rflowcyt

April 19, 2009

`ContourScatterPlot` *Image and Contour Bivariate Plot*

Description

To make a bivariate image with a rectangular grid and a superimposed contour plot of two variables or to make a bivariate hexbin image plot from a hexagon grid with NO superimposed contour plot.

Usage

```
ContourScatterPlot(xvar, yvar,
                   status=NULL,
                   type.CSP=c("count.diff", "p.hat", "p.hat.norm", "z.stat"),
                   xlab = NULL, ylab = NULL, main = NULL,
                    x.grid = round(seq(range(xvar)[1],
                                   ceiling(diff(range(xvar))/25)*25+range(xvar)[1
                                   by=25),0),
                    y.grid =round(seq(range(yvar)[1],
                                   ceiling(diff(range(yvar))/25)*25+range(yvar)[1
                                   by=25),0),
                   lattice=FALSE,
                   hexbin.plotted=FALSE,
                   hexbin.style=c("colorscale", "lattice", "centroids",
                             "nested.lattice", "nested.centroids"),
                   n.hexbins=100, numlev = 5, xaxt="s", yaxt="s",image.col = hea
                   ...)
```

Arguments

`xvar`	numerical vector of the x-variable
`yvar`	numerical vector of the y-variable
`status`	numerical binary 0, 1 vector denoting the status of the observations; default is NULL
`type.CSP`	character string denoting the type of value to be estimated using the 'status' for each cell grid: the difference in counts ("count.diff"), the proportion ("p.hat"), the normalized proportion at 0.5 ("p.hat.norm"), the z.statistic ("z.stat"), see make.density for details.

`xlab`	character string of the x-variable name
`ylab`	character string of the y-variable name
`main`	character string of title of the plot
`x.grid`	numerical vector of the x-axis breaks for the image plot using the rectangular grid; default is a vector of values within the range of 'xvar' separated by 25 units increments.
`y.grid`	numerical vector of the y-axis breaks for the image plot using the rectangular grid; default is a vector of values within the range of 'yvar' separated by 25 units increments.
`hexbin.plotted`	boolean; if TRUE then the grid cells/compartments are hexagons; otherwise the grid cells are rectangular; default value is FALSE
`lattice`	logical
`n.hexbins`	number of xbins for hexagon binning; default is 100
`hexbin.style`	the style of hexbin plot; default is "colorscale"
`image.col`	vector of color or color type for the image plot with the rectangular grid; default=heat.colors(10)
`numlev`	number of levels for the contour plot superimposed on the image plot using a rectangular grid; default value=5
`xaxt`	if "s", then the x-axis is plotted, if "n" then there is no x-axis plotted
`yaxt`	if "s", then the y-axis is plotted, if "n" then there is no y-axis plotted
`...`	if hexbin.plotted=TRUE, the other options/arguments under `plot.hexbin` (library(hexbin)) can be used; if hexbin.plotted=FALSE, then other options under `contour` (library(base)) can be used

Details

This function calls make.grid or make.density for the values in the rectangular grid which make up the image plot. This procedure produces rectangular cells for the resulting grid, but if there is a library(hexbin) and the user wants hexagon cells in the image grid, hexbin cells are produced in the grid. A superimposed contour plot is available for the rectangular-celled image grid, but not available for the hexbin image grid.

Other image colors (image.col) may be used. See documentation for `heat.colors`.

Value

Image plot with a superimposed contour plot along with a legend roughly describing the values associated with the color scheme. The white-colored grid cells correspond to those with no observations.

Warning

The number of image colors used may vary from one plot to another, and users should be warned that a different number of colors, ie, heat.colors(2) (as default) may be used if there are few variations/clusters in the data.

The user should use more colors, ie, heat.colors(10) or heat.colors(5), etc. to account for more variation in the data, if there is a lot of variation that is apparent. An error message to use gray or psuedo.cube colors will prompt the user in such cases that will need a change (usually a decrease) in the number of image colors.

Gating (both interactive and non-interactive currently works only with the bivariate image plot using a rectangular and not hexagonal grid (ie, with the option hexbin.plotted=FALSE).

Author(s)

A. J. Rossini, J. Y. Wan

See Also

`make.grid`, `legend.CSP`, `image`, `contour`, `heat.colors`, **hexbin**, 'plot.hexbin',

Examples

```
##Example I: with a  FSC object
  if (require(rfcdmin)){
    data.there<-is.element("MC.053",objects())
    if ((sum(data.there) != length(data.there))) {
      ## obtaining the FCS objects from FHCRC data
      data(MC.053min)
    }
    ## obtain the two column variables
    xvar<-MC.053@data[,1]
    yvar<-MC.053@data[,2]

    ## have an example plot
    if (interactive()==TRUE) {

      ## rectangular cells with the contour plot
      ContourScatterPlot(xvar, yvar,
                         xlab=colnames(MC.053@data)[1],
                         ylab=colnames(MC.053@data)[2],
                         main="Individual 042402c1.053",
                         hexbin.plotted=FALSE,
                         numlev=25, image.col=heat.colors(15),
                         plot.legend.CSP=TRUE)

      ## hexagon cells without contour lines; default n.hexbins=100
      ContourScatterPlot(xvar, yvar,
                         xlab=colnames(MC.053@data)[1],
                         ylab=colnames(MC.053@data)[2],
                         main="Individual 042402c1.053",
                        hexbin.plotted=TRUE)
      ## finer hexgonal binning
        ContourScatterPlot(xvar, yvar,
                         xlab=colnames(MC.053@data)[1],
                         ylab=colnames(MC.053@data)[2],
                         main="Individual 042402c1.053",
                         hexbin.plotted=TRUE, n.hexbins=300)

## and with some additional
      ## plot.hexbin options
      ContourScatterPlot(xvar, yvar,
                          xlab=colnames(MC.053@data)[1],
                          ylab=colnames(MC.053@data)[2],
                          main="Individual 042402c1.053", hexbin.plotted=TRUE,
                          minarea=1, maxarea=1)
      ## different hexbin styles

      ContourScatterPlot(xvar, yvar,
                          xlab=colnames(MC.053@data)[1],
                          ylab=colnames(MC.053@data)[2],
                          main="Hexbin.style=colorscale", hexbin.plotted=TRUE,
                          hexbin.style="colorscale")
      ContourScatterPlot(xvar, yvar,
                          xlab=colnames(MC.053@data)[1],
                          ylab=colnames(MC.053@data)[2],
                          main="Hexbin.style=lattice", hexbin.plotted=TRUE,
                          hexbin.style="lattice")
      ContourScatterPlot(xvar, yvar,
                          xlab=colnames(MC.053@data)[1],
                          ylab=colnames(MC.053@data)[2],
                          main="Hexbin.style=centroids", hexbin.plotted=TRUE,
                          hexbin.style="centroids")

      ContourScatterPlot(xvar, yvar,
                          xlab=colnames(MC.053@data)[1],
                          ylab=colnames(MC.053@data)[2],
                          main="Hexbin.style=nested.lattice", hexbin.plotted=TRUE,
                          hexbin.style="nested.lattice")
      ContourScatterPlot(xvar, yvar,
                          xlab=colnames(MC.053@data)[1],
                          ylab=colnames(MC.053@data)[2],
                          main="Hexbin.style=nested.centroids", hexbin.plotted=TRUE,
                          hexbin.style="nested.centroids")
      }
## See example(make.density) for examples of 'image' of
## grid images with values estimated from 'status'; ie plots of
## differences between stimulated and unstimulated
## HIV-protein 'status' scenarios

if ( ( sum(data.there) != length(data.there) )){
      ## obtaining the FCS objects from VRC data
      data(VRCmin)
  }

var1<-st.DRT@data[,4]
var2<-st.DRT@data[,5]
var1.2<-unst.DRT@data[,4]
var2.2<-unst.DRT@data[,5]

col.nm<-colnames(st.DRT@data)

## The status where 1=stimulated
## 0 = unstimulated
status<-c(rep(1, dim(st.DRT@data)[1]), rep(0, dim(unst.DRT@data)[1]))
x <- c(var1, var1.2)
y <-c(var2, var2.2)

if (interactive()){
par(mfrow=c(3,4))
ContourScatterPlot(var1, var2,
  main="make.grid: Counts for stimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], image.col=heat.colors(20),plot.legend.CSP=TRUE)

ContourScatterPlot(x, y,
  main="make.grid: Counts for unstimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], image.col=heat.colors(20),plot.legend.CSP=TRUE)

## white cells are those with NO data
ContourScatterPlot(x, y,  status=status,
  type.CSP="count.diff",
  main="Count difference between Stimulated and unstimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], image.col=c("brown","lightyellow"))

ContourScatterPlot(x, y,  status=status,
  type.CSP="p.hat",
  main="Proportion of Stimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], image.col=c("brown","lightyellow"))

ContourScatterPlot(x, y,  status=status,
   main="Normalized proportion of Stimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], image.col=c("brown","lightyellow"))

ContourScatterPlot(x, y,  status=status,
   main="z statistic",
   xlab=col.nm[4],
   ylab=col.nm[5], image.col=c("brown","lightyellow"))
}

}

##Example II: with a  CytoFrame object
 if (require(rfcdmin)) {

 ##obtaining the location of the fcs files in the data
  pathFiles<-system.file("bccrc", package="rfcdmin")
  drugFiles<-dir(pathFiles)

 ## reading in the FCS files
  drugData<-read.series.FCS(drugFiles,path=pathFiles,MY.DEBUG=FALSE)
  xvar <- fluors(drugData[[1]])[,1]
  yvar <- fluors(drugData[[1]])[,2]
  if (interactive()==TRUE) {
    ContourScatterPlot(xvar, yvar,
                         xlab=colnames(exprs(drugData[[1]]))[1],
                         ylab=colnames(exprs(drugData[[1]]))[2],
                         main="Contour plot",
                         hexbin.plotted=FALSE,
                         numlev=25, image.col= c("gray82", "blue"),
                         plot.legend.CSP=TRUE)
                         }
}
```

`"FCS-class"` *Class "FCS" : Flow Cytometry Standard*

Description

This class represents objects read from raw binary Flow Cytometry Standard (FCS) files. These files contain a data portion, consisting of immunofluorescence and other column variables for each cell or row observation, and a metadata portion, which contains information such as parameter shortnames, longnames, ranges and data dimensions as well as file information.

Objects from the Class

Objects can be created by calls of the form `new("FCS", ...)`.

Slots

`data`: Object of class `"matrix"` which holds integer data such that the columns are the variables (usually immunofluorescence measurements) and the rows are the cell observations.

`metadata`: Object of class `"FCSmetadata"` which holds information about the file, data, and column variables among other items in the header of the original raw FCS binary file.

Methods

"[" `signature(x = "FCS")`: Extracts the data

"[<-" `signature(x = "FCS")`: Replaces or sets the data

"[[" `signature(x = "FCS")`: Extracts the metadata

"[[<-" `signature(x = "FCS")`: Replaces or sets the metadata

addParameter `signature(x = "FCS", colvar = "vector")`: Adds a column parameter to the data

checkvars `signature(x = "FCS")`: Checks the compatibility of the metadata against the data dimensions and column/parameter names and ranges

coerce `signature(from = "FCS", to = "matrix")`: Returns the data as a matrix

coerce `signature(from = "FCS", to = "data.frame")`: Returns the data as a data.frame

coerce `signature(from = "matrix", to = "FCS")`: Returns an FCS object with data and default prototype metadata

coerce `signature(from = "data.frame", to = "FCS")`: Returns an FCS object with data and default prototype metadata

dim.FCS `signature(x = "FCS")` : Returns the dimensions (ie, the number of rows and columns respectively) of the data matrix; the output is a vector

equals `signature(x = "FCS", y = "FCS")`: Compares the equality of two objects in terms of data and metadata correspondence

fixvars `signature(x = "FCS")`: Sets the discrepant metadata slots to values in from the data

fluors `signature(x = "FCS")`: Returns the complete data portion of the object

metaData `signature(x = "FCS")`: Returns the complete metadata portion of the object

"plot-methods" `signature(x = "FCS", y = "missing")`: Plots the object as a pairs plot (with rectangular binned contour-image plots or hexagonal binned image plots) or as a joint or marginal image parallel coordinates plot

"print-methods" `signature(x = "FCS")`: Prints a brief description about the original filename, dimensions of the data, and the original status of the current object's data

"show-methods" `signature(object = "FCS")`: Prints a brief description about the original filename, dimensions of the data, and the original status of the current object's data

"summary-methods" `signature(object = "FCS")`: Summaries the data's dimensions, five-number summaries on the column parameters, the information contained in the metadata

Note

The function `read.FCS` is used to read in a raw binary FCS files and output a "FCS-class" object.

Author(s)

A.J. Rossini, J.Y. Wan, and Zoe Moodie

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics : New York, 2001. pp.279-283.

Jerome H. Friedman and Nicholas I. Fisher. Bump Hunting in High-Dimensional Data. Tech Report. October 28, 1998.

J. Paul Robinson, et al. Current Protocols in Cytometry. John Wiley & Sons, Inc : 2001.

Mario Roederer and Richard R. Hardy. Frequency Difference Gating: A Multivariate Method for Identifying Subsets that Differe between Samples. Cytometry, 45:56-64, 2001.

Mario Roederer and Adam Treister and Wayne Moore and Leonore A. Herzenberg. Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences. Cytometry, 45:37-46, 2001.

Keith A. Baggerly. Probability Binning and Testing Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared Test. Cytometry, 45:141-150, 2001.

See Also

`read.FCS`, `"FCSgate-class"`, `"FCSsummary-class"`, `"FCSmetadata-class"`, `"plot-methods"`, `"print-methods"`, `"show-methods"`, `"summary-methods"`, `"coerce-methods"`, `"[-methods"`, `"[[-methods"`, `"[<--methods"`, `"[[<--methods"`, `checkvars`, `fixvars`, `equals`, `addParameter`, `fluors`, `metaData`, `dim.FCS`

Examples

```
## a default FCS object
default.FCSobj<-new("FCS")

## making my own FCS object
## first making up the data
dummy.data<-matrix(1:1000, ncol=10)
colnames(dummy.data)<-paste("foo", 1:10, sep="")

## second making up the metadata
##   default FCSmetadata
dummy.metadata<-new("FCSmetadata")
##   user-defined metadata
```

```
foo.metadata<-new("FCSmetadata", mode="none", size=100, nparam=10,
shortnames=paste("V", 1:10, sep=""), longnames=colnames(dummy.data),
paramranges=unlist(apply(dummy.data, 2, max)), filename="",
objectname="foo.FCSobj", fcsinfo=list("extraInfo1"="dummy FCS",
"extraInfo2"=9:20))

foo.FCSobj<-new("FCS", data=dummy.data, metadata=foo.metadata)

dummy.FCSobj<-new("FCS", data=matrix(), metadata=dummy.metadata)

## extraction of the metadata
foo.FCSobj[["size"]]
## replacement of the metadata
 ## introduce an error in the column length
foo.FCSobj[["nparam"]]<-0

## extraction of the data

first.ten.obs<-foo.FCSobj[1:10,]
## replacement of the data
foo.FCSobj[1:10,]<-matrix(1:100, ncol=10)
## addParameter
foo.FCSobj<-addParameter(foo.FCSobj, 1:100, shortname="newvar",
longname="newlymadevariable", use.shortname=FALSE)

## replacement of the metadata
 ## introduce an error in the column length
foo.FCSobj[["nparam"]]<-0

## checkvars
correct.status.is.FALSE<-checkvars(foo.FCSobj)
## coerce FCS to matrix
coerced.mat<-as(foo.FCSobj, "matrix")
is(coerced.mat, "matrix")
## coerce FCS to data.frame
coerced.df<-as(foo.FCSobj, "data.frame")
is(coerced.df, "data.frame")
## coerce matrix to FCS
FCSobj1<-as(coerced.mat, "FCS")
is(FCSobj1, "FCS")
## coerce data.frame to FCS
FCSobj2<-as(coerced.df, "FCS")
is(FCSobj2, "FCS")

##obtaining the dimensions of the data
dim.FCS(FCSobj2)

## equals

## should be TRUE
equals(FCSobj1, FCSobj2, check.filename=TRUE, check.objectname=TRUE)

## default does not check filename or objectname equality
## should be FALSE
equals(foo.FCSobj, dummy.FCSobj)

## fixvars
```

```
foo.FCSobj<-fixvars(foo.FCSobj)
## fluors
data.mat<-fluors(foo.FCSobj)
## metaData
metadata.ls<-metaData(foo.FCSobj)
## plot
## not interesting to plot dummy data

## default plot is pairs.CSP <pairs plot with Contour-images>
## plot(foo.FCSobj)

## can do joint image.parallel.coordinates pairs plots
## plot(foo.FCSobj, image.parallel.plot=TRUE)

## can do marginal image parallel coordinates pairs plots
## plot(foo.FCSobj, image.parallel.plot=TRUE, joint=FALSE)

## print
print(foo.FCSobj)
foo.FCSobj

## show
show(foo.FCSobj)

## summary
summary(foo.FCSobj)
summary(dummy.FCSobj)
```

`"FCSgate-class"` *Class "FCSgate" Flow Cytometry Standard extension to gating*

Description

This class of objects extends the class `FCS-class` to incorporate information from gating which is a procedure by which rows or cells from the data are selected via one or two dimensional value restrictions or gating ranges.

Objects from the Class

Objects can be created by calls of the form `new("FCSgate", ...)`. Essentially this new object includes the `FCS-class` object.

Slots

`gate`: Object of class `"matrix"` containing the gating indices such that each column corresponds to a different gating procedure/index and the rows correspond to the positions of the **original** row/cell observations.

`history`: Object of class `"vector"` containing the gating history strings such that each vector element corresponds to a different gating procedure/index and each string contains information about the particular gate, column variables that were used, and other additional comments.

`extractGatedData.msg`: Object of class `"vector"` containing strings describing any extraction that took place corresponding to each gating procedure/index and history string; each string contains information about the particular corresponding gate column position and gate name and what value index was for inclusion/selection (ie, IndexValue.In)

`current.data.obs`: Object of class `"vector"` contains the current data positional values from the original data

`data`: Object of class `"matrix"` which holds integer data such that the columns are the variables (usually immunofluorescence measurements) and the rows are the cell observations.

`metadata`: Object of class `"FCSmetadata"` which holds information about the file, data, and column variables among other items in the header of the original raw FCS binary file.

Extends

Class `"FCS"`, directly.

Methods

No methods defined with class "FCSgate" in the signature.

