()["tcltk"]){
  progress(message="This is a progress display...", sub="(step 1 of 50)")
  for(i in 1:50) {
    zz = rnorm(1e5)
    updateProgress(i*2, autoKill=TRUE, sub=paste("(step", i, "of 50)"))
  }
}

```

threePanelPlot *Visualize cytometry data*

Description

Function to visualize multivariate (cytometry) data in three two-dimensional plots.

Usage

```

threePanelPlot(data, x.panels = c(1, 4, 5), y.panels = c(2, 3, 6),
  tot.width = 15, tot.height = 5.4, maxcells = 20000,
  limits = c(0, 1023), remove.extremes = TRUE,
  plotTitle = "Three-Panel Plot", use.smoothScatter = TRUE,
  palette = colorRampPalette(brewer.pal(9, "Blues")),
  new.device = TRUE, verbose = TRUE,
  addPoints = NULL, addCol = "red", ...)

```

Arguments

data	data matrix to visualize
x.panels	which variables (columns) are to be plotted at the x-axis of the three variables
y.panels	which variables (columns) are to be plotted at the y-axis of the three variables
tot.width	width of a new device to open, see argument <code>new.device</code>
tot.height	height of a new device to open, see argument <code>new.device</code>
maxcells	maximum number of observations (cells) for plotting; higher numbers reduce performance
limits	minimum and maximum value (theoretically) observed in the data; e.g., with 10-channel digitized data it is <code>c(0,1023)</code>
remove.extremes	logical; are extreme values (equal to theoretical <code>limits</code>) to be removed before plotting
plotTitle	title for the plot
use.smoothScatter	logical, should the function <code>smoothScatter</code> be employed for plotting the data (plots data densities rather than individual points)
palette	if <code>smoothScatter</code> is used, which colour palette is it to use
new.device	logical; should a new device be opened for the three plots; if <code>FALSE</code> the three plots will be plotted to the currently active device
verbose	logical; do you want extended output to <code>STDOUT</code>

`addPoints` should special points be marked after plotting the data; is expected to be a subset of argument `data` with the same number of columns (=variables); if `NULL` no points are marked

`addCol` in which colour are the points in `addPoints` to be marked

`...` further arguments passed on to `plot.default`

Value

no value is returned; the function is called to produce three plots

Author(s)

Joern Toedling <toedling@ebi.ac.uk>

See Also

[plot.default](#)

Examples

```
# generate some data:
toyData <- cbind(matrix(pmax(0,pmin(runif(3000)+rnorm(3000),4)),ncol=3),
                 matrix(pmax(0,pmin(rnorm(3000,2,1),4)),ncol=3))
colnames(toyData) <- paste("Var",1:6,sep=" ")
toyQuantiles <- apply(toyData,2,quantile,probs=c(0.25,0.5,0.75))

# plot it and mark the quantiles:
threePanelPlot(toyData,addPoints=toyQuantiles,
               addCol=c("orange","red","purple"),limits=c(0,4),pch=20)
```

thresholds	<i>Discretize a two-dimensional data space into quadrants by applying thresholds</i>
------------	--

Description

Discretize a two-dimensional data space into quadrants by applying thresholds.

Usage

```
thresholds(x, y, xthr, ythr)
```

Arguments

`x` Vector containing x or matrix containing x and y values of bivariate data.

`y` Optional vector containing y values of bivariate data.

`xthr` x value separating 'left' and 'right'.

`ythr` y value separating 'up' and 'down'.

Details

The function returns a 2x2 matrix giving the counts for each quadrant. Events with values equal to the thresholds are counted to the left or down respectively.

Value

2x2 matrix.

Author(s)

Florian Hahne

Examples

```
thresholds(cbind(c(1, 1, 2, 2, 2, 4), c(1, 4, 2, 4, 5, 4)), xthr=3, ythr=3)
```

touchFCS

Check for FCS files

Description

The function reads the header of a file or of a range of files and checks whether they are valid FCS 2.0 or FCS 3.0 files.

Usage