Note

The methods `createGate` and `icreateGate`, functionally without plots or interactively with plots, respectively, extends the `FCS-class` to the FCSgate-class. Some interactive gating schemes are noted in `FHCRC.HVTNFCS` and `VRC.HVTNFCS`. Further testing after gating is implemented by `runflowcytests` on the particular variable of interest which is usually the Interferon Gamma Immunofluoroescence measurement.

Author(s)

A.J. Rossini, J.Y. Wan, and Zoe Moodie

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics : New York, 2001. pp.279-283.

Jerome H. Friedman and Nicholas I. Fisher. Bump Hunting in High-Dimensional Data. Tech Report. October 28, 1998.

J. Paul Robinson, et al. Current Protocols in Cytometry. John Wiley & Sons, Inc : 2001.

Mario Roederer and Richard R. Hardy. Frequency Difference Gating: A Multivariate Method for Identifying Subsets that Differe between Samples. Cytometry, 45:56-64, 2001.

Mario Roederer and Adam Treister and Wayne Moore and Leonore A. Herzenberg. Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences. Cytometry, 45:37-46, 2001.

Keith A. Baggerly. Probability Binning and Testing Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared Test. Cytometry, 45:141-150, 2001.

See Also

`createGate`, `icreateGate`, `extractGatedData`, `extractGateHistory`, `FHCRC.HVTNFCS`, `VRC.HVTNFCS`, `"FCS-class"`, `runflowcytests`

Examples

```
default.FCSgateobj<-new("FCSgate")
```

`"FCSggobi-class"` *Class "FCSggobi" : Dynamic Plots*

Description

This class supports the plotting of "FCS-class" objects.

Objects from the Class

Objects can be created by calls of the form `new("FCSggobi", ...)`.

Slots

`dataName`: Object of class `"character"`.

`ggobiLink`: Object of class `"list"`.

Methods

No methods defined with class "FCSggobi" in the signature.

Note

Still in progress of coding

Author(s)

A.J. Rossini

References

See 'library(ggobi)'.

See Also

'ggobi' in 'library(ggobi)', `xgobi.FCS`

`"FCSmetadata-class"` *Class "FCSmetadata" Metadata portion of a Flow Cytometry Standard object*

Description

Information from the HEADER and TEXT of a raw binary FCS file about the data and other parameters are stored in the metadata.

Objects from the Class

Objects can be created by calls of the form `new("FCSmetadata", ...)`.

Slots

`mode`: Object of class `"character"` the "$MODE" mode of the raw binary FCS file

`size`: Object of class `"numeric"` the "$TOT" row dimension of the data; describing the number of observations or cells

`nparam`: Object of class `"numeric"` the "$PAR" column dimension of the data; describing the number of parameters

`shortnames`: Object of class `"vector"` the "$PnN" short names corresponding to the column variables of the data; these names are generally non-descript and are not used as the names of the columns of the data

`longnames`: Object of class `"vector"` the "$PnS" long names used for the column variables of the data

`paramranges`: Object of class `"vector"` the "$PnR" maximum value corresponding to the column variables

`filename`: Object of class `"character"` path and/or name of the **original** raw binary FCS object

`objectname`: Object of class `"character"` the name of the original, `FCS-class` object

`original`: Object of class `"logical"` the original status of the current object

`fcsinfo`: Object of class `"list"` the other parameters and values in the HEADER and TEXT of the raw binary FCS file

Methods

"[" `signature(x = "FCSmetadata")`: Extracts the metadata slots or metadata@fcsinfo slots by using a single character name index; Extracts the metadata@fcsinfo slots by using a single or vector of numerical indicies

"[<-" `signature(x = "FCSmetadata")`: Replaces the metadata slots or metadata@fcsinfo slots by using a single character name index; Replaces the metadata@fcsinfo slots by using a single or vector of numerical indicies;Adds a new slot to the metadata@fcsinfo

"[[" `signature(x = "FCSmetadata")`: Extracts the metadata slots or metadata@fcsinfo slots by using a single character name index; Extracts the metadata@fcsinfo slots by using a single or vector of numerical indicies

"[[<-" `signature(x = "FCSmetadata")`: Replaces the metadata slots or metadata@fcsinfo slots by using a single character name index; Replaces the metadata@fcsinfo slots by using a single or vector of numerical indicies;Adds a new slot to the metadata@fcsinfo

"print-methods" `signature(x = "FCSmetadata")`: prints the original status, the objectname, filename, and dimensions of the data

"show-methods" `signature(object = "FCSmetadata")`: same as 'print'

"summary-methods" `signature(object = "FCSmetadata")`: summaries the metadata in a string output

Note

For more information about the different parameters in the metadata@fcsinfo slot, please look at the documentation for `read.FCS`.

Author(s)

A.J. Rossini, J.Y. Wan, and Zoe Moodie

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics : New York, 2001. pp.279-283.

Jerome H. Friedman and Nicholas I. Fisher. Bump Hunting in High-Dimensional Data. Tech Report. October 28, 1998.

J. Paul Robinson, et al. Current Protocols in Cytometry. John Wiley & Sons, Inc : 2001.

Mario Roederer and Richard R. Hardy. Frequency Difference Gating: A Multivariate Method for Identifying Subsets that Differe between Samples. Cytometry, 45:56-64, 2001.

Mario Roederer and Adam Treister and Wayne Moore and Leonore A. Herzenberg. Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences. Cytometry, 45:37-46, 2001.

Keith A. Baggerly. Probability Binning and Testing Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared Test. Cytometry, 45:141-150, 2001.

See Also

`read.FCS`, `"FCS-class"`, `"print-methods"`, `"show-methods"`, `"summary-methods"`, `"[-methods"`, `"[[-methods"`, `"[<--methods"`, `"[[<--methods"`

Examples

```
default<-new("FCSmetadata")

some.meta<-new("FCSmetadata", fcsinfo=list("comment"=rep("none", 10)),
mode="none", nparam=0, size=0)

## extract/subset the metadata
some.meta[["nparam"]]
some.meta["paramranges"]
## replace the metadata/subsetassign the metadata
## 3 parameters with ranges
some.meta[["nparam"]]<-3
some.meta["paramranges"]<-rep(1,3)
## show
show(some.meta)
## print
print(some.meta)
some.meta
## summary
summary(some.meta)
```

`"FCSsummary-class"` *Class "FCSsummary" Summary object for a "FCS-class"*

Description

The data summary statistics along with metadata output help summarize a "FCS-class" object using the "summary" method.

Objects from the Class

Objects can be created by calls of the form `new("FCSsummary", ...)`.

Slots

`num.cells`: Object of class `"numeric"` the number of cells or rows from the data

`num.param`: Object of class `"numeric"` the number of parameters or columns from the data

`univariate.stat`: Object of class `"matrix"` five-number summary including the standard deviation of all the column variables

`metadata.info`: Object of class `"list"` with the following slots: "Description", "ColumnParametersSummary", and "fcsinfoNames".

Methods

"print-methods" `signature(x = "FCSsummary")`: prints the output of the summary statistics of the data and the metadata

"show-methods" `signature(object = "FCSsummary")`: same as "print"

Author(s)

A.J. Rossini, J.Y. Wan, and Zoe Moodie

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics : New York, 2001. pp.279-283.

Jerome H. Friedman and Nicholas I. Fisher. Bump Hunting in High-Dimensional Data. Tech Report. October 28, 1998.

J. Paul Robinson, et al. Current Protocols in Cytometry. John Wiley & Sons, Inc : 2001.

Mario Roederer and Richard R. Hardy. Frequency Difference Gating: A Multivariate Method for Identifying Subsets that Differe between Samples. Cytometry, 45:56-64, 2001.

Mario Roederer and Adam Treister and Wayne Moore and Leonore A. Herzenberg. Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences. Cytometry, 45:37-46, 2001.

Keith A. Baggerly. Probability Binning and Testing Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared Test. Cytometry, 45:141-150, 2001.

See Also

`"FCS-class"`, `"show-methods"`, `"print-methods"`

Examples

```
default.sum<-new("FCSsummary")

## show, print
default.sum
```

`FHCRC.HVTNFCS`	*Fred Hutchinson Cancer Research Center Sequential Gating Procedure proposed by Julie McElrath's Lab*

Description

This function uses `icreateGate` and `createGate` to select the datapoints which are of particular interest. The selection process is realized in an index column which is added to the data of the FCS object. In particular, after a series of gating/datapoint selection sequences, the interferon gamma variable is of interest.

To row reduce the data of the FCS object, the function, `extractGatedData` should be used on the last gate index to obtain the rows/cells and then should be used again to subset across columns to obtain the gamma interferon column.

Usage

```
FHCRC.HVTNFCS(myFCSobj, gate1.vars = c(1, 2), gate2.vars = c(5, 7),
                   gate3.vars = c(3, 4),MY.DEBUG = FALSE)
```

Arguments

`myFCSobj`	a FCS object
`gate1.vars`	The vector of column variable positions corresponding to Forward Scatter and Side Scatter variables for the first gate; default is column positions 1 and 2 respectively
`gate2.vars`	The vector of column variable positions corresponding to cd3 and cd8 variables for the second gate; default is column positions 5 and 7 respectively
`gate3.vars`	The vector of column variable positions corresponding to cd69 and Interferon Gamma variables for gate 3; default is column positions 3 and 4 respectively
`MY.DEBUG`	if TRUE, then will print the debugging statements; otherwise, if FALSE, then will surpress the debugging statements; default is FALSE

Details

The Selection Sequence made by Julie McElrath's Lab is the following:

gate1:bidcut: Forward Scatter VS Side Scatter

single gate (Select the lymphocytes–central cluster)

gate2:bidcut: cd3 VS cd8

gate 2.1: (Select cd3+/cd8-)

gate 2.2: (Select cd3+/cd8+)

gate3:biscut: cd69 vs Interferon Gamma

gate 3.1: (Select +/+ which are the cd4+ cells (from gate2.1)) gate 3.2: (Select +/+ which are the cd8+ cells (from gate2.2))

In General, the types of Gating/Cutting that are used in this gating scheme are the following:

uniscut = univariate single cut (Selection of the positive/right half)

biscut = bivariate single cut (Selection of the +/-, -/-. +/+, or -/+ quadrant)

bidcut = bivariate double cut (Selection of the center rectangle that results)

Value

`FCS object`	with the following slots:
`data`	A augmented dataframe with the added-on gating column variables/indices
`metadata`	a FCSmetadata object with the information about the gating column variables: $PnR (gating range), $PnN (gating variable's shortname/unused name in the data of the FCS object), $PnS (gating variable's longname/used name), and other slot information

WARNING

This gating scheme is not standard, and there may have been changes to the gating scheme. This gating scheme only serves as an example, which demonstrates the use of `createGate`,`icreateGate` and `"[[-methods"` which extracts the metadata information (eg. in order to obatin information about a previous gating index/column variable

Note

The "FHCRC" data from the **rfcdorig** package can be used for this sequential gating scheme.

Author(s)

A.J. Rossini and J.Y. Wan

References

Julie McElrath, PhD

See Also

`createGate`, `icreateGate`, `showgate.FCS`, `VRC.HVTNFCS`, `plotvar.FCS`, `"[-methods"`, `"[[-methods"`

Examples

```
if (require(rfcdmin)){

  data.there<-is.element("MC.053",objects())
  if ( ( sum(data.there) != length(data.there) )){
    ## obtaining the FCS objects from VRC data
    data(MC.053min)
  }

  if (interactive()==TRUE){
    par(mfrow=c(4,2))
    MC.053.FHCRC<-FHCRC.HVTNFCS(MC.053)
  }
}
```

`ImageParCoord` *Image Parallel Coordinates Plot: Joint and marginal*

Description

This function constructs an image plot in which a rectangular grid structure displays the change of observations from the value of one variable to the value of the next variable. The vertical axis of the image plot denotes the value of the variables that are labeled on the horizontal axis. Traditionally, the lines in a parallel coordinates plot represent the movement of each observation from one variable to the next, but in this case a colored image transition column will represent the movement of observations from cell to cell in the image grid produced by horizontal bins on the vertical axis and vertical divisions between variables and transitions between variables labeled on the horizontal axis. Lines with scaled widths overlaying the image plot indicate the movement of observations from binned values of one variable to the binned values of another (either marginally and only between pairs of variables using `ImageParCoord` OR jointly across all variables using `JointImageParCoord`). Histograms for each variable and the transitions between the variables can be plotted as well.

Usage

```
ImageParCoord(x,
              num.bins=10,
              range.var=range(x),
              break10 = NULL,
              joint=FALSE,
              title="",
              use.shortnames=FALSE,
              color.image=gray((25:5/25)[-c(1,2,3, 4, 5, 6)]),
              xwidth.scale=5,
              ntrans=1,
              legend.plotted=TRUE,
              legend.shrink = 0.9,
              hist.plotted=FALSE,
              image.plotted=TRUE,
              para.plotted=FALSE,
              lines.plotted=TRUE,
              lwd.vec=1:7,
              lty.vec=rep(1,7),
              col.vec=7:1,
              range.image=c(0,dim(x)[1]),
              horizontal.legend = TRUE,
              offset.legend=0.03,
              nlevel.legend=length(color.image),
              xlab.image="",
              ylab.image="Bins",
                MY.DEBUG=TRUE,...)

JointImageParCoord(x,
    num.bins=10,
    range.var=range(x),
    break10=NULL,
    title="",
    use.shortnames=FALSE,
    color.image=gray((25:5/25)[-c(1,2,3, 4, 5,6)]),
    xwidth.scale=5,
    ntrans=1,
    legend.plotted=TRUE,
    legend.shrink = 0.9,
    hist.plotted=FALSE,
    image.plotted=TRUE,
    para.plotted=FALSE,
    lines.plotted=TRUE,
    lwd.vec=1:7,
    lty.vec=rep(1,7),
    col.vec=7:1,
    range.image=c(0, dim(x)[1]),
    horizontal.legend = TRUE,
    offset.legend=0.03,
    nlevel.legend=length(color.image),
    xlab.image="",
    ylab.image="Bins",
    MY.DEBUG=TRUE,...)
```

Arguments

`x`	data matrix from a FCS object; data has columns as the variables and rows as the cells and assume that all column variables are of the same unit and range
`num.bins`	numeric value denoting the number of horizontal bins on the vertical axis to determine how well-defined/sharp the columns of the image plot are; default value is 10 bins
`range.var`	a 2-dimensional vector denoting the minimum value and the maximum value of the variables to be plotted; default is the range of the FCS object data
`break10`	vector denoting the breaks for the binning on the vertical axis; default is equal interval binning denoted by num.bins unless otherwise specified; the breaks must include the range of the variable; each bin is denoted by an open lower value and a closed upper value, ie, (a,b] where a and b are breakpoints and a<b.
`joint`	boolean; if TRUE then the plots will be joined; default value is TRUE
`title`	character string denoting the title of the image plot; default value is an empty string
`use.shortnames`	Boolean; if TRUE, then the shortnames of the variables will be used in labeling in the plots; otherwise if FALSE, the longnames of the variables will be used; default is FALSE
`color.image`	the color scheme for the image plot; default is gray((25:5/25)[-c(1,2,3, 4, 5, 6)])

`xwidth.scale`	numeric value denoting the horizontal width of the variable and the transitions blocks; default value is 5 units of width
`ntrans`	numeric value denoting the number of transition columns between each pair of variables; default is 1 transition column between each pair of variables
`legend.plotted`	Boolean; if TRUE then the legend is produced in a separate graph/plot; otherwise if FALSE, then no legend plot is made; default is TRUE
`legend.shrink`	numeric to reduce the size of the legend
`hist.plotted`	Boolean; if TRUE then the histogram plots of the variables and the transitions are made; otherwise if FALSE, there is no histogram plots; default value is FALSE
`image.plotted`	Boolean; if TRUE, then the image parallel coordinates plot is displayed; otherwise if FALSE, the plot is surpressed; default is TRUE
`para.plotted`	Boolean; if TRUE, then the parallel coordinates plot is displayed; otherwise if FALSE, the plot is surpressed; default is TRUE
`lines.plotted`	Boolean; if TRUE, then superimposed binned parallel coordinate lines displayed on top of the existing plot; otherwise if FALSE, the plot is surpressed; default is TRUE; Note that image.plotted has to be TRUE to see the superimposed image and parallelCoordinates lines
`lwd.vec`	vector denoting the line width sizes to be used in the lines overlaying the image parallel coordinates plot; default value is an integer vector from 1 to 7
`lty.vec`	vector denoting the line type (solid or dotted, etc) for the corresponding line width in lwd.vec; the default is to have a solid line for each line width
`col.vec`	vector denoting the color for each line with the corresponding line width in lwd.vec and line type in lty.vec; the default is to have colors ranging from yellow to black (in that order).
`range.image`	2-dimensional numerical vector denoting the range of the number of counts in the image block to be plotted. The default value is to have a vector with a mininum value of zero and to have a maximum dependent on the number of cells/rows and bins
`horizontal.legend`	default value is TRUE
`offset.legend`	default value is 0.03
`nlevel.legend`	default value is the length of the color.image vector
`xlab.image`	a character string denoting the label of the horizontal x-axis on the image plot; default value is an empty string
`ylab.image`	a character string denoting the label of the vertical y-axis on the image plot; default value is "Bins"
`MY.DEBUG`	a boolean; if TRUE then debugging statements for the binning are output, otherwise if FALSE, the statements are surpressed; default is TRUE
`...`	graphical parameters for `plot` may also be passed as arguments to this function

Details

The result is to have an image block or matrix. Each variable was binned according to the number of bins specified by the option num.bins.

A point-slope line formula was used to determine the counts in the transition block (a matrix of the same transition column across a certain number of rows defined by ntrans and x.width options) between two variables. For each pair of column variables, the horizontal positions of the two variables were regressed on the bin position of the particular observation in order to obtain a point-slope line formula. Thus, for each row observation, one could predict the particular bin that it passed through for the transition block between two known bin values of the two variables.

The following is the point-slope formula for each pair of column variables:

$bin.predicted = slope * (xpos.trans - xpos.V1) + bin.V1$

$bin.predicted$ a row observation's predicted bin value for the specific transition column

$slope$ the slope of the line determined by dividing the difference between the bin values of variable 1 (V1) and variable 2 (V2) by the difference between the horizontal, x-axis positions of V1 and V2: $slope = (bin.V2 - bin.V1)/(xpos.V2 - xpos.V1)$

$xpos.trans$ the x-axis, horizontal position of the transition column for the particular row observation

$xpos.V1$ the x-axis, horizontal position of V1 for the particular row observation

$bin.V1$ a row observation's bin value for V1

Please note that the lines are only marginal. They denote the number of cells moving only between adjacent pairs of variables. To view the cells jointly across all variables, the function `JointImageParCoord` should be used.

The line widths in the image parallel coordinates plot were scaled by the following equation:

$x = (m - 1) * ((n - i)(a - i)) + 1$

x is the scaled size for a particular line

m is the maximum line width size denoted by max.lwd from the function signature

n is the number of observations denoted by the line

i is the minimum number of observations denoted by a line

a is the maximum number of observations denoted by a line

Value

The image parallel coordinates plot with overlayed lines and a legend for the lines, the traditional parallel coordinates plot without the image, and histograms of the variables and the transitions are displayed upon user request as well as a list of the following:

`image.block`	a matrix denoting the number of observations in each cell of the image plot
`line.info`	list of matrices in which each matrix corresponds to the the line information between a pair of variables. Each matrix has three columns. The first two columns are the values of unique bin patterns between the pair of column variables, and the third column is the number of observations with that particular pattern.
`breaks`	vector of breaks for binning on the vertical axis for the values of the variables

WARNING

On some workstations, some colors may not be able to be allocated using `rainbow` or `heat.colors` as the image.color.

Note

Other color images can be used (see the example), but please be advised of the color scheme.

Probability binning can be incorporated by using the signature option break10 to denote the breaks from probability binning.

Author(s)

A.J. Rossini and J.Y. Wan

See Also

`parallelCoordinates`, `rainbow`, `heat.colors`, `ContourScatterPlot`, `ProbBin.FCS`, `gate.IPC`

Examples

```
 if (require(rfcdmin)){

   data.there<-is.element(c("st.1829", "unst.1829", "unst.DRT", "st.DRT"),objects())
   if ( ( sum(data.there) != length(data.there) )){
     ## obtaining the FCS objects from VRC data
     data(VRCmin)
   }

   if (interactive()==TRUE){
   par(mfrow=c(3,3))

   ImageParCoord(unst.1829@data[1:1000, 1:3], num.bins=16,
             title="1000 obs 16 bins 5 trans", ntrans=5)
   ## joint line plot
   ImageParCoord(unst.1829@data[1:1000,c(1,2,3)], num.bins=16,joint=TRUE,
             title="1000 obs 16 bins 5 trans", ntrans=5,legend.plotted=FALSE)

   ## color image is changed
   ImageParCoord(unst.1829@data[1:1000,c(1,2,3)], num.bins=20,
             title="1000 obs 20 bins 5 trans", color.image=rainbow(16,
             start=.4, end=.1), ntrans=5)

   par(mfrow=c(3,3))
   ImageParCoord(unst.1829@data[1:1000,c(1,2,3)], num.bins=20,
             title="1000 obs 20 bins 10 trans", ntrans=10)
   ## joint line plot

   ImageParCoord(unst.1829@data[1:1000,c(1,2,3)], num.bins=20,joint=TRUE,
             title="1000 obs 20 bins 10 trans", ntrans=10)

   ## plot the parallel coordinates plot also
   par(mfrow=c(2,2))
   ImageParCoord(unst.1829@data[1:1000,c(1,2,3)], 1:1000, num.bins=16,
              color.image=gray((25:5/25)[-c(1, 2, 3, 4, 5, 6,7)]),
              title="1000 obs 16 bins 5 trans", ntrans=5,
              para.plotted=TRUE)

  ## plot the parallel coordinates plot also
  par(mfrow=c(2,2))
  ImageParCoord(unst.1829@data[1:1000,c(1,2,3)],joint=TRUE,
              1:1000, num.bins=16,
              color.image=gray((25:5/25)[-c(1, 2, 3, 4, 5, 6,7)]),
              title="1000 obs 16 bins 5 trans", ntrans=5,
              para.plotted=TRUE)

  ##histograms only
  par(mfrow=c(3,3))
  ImageParCoord(unst.1829@data[1:1000,c(1,2,3)], num.bins=10,
             title="1000 obs 10 bins 1 trans",
             ntrans=1, hist.plotted=TRUE,
             image.plotted=FALSE, legend.plotted=FALSE,
             lines.plotted=FALSE)

  ## histograms and images
  par(mfrow=c(3,3))
  ImageParCoord(unst.1829@data[1:1000,c(1,2,3)],
           num.bins=10,
           title="1000 obs 10 bins 5 trans",
           ntrans=5, hist.plotted=TRUE)

  ## legend only
  ImageParCoord(unst.1829@data[1:1000,c(1,2,3)], num.bins=10,
             title="1000 obs 10 bins 5 trans", ntrans=5, legend.plotted=TRUE,
             image.plotted=FALSE, lines.plotted=FALSE)

 ImageParCoord(unst.1829@data[1:1000,c(1,2,3)],joint=TRUE,
             num.bins=10,
             title="1000 obs 10 bins 5 trans",
             ntrans=5, legend.plotted=TRUE,
             image.plotted=FALSE, lines.plotted=FALSE)
  }
}
```

KS.flowcytest *Kolmogorov Smirnoff Test 2-sample*

Description

Provides a Kolmogorov Smirnoff 2-sample Test to determine if the distribution of the control data is different from the distribution of the stimulated data (for which both datasets are of the same variable). See also the function 'ks.test' in the **stats**. A density plot made by the function 'bkde' in **KernSmooth** package is also shown.

Usage

```
KS.flowcytest(controldata, stimuldata,
                title="", varname = "", yupper = 0.01,
                xlimit = c(0, 1025), alternative="two.sided",
                KS.plotted=TRUE,
                MY.DEBUG=TRUE,...)
```

Arguments

`controldata`	a vector of numeric values of the control data
`stimuldata`	a vector of numeric values of the stimulated/case data
`title`	character string of the plot title
`varname`	character string of the name of the variable
`yupper`	the upper limit of the densities calculated
`xlimit`	a vector indicating the range of the controldata and the stimuldata
`alternative`	character string of the alternative hypothesis: 1. "two sided" : Two sided alternative hypothesis 2. "less": One sided alternative hypothesis: controldata distribution is less than the stimuldata distribution 3. "greater" One sided alternative hypothesis: controldata distribution is greater than the stimuldata distribution
`KS.plotted`	boolean to display the corresponding plot; default is TRUE and the plot will be displayed
`MY.DEBUG`	boolean; if TRUE, the test is printed out with comments; if FALSE then these comments are surpressed
`...`	parameters for the stimuldata distribution specified in `ks.test`

Details

In general, the control and the stimulated data come from the Interferon Gamma Data Variable of a FCS R object.

Value

`pval.2sid.KS`	p value of the two sided Kolmogorov Smirnoff test
`Alt.Hypoth.KS`	The Alternative Hypthesis as a string
`method.KS`	the method used
`dataname.KS`	the name of the data

A superimposed plot of the densities of the control and the stimulated dataset is also displayed.

WARNING

Usually the FCS object is gated and subset prior to this testing and analysis.

Note

Other flowcytests are available such as `pkci2.flowcytest`, `ProbBin.flowcytest`, `KS.flowcytest`, which test the equivalence of two sample distributions. Generally, comparing the control and stimulated samples of the interferon gamma variable is of interest.

Author(s)

A.J. Rossini and J.Y. Wan

References

See ks.test

See Also

pkci2.flowcytest, ProbBin.flowcytest, runflowcytests, ks.test, bkde

Examples

```
## different distributions
control<-rnorm(1000, mean=3, sd=.7)
stimulated<-rnorm(1000, mean=2, sd=.5)

if (interactive()==TRUE) {
  output.same <- KS.flowcytest(control, stimulated,
                               title="Different Distributions",
                               varname="Interferon Gamma",
                               yupper=1, xlimit=c(-5,8))
}
## same distribution
stimulated2<-rnorm(1000, mean=3, sd=.7)
if (interactive()==TRUE) {
  output.diff <- KS.flowcytest(control, stimulated2,
                               title="Same Distributions",
                               varname="Interferon Gamma",
                               yupper=1, xlimit=c(-5,8))
}

## obtaining the FCS objects from VRC data
if (require(rfcdmin)) {
  data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
  if ((sum(data.there) != length(data.there))) {
    ## obtaining the FCS objects from VRC data
    data(VRCmin)
  }

  ## This only serves as an example.  Usually the FCS object is
  ## gated and then subset

  ## HIV negative individual 1829
  ## only the first 2000 cells are selected

  IFN.control<-unst.1829@data[1:2000,4]
  IFN.stimul<-st.1829@data[1:2000,4]

  if (interactive()==TRUE){
    KS.flowcytest(IFN.control, IFN.stimul,
      title="HIV Negative Individual 1829", varname="Interferon Gamma",
      yupper=.006)
  }
  ## HIV positive individual DRT
```

```
        ## only the first 2000 cells are selected

        IFN.control2<-unst.DRT@data[1:2000,4]
        IFN.stimul2<-st.DRT@data[1:2000,4]

        if (interactive()){

        KS.flowcytest(IFN.control2, IFN.stimul2,
              title="HIV Positive Individual DRT", varname="Interferon Gamma",
              yupper=.006)
      }
      ## This is an artifical example, but one would expect the
      ## distributions of the stimulated and control samples
      ## to be the same in the HIV negative individual 1829
      ## and to be different in the HIV positive individual DRT
      ## The test in this example is a bit contrived but
      ## the bigger picture is achieved.
    }
```

MODE — *Estimate the highest mode of a multimodale distribution*

Description

MODE returns the highest mode of a multimodale distribution estimate for a given data vector

Usage

```
MODE(x, na.rm=TRUE)
```

Arguments

`x`	numeric vector
`na.rm`	logical

Value

`x`	highest mode

Author(s)

Nolwenn Le Meur

See Also

`plotQA.FCS`

Examples

```
set.seed(12345)
x<-rnorm(50)
h<-MODE(x)
```

`PercentPos.FCS` *Calculate the Percent Positive given a percentile*

Description

From a sample of observations, the percentile for a given percent is computed as the value in which there is a given percent of observations that are lower than it. Using `percentile.FCS` will obtain the percentile of interest in a given vector of values.

Given a sample of observed values, the percent positive over a certain percentile value will be calculated and output by using `PercentPos.FCS`.

Usage

```
percentile.FCS(x.vector, percent = 0.999)

PercentPos.FCS(st.data, percentile)
```

Arguments

`x.vector`	numerical vector of observations usually from the control data
`percent`	numeric; the percent at which to obtain the percentile
`st.data`	numerical vector of observations; usually of the cytokine response of the stimulated sample
`percentile`	numerical value of the threshold; usually the 99.9th percentile of the corresponding unstimulated/control sample

Details

Specifically `percentile.FCS` is used to obtain the percentiles for `PercentPos.FCS` and `ROC.FCS` in the analysis of the upper tail distributions of the stimulated and controls samples of cytokine responses, especially of the Interferon Gamma variable, among HIV positive and HIV negative individuals. This function and analysis can be applied to different scenerios as well.

Usually the Interferon Gamma variable from the FCS object (after gating and subsequent subsets (See `createGate` and `extractGatedData`)), is of interest. The percentile is obtained from the unstimulated or control sample and 100* Percent positives among the cells/observations of the stimulated sample is obtained based on the 99.9th percentile of the control sample. There are differences in the tails of these distributions (stimulated versus control) between HIV positive and HIV negative samples that might better distinguish HIV positive and HIV negative samples. This method was proposed by Zoe Moodie.

Value

For `percentile.FCS`:

the percentile is returned; the percentile is defined as the numeric value of the observation at the which there is a given percent of observations below this value; the value's label or name is the position of the value in the input vector 'controldata'

For `PercentPos.FCS`:

`percent.pos`	the fraction of the observations above or equal to the threshold/percentile
`total.num`	total number of observations in the sample

Note

Please note that Percentage Positive = 100 * (percent positive).

Author(s)

A.J. Rossini and J.Y. Wan

References

Zoe Moodie and Mario Roederer

See Also

data 'PerPosROC' in **rfcdorig** package, `ROC.FCS`

Examples

```
if (require(rfcdmin)){

data.there<-is.element(c("st.1829", "unst.1829", "unst.DRT", "st.DRT"),objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(VRCmin)
}
#hiv negative one individual, 1829
#stimulated sample
INFg.st.neg<-st.1829@data[,4]
#control sample
INFg.unst.neg<-unst.1829@data[,4]

#hiv positive one individual, DRT
#stimulated sample
INFg.st.pos<-st.DRT@data[,4]
#control sample
INFg.unst.pos<-unst.DRT@data[,4]

c.neg<-percentile.FCS(INFg.unst.neg)
c.pos<-percentile.FCS(INFg.unst.pos)

#percent positive for two individuals
p.neg<-PercentPos.FCS(INFg.st.neg, c.neg)
p.pos<-PercentPos.FCS(INFg.st.pos, c.pos)

### percentage positive
ptg.neg<-100*p.neg$percent.pos
ptg.pos<-100*p.pos$percent.pos
}
```

ProbBin.FCS — *ProbBin.FCS R-object: Probability binning of 2 samples*

Description

Constructs a list of histogram objects and other variables on the probability binning between 2 samples, usually the stimulated and unstimulated data (post gating).

Usage

```
ProbBin.FCS(controldata, stimuldata, N, varname = "",
PBspec = c("by.control", "combined"), MY.DEBUG = TRUE, ...)
```

Arguments

`controldata`	a vector of the unstimulated sample data (of 1 variable)
`stimuldata`	a vector of the stimulated sample data (of 1 variable)
`N`	the number of observations per a bin
`varname`	character string of the name of the variable (optional)
`PBspec`	The type of probability binning either:
"by.control"	in which the breaks for the bins are based on the unstimulated having N observations in each bin
"combined"	in which the breaks for the bins are based on the combined dataset (stimulated and unstimulated) having N observations in each bin
`MY.DEBUG`	If TRUE, then debugging statements will be printed; default is TRUE.
`...`	other options besides 'plot' and 'br' in hist function

Details

Based on either the control data or the combined data, breaks for the bins are determined by having a specific number of observations fall in each bin. These breaks are then applied to the stimulated data or both the control and stimulated data, respectively. The resulting two histograms (one of the stimulated data and the other of the control data) are the result of this probability binning method.

Value

`unst.hist`	histogram object of the control/unstimulated data
`st.hist`	histogram object of the stimulated data
`PB`	type of Probability binning: either "by.control" or "combined"
`N.in.bin`	number in each bin
`varname`	character string of the variable name

WARNING

Gating and subsetting should precede the analysis and the use of this function. It is a good idea to implement `icreateGate` or `createGate` and `extractGatedData` before this analysis on univariate data.

Note

Further graphing & testing can be implemented via the following functions in rflowcyt package:plot.ProbBin.FCS, summary.ProbBin.FCS, ProbBin.flowcytest

Author(s)

Zoe Moodie, A.J. Rossini, J.Y. Wan

References

Mario Roederer, et al. "Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences" Cytometry 45:37-46 (2001).

See Also

hist, breakpoints.ProbBin, plot.ProbBin.FCS, summary.ProbBin.FCS, ProbBin.flowcytest, is, as

Examples

```
if (require(rfcdmin)){

data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(VRCmin)
}
## This only serves as an example.
## Gating/subsetting should precede this analysis
IFN.gamma.1<-unst.1829@data[1:2000,4]
IFN.gamma.2<-st.1829@data[1:2000,4]

#Probability binning using the control dataset to determine the breaks
PB1<-ProbBin.FCS(IFN.gamma.1, IFN.gamma.2, 200,
varname=colnames(unst.1829@data)[4], PBspec="by.control",MY.DEBUG=FALSE)

## Probability Binning using the combined dataset (control & stimulated)
## to determing the breaks
PB2<-ProbBin.FCS(IFN.gamma.1, IFN.gamma.2, 200,
varname=colnames(unst.1829@data)[4], PBspec="combined",MY.DEBUG=FALSE)
}
```

`ProbBin.flowcytest` *Test the equivalence of two univariate sample distributions by using Probability Binning and plots the probability-binned histograms of the two samples*

Description

This function will create a probability binning object called `ProbBin.FCS` and will perform summary statistics and a plot of the two resulting probability-binned histograms. There can be probability binning based on the combined data of the two samples or just based on one sample, which is labled as the control.

Usage

```
ProbBin.flowcytest(controldata,
   stimuldata, N = 100, varname = "",
   AnalyType = c("combined", "by.control"),
   title = "",
   MY.DEBUG = FALSE,
   PBobj.plotted=TRUE,
   plots.made=c("both", "stimulated", "unstimulated"),
   ...)
```

Arguments

`controldata`	numerical vector of the control sample univariate data
`stimuldata`	numerical vector of the stimulated sample of the univariate data
`N`	The nummber of observations in each bin on the data specified in the AnalyType option
`varname`	character string of the variable being investigated (usually, in this analysis, the interferon gamma variable is used after gating and subsetting of the FCS object)
`AnalyType`	Probability Binning either "by.control" or based on the "combined" (control and stimulated) data
`title`	character string denoting the title of the plots
`MY.DEBUG`	boolean; if TRUE, debugging statements are printed; default is FALSE
`PBobj.plotted`	boolean; if TRUE then histograms of the ProbBin.FCS object will be plotted; if FALSE, then these plots are surpressed; default is TRUE
`plots.made`	character string denoting which histogram plot should be displayed; default is "both"
`...`	more plotting options; see `plot.ProbBin.FCS` and `hist` for details

Details

The testing performed are summarized in `summary.ProbBin.FCS`, and the plots are produced by `plot.ProbBin.FCS`.