```
touchFCS(path = ".", file)
```

Arguments

path	character, the path to a folder containing files
file	character, the path to a single file

Details

The user may either specify the path to a directory in which to search for FCS files or the path to a single file.

Value

A character vector with names of the valid FCS files found.

Author(s)

fhahne

vpLocation	<i>Absolute location of current viewport</i>
------------	--

Description

Calculates the absolute location and size of the current grid viewport in inches and pixels.

Usage

```
vpLocation()
```

Details

This function may be used to get the absolute location of the current viewport on the current graphics device. It uses function [devRes](#) to get the device resolution for calculating pixel values. Locations are given by the two extreme coordinates in x and y direction.

Value

A list with items `location`, `size`, `ilocation` and `isize`, the location and size of the viewport in pixels and inches respectively.

Author(s)

Florian Hahne

See Also

[plotPlate](#), [devRes](#)

Examples

```
vpLocation()
```

Index

*Topic **IO**

readCytoSet, 19
readFCS, 20
readFCSaux, 21
touchFCS, 26

*Topic **classes**

cytoFrame-class, 5
cytoSet-class, 7
gate-class, 12
gateSet-class, 14

*Topic **datasets**

cframe, 9
cset, 9

*Topic **hplot**

barploterrbar, 3
plotPlate, 17
threePanelPlot, 24

*Topic **manip**

analysePlate, 1
as.all, 2
csApply, 4
getPradaPar, 15

*Topic **misc**

progress, 23

[, cytoFrame, ANY, ANY, ANY-method
(*cytoFrame-class*), 5
[, cytoSet, ANY, missing, missing-method
(*cytoSet-class*), 7
[, gateSet, ANY, missing, missing-method
(*gateSet-class*), 14
[[, cytoSet, ANY, missing-method
(*cytoSet-class*), 7
[[, gateSet, ANY, missing-method
(*gateSet-class*), 14
[[<- , cytoSet-method
(*cytoSet-class*), 7
\$.cytoFrame (*cytoFrame-class*), 5

analysePlate, 1
AnnotatedDataFrame, 19
appendGates (*gateSet-class*), 14
appendGates, gateSet_method
(*gateSet-class*), 14

appendGates, cytoFrame-method
(*cytoFrame-class*), 5
appendGates, gateSet-method
(*gate-class*), 12
applyGate (*gateSet-class*), 14
applyGate, cytoFrame, character-method
(*cytoFrame-class*), 5
applyGate, cytoFrame, gate-method
(*cytoFrame-class*), 5
applyGate, cytoFrame, gateSet-method
(*cytoFrame-class*), 5
applyGate, cytoFrame, logical-method
(*cytoFrame-class*), 5
applyGate, cytoFrame, numeric-method
(*cytoFrame-class*), 5
applyGate, matrix, gate-method
(*gate-class*), 12
applyGate, matrix, gateSet-method
(*gateSet-class*), 14
as, 2
as.all, 2
as.gateSet (*gate-class*), 12
as.gateSet, gate-method
(*gate-class*), 12

barplot, 3
barploterrbar, 3
cframe, 9
colnames, cytoFrame-method
(*cytoFrame-class*), 5
colnames, cytoSet-method
(*cytoSet-class*), 7
colnames<- , cytoFrame-method
(*cytoFrame-class*), 5
colnames<- , cytoSet-method
(*cytoSet-class*), 7
colorRampPalette, 17
combineFrames, 4
combineGates (*gate-class*), 12