Value

A list consisting of:

`PBinType`	Type of Probability Binning:
"by.control"	uses the control dataset to obtain the breaks/cutoffs to bin the stimulated dataset given a certain number of observations in each bin of the control dataset
"combined"	uses the combined dataset (both control and stimulated datasets) to obtain the breaks/cutoffs for the bins given a certain number in each bin

`control.bins`	single column matrix of the counts in each bin of the control dataset
`stim.bins`	single column matrix of the counts in each bin of the stimulated dataset
`total.control`	numeric; total number in the control dataset
`total.stim`	numeric; total number in the stimulated dataset
`T.chi.unadj`	Roederer's unadjusted normalized PB metric statistic which is normalized by subtracting off the mean and then dividing by the standard deviation. This statistic is approximately standard normal.
`p.val.2tail.z.unadj`	Two-tailed standard normal p-value corresponding to the Roederer's unadjusted normalized PB metric statistic which is approximated as a standard normal
`p.val.1tail.z.unadj`	Upper standard normal one-tailed p-value corresponding to the Roederer's unadjusted PB metric statistic which is approximated as a standard normal
`PBmetric.unadj`	Roederer's unadjusted PB metric which is ((n.c + n.s)/(2*nc.*n.s))*Chi-squared or an unadjusted chi-squared statistic, where n.c is the number of control observations (unbinned) and n.s is the number of stimulated observations (unbinned)
`PBmetric.adj`	Baggerly's adjusted PB metric statistic which is a Chi-squared statistic
`PB.df`	The degrees of freedom of the PB metric (adjusted and unadjusted) which is B-1, where B is the number of bins in the eitherthe control or the stimulated binned data
`p.val.1tail.chi.adj`	Upper one-tailed chi-squared p-value corresponding to Baggerly's adjusted PB metric
`T.chi.adj`	Baggerly's PB metric which is normalized by subtracting off the mean and dividing by the standard deviation; This normalized statistic is approximately standard normal.
`p.val.1tail.z.adj`	Upper one-tailed standard normal p-value corresponding to the Baggerly's adjusted normalized PB metric statistic which is approximated as a standard normal
`p.val.2tail.z.adj`	Standard normal two-tailed p-value corresponding to the Baggerly's adjusted PB metric statistic which is approximated as a standard normal
`pearson.stat`	Pearson's Chi-Squared Statistic with degrees of freedom 2B-1, where B is the number of bins in either the control or the stimulated binned data
`pearson.df`	the degrees of freedom for the chi-squared statistic
`pearson.p.value`	The p-value corresponding to the chi-squared distribution
`pearson.method`	string of the indicating the type of test and options performed
`pearson.dataname`	string of the name(s) of the data
`pearson.observed`	a vector of the observed counts
`pearson.expected`	a vector of the expected counts under the null hypothesis

`pearson.p.val.PB.df`
: Fisher's Chi-squared statistic with degrees of freedom B-1, where B is the number of bins in either the control or the stimulated binned data

Two histograms, one of each sample, are also plotted.

WARNING

Usually the FCS object is gated and subset prior to this testing and analysis.

Note

Other flowcytests are available such as `pkci2.flowcytest`, `ProbBin.flowcytest`, `KS.flowcytest`, which test the equivalence of two sample distributions. Generally, comparing the control and stimulated samples of the interferon gamma variable is of interest.

Author(s)

A.J. Rossini and J.Y. Wan

References

Keith A. Baggerly "Probability Binning and Test Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared test" Cytometry 45: 141:150 (2001).

Mario Roederer, et al. "Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences" Cytometry 45:37-46 (2001).

See Also

`pkci2.flowcytest`, `WLR.flowcytest`, `KS.flowcytest`, `runflowcytests`, `summary.ProbBin.FCS`, `ProbBin.FCS`, `plot.ProbBin.FCS`, `hist`

Examples

```
if (require(rfcdmin)){

data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(VRCmin)
}

## This only serves as an example.  Usually the FCS object is
## gated and then subset

## HIV negative individual 1829
  IFN.control<-unst.1829@data[1:2000,4]
  IFN.stimul<-st.1829@data[1:2000,4]

## probability binning based on the combined data of both samples
if (interactive()==TRUE){
par(mfrow=c(2,2))
test1.out<-ProbBin.flowcytest(IFN.control, IFN.stimul, varname="Interferon Gamma",
AnalyType="combined", N=200, title="HIV negative individual 1829")
}
## HIV positive individual DRT
  IFN.control2<-unst.DRT@data[1:2000,4]
  IFN.stimul2<-st.DRT@data[1:2000,4]

## probability binning based on the control data only
if (interactive()==TRUE){
test2.out<-ProbBin.flowcytest(IFN.control2, IFN.stimul2,
varname="Interferon Gamma", AnalyType="by.control",
N=100, title="HIV negative individual 1829")
}
## This is an artifical example, but one would expect the
## distributions of the stimulated and control samples
## to be the same in the HIV negative individual 1829
## and to be different in the HIV positive individual DRT
## The test in this example is a bit contrived but
## the bigger picture is achieved.
}
```

`ROC.FCS` *ROC (Receiver Operating Characteristic) Curve: Percentage Positives for Flow Cytometry data*

Description

This function plots an ROC curve based on cutoff values from the observed combined dataset of hivpos and hivneg, which both are vectors of patient-specific percentage positives based on the 99.9th percentile of the corresponding control sample distribution. The output contains the sensitivities, 1-specificity,and the observed dataset, cutoff values.

Usage

```
ROC.FCS(hivpos, hivneg, lineopt = 1, colopt = 1, overlay = FALSE)
```

Arguments

`hivpos`	numerical vector of percentage positives for the HIV positive individuals/samples for a given condition
`hivneg`	numerical vector of the percentage positives for the HIV negative individuals/samples for a given condition
`lineopt`	numerical value for the lty option of the plot (line type)
`colopt`	numerical value for the col option of the plot (color type)
`overlay`	Boolean expression as to whether or not the plot is an overlay

Details

See 'PerPosROC' in the 'rfcdorig' package for a description of the input data and how percentage positives are defined.

The ROC curve in the example demonstrates that there is higher predictive ability of using the GAG stimulated samples rather than the PolA or PolB stimulated samples.

Value

Let T be the the percentage positives, c be a given value in c.obs, and HIV+ defined as among HIV positive individuals, and HIV- defined as among HIV negative individuals.

`sensitivity`	numerical vector of the sensitivity=P(T>c ∣ HIV+) calculated corresponding to a given cut-off in c.obs
`spec.complement`	numerical vector of 1-specificity= P(T>c ∣ HIV -)corresponding to a given cut-off in c.obs
`c.obs`	a numerical vector of the cutoffs which were taken to be the values of the observations (the values of the percentage positives of both the HIV positive and HIV negative data)

Author(s)

A.J. Rossini and J.Y. Wan

References

Zoe Moodie and Mario Roederer

See Also

`PercentPos.FCS`, data 'PerPosROC' in 'rfcdorig' package, `percentile.FCS`

Examples

```
if (require(rfcdmin)){

data(PerPosROCmin)

#plotting the gag stimulated 100* percent positives
if (interactive()==TRUE){
GAG<-ROC.FCS(hivpos.gag, hivneg.gag)
#plotting the pola stimulated 100* percent positives
POLA<-ROC.FCS(hivpos.pola, hivneg.pola, lineopt=2, colopt=2, overlay=TRUE)
#plotting the polb stimulated 100* percent positives
POLB<-ROC.FCS(hivpos.polb, hivneg.polb, lineopt=4, colopt=3, overlay=TRUE)
legend(0.7, 0.7, c("gag", "polA", "polB"), col = c(1,2,3), lty=c(1,2,4))
}

}
```

`VRC.HVTNFCS` *Sequential Gating Scheme from Vaccine Research Center (VRC), NIH, Bethesda, MD; Mario Roederer, PhD*

Description

This function uses `icreateGate` and `createGate` to select the datapoints which are of particular interest. The selection process is realized in an index column which is added to the data of the FCS object. In particular, after a series of gating/datapoint selection sequences, the interferon gamma variable is of interest.

To row reduce the data of the FCS object, the function, `extractGatedData` should be used on the last gate index to obtain the rows/cells and then should be used again to subset across columns to obtain the gamma interferon column.

Usage

```
VRC.HVTNFCS(myFCSobj, gate1.vars = c(1, 2), gate2.vars = c(7, 5),
                  gate3.vars = c(5, 3),MY.DEBUG = FALSE)
```

Arguments

`myFCSobj`	a FCS object
`gate1.vars`	The vector of column variable positions corresponding to Forward Scatter and Side Scatter variables for the first gate; default is column positions 1 and 2 respectively
`gate2.vars`	The vector of column variable positions corresponding to cd3 and cd4 variables for the second gate; default is column positions 7 and 5 respectively
`gate3.vars`	The vector of column variable positions corresponding to cd4 and cd8 variables for gate 3; default is column positions 5 and 3 respectively
`MY.DEBUG`	if TRUE, then will print the debugging statements; otherwise, if FALSE, then will surpress the debugging statements; default is FALSE

Details

The Selection Sequence proposed by Mario Roederer is the following:

gate1:bipcut: Forward Scatter VS Side Scatter (Select the lymphocytes–central cluster)

gate2:bidcut: cd3 VS cd4 (want cd3+ cells) (Select the cd3 positive cells on the right of the cutoff)

gate3:biscut: cd4 vs cd8 gate 3.1: (Select cd4+/cd8- cells) (+/- quadrant) gate 3.2: (Select cd4-/cd8+ cells) (-/+ quadrant)

In General, Types of Gating/Cutting:

uniscut = univariate single cut (Selection of the positive/right half)

biscut = bivariate single cut (Selection of the +/-, -/-. +/+, or -/+ quadrant)

bidcut = bivariate double cut (Selection of the center rectangle that results)

Value

`FCS object`	with the following slots:
`data`	A augmented dataframe with the added-on gating column variables/indices
`metadata`	a FCSmetadata object with the information about the gating column variables: $PnR (gating range), $PnN (gating variable's shortname/unused name in the data of the FCS object), $PnS (gating variable's longname/used name), and other slot information

WARNING

This gating scheme is not standard, and there may have been changes to the gating scheme. This gating scheme only serves as an example, which demonstrates the use of `createGate`,`icreateGate` and `extractGateHistory` which extracts the gating information (eg. in order to obtain information about a previous gating index/column variable)

Note

The "VRC" data from the "rfcdorig" package can be used for this sequential gating scheme.

Author(s)

A.J. Rossini & J.Y. Wan

References

Mario Roederer, PhD

See Also

`createGate`, `icreateGate`, `FHCRC.HVTNFCS`, `plotvar.FCS`, `extractGatedData`, `extractGateHistory`

Examples

```
if (require(rfcdmin)){

data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(VRCmin)
}



# HIV positive individual
if (interactive()==TRUE){
par(mfrow=c(4,2))
st.DRT.VRC<-VRC.HVTNFCS(st.DRT)
}

}
```

WLR.flowcytest — *Weighted Logrank Test for testing the differences between time-to-event, survival curves*

Description

Using a survival method developed by Flemming and Harrington, this function examines the difference in the survival curves of two samples in order to determine a distribution difference between the two samples. A plot of the two super-imposed survival curves is displayed.

Usage

```
WLR.flowcytest(controldata, stimuldata, title = "", varname = "",
               na.action.WLR = options()$na.action, rho.test = 0,
               WLR.plotted=TRUE, MY.DEBUG=TRUE)
```

Arguments

`controldata`	numerical vector of observations of the control data for one variable
`stimuldata`	numerical vector of observations of the stimulated data for the same variable as the control
`title`	character string describing the title
`varname`	character string describing the name of the variable
`na.action.WLR`	a missing-data filter function. This is applied to the 'model.frame' after any subset argument has been used. Default is 'options()$na.action' (as quoted from the 'survdiff' documentation from the **survival** package.)
`rho.test`	the exponent, ρ in $S(t)\hat{}\rho$, where S is the Kaplan-Meier estimate of survival; A ρ value of 0 specifies using the weighted log-rank test, and a value of 1 specifies using the Peto & Peto modification of the Gehan-Wilcoxon test.
`WLR.plotted`	boolean; if TRUE, then plot is made; otherwise if FALSE, plotting is surpressed; default=TRUE
`MY.DEBUG`	boolean; if TRUE, the test is printed out with comments; if FALSE then these comments are surpressed

Details

The null hypothesis is that the two survival curves are the same in both samples. If there is a significant difference then a large chi-squared one statistic corresponding to a small p-value (usually $<$ 0.05, where the Type I error rate=alpha=0.05) will suggest this significance.

This function uses 'survdiff' in the **survival** package. The following is a direct quote from the 'survdiff' documentation: "This function (survdiff) implements the G-rho family of Harrington and Fleming (1982), with weights on each death of $S(t)\hat{}\rho$, where S is the Kaplan-Meier estimate of survival.With '$\rho = 0$' this is the log-rank or Mantel-Haenszel test, and with '$\rho = 1$' it is equivalent to the Peto & Peto modification of the Gehan-Wilcoxon test."

In this flowcytometry analysis, we are not dealing with the proportion of survival, persay, but instead in terms of the proportion of observations/cells beyond a certain value of the interferon gamma variable.

Value

`p.val.1sid.chisq.WLR`	p-value associated with a chi-squared statistic with one degree of freedom
`chisq.WLR`	the chi-squared statistic in the test of the difference in survival curves
`n.WLR`	a numeric vector of the number of subjects in the control and the stimulated samples, respectively
`obs.WLR`	numeric vector of the weighted observed number of events in each sample, control and stimulated, respectively
`exp.WLR`	numeric vector of the weighted expected number of events in each sample, control and stimulated, respectively
`var.WLR`	the variance matrix of the test (control, stimulated)

A survival plot is also made with the two survival curves, labeled "Control" and "Stimulated" and super-imposed on one plot.

WARNING

Usually the FCS object is gated and subset prior to this testing and analysis. Also this function requires the library `survival`.

Note

Other flowcytests are available such as `pkci2.flowcytest`, `ProbBin.flowcytest`, `KS.flowcytest`, which test the equivalence of two sample distributions. Generally, comparing the control and stimulated samples of the interferon gamma variable is of interest.

Author(s)

A.J. Rossini and J.Y. Wan

References

Harrington, D. P. and Fleming, T. R. (1982). A class of rank test procedures for censored survival data. Biometrika 69, 553-566.

See Also

`pkci2.flowcytest`, `ProbBin.flowcytest`, `KS.flowcytest`, `runflowcytests`, the function 'survdiff' in the **survival** package.

Examples

```
if (require(rfcdmin)){

data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(VRCmin)
}

## This only serves as an example.  Usually the FCS object is
## gated and then subset
## HIV negative individual 1829
  IFN.control<-unst.1829@data[1:2000,4]
  IFN.stimul<-st.1829@data[1:2000,4]


if (interactive()==TRUE){
par(mfrow=c(2,2))
WLR.flowcytest(IFN.control, IFN.stimul,
title="HIV negative individual 1829",
varname="Interferon Gamma")
}
## HIV positive individual DRT
  IFN.control2<-unst.DRT@data[1:2000,4]
  IFN.stimul2<-st.DRT@data[1:2000,4]

if (interactive()==TRUE){
WLR.flowcytest(IFN.control2, IFN.stimul2,
title="HIV positive individual DRT",
varname="Interferon Gamma")
}
## This is an artifical example, but one would expect the
## distributions of the stimulated and control samples
## to be the same in the HIV negative individual 1829
## and to be different in the HIV positive individual DRT
## The test in this example is a bit contrived but
## the bigger picture is achieved.
}
```

`add.parallel.coordinates`

Add a parallel coordinates line to an existing plot

Description

This function will allow the user to add a parallel coordinates line to an existing plot. The single line can be specified with a certain scale, color, line type, and line width as well as with other `line` options.

Usage

```
add.parallel.coordinates(x, varlabpos = 1:length(x), scaled = FALSE, lty = 1, cc
```

Arguments

`x`	is a vector of variable values made for one cell/individual; the length corresponds to the number of variables on the horizontal x-axis
`varlabpos`	a vector denoting the positions on the x-axis to plot values
`scaled`	Boolean; If TRUE, then the values of x will be on a (0,1) scale; if FALSE, then the original values of x are to be plotted on the vertical axis.
`lty`	numerical value denoting the line type; see `par` for descriptions
`col`	color of the line
`lwd`	line width
`...`	other options from the `lines` function

Value

A parallel coordinates line will be added to the exisiting plot.

Note

This function is deprecated, please use `add.parallelCoordinates`.

Author(s)

A.J. Rossini, J.Y. Wan

See Also

`plot`, `par`, `lines`, `parallelCoordinates`, `ImageParCoord`

Examples

```
if (require(rfcdmin)){

data.there<-is.element("MC.053",objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(MC.053min)
}

dataMC<-MC.053@data


if (interactive()){
par(mfrow=c(2,2))
### subset the data to the first 5 observations because it is too huge
parallelCoordinates(dataMC[c(1:5),-6])

## adding in the 6-th row observation
add.parallel.coordinates(dataMC[6,-6], col="red")

### the same plot is scaled to 0,1 range
parallelCoordinates(dataMC[c(1:5),-6], scaled=TRUE)
## adding in the 6-th row observation
add.parallel.coordinates(dataMC[6,-6], scaled=TRUE, col="red")

## positions on the horizontal x-axis
parallelCoordinates(dataMC[c(1:5),1:4], varlabpos=c(1, 5, 8, 16))
## adding in the 6-th row observation
add.parallel.coordinates(dataMC[6,1:4], varlabpos=c(1,5,8,16),
col="red")
}

}
```

`add.parallelCoordinates` *Add a parallel coordinates line to an existing plot*

Description

This function will allow the user to add a parallel coordinates line to an existing plot. The single line can be specified with a certain scale, color, line type, and line width as well as with other `line` options.

Usage

```
add.parallelCoordinates(x, varlabpos = 1:length(x), scaled = FALSE, lty = 1, col
```

Arguments

`x`	is a vector of variable values made for one cell/individual; the length corresponds to the number of variables on the horizontal x-axis
`varlabpos`	a vector denoting the positions on the x-axis to plot values
`scaled`	Boolean; If TRUE, then the values of x will be on a (0,1) scale; if FALSE, then the original values of x are to be plotted on the vertical axis.
`lty`	numerical value denoting the line type; see `par` for descriptions
`col`	color of the line
`lwd`	line width
`...`	other options from the `lines` function

Value

A parallel coordinates line will be added to the exisiting plot.

Author(s)

A.J. Rossini, J.Y. Wan

See Also

`plot`, `par`, `lines`, `parallelCoordinates`, `ImageParCoord`

Examples

```
if (require(rfcdmin)){

data.there<-is.element("MC.053",objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(MC.053min)
}

dataMC<-MC.053@data
if (interactive()){
par(mfrow=c(2,2))
### subset the data to the first 5 observations because it is too huge
parallelCoordinates(dataMC[c(1:5),-6])

## adding in the 6-th row observation
add.parallelCoordinates(dataMC[6,-6], col="red")

### the same plot is scaled to 0,1 range
parallelCoordinates(dataMC[c(1:5),-6], scaled=TRUE)
## adding in the 6-th row observation
add.parallelCoordinates(dataMC[6,-6], scaled=TRUE, col="red")

## positions on the horizontal x-axis
parallelCoordinates(dataMC[c(1:5),1:4], varlabpos=c(1, 5, 8, 16))
## adding in the 6-th row observation
add.parallelCoordinates(dataMC[6,1:4], varlabpos=c(1,5,8,16),
col="red")
}

}
```

`"addParameter-methods"` *Add a column data variable to the data of a FCS object*

Description

This function enables the user to add a column data variable, "colvar", (which specifies a value for each row/cell) to the data of a "FCS" object and updates the data information in the metadata of a FCS object.

Methods

x = "FCS", colvar = "vector" Adds colvar to the data portion of the "FCS" object; colvar must agree in length with the row dimension of the data matrix

x = "FCS", colvar = "vector", shortname="", longname="", use.shortname=FALSE Other unlisted options in the signature include:

(1) shortname : character string denoting the name of colvar; default value is "".

(2) longname : character string denothing the long name of colvar; default value is "".

(3) use.shortname : boolean; if TRUE then the shortname is assigned to the column variable in the data, otherwise the longname is used; default value is FALSE

`boxplot.FCS` *Create boxplots one parameter of one (or more) FCS object(s)*

Description

Produce box-and-whisker plot(s) of a single column variable specified from the data of one (or more) FCS object(s).

Usage

```
boxplot.FCS(x, varpos=c(1),groups=NULL, xlab, ylab, col,
alternating=TRUE, do.out = FALSE, ...)
```

Arguments

`x`	a list of one (or more) FCS object(s) or a cytoSet object
`varpos`	the numerical column variable position of the data of the FCS object
`groups`	a variable or expression to be evaluated in the data frame specified by 'data', expected to act as a grouping variable within each panel, typically used to distinguish different groups by varying graphical parameters like color and line type
`xlab`	a title for the x axis
`ylab`	a title for the y axis
`col`	The colors for lines and points. Multiple colors can be specified so that each point can be given its own color. If there are fewer colors than points they are recycled in the standard fashion. Lines will all be plotted in the first colour specified.
`alternating`	logical specifying whether axis labels should alternate from one side of the group of panels to the other (for more details see `xyplot`)
`do.out`	logical to specify if the outlier values should be displayed (default is FALSE)
`...`	any other arguments are passed to the `boxplot` function

Details

If several FCS objects are supplied parallel boxplots will be plotted. Other options from the functions `plot`, `boxplot`.

Value

The `boxplot` will output a list with the following components:

`stats`	any other arguments are passed to the `boxplot` function
`n`	any other arguments are passed to the `boxplot` function
`conf`	any other arguments are passed to the `boxplot` function
`out`	any other arguments are passed to the `boxplot` function
`group`	any other arguments are passed to the `boxplot` function
`names`	any other arguments are passed to the `boxplot` function

a vector of names for the groups

Author(s)

N. Le Meur

See Also

boxplot, boxplot.stats

Examples

```
## Example I:
require(rfcdmin)
data(flowcyt.data)

## Draw a boxplot for the Foward Scatter parameter for the time points 1
## and 6 (in this experiment, each time point corresponds to a column of
## a 96 wells plates)
mat <- matrix(c(1:2),1,2,byrow=TRUE)
nf <- layout(mat,respect=TRUE)
boxplot.FCS(flowcyt.data[1:8],varpos=c(1),col=c(1:8),main="FSC across stains time point
boxplot.FCS(flowcyt.data[65:72],varpos=c(1),col=c(1:8),main="FSC across stains time poir

##Example II:
## Read a serie of FCS files
if (require(rfcdmin)) {

##obtaining the location of the fcs files in the data
 pathFiles<-system.file("bccrc", package="rfcdmin")
 drugFiles<-dir(pathFiles)

## reading in the FCS files
 drugData<-read.series.FCS(drugFiles,path=pathFiles,MY.DEBUG=FALSE)
 }

##Draw a boxplot for the Foward Scatter parameter
##for the differents aliquots (of the same cell line)
##tested with different compounds.
boxplot.FCS(drugData,varpos=c(1),col=c(1:8),main="FSC of differents aliquots from the sa
```

---

breakpoints.ProbBin
Obtain break points for Probability binning

Description

To define the break points in data.var in which there are N observations in each bin.

Usage

```
breakpoints.ProbBin(data.var, N)
```

Arguments

`data.var`	a vector of numeric data values for the break points to be determined
`N`	the number of data points between two breaks

Details

This function is used to determine the break points that can be used to specify a `ProbBin.FCS` object as well as a `hist` object.

Please note that each bin in the histograms (in `ProbBin.FCS`) will be determined such that the end point is included (ie, for a<b, (a,b] is the bin interval for break points a & b.

Thus, the output of this function will have min(data.var)-1 as the first break point and max(data.var) as the last break point such that (min(data.var)-1, min(data.var)] is the first bin/interval of the break points.

Value

a vector of the numerical breaks

Author(s)

Zoe Moodie, A.J. Rossini, J.Y. Wan

References

Mario Roederer, et al. "Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences" Cytometry 45:37-46 (2001).

See Also

`ProbBin.FCS` `hist`

Examples

```
  x <- 1:23
  N <- 3

## making a series of cutpoints which have
## an equal number of counts in each bin
  breaks <- breakpoints.ProbBin(x, N)

  hist(x, br=breaks, plot=FALSE)
```

`"checkvars-methods"`
Checks the ranges, dimensions, and names of the metadata based on the current data of an FCS R-object.

Description

Any discrepancy between the metadata and the data of the FCS object is considered as a failure to pass the check. The following is a description of the checks:

1. Dimension check We always check the dimensions (ie, if the data dimensions match with size ("$TOT") and nparam ("$PAR") that are specified in the metadata).

2. Parameter Name check We check the names of the metadata with the names of the data column parameters. Either only the longnames ("$PnS") or the shortnames ("$PnN") of the metadata are checked against the names of the data. Please take note that both ("$PnS") and ("$PnN") ARE NOT BOTH checked.

3. Column Variable Range Check We check the paramranges ("$PnR") specified in the metadata with the column parameter ranges of the data; if the paramranges do not exist in the metadata, then it is noted in the debugging statements.

   Please note that if the metadata@original is FALSE, then the metadata slotNames have a "RFACSadd> > " suffix and are located in metadata@fcsinfo in order to store the current data descriptives. The original data descriptives can be retrieved/checked when metadata@original is set to TRUE; otherwise the current metadata information about the data is retrieved/checked even when the "RFACSadd> > " suffix is not noted in the character index.

   (ie) If metadata@original is FALSE, then metadata[["size"]] will return metadata[["RFACSadd> > $TOT"]], the current row length of the data, while metadata@size will return the number of rows for the original data.

   Note that metadata@original is changed only when a parameter column is added to the data using `addParameter-methods`, when rows of the data are extracted using `extractGatedData` or if the user decides to change the value metadata@original. Using `"["-methods` and `"[<-"-methods` on a "FCS" object will not change the value of metadata@original.

Methods

x = "FCS" boolean value is returned; TRUE if the check passes and FALSE if it does not pass the check.

x = "FCS", MY.DEBUG=TRUE, range.max=NULL Other options in the signature include:

(1) MY.DEBUG : boolean value; if TRUE, then the output statements are printed, otherwise if FALSE, then the statements are surpressed; default is TRUE.

(2) range.max : numeric value describing the true maximum of the data that the checks on the ranges will be compared; default is NULL (ie, the maximum of each column variable in the data is the truth)

coerce-FCSformat *Convert Data Objects*

Description

Convert between rflowcyt and prada data objects.

Details

Objects can be converted (coerced) from one class to another using `as(object, Class)` where `object` is an object to convert and `Class` is the name of the class to convert to. The following conversions are provided:

From:	To:
`FCS`	`cytoFrame`
`cytoFrame`	`FCS`

Note that `cytoFrame` objects are coerced to `cytoFrame` in such a way that the metadata are not stored in the exact same order.

Author(s)

N. Le Meur

See Also

`as` in the `methods` package.

Examples

```
x <- new("FCS")
y <- as(x,"cytoFrame")

##z <- new("cytoFrame")
##z@exprs <- matrix(rnorm(5*2),5,2)
##y <- as(z,"FCS")
```

"coerce-methods" *Coercing an object class to another class*

Description

This method will coerce an object to a specific class using the following call:

`as("class", object)`

where "class" is a specific class detailed below, and 'object' is the specific object to be coerced.

Methods

from = "ANY", to = "array" Coercion or force "ANY" object into "array" object

from = "ANY", to = "call" Coercion or force "ANY" object into "call" object

from = "ANY", to = "character" Coercion or force "ANY" object into "character" object

from = "ANY", to = "complex" Coercion or force "ANY" object into "complex" object

from = "ANY", to = "environment" Coercion or force "ANY" object into "environment" object

from = "ANY", to = "expression" Coercion or force "ANY" object into "expression" object

from = "ANY", to = "function" Coercion or force "ANY" object into "function" object

from = "ANY", to = "integer" Coercion or force "ANY" object into "integer" object

from = "ANY", to = "list" Coercion or force "ANY" object into "list" object

from = "ANY", to = "logical" Coercion or force "ANY" object into "logical" object

from = "ANY", to = "matrix" Coercion or force "ANY" object into "matrix" object

from = "ANY", to = "name" Coercion or force "ANY" object into "matrix" object

from = "ANY", to = "numeric" Coercion or force "ANY" object into "numeric" object

from = "ANY", to = "single" Coercion or force "ANY" object into "single" object

from = "ANY", to = "ts" Coercion or force "ANY" object into "ts" object

from = "ANY", to = "vector" Coercion or force "ANY" object into "vector" object

from = "ANY", to = "NULL" Coercion or force "ANY" object into "NULL" object

from = "FCS", to = "matrix" Coercion or force "FCS" object into "matrix" object by returning only the data matrix of the "FCS" object

from = "FCS", to = "data.frame" Coercion or force "FCS" object into "data.frame" object by returning only the data data.frame of the "FCS" object

from = "matrix", to = "FCS" Coercion or force "matrix" object into "FCS" object by setting the "matrix" object as the 'data' slot and having a default 'metadata' slot of class "FCSmetadata".

from = "data.frame", to = "FCS" Coercion or force "data.frame" object into "FCS" object by setting the "data.frame" object as the 'data' slot and having a default 'metadata' slot of class "FCSmetadata".

`convertS3toS4` *Converts S3 class FCS object to S4 class FCS object*

Description

This function will update any S3 class `FCS` object to S4 class.

Usage

```
convertS3toS4(S3file, myFCSobj.name = "", fileName = "")
```

Arguments

`S3file`	S3 Class FCS object location and filename
`myFCSobj.name`	character string indicating the FCS object name
`fileName`	character string indicating the file name of the binary raw FCS data, from which the FCS object originates and which is read by `read.FCS`

Details

The FCS object is obtained as the result of `read.FCS` which has been currently updated to output FCS objects as class S4 instead of S3.

Value

A Class S4 `FCS` object with the following slots:

`data`	matrix of the data, where the rows are the cells/observations and the columns are the different fluoroescence measurements
`metadata`	of class `FCSmetadata` with the following slots:
mode	the mode of the raw binary file
size	numeric value describing the total number of rows or observations/cells
nparam	numeric value describing the number of columns or parameters
shortnames	a vector of the short names of the column variables of the data
longnames	a vector of the long names of the column variables of the data
paramranges	the vector of corresponding ranges or maximum values for each column variable
filename	character string of the name of the raw data file from which the object originates
objectname	character string of the name of the FCS S4 object
original	Boolean value indicating whether the data is the original
fcsinfo	list of other parameters

Author(s)

A.J. Rossini and J.Y. Wan

See Also

`read.FCS`, `FCS`

Examples

```
if (require(rfcdmin)){

   ## if previously read-in as S3 FCS object
   facscan256.fcs<- paste(system.file("fcs", package="rfcdmin"),
                              "facscan256.fcs",
                              sep="/")

   ## reading in the FCS files
   FCSobj.S3<-read.FCS(facscan256.fcs, UseS3=TRUE)

   ## convert to S4 FCS

   FCSobj.S4<- convertS3toS4(FCSobj.S3,
                               myFCSobj.name="FCSobj.S4",
                               fileName=facscan256.fcs)
}
```

`createGate` *Gating of a FCS object: Making a Gating/Selection index column for subsequent extraction*

Description

After the gating procedure, which can be implemented either non-interactively by `createGate` or interactively by `icreateGate`, a `FCSgate` class object is returned with a column variable of indices in which 1 denotes inclusion and 0 denotes inclusion or exclusion, respectively, from the gating ranges or thresholds added as a column to the "gate" matrix, and information: $PnR (gating range), $PnS (longname of the gating index), $PnN (shortname of the gating index) will be added in the "history" string. The message "NONE" is added or updated in the corresponding "extractGatedData.msg" slot. The "current.data.obs" vector is not changed. The interactive gating here will provide contour-image plots and allow the user to input the gatingrange after viewing these plots.

Usage

```
createGate(x, varpos = NULL, gatingrange = NULL, type = c("uniscut",
"bidcut", "biscut", "bipcut"),
biscut.quadrant = c("+/+", "-/-", "-/+", "+/-"),
prev.gateNum = NULL, prev.IndexValue.In = NULL,
comment = "", MY.DEBUG = FALSE)

icreateGate(x, varpos = NULL, gatingrange = NULL, type = NULL,
biscut.quadrant = NULL, prev.gateNum = NULL,
prev.IndexValue.In = NULL,
comment = NULL,
pchtype=".",
MY.DEBUG = TRUE,
prompt.all.options=TRUE)
```

Arguments

`x`	a `FCS` object
`varpos`	one numeric position or vector of two positions of the column variable(s) to gate upon (note: x is the horizontal axis/variable and y is the vertical axis/variable)
`gatingrange`	gating threshold range in one of the following formats for each type of gating:
"uniscut"	univariate single cut; gatingrange=x1 (will select/include all points $>\ = x1$), x1 is numeric value
"bidcut"	bivariate double cut: gatingrange=c(x1,x2, y1,y2), a numeric vector of lowerbound, upperbound cutoffs for x and y variables
"biscut"	bivariate single cut:gatingrange=c(x1,y1), a numeric vector of the cutoffs for x and y variables
"bipcut"	bivariate polygonal cut: polygonal thresholds for an n−sided polygon has: (gatingrange=c(c(x1, x2, ...,xn, x1), c(y1, y2, ...,yn, y1)), a vector of vectors which denote the outer points of the polygonal vertices)
`type`	character string of the type of cut/gating:

"uniscut"	univariate single cut: selects datapoints that are greater than or equal to the cutoff value denoted in gatingrange
"bidcut"	bivariate double cut: selects datapoints in the central rectangle formed by two vertical lines (x variable cutoffs) and two horizontal lines (y variable cutoffs)
"biscut"	bivariate single cut: cuts graph into quadrants (selects datapoints in the quadrant denoted by biscut.quadrant)
"bipcut"	bivariate polygonal cut: selects the datapoints in a polygon
`biscut.quadrant`	character string value denoting the (x,y) quadrant that is to be selected; Values are one of the following:
"+/+"	selects the upper right quadrant, where x is positive and y is positive
"−/+"	selects the upper left quadrant, where x is negative and y is positive
"+/−"	selects the lower right quadrant, where x is positive and y is negative
"−/−"	selects the lower left quadrant, where x is negative and y is negative
`prev.gateNum`	numeric column number of the previous subset/gate index in the "gate" matrix of x that should be carried over to this gate. *NOTE: The datapoints not selected in the index specified by prev.colNum will not be selected in this gate either*
`prev.IndexValue.In`	the value of inclusion for the gating index specified by "prev.gateNum"
`comment`	character string denoting the importance of the gating; default is the empty string
`pchtype`	The type of point to plot observations that have been selected using showgate.FCS; default is using "."
`MY.DEBUG`	If TRUE, prints out debugging statements; otherwise if FALSE, the debugging statements are surpressed; default is TRUE
`prompt.all.options`	boolean; if TRUE all other options about the display of plots are prompted for user input in the interactive gating; otherwise, if FALSE, these prompts are surpressed; default is TRUE

Details

If any options in the signature for `icreateGate` are not specified, then these options are prompted for the user to input values.

Use `extractGateHistory` to obtain information about the particular gating/selection index from the "history" string.

Usually the function `extractGatedData` is used to row reduce the data of the FCS object.

For an example of a sequential interactive gating scheme please use `FHCRC.HVTNFCS` for the FCS objects in data(FHCRC) of the 'rfcdorig' package and use `VRC.HVTNFCS` for the FCS objects in data(VRC) of the 'rfcdorig' library.

For basic, non-interactive gating, use `createGate`, and for basic, non-interactive subsetting or data extraction after gating use `extractGatedData`. For basic, non-interactive plotting, use `plotvar.FCS` to plot column variables in an FCS object and `showgate.FCS` to graph the gate and color-in the selected datapoints.

When all gating parameters are input in `icreateGate`, and "prompt.all.options" is set to FALSE, then a gating index is created and appended to the 'gate' matrix and the corresponding plot is shown with the gate without any user input prompts. See 'examples' for details.

Value

A `FCSgate` S4 object is returned that extends the `FCS` object to contain additional slots:

`gate`	a matrix whose columns are the gating indices for the original data
`history`	vector which corresponds to each column gating index in "gate" and holds information about what variables and type of gate that was implemented and for what ranges of values
`extractGatedData.msg`	vector of strings to specify what if any extraction has been implemented using `extractGatedData`; "NONE" specifies no extraction has been implemented on the data for that particular corresponding gating index
`current.data.obs`	vector of the original data row positions that are currently still in the data matrix

Author(s)

A.J. Rossini and J.Y. Wan

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics : New York, 2001. pp.279-283.

Jerome H. Friedman and Nicholas I. Fisher. Bump Hunting in High-Dimensional Data. Tech Report. October 28, 1998.

J. Paul Robinson, et al. Current Protocols in Cytometry. John Wiley & Sons, Inc : 2001.

Mario Roederer and Richard R. Hardy. Frequency Difference Gating: A Multivariate Method for Identifying Subsets that Differe between Samples. Cytometry, 45:56-64, 2001.

Mario Roederer and Adam Treister and Wayne Moore and Leonore A. Herzenberg. Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences. Cytometry, 45:37-46, 2001.

Keith A. Baggerly. Probability Binning and Testing Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared Test. Cytometry, 45:141-150, 2001.