cov.rob, 12
csApply, 4
cset, 9
cytoFrame, 7, 8, 13–15, 20

- cytoFrame (*cytoFrame-class*), 5
- cytoFrame-class, 5
- cytoSet, 4, 6, 20
- cytoSet (*cytoSet-class*), 7
- cytoSet-class, 7
- description, cytoFrame-method
(*cytoFrame-class*), 5
- description<-, cytoFrame, character-method
(*cytoFrame-class*), 5
- devDims, 10
- devRes, 11, 27
- dir, 19
- drawGate, 13, 14
- drawGate (*gate-class*), 12
- drawGate, cytoFrame-method
(*cytoFrame-class*), 5
- drawGate, matrix-method
(*cytoFrame-class*), 5
- environment, 7, 8
- exprs, cytoFrame-method
(*cytoFrame-class*), 5
- exprs<-, cytoFrame, matrix-method
(*cytoFrame-class*), 5
- fitNorm2, 11, 16, 17
- gate, 6, 8, 14, 15
- gate (*gate-class*), 12
- gate, cytoFrame-method
(*cytoFrame-class*), 5
- gate-class, 12
- gate<- (*gate-class*), 12
- gate<-, cytoFrame, gate-method
(*cytoFrame-class*), 5
- gate<-, cytoFrame, gateSet-method
(*cytoFrame-class*), 5
- gateSet, 6, 8, 13, 14
- gateSet (*gateSet-class*), 14
- gateSet-class, 14
- getPradaPar, 15
- hist, cytoFrame-method
(*cytoFrame-class*), 5
- hist, cytoSet-method
(*cytoSet-class*), 7
- imageMap, 18, 19
- killProgress (*progress*), 23
- length, cytoSet-method
(*cytoSet-class*), 7
- length, gateSet-method
(*gateSet-class*), 14
- lines, gate-method (*gate-class*), 12
- names, gate-method (*gate-class*), 12
- names, gateSet-method
(*gateSet-class*), 14
- names<-, gate-method (*gate-class*),
12
- names<-, gateSet-method
(*gateSet-class*), 14
- ncol, cytoFrame-method
(*cytoFrame-class*), 5
- nrow, cytoFrame-method
(*cytoFrame-class*), 5
- pData, cytoSet-method
(*cytoSet-class*), 7
- phenoData, 7, 8
- phenoData, cytoSet-method
(*cytoSet-class*), 7
- phenoData<-, cytoSet, AnnotatedDataFrame-method
(*cytoSet-class*), 7
- plot, cytoFrame, missing-method
(*cytoFrame-class*), 5
- plot, cytoSet, missing-method
(*cytoSet-class*), 7
- plot.default, 25
- plotNorm2, 12, 16
- plotPlate, 10, 11, 17, 27
- progress, 23
- read.AnnotatedDataFrame, 19
- read.fcs (*readFCS*), 20
- readCytoSet, 6–8, 19, 20, 21
- readFCS, 6, 20, 21, 22
- readFCSaux, 21
- readFCSdata, 20
- readFCSdata (*readFCSaux*), 21
- readFCSgetPar (*readFCSaux*), 21
- readFCSheader (*readFCSaux*), 21
- readFCSstext (*readFCSaux*), 21
- removeCensored, 22
- removeCensored, cytoFrame-method
(*removeCensored*), 22
- removeCensored, data.frame-method
(*removeCensored*), 22
- removeCensored, matrix-method
(*removeCensored*), 22
- sapply, 4, 5
- setPradaPars (*getPradaPar*), 15
- show, cytoFrame-method
(*cytoFrame-class*), 5

show, cytoSet-method
 (*cytoSet-class*), 7

show, gate-method (*gate-class*), 12

show, gateSet-method
 (*gateSet-class*), 14

smoothScatter, 24

split, cytoSet, ANY, ANY-method
 (*cytoSet-class*), 7

split, cytoSet, ANY-method
 (*cytoSet-class*), 7

split, cytoSet-method
 (*cytoSet-class*), 7

tapply, 1

threePanelPlot, 24

thresholds, 25

touchFCS, 26

updateProgress (*progress*), 23

vpLocation, 27