See Also

`extractGatedData`, 'FHCRC' data in the 'rfcdorig' package, `FHCRC.HVTNFCS`, 'VRC' data in the 'rfcdorig' package, `VRC.HVTNFCS`,`extractGateHistory`

Examples

```
## example of interactive gating

    if (require(rfcdmin)) {
      data.there<-is.element("MC.053",objects())
      if ((sum(data.there) != length(data.there))) {
        ## obtaining the FCS objects from VRC data
        data(MC.053min)
      }

      if (interactive()==TRUE) {
        ## icreateGate: The following will prompt the user for
        ## plotting and gating information.

        ## put two plots on one row
        par(mfrow=c(2,2))

        ## uniscut: univariate single cut
        MC.053.iuniscut<-icreateGate(MC.053, varpos=2,
               gatingrange=250, type="uniscut")
        ## IndexValue.In = 1

        ## bidcut: bivariate double cut
        MC.053.ibidcut<-icreateGate(MC.053.iuniscut,
            prev.gateNum=1,prev.IndexValue.In=1, type="bidcut")

        ## biscut: bivariate single cut
        MC.053.ibiscut<-icreateGate(MC.053.ibidcut, type="biscut")
        ## prev.gateNum=2


        ## bipcut: bivariate polygonal cut
        MC.053.ibipcut<-icreateGate(MC.053.ibiscut, type="bipcut")
        ## prev.gateNum=3

        ## user-chosen gate
        MC.053.iuser<-icreateGate(MC.053)

    }

      ## example of creating a gate when parameters are known

      ## uniscut: univariate single cut

      MC.053.gated<-createGate(MC.053, varpos=2, type="uniscut",
                             gatingrange=300, comment="Example")

      if (interactive()){
      ## corresponding icreateGate with a plot and no prompts
      MC.053.igated<-icreateGate(MC.053, varpos=2, type="uniscut",
                             gatingrange=300, comment="plot and gate shown",
                             prompt.all.options=FALSE)
      }
      ## bidcut: bivariate double cut

      MC.053.gated1<-createGate(MC.053, varpos=c(1,2), type="bidcut",
                              gatingrange=c(250, 500, 0,250),
                              comment="Example")
      if (interactive()){
      ## corresponding icreateGate with a plot and no prompts
      MC.053.igated1<-icreateGate(MC.053, varpos=c(1,2), type="bidcut",
                             gatingrange=c(250, 500, 0,250),
                             comment="plot and gate shown",
                             prompt.all.options=FALSE)
      }
      ## biscut: bivariate single cut

      MC.053.gated<-createGate(MC.053, varpos=c(3,4), type="biscut",
                             gatingrange=c(250, 500),
                                  biscut.quadrant="+/-", comment="Example")

      if (interactive()){
      ## corresponding icreateGate with a plot and no prompts
      MC.053.igated<-icreateGate(MC.053, varpos=c(1,2), type="biscut",
                                  gatingrange=c(250, 500),
                                  biscut.quadrant="+/-",
                                  comment="plot and gate shown",
                                  prompt.all.options=FALSE)
      }
      ## bipcut: bivariate polygonal cut

      x.coord<-c(200, 200, 600, 600, 200)
      y.coord<-c(200, 600, 600, 200, 200)
      MC.053.gated2<-createGate(MC.053, varpos=1:2, type="bipcut",
                                   gatingrange=cbind(x.coord, y.coord),
                                   comment="Example")
      if (interactive()){
      ## corresponding icreateGate with a plot and no prompts
      MC.053.igated2<-icreateGate(MC.053, varpos=c(1,2), type="bipcut",
                                  gatingrange=c(x.coord, y.coord),
                                  comment="plot and gate shown",
                                  prompt.all.options=FALSE)
      }
    }
```

`cytoSet-class` *'cytoSet': a class for storing raw data from a quantitative cell-based assay*

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.

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)

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

See Also

readCytoSet, cytoFrame-class

Examples

```
if (require(prada)) {
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, ]

plot(cset[[2]])
}

if (require(rfcdmin) && require(prada)) {

 ##obtaining the location of the fcs files in the data
  pathFiles<-system.file("bccrc", package="rfcdmin")
  drugFiles<-dir(pathFiles)

 ## reading in the FCS files
  drugData<-readCytoSet(path=system.file("bccrc", package="rfcdmin"),
      pattern="[A-Z][0-9][0-9]$")
  }
```

`"dim.FCS-methods"` *Obtaining the dimensions of the data of an "FCS-class" object*

Description

This function returns the dimensions of the data such that the number of rows and the number of columns, respectively, are output in a vector. The number of rows corresponds to the number of cell observations, and the number of columns correspond to the number of parameters or fluorescence measurements and other integer-measured variables.

Methods

x Extracts the dimensions of the data

`emp.f` *Create a guassian kernel density*

Description

emp.f creates a guassian kernel density estimate for x using a bandwidth h

Usage

```
emp.f(x, h)
```

Arguments

`x`	the data vector
`h`	the bandwidth, should be on scale of standardized x's

Details

the definition of bandwidth is different than R's density function, thus will not give you the same reult. Also, emp.f finds the density estimate at every 0.02 values of x. Also, this rescales x by median and the mad for a comparable unit

Value

`f`	the density at specific x
`x`	the values along the x axis every 0.02 values, going from midpoint between minimum and 2nd smallest to the largest and 2nd largest values of x

...

Author(s)

Kevin Rader

References

B.W. Silverman (1981),Using Kernel Density Estimates to Investigate Multimodatlity. J.R. Statist. Soc. B,43,1,97-99.

See Also

`get.h`, `get.p`, `get.num.modes`

Examples

```
set.seed(12345)
x<-runif(50)
f<-emp.f(x,0.5)
```

`"equals-methods"` *Checks equality of two "FCS-class" objects*

Description

All the contents in the metadata and data portions of two input FCS objects are compared for equality. By default, the filename and objectname slots in the metadata are not compared. A boolean value is output specifying the status of the check on equality.

Methods

x = "FCS", y = "FCS" boolean value is output; if TRUE then the two FCS objects are the same, if FALSE then the two FCS objects are different.

Additional input signature options include:

check.filename boolean; if TRUE then the original filenames in the metadata are compared and checked; default is FALSE

check.objectname boolean; if TRUE, then the current object names in the metadata are compared and checked; default is FALSE

`extractGateHistory` *Extracting the gating information from the history*

Description

The history string corresponding to a specific gating Index specified by 'gateNum' is retrieved and output as a list of specific components.

Usage

```
extractGateHistory(x, gateNum)
```

Arguments

`x`	a "FCSgate" object created after using `createGate`
`gateNum`	the numeric column position of the gating index in the 'gate' matrix

Value

`gateNum`	the numeric column position of the gating index in the 'gate' matrix
`gateName`	character name of the gating index specified in the 'gate' matrix
`type`	type of gating (ie, "biscut", "uniscut", "bipcut", "bidcut")
`biscut.quadrant`	the quadrant specified (ie, ("+/+", "-/-", "+/-", "-/+"))
`data.colpos`	the gated parameter column positions in the 'data' matrix
`data.colnames`	the gated parameter column names in the 'data' matrix
`IndexValue.In`	the value of the index that specifies inclusion or selection
`gatingrange`	the vector of gating threshold(s)
`prev.gateNum`	the previous or most prior gating index column position in the 'gate' matrix
`prev.gateName`	the previous or most prior gating index column name in the 'gate' matrix
`comment`	character string of the user-defined comment

Author(s)

A.J. Rossini and J.Y. Wan

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics : New York, 2001. pp.279-283.

Jerome H. Friedman and Nicholas I. Fisher. Bump Hunting in High-Dimensional Data. Tech Report. October 28, 1998.

J. Paul Robinson, et al. Current Protocols in Cytometry. John Wiley & Sons, Inc : 2001.

Mario Roederer and Richard R. Hardy. Frequency Difference Gating: A Multivariate Method for Identifying Subsets that Differe between Samples. Cytometry, 45:56-64, 2001.

Mario Roederer and Adam Treister and Wayne Moore and Leonore A. Herzenberg. Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences. Cytometry, 45:37-46, 2001.

Keith A. Baggerly. Probability Binning and Testing Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared Test. Cytometry, 45:141-150, 2001.

See Also

`FCS-class`, `FCSgate-class`, `createGate`, `\code{createGate}`

Examples

```
 if (require(rfcdmin)) {
      data.there<-is.element("MC.053",objects())
      if ((sum(data.there) != length(data.there))) {
        ## obtaining the FCS objects from VRC data
        data(MC.053min)
      }

####  foo1 : Gating type: uniscut, univariate single cut
foo1 <- createGate(MC.053, varpos=4, gatingrange=256,
                    type="uniscut", MY.DEBUG=TRUE)

#### foo2.3 : Gating type : biscut -/-
foo2.3 <- createGate(foo1, varpos=c(1,2),
                      gatingrange=c(256, 300),
                      type="biscut",
                      biscut.quadrant="-/-",
                      prev.gateNum=NULL,
                      MY.DEBUG=TRUE)

## obtain gate information for first uniscut gate
gate.info1<-extractGateHistory(foo1, gateNum=1)

## obtain gate information for the second biscut gate
gate.info2<-extractGateHistory(foo2.3, gateNum=2)

### foo2.3.1 : extraction
foo2.3.1 <- extractGatedData(foo2.3, gateNum=2,
                              IndexValue.In=1,
                              MY.DEBUG=TRUE)
## obtain the second biscut gate information after
## subset/extraction of row observations
gate.info2.1<-extractGateHistory(foo2.3.1, gateNum=2)
}
```

`extractGatedData` *Extract the data of a FCS object using a specified Gating Index*

Description

This function will subset/reduce the rows of the data of an FCS object according to a column index of the "gate" matrix, which is created by using the function `createGate-methods`.

Usage

```
extractGatedData(x, gateNum = NULL, IndexValue.In = 1, MY.DEBUG = FALSE)
```

Arguments

`x`	an "FCSgate" object obtained from `createGate`
`gateNum`	the column position of the gating index that is specified in the "gate" matrix
`IndexValue.In`	either 0 or 1 depending on what value should be set for inclusion in the extraction. The default is the value 1.
`MY.DEBUG`	a boolean value that prints out debugging comments The default is FALSE and no debugging comments are printed.

Details

A "FCSgate" object with data having a reduced row length will be output along with an update to the following slots: "extractGatedData.msg" (The gateNum along with the inclusion value will be noted as a string), "current.data.obs" (the index of original data row positions that are currently in the data will be noted), and "metadata" (data dimension information will be updated along with the original status being changed to FALSE).

Value

A "FCSgate" S4 object is returned that extends the "FCS" object to contain additional slots:

`gate`	a matrix whose columns are the gating indices for the original data
`history`	vector which corresponds to each column gating index in "gate" and holds information about what variables and type of gate that was implemented and for what ranges of values
`extractGatedData.msg`	vector of strings to specify what if any extraction has been implemented using `extractGatedData`; "NONE" specifies no extraction has been implemented on the data for that particular corresponding gating index
`current.data.obs`	vector of the original data row positions that are currently still in the data matrix

Author(s)

A.J. Rossini and J.Y. Wan

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics : New York, 2001. pp.279-283.

Jerome H. Friedman and Nicholas I. Fisher. Bump Hunting in High-Dimensional Data. Tech Report. October 28, 1998.

J. Paul Robinson, et al. Current Protocols in Cytometry. John Wiley & Sons, Inc : 2001.

Mario Roederer and Richard R. Hardy. Frequency Difference Gating: A Multivariate Method for Identifying Subsets that Differe between Samples. Cytometry, 45:56-64, 2001.

Mario Roederer and Adam Treister and Wayne Moore and Leonore A. Herzenberg. Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences. Cytometry, 45:37-46, 2001.

Keith A. Baggerly. Probability Binning and Testing Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared Test. Cytometry, 45:141-150, 2001.

See Also

`FCS-class`, `FCSgate-class`, `createGate`

Examples

```
 if (require(rfcdmin)) {
      data.there<-is.element("MC.053",objects())
      if ((sum(data.there) != length(data.there))) {
        ## obtaining the FCS objects from VRC data
        data(MC.053min)
      }

####  test1 : Gating type: uniscut, univariate single cut
test1 <- createGate(MC.053, varpos=1, gatingrange=256,
                    type="uniscut", MY.DEBUG=TRUE)

#### test2.3 : Gating type : biscut -/-
test2.3 <- createGate(test1, varpos=c(1,2),
                      gatingrange=c(256, 300),
                      type="biscut",
                      biscut.quadrant="-/-",
                      prev.gateNum=NULL,
                      MY.DEBUG=TRUE)

### test 2.3.1 : extraction
test2.3.1 <- extractGatedData(test2.3, gateNum=2,
                              IndexValue.In=1,
                              MY.DEBUG=TRUE)
}
```

`fcs.type` *Objects providing parameters for the raw FCS file types*

Description

The `fcs.type` objects define the parameters needed for reading in certain raw FCS files into R via the use of `read.FCS`. Currently this is just a script file defining certain `fcs.type` objects, but ultimately this will be an environment. There are certain `read.FCS` parameters that are known to be compatible for certain types of cytometers. The `fcs.type` objects may be optionally used during the reading in of raw FCS files into R and result in FCS R-objects (FCS objects).

Usage

```
fcs.type.default
```

Arguments

No arguments.

Details

A fcs.type is a list of the following:

version raw FCS version number; value="1.0" or "2.0" or "3.0"

byte.size The byte size for the file (8 bits is one byte); value=1 or 2 or 4, etc.

signed boolean; If the data is signed; value=FALSE or TRUE

endian The endian of the file depending on the endian of the platform; Usually the value of endian is "big" (if both the file and platform endian are "big") or "little" (if both the platform and the file endian are "little") or "auto", then the read.FCS will automatically detect the endian compatibility with the platform system (See `readBin` for more details.)

The fcs.types are the following:

1. fcs.type.default a list of the following options and values:

version "2.0"

byte.size 2

signed TRUE

endian "auto"

2. fcs.type.cellquest.3.1.FACScan a list of the following options and values:

version "2.0"

byte.size 1

signed FALSE

endian "auto"

3. fcs.type.LSR256 a list of the following options and values:

version "2.0"

byte.size 1

signed FALSE

endian "auto"

4. fcs.type.FACStar256 a list of the following options and values:

version "2.0"

byte.size 1

signed FALSE

endian "auto"

5. fcs.type.facscan256 a list of the following options and values:

version "2.0"

byte.size 1

signed FALSE

endian "auto"

6. fcs.type.cellquest.3.1.FACS.Vantage a list of the following options and values:

version "2.0"

byte.size 2

signed TRUE

endian "auto"

7. fcs.type.cellquest.3.3 a list of the following options and values:

version "2.0"

byte.size 2

signed TRUE

endian "auto"

8. fcs.type.LYSYS a list of the following options and values:

version "2.0"

byte.size 2

signed TRUE

endian "auto"

9. fcs.type.DiVa1024 a list of the following options and values:

version "2.0"

byte.size 2

signed TRUE

endian "auto"

10. fcs.type.FACSCalibur1024 a list of the following options and values:

version "2.0"

byte.size 2

signed TRUE

endian "auto"

11. fcs.type.LSR1024 a list of the following options and values:

version "2.0"

byte.size 2

signed	TRUE
endian	"auto"

12. fcs.type.facscan1024 a list of the following options and values:

version	"2.0"
byte.size	2
signed	TRUE
endian	"auto"

Value

With the help of fcs.type, the raw FCS file will be read into a FCS R object that can be implemented for further analysis in R.

Author(s)

A.J. Rossini and J.Y. Wan

References

Peter Rabinovitch

See Also

`read.FCS`, `readBin`

Examples

```
if (require(rfcdmin)) {
  ## obtaining the location of the fcs files in the data
  get.path<-function(filename) {
    datadir<-system.file("fcs", package="rfcdmin")
    return(paste(datadir, filename, sep="/"))
  }


  FF256 <-read.FCS(get.path("facscan256.fcs"),
                             fcs.type=fcs.type.facscan256)
}
```

`"fixvars-methods"` *Checks and fixes the ranges, dimensions, and names of the metadata based on the current data of an FCS R-object.*

Description

Any discrepancy between the metadata and the data of the FCS object is considered as a failure to pass the check and will be updated with the descriptives from the data. The following is a description of the checks and fixes:

1. Dimension check and fix We always check the dimensions (ie, if the data dimensions match with size ($TOT) and nparam ($PAR) that are specified in the metadata. If they are not in check, then the metadata parameters are changed to reflect the values of the data dimensions.
2. Parameter Name check and fix We check the names of the metadata with the names of the data column parameters. Either only the longnames ($PnS) or the shortnames ($PnN) of the metadata are checked against the names of the data. Please take note that both ($PnS) and ($PnN) ARE NOT BOTH checked. Depending on the number of discrepancies (ie, the one with the least number of discrepanices; by default the longnames if there is a tie), either the longnames or the shortnames of the metadata are replaced with the column names of the data.
3. Column Variable Range Check We check the paramranges ($PnR) specified in the metadata with the column parameter ranges of the data; if there are any discrepancies, then the paramranges are replaced with the maximum values of the data columns.

   Please note that if the metadata@original is FALSE, then the metadata slotNames have a "RFACSadd> > " suffix and are located in metadata@fcsinfo in order to store the current data descriptives. The original data descriptives can be retrieved/checked when metadata@original is set to TRUE; otherwise the current metadata information about the data is retrieved/checked even when the "RFACSadd> > " suffix is not noted in the character index.

   (ie) If metadata@original is FALSE, then metadata[["size"]] will return metadata[["RFACSadd> > $TOT"]], the current row length of the data, while metadata@size will return the number of rows for the original data.

   Note that metadata@original is changed only when a parameter column is added to the data using `addParameter-methods`, when rows of the data are extracted using `extractGatedData` or if the user decides to change the value metadata@original. Using `"["-methods` and `"[<-"-methods` on a `FCS` object will not change the value of metadata@original.

Methods

x = "FCS" A `FCS`object will be returned with any fixes to the metadata.

x = "FCS", x.name="", MY.DEBUG=TRUE, range.max=NULL Other options in the signature include:

(1) x.name : character string of the true object name; default is "" (ie, the objectname in the metadata will be regarded as the true object name)

(2) MY.DEBUG : boolean value; if TRUE, then the output statements are printed, otherwise if FALSE, then the statements are surpressed; default is TRUE.

(3) range.max : numeric value describing the true maximum of the data that the checks on the ranges will be compared; default is NULL (ie, the maximum of each column variable in the data is the truth)

`"fluors-methods"` *Obtaining the Data of Fluorescence Measurements from a FCS object*

Description

This method is used to obtain the data matrix of the `FCS` object.

Methods

x = "FCS" The input `FCS` object has data and metadata constituents, and the output of the function will be the extraction of the data portion of the input object.

`gate.IPC` *Interactive gating of an Image Parallel Coordinates Plot*

Description

This function will plot an image parallel coordinates plot and allows to user to click on the plot to indicate the cutoff value of the variable that is to be gated. On this single variable, the plot will be divided and two subsequent subplots (ie, two image parallel coordinates plots) will be shown.

Usage

```
gate.IPC(myFCSobj, var.gate,
      var.pos=1:(dim(myFCSobj@data)[2]),
      num.bins=10,
      joint=FALSE,
      range.var=range(myFCSobj@data[,var.pos]),
      break10 =seq(range.var[1]-1, range.var[2],
                        by=range.var[2]/num.bins),
      title="",
      use.shortnames=FALSE,
      color.image=gray((25:5/25)[-c(1,2,3, 4, 5, 6)]),
      xwidth.scale=5,
      ntrans=1,
      hist.plotted=FALSE,
      image.plotted=TRUE,
      para.plotted=FALSE,
      lines.plotted=TRUE,
      legend.plotted=TRUE,
      lwd.vec=1:7,
      lty.vec=rep(1,7),
      col.vec=7:1,
      range.image=c(0, dim(myFCSobj@data)[1]),
      shrink.legend=TRUE,
      horizontal.legend = TRUE,
      offset.legend=0.03,
      nlevel.legend=length(color.image),
      xlab.image="",
      ylab.image="Bins",
      MY.DEBUG=FALSE,...)
```

Arguments

`myFCSobj`	FCS object to be gated/subsetted on an image parallel coordinates plot
`var.gate`	numerical column position of the variable to be gated in the data component of myFSobj
`var.pos`	a vector of the column positions of the variables of interest in the data of the FCS object to be shown in the image parallel coordinates plot;default is all the columns will be shown in the plots
`num.bins`	a vector consisting of the row positions of the cells to be analyze; default is 10
`joint`	Boolean; If TRUE, then the joint image parallel coordinate plots will be shown for the pre-gated and post-gated data; if FALSE, then the mariginal lines for the image parallel coordinate plots will be displayed; default is FALSE
`range.var`	a 2-dimensional vector denoting the minimum value and the maximum value of the variables to be plotted; default is c(0,1024), where 0 is the minimum value and 1024 is the max value
`break10`	vector denoting the breaks for the binning on the vertical axis; default is equal interval binning denoted by num.bins unless otherwise specified; the breaks must include the range of the variable; each bin is denoted by an open lower value and a closed upper value, ie, (a,b] where a and b are breakpoints and a<b.
`title`	character string denoting the title of the image plot; default value is an empty string
`use.shortnames`	Boolean; if TRUE, then the shortnames of the variables will be used in labeling in the plots; otherwise if FALSE, the longnames of the variables will be used; default is FALSE
`color.image`	the color scheme for the image plot; default is gray((25:5/25)[-c(1,2,3, 4, 5, 6)])
`xwidth.scale`	numeric value denoting the horizontal width of the variable and the transitions blocks; default value is 5 units of width
`ntrans`	numeric value denoting the number of transition columns between each pair of variables; default is 1 transition column between each pair of variables
`hist.plotted`	Boolean; if TRUE then the histogram plots of the variables and the transitions are made; otherwise if FALSE, there is no histogram plots; default value is FALSE
`image.plotted`	Boolean; if TRUE, then the image parallel coordinates plot is displayed; otherwise if FALSE, the plot is surpressed; default is TRUE
`para.plotted`	Boolean; if TRUE, then the parallel coordinates plot is displayed; otherwise if FALSE, the plot is surpressed; default is TRUE
`lines.plotted`	Boolean; if TRUE, then the image plot with the superimposed lines displayed; otherwise if FALSE, the plot is surpressed
`legend.plotted`	Boolean; if TRUE, then the legend for the superimposed lines denoting particular counts will be diplayed; otherwise if FALSE, the legend display is surpressed

`lwd.vec`	vector denoting the line width sizes to be used in the lines overlaying the image parallel coordinates plot; default value is an integer vector from 1 to 7
`lty.vec`	vector denoting the line type (solid or dotted, etc) for the corresponding line width in lwd.vec; the default is to have a solid line for each line width
`col.vec`	vector denoting the color for each line with the corresponding line width in lwd.vec and line type in lty.vec; the default is to have colors ranging from yellow to black (in that order).
`range.image`	2-dimensional numerical vector denoting the range of the number of counts in the image block to be plotted. The default value is to have a vector with a mininum value of zero and to have a maximum dependent on the number of cells/rows and bins
`shrink.legend`	boolean; if TRUE then the legend will be ; default value is TRUE
`horizontal.legend`	default value is TRUE
`offset.legend`	default value is 0.03
`nlevel.legend`	default value is the length of the color.image vector
`xlab.image`	a character string denoting the label of the horizontal x-axis on the image plot; default value is an empty string
`ylab.image`	a character string denoting the label of the vertical y-axis on the image plot; default value is "Bins"
`MY.DEBUG`	boolean value; if TRUE, debugging statements are printed, otherwise if FALSE, the statements are surpressed; default is FALSE
`...`	graphical parameters for `plot` may also be passed as arguments to this function

Details

The gating will be made on the image parallel coordinates plot without the lines drawn; this plot is the last plot to be displayed. The user should make a right click on the variable value displayed on the vertical axis. This variable value will denote the cutoff. The subsequent plots of the subsets will be made on the data such that the first subset will include row observations whose gated variable values are less than or equal to the cutoff of the gated variable across all other variables of interest and that the second subset/subplot will include row observations of whose gated variable's values are strictly greater than the cutoff.

Value

The first series of histograms, and parallel coordinates plots, and image parallel coordinates plots with superimposed lines and legends are displayed optionally by the user.

The second single image parallel coordinates plot is the one, in which the gating or threshold in which to subset is obtained by right clicking on the plot.

`info.total`

image.block	a matrix denoting the number of observations in each cell of the total image plot
line.info	total plot's list of matrices in which each matrix corresponds to the the line information between a pair of variables. Each matrix has three columns. The first two columns are the values of unique bin patterns between the pair of column variables, and the third column is the number of observations with that particular pattern.

breaks	total plot's vector of breaks for binning on the vertical axis for the values of the variablesDescription of 'comp1'
`info.sub1`	
image.block	a matrix denoting the number of observations in each cell of the first subsetted image plot
line.info	first subset's list of matrices in which each matrix corresponds to the the line information between a pair of variables. Each matrix has three columns. The first two columns are the values of unique bin patterns between the pair of column variables, and the third column is the number of observations with that particular pattern.
breaks	first subset's vector of breaks for binning on the vertical axis for the values of the variablesDescription of 'comp2'
`info.sub2`	
image.block	a matrix denoting the number of observations in each cell of the second subsetted image plot
line.info	second subset's list of matrices in which each matrix corresponds to the the line information between a pair of variables. Each matrix has three columns. The first two columns are the values of unique bin patterns between the pair of column variables, and the third column is the number of observations with that particular pattern.
breaks	second subset's vector of breaks for binning on the vertical axis for the values of the variablesDescription of 'comp1'
`obspos.sub1`	first subset's vector of numerical row observation positions of the data component of myFCSobj
`obspos.sub2`	second subset's vector of numerical row observation positions of the data component of myFCSobj
`FCSgateobj`	An FCS gate object that resulted from the gating

Author(s)

A.J. Rossini & J.Y. Wan

See Also

`ImageParCoord`, `JointImageParCoord`, `hist`, `plot`

Examples

```
if (require(rfcdmin)){
data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(VRCmin)
}

## make a smaller data for example
## first 1000 row observations
example.fcs<-unst.DRT[1:1000,]
if (!checkvars(example.fcs)){
example.fcs<-fixvars(example.fcs)
}

if (interactive()==TRUE){

## Joint parallel coordinates image
par(mfrow=c(4,3))
## gating the first column variable
## showing the image parallel coordinates
##    for column variables 1 through 5
gate.IPC(example.fcs, 1, var.pos=1:5, num.bins=10,joint=TRUE,
              title="Joint 10 bins 5 trans", ntrans=5)

## marginal parallel coordinate image
## gating the second column variable
par(mfrow=c(4,3))
gate.IPC(example.fcs, 2, var.pos=1:5, num.bins=10,joint=FALSE,
              title="Marginal 10 bins 5 trans", ntrans=5)
}
}
```

---

get.h *Estimate the critical bandwidth for specific number of modes*

Description

get.h finds the critical bandwidth for specific number of modes. That is, it finds the smallest bandwidth for which "m" modes are present for a kernel density estimator.

Usage

```
get.h(x, m = 1, prec = 0.001, hmin = 0, hmax = 1)
```

Arguments

`x`	the data vector in which to find the critical bandwidth
`m`	the number of modes for the critical bandwidth
`prec`	the precision for the resulting bandwidth
`hmin`	the minimum value to start searching for the critical bandwidth, h
`hmax`	the maximum value to start searching for the critical bandwidth, h

Details

get.h uses the Gaussian kernel to estimate the density of a data vector given by x. The bandwidth determines the spread of each data point. Thus a larger bandwidth leads to a smoother density estaimate. get.h finds the smallest bandwidth in which "m" modes are still present.

Value

`h`	the critical bandwidth, rescaled for the standardized x-values for direct comparison

Author(s)

Kevin Rader

References

B.W. Silverman (1981),Using Kernel Density Estimates to Investigate Multimodatlity. J.R. Statist. Soc. B,43,1,97-99.

See Also

`get.p`, `emp.f`, `get.num.modes`

Examples

```
set.seed(12345)
x <- c(rnorm(20,0),rnorm(20,3))
get.h(x)
```

`get.num.modes` *Number of modes of a gaussian kernel*

Description

get.num.modes returns the number of modes of the gaussian kernel estimate for a given data vector and bandwidth on the standardized scale

Usage

```
get.num.modes(x, h)
```

Arguments

`x`	the data vector
`h`	the bandwidth for the standardized data vector

Value

`x`	number of modes

Author(s)

Kevin Rader

References

B.W. Silverman (1981),Using Kernel Density Estimates to Investigate Multimodatlity. J.R. Statist. Soc. B,43,1,97-99.

See Also

`get.h`, `get.p`, `emp.f`

Examples

```
set.seed(12345)
x<-rnorm(50)
h<-get.h(x)
num<-c(get.num.modes(x,h),get.num.modes(x,h-0.005))
num
```

`get.p` *Test if the kernel density estimate given by x and h0 has at most m modes*

Description

This function returns the p-value of rejecting the null hypothesis that the kernel density estimate given by x and h0 has at most m modes.

Usage

```
get.p(x,h0,m=1,num.sim=200)
```

Arguments

`x`	the data vector
`h0`	the bandwidth for the gaussian kernel density estimate for the standardized data
`m`	the number iof modes we are trying to reject is the maximum
`num.sim`	the number of bootstrap simulations to determine this p-value

Value

returns the p-value of the test

Author(s)

Kevin Rader

References

B.W. Silverman (1981),Using Kernel Density Estimates to Investigate Multimodatlity. J.R. Statist. Soc. B,43,1,97-99.

See Also

`get.h`, `emp.f`, `get.num.modes`

Examples

```
set.seed(12345)
x1<-matrix(rnorm(50),ncol=1)
x2<-matrix(c(rnorm(25,mean=-2),rnorm(25,mean=2)),ncol=1)
h1<-get.h(x1,m=1,prec=0.001)
h2<-get.h(x2,m=1,prec=0.001)
p1<-get.p(x1,h1,1,100)
p2<-get.p(x2,h2,1,100)
c(p1,p2)
```

`"ggobi-methods"` *Dynamic Plotting and Viewing the "FCS" object data high-dimensionally*

Description

See 'ggobi' in 'library(ggobi)' for details.

Methods

fcsobject = "FCS" views the FCS object

`legend.CSP` *Makes a rough legend for the ContourScatterPlot*

Description

The color scheme used for the image plot within the ContourScatterPlot is scaled according the rough estimates of the breaks. Any white-colored cells in an image or ContourScatterPlot is considered to be NA.

Usage

```
legend.CSP(z, n,
   border = if (n < 32) "light gray" else NA,
   main = paste("color palettes;  n=", n),
   ch.col = c("rainbow(n, start=.7, end=.1)",
             "heat.colors(n)", "terrain.colors(n)",
             "topo.colors(n)", "cm.colors(n)"),
             breaks = seq(range(z, na.rm = TRUE)[1],
                   range(z, na.rm = TRUE)[2],
                   by = diff(range(z, na.rm = TRUE))/n))
```

Arguments

`z`	The matrix grid used for the image plot; this matrix is produced via make.grid or make.density
`n`	The number of color levels
`border`	The border of the legend plot
`main`	The main title of the legend plot
`ch.col`	the color palette used
`breaks`	the breaks used to scale the color scheme

Details

This legend is used as a rough approximation and is produced in a plot entirely separate.

Value

Plot of the color scheme scaled by ranges of the values of the grid cells in the image plot produced by 'z' input.

Author(s)

A.J. Rossini and J.Y. Wan

References

The code was obtained from the example of heat.colors

See Also

heat.colors, ContourScatterPlot, image

Examples

```
  if (require(rfcdmin)){
   data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
   if ( ( sum(data.there) != length(data.there) )){
      ## obtaining the FCS objects from VRC data
      data(VRCmin)
  }

var1<-st.DRT@data[,4]
var2<-st.DRT@data[,5]

col.nm<-colnames(st.DRT@data)

## matrix of counts
count.output1<-make.grid(var1, var2)
mat.counts1<-count.output1$z
if (interactive()){
par(mfrow=c(2,2))

image(mat.counts1,
  main="make.grid: Counts for stimulated",
   xlab=col.nm[4],yaxt="n", xaxt="n",
   ylab=col.nm[5], col=heat.colors(20))

## legend describes the counts in each cell
legend.CSP(mat.counts1, 20, ch.col="heat.colors(n)")

image(mat.counts1,yaxt="n", xaxt="n",
  main="make.grid: Counts for stimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], col=topo.colors(20))

legend.CSP(mat.counts1, 20, ch.col="topo.colors(n)")

}
}
```

`make.grid` *Make a matrix of values allocated in a two dimensional grid*

Description

A two-dimensional plot can be subdivided via grid marks and lines. Each component of the resulting grid is called a cell. The function `make.grid` determines a matrix of values corresponding to the number of observations that lie within each cell of the grid. The function `make.density` estimates the values allocated to each grid cell by a 'status' binary variable. The values are estimated to be either a difference in counts between the two status categories, a proportion, a normalized proportion, and a z statistic for each cell such that an image or ContourScatterPlot plot can be implemented.

Usage

```
make.grid(x, y, x.grid = seq(0, 1025, by = 25),
           y.grid = seq(0, 1025, by = 25))

make.density(x, y, status = NULL,
          x.grid = seq(0, 1025, by = 25),
          y.grid = seq(0, 1025, by = 25),
          type.CSP = c("count.diff", "p.hat", "p.hat.norm", "z.stat"))
```

Arguments

`x`	a vector of data values for the x-axis
`y`	a vector of data values for the y-axis
`status`	a vector of 0, 1 values denoting two categories
`x.grid`	a vector of grid marks to allocate x
`y.grid`	a vector of grid marks to allocate y
`type.CSP`	character string denoting the type.CSP of value to be estimated using the 'status' for each cell grid

Details

The following details the options for 'type.CSP':

"count.diff" The cell value is the count difference between the two 'status' categories

"p.hat" The grid cell value is the proportion of observations with 'status'==1 for that grid cell.

"p.hat.norm" The grid cell value is the following:

(ie, (p.hat - 0.05)/sqrt((0.05 * (1-0.05)) /n)

p.hat is the proportion in 'status'==1

where n is the number of cells in the grid with information. The default is to set the z statistic to zero for the cells with no information in either status. The value 0.5 is considered to be the case of no difference when the counts of both categories of 'status' are the same in the grid cell.

"z.stat" The cell value is a z statistic computed as the following:

(ie, (p.hat - p.bar)/se(p.bar))

p.hat is the proportion in 'status'==1

p.bar is the average of p.hat over the whole grid

se(p.bar)=sqrt((1-p.bar)(p.bar)/n), where n is the number of cells in the grid with information.

Value

`z`	matrix of values corresponding to the counts in an x-y grid
`n.cells`	(only output for 'make.grid'); number of total observations in z
`type.CSP`	(only output for 'make.density'); the type.CSP of value in each cell.

Note

In the base package, the function `image` could make a plot with the resulting matrix of values.

Author(s)

Zoe Moodie, A.J. Rossini, J.Y. Wan

See Also

`image`, `ContourScatterPlot`, `pairs.CSP`, `legend.CSP`, `heat.colors`

Examples

```
  if (require(rfcdmin)){
   data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
   if ( ( sum(data.there) != length(data.there) )){
      ## obtaining the FCS objects from VRC data
      data(VRCmin)
  }

var1<-st.DRT@data[,4]
var2<-st.DRT@data[,5]
var1.2<-unst.DRT@data[,4]
var2.2<-unst.DRT@data[,5]

col.nm<-colnames(st.DRT@data)

## The status where 1=stimulated
## 0 = unstimulated
status<-c(rep(1, dim(st.DRT@data)[1]), rep(0, dim(unst.DRT@data)[1]))
x <- c(var1, var1.2)
y <-c(var2, var2.2)

count.output1<-make.grid(var1, var2)
count.output0<-make.grid(var1.2, var2.2)

## matrix of counts
mat.counts1<-count.output1$z
mat.counts0<-count.output0$z
##total observations
total.stimulated<-count.output1$n.cells
total.unstimulated<-count.output0$n.cells

count.diff.output <-make.density(x, y, status=status, type.CSP="count.diff")
## matrix of cont differences between the status categories
mat.count.diff <-count.diff.output$z

p.hat.output <-make.density(x, y, status=status, type.CSP="p.hat")
## matrix of cont differences between the status categories
mat.p.hat <-p.hat.output$z

p.hat.norm.output <-make.density(x, y, status=status, type.CSP="p.hat.norm")
## matrix of cont differences between the status categories
mat.p.hat.norm <-p.hat.norm.output$z

z.stat.output <-make.density(x, y, status=status, type.CSP="z.stat")
## matrix of cont differences between the status categories
mat.z.stat <-z.stat.output$z

if (interactive()){
par(mfrow=c(3,2))

image(mat.counts1,yaxt="n", xaxt="n",
  main="make.grid: Counts for stimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], col=heat.colors(20))

image( mat.counts0,yaxt="n", xaxt="n",
  main="make.grid: Counts for unstimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], col=heat.colors(20))

image( mat.count.diff,yaxt="n", xaxt="n",
  main="make.density: Count Difference (Stimulated-Unstimulated)",
   xlab=col.nm[4],
   ylab=col.nm[5], col=heat.colors(20))

image( mat.p.hat,yaxt="n", xaxt="n",
  main="make.density: Proportion of Stimulated",
   xlab=col.nm[4],
   ylab=col.nm[5], col=heat.colors(20))

image( mat.p.hat.norm,main="make.density: Normalized proportion of Stimulated",
   xlab=col.nm[4],yaxt="n", xaxt="n",
   ylab=col.nm[5], col=heat.colors(20))

image( mat.z.stat, main="make.density: z statistic",
   xlab=col.nm[4],yaxt="n", xaxt="n",
   ylab=col.nm[5], col=heat.colors(20))

}
}
```

`"metaData-methods"` *Extraction of the FCSmetadata-class object from a FCS-class object*

Description

The metadata constituent is extracted from an FCS-class object.

Methods

x = "FCS" Extraction of a FCSmetadata-class object from a FCS-class object

---

`pairs.CSP` *Contour/Hexbin Scatterplot Matrices*

Description

A pairs plotting of histograms and rectangular-binned or hexagonal-binned image plots are produced using `hist` and `ContourScatterPlot`, respectively.

Usage

```
pairs.CSP(x,
          status=NULL,
          box.idx.list=NULL,
          type.CSP=c("count.diff",
                  "p.hat",
                  "p.hat.norm",
                  "z.stat"),
          alternate.hexbinplot=FALSE,
          n.hexbins=100,
          range.x=range(x),
          varlabpos=round(seq(range.x[1],
                ceiling(diff(range.x)/150)*150+range.x[1],
                by=150),0),
          cutoffs = seq(range.x[1],
                  ceiling(diff(range.x)/25)*25+range.x[1],
                  by=25),

           labels = colnames(x),

           panel = ContourScatterPlot,
           main="",

           image.col=heat.colors(10),
           numlev=5,...,
           lower.panel = legend.CSP,
           upper.panel = panel,
           overlay.panel=rect.box.idx,
           border.boxes=1:length(box.idx.list),
           lwd.boxes=rep(3,length(box.idx.list)),
           lty.boxes=rep(1,length(box.idx.list)),

           label.pos = 0.5,
           cex.labels = NULL,
           font.labels = 1,
           rowlattop = TRUE,
           gap=1,
           ch.col=c("heat.colors(n)",
                   "rainbow(n, start=.7, end=.1)",
                   "terrain.colors(n)",
                   "topo.colors(n)",
                   "cm.colors(n)"))
```

Arguments

`x`	matrix of data in which the columns are the variables and the rows are the individual observations
`status`	numerical binary 0, 1 vector denoting the status of the observations; default is NULL
`box.idx.list`	a list of vectors indicating the positions of 'x' which form a box to be overlayed on the binned plot in the upper and lower panels of the hexbin plot and the only the upper panel of the rectangular-binned plot by default
`type.CSP`	character string denoting the type of value to be estimated using the 'status' for each cell grid: the difference in counts ("count.diff"), the proportion ("p.hat"), the normalized proportion at 0.5 ("p.hat.norm"), the z.statistic ("z.stat"), see make.density for details.
`alternate.hexbinplot`	Boolean; if TRUE then alternate hexbin pairs plot is used; otherwise the ContourScatterPlot with rectangular bins is implemented
`n.hexbins`	number of bins for hexbin call; default is 100
`range.x`	vector denoting the min and the max of the observation values across all variable columns
`varlabpos`	vector of position of the variable values in which to label the x and y axes
`cutoffs`	the cutoffs for the x and y axes of the rectangular bins when alternate.hexbinplot is FALSE
`labels`	the labels for the diagonals when alternative.hexbinplot is TRUE
`panel`	default panel function; currently this is the contour scatter plot with rectangular bins; this option is ignored when 'alternate.hexbinplot' is TRUE
`main`	the main title for the rectanglar Contour scatter plot when alternative.hexbinplot is FALSE
`image.col`	image colors for the rectangular bins when alternative.hexbinplot is FALSE
`numlev`	number of levels for the contours for the rectangular bins when alternative.hexbinplot is FALSE
`...`	other options in `hexagons` or `ContourScatterPlot`
`lower.panel`	function for the lower panels of the pairs plot; currently this is fixed as a hexbin (when 'alternate.hexbinplot' is TRUE) or the legend.CSP (when 'alternative.hexbinplot' is FALSE)
`upper.panel`	function for the upper panels of the pairs plot; currently this is fixed as a hexbin or contour scatter plot
`overlay.panel`	Function which describes the overlay image on the panels; currently this option only works with the 'rect.box.idx' function and other functions that have the same signature

`border.boxes`	vector of corresponding border colors for each of the boxes in 'box.idx.list'
`lwd.boxes`	vector of corresponding widths for each of the outlined boxes in 'box.idx.list'; default is for all the boxes to have lwd = 3
`lty.boxes`	vector of corresponding line types for each of the outlined boxes in 'box.idx.list'; default is for all the boxes to have lty = 1
`label.pos`	position of the labels on the diagonal panels which are currently fixed as histograms; this option is not in use currently.
`cex.labels`	cex for the labels, used only when 'alternative.hexbinplot' is TRUE
`font.labels`	font for the labels, used only when 'alternative.hexbinplot' is TRUE
`row1attop`	boolean if row 1 is at the top, used only when 'alternative.hexbinplot' is TRUE
`gap`	used only when 'alternative.hexbinplot' is TRUE
`ch.col`	character string denoting the type of color palette used for the rectangular-binned image to be displayed in the legend when 'aternate.hexbinplot' is FALSE; default is "heat.colors(n)"

Details

There are no legends for the hexagonal (when 'alternate.hexbinplot' is TRUE) but there is a roughly estimate legend available for the rectangular binning (when 'alternate.hexbinplot' is FALSE) in the pairs plot.

Value

A pairs plot is displayed. NOTE: The histograms on the diagonals are of the whole dataset regardless of the value of the cells in each ContourScatterPlot.

Author(s)

J.Y. Wan and A.J. Rossini

References

Hexbin, other papers.

See Also

objects to See Also as 'hexbin' in the **hexbin** package

Examples

```
if (interactive()){
  if (require(rfcdmin)){
    data.there<-is.element(c("st.1829", "unst.1829", "st.DRT",
                  "unst.DRT"),objects())

    if ( ( sum(data.there) != length(data.there) )){
       ## obtaining the FCS objects from VRC data
      data(VRCmin)
    }

    ## subsetting the data for quicker plot display of less data
    data.mat1<-st.DRT@data[1:10000, 1:5]
```

```
      ## hexagonal binning

      pairs.CSP(data.mat1, alternate.hexbinplot=TRUE)

      ## rectangular binning with legends

      pairs.CSP(data.mat1, numlev=3,
         image.col=heat.colors(20))

      ## rectangular binning without legends

      pairs.CSP(data.mat1, numlev=3,
         image.col=heat.colors(20),
         lower.panel=ContourScatterPlot)

      ## putting a box around the observations
      ## greater than 500 for the second variable
      ##  less than 200 for the first variable
      idx1<-which(data.mat1[,2] > 500)  ## green box
      idx2<-which(data.mat1[,1] < 200)  ## blue box

      box.idx.list<-list(idx1, idx2)
      ## hexbin plots
      pairs.CSP(data.mat1, box.idx.list=box.idx.list,
            alternate.hexbinplot=TRUE, border.vec=c("green", "blue"))
      ## rectangular binned plots
      pairs.CSP(data.mat1, box.idx.list=box.idx.list,
            alternate.hexbinplot=FALSE,border.vec=c("green", "blue"),
            lower.panel=ContourScatterPlot)
    }
  }
```

parallelCoordinates

Parallel coordinates: Plotting each observation across all variables

Description

To view multi-dimensional data, a parallel coordinates plot is made such that each row is treated as an observation which is plotted across all column variables. The two dimensional plot which results has the column variables on the horizontal axis and the values of the column variables on the vertical axis. Care should be taken to note that each line drawn corresponds to a row observation. Also the units of measurements should be the same among all column variables. Note that the row observations can be grouped visually by specifying group line options such as line type, color, or width. The data can also be scaled to have a range of (0,1).

Usage

```
parallelCoordinates(x, varlabpos=1:dim(x)[2],
                        variable.names=colnames(x),my.ylab ="",
                        my.ylim =c(min(x),max(x)),
                        at.y=seq(min(x), max(x),
                             by=(max(x)-min(x))/20),
                        each.ylab=at.y, scaled = FALSE,
                        group = rep(1, dim(x)[1]),
                        group.lty=group, group.col=group,
                        group.lwd=group, superimpose=FALSE,...)
```

Arguments

`x`	matrix of the data (rows are the observations & columns are the variables)
`varlabpos`	numerical vector denoting the position of the variables/variable labels on the horizontal axis; default is a vector of 1 to the number of variables
`variable.names`	a vector of strings denoting the names of each variable; default value is the column names of the input matrix, x
`my.ylab`	character string denoting the name/label of the vertical y-axis; default value is "Values"
`my.ylim`	two-dimensional vector denoting the range of the vertical y-axis, ie, the range of the variables; default is the vector of the min and the max of the input matrix, x
`at.y`	vector of the vertical y-axis values at which labels will be shown on the plot; default is a vector of the minimum to the maximum by increments of one-twentieth of the difference between the mininum and the maximum
`each.ylab`	vector of the vertical y-axis labels; default is the numerical values of at.y
`scaled`	boolean; if TRUE, then the data is scaled to a range of [0,1]
`group`	a vector of indicating which group the row observations are in; default is all the row observations are in one group
`group.lty`	vector corresponding to each data row's line type corresponding to the group that each row observation is in; default is the vector value of group
`group.col`	vector corresponding to each data row's line type corresponding to the group that each row observation is in; default is the vector value of group
`group.lwd`	vector corresponding to each data row's line type corresponding to the group that each row observation is in; default is the vector value of group
`superimpose`	Boolean, if TRUE then parallel coordinate lines will be added to the existing plot; otherwise a new parallel coordinate plot will be made; default is FALSE
`...`	`plot` options

Value

A parallel coordinates plot in which row observations are plotted across all column variables in a plot with x-axis= names of the column variables and y-axis=values of the column variables.

WARNING

The dataset may have to be subsetted before implementing this function because the plot may take a long time to finish and may not be readable.

If the at.y option is not within the range of the column variables, then the range will be changed appropriately, but the interval or the difference between two elements of at.y will remain the same in order to keep the specified spacings of the y labels/tick marks.

If the each.ylab vector is different in length with the number of tick marks specified by at.y for the vertical axis, then by default the each.ylab will be the values of at.y. In other words, the labels will be the number values specified by at.y.

Author(s)

A.J. Rossini, J.Y. Wan

See Also

pairs, plot, ImageParCoord

Examples

```
if (require(rfcdmin)){
  data.there<-is.element("MC.053",objects())
    if ((sum(data.there) != length(data.there))) {
      ## obtaining the FCS objects from FHCRC data
      data(MC.053min)
    }

  dataMC<-MC.053@data


  if (interactive()) {
   par(mfrow=c(2,2))

    ### subset the data to the first 5 observations because it is too huge
    parallelCoordinates(dataMC[c(1:5),-6])
    ### the first 2 rows are a group and the last 3 rows are a different group
    parallelCoordinates(dataMC[c(1:5),-6], group=c(1,1,2,2,2))

    ### the same plot is scaled to 0,1 range
    parallelCoordinates(dataMC[c(1:5),-6], scaled=TRUE)
    parallelCoordinates(dataMC[c(1:5),-6], scaled=TRUE, group=c(1,1,2,2,2))

    parallelCoordinates(dataMC[c(1:5),1:4])
    ## changing the positions of the variables to the 1st, 5th, 8th, 16th
    ## positions on the horizontal x-axis
    parallelCoordinates(dataMC[c(1:5),1:4], varlabpos=c(1, 5, 8, 16))

    parallelCoordinates(dataMC[c(1:5),1:3])
    ## having the variable positions out of order of how they are plotted
    parallelCoordinates(dataMC[c(1:5),1:3], varlabpos=c(1, 15, 8))

    ## changing the labels of the vertical y-axis
    parallelCoordinates(dataMC[c(1:5),1:3], at.y=c(0, 500,
    1000),my.ylim=c(0, 1000),
    each.ylab=c("zero", "five hundred", "one thou"))
  }
}
```

`pkci2.flowcytest` *Testing the difference of upper-tail distributions of two samples*

Description

This function calculates a cut-off value designating the lower bound of the upper tail as k.hat.pkci2, the given percentile of the control sample, and a 95% confidence interval to test for a significant difference in proportion of stimulated cells and control cells above the threshold, k.hat.pkci2.

Usage

```
pkci2.flowcytest(controldata, stimuldata, crit = 0.999, alpha = 0.05)
```

Arguments

`controldata`	vector of data for control cells
`stimuldata`	vector of data for stimulated cells
`crit`	the percent of control sample below the threshold, k.hat.pkci2
`alpha`	The Type I error rate for construction of (1-alpha)% confidence interval

Details

Sometimes the difference in two sample distributions (control and stimulated) lies in the upper tail (usually at k.hat.pkci2 threshold which is the 99.9th percentile of the control sample). This function applies a standard normal test of the difference of two proportions (One proportion is obtained from the control sample, and one proportion is obtained from the stimulated sample. Both proportions are defined as the proportion of cells within that particular sample that are above the k.hat.pkci2 threhold value.) Please note that the standard normal approximation is used because it is assumed that the control and the stimulated samples are large in size (over 100 observations).

The null hypothesis of the test is that the proportion of the control sample above the k.hat.pkci2 threshold is the same as the proportion of the stimulated sample above the k.hat.pkci2 (ie, the distribution of cells in the tails of both the control and the stimulated samples are the same.)

Two alternative hypotheses are investigated. The one-sided alternative hypothesis states that the stimulated proportion is greater than the control proportion. The two-sided alternative hypothesis is that the stimulated proportion is not equal to the control proportion.

The respective p-values and a 95% confidence interval is obtained from the Z statistic (standard normal statistic).

Value

`k.hat.pkci2`	the threshold which is the 100*crit-th percentile of the control sample, where crit is the user input value
`pc.hat.pkci2`	the proportion of control cells/data above the k.hat.pkci2 threshold
`ps.hat.pkci2`	the proportion of stimulated cells/data above the k.hat.pkci2 threshold
`lb.pkci2`	The numeric lower bound of the 95% confidence interval from the Z statistic of the test
`up.pkci2`	The numeric upper bound of the 95% confidence interval from the Z statistic of the test
`test.1pkci2`	0,1 indicator for the one-sided test: 1= reject the null hypothesis, 0=cannot reject the null hypothesis
`pval1.pkci2`	p-value of the one-sided test; Pr(Z > z.statistic)
`test.2pkci2`	0,1 indicator for the two-sided test: 1= reject the null hypothesis, 0=cannot reject the null hypothesis
`pval2.pkci2`	p-value of the two-sided test; Pr(|Z| > z.statistic)= Pr(Z > z.statistic) + Pr(Z <-z.statistic)

WARNING

Usually the FCS object is gated and subset prior to this testing and analysis.

Note

Other flowcytests are available such as `WLR.flowcytest`, `ProbBin.flowcytest`, `KS.flowcytest`, which test the equivalence of two sample distributions. Generally, comparing the control and stimulated samples of the interferon gamma variable is of interest.

Author(s)

Zoe Moodie and A.J. Rossini and J.Y. Wan

References

Zoe Moodie, PhD Statistical Center for HIV/AIDS Research and Prevention (SCHARP) Fred Hutchison Cancer Research Center Seattle, WA 98109-1024

See Also

`WLR.flowcytest`, `ProbBin.flowcytest`, `KS.flowcytest`, `runflowcytests`, `qnorm`, `pnorm`

Examples

```
if (require(rfcdmin)){
data.there<-is.element(c("st.1829", "unst.1829", "st.DRT", "unst.DRT"),objects())
if ( ( sum(data.there) != length(data.there) )){
## obtaining the FCS objects from VRC data
data(VRCmin)
}


## This only serves as an example.  Usually the FCS object is
## gated and then subset

## HIV negative individual 1829
  IFN.control<-unst.1829@data[1:2000,4]
  IFN.stimul<-st.1829@data[1:2000,4]

  output1.pkci2<-pkci2.flowcytest(IFN.control, IFN.stimul, crit=.9999)

## HIV positive individual DRT
  IFN.control2<-unst.DRT@data[1:2000,4]
  IFN.stimul2<-st.DRT@data[1:2000,4]
  output2.pkci2<-pkci2.flowcytest(IFN.control2, IFN.stimul2, crit=.9999)

## This is an artifical example, but one would expect the
## distributions of the stimulated and control samples
## to be the same in the HIV negative individual 1829
## and to be different in the HIV positive individual DRT
## The test in this example is a bit contrived but
## the bigger picture is achieved.
}
```

`"plot-methods"` *Graphical representation of an object*

Description

The default action is a graphical plot of the object.

Methods

x = "ANY", y = "ANY" A scatterplot or other graphical representation is produced.

x = "FCS", y = "missing" The default action is contour-image pairs plotting for all the column variables.

x = "FCS", y = "missing", image.parallel.plot=FALSE, joint=TRUE, ... An optional image parallel coordinates plotting (either marginal or joint) for each row/cell across all column variables can also be displayed.

The optional signature details are listed below:

image.parallel.plot boolean; if true the image parallel coordinates plot will be implemented instead of default pairs plot; default value of FALSE

joint boolean; if image.parallel.plot is TRUE, then this boolean establishes if the image parallel coordinates plot is joint or not

... optional additional plot variables; See `ImageParCoord` or `pairs.CSP` for additional information on image parallel coordinates plotting and pairs contour-image plotting, respectively.

x="PRIM.step", y="missing" Trajectory plot using the 'trajectory.pl' function in the **rfcprim** pacakge is displayed for the step.

x="PRIM.step.set", y="missing" Trajectory plot using the 'trajectory.pl' function in the **rfcprim** is displayed for the peeling and the expansion steps.

x="PRIM.crossval.step", y="missing" Trajectory plot using the 'trajectory.pl' function in the **rfcprim** is displayed for the peeling and the expansion steps for each testdata set.

x="PRIM.rule", y="missing" Trajectory plots for all 3 steps is displayed.

`plot.ProbBin.FCS` *Plots a ProbBin.FCS object*

Description

A ProbBin.FCS object plot results in two histograms–one for the stimulated sample and one for the unstimulated sample.

Usage

```
plot.ProbBin.FCS(x, xlab=x$varname,
                 xlim=c(min(c(round(range(x$st.hist$breaks),1) + 1,
                    round(range(x$unst.hist$breaks),1) + 1)),
                    max(c(round(range(x$st.hist$breaks),1) + 1,
                    round(range(x$unst.hist$breaks),1) + 1))),

               main="",
               labels=FALSE,
               freq=FALSE, plots.made=c("both", "stimulated", "unstimulated"), ..
```

Arguments

`x`	ProbBin.FCS object
`xlab`	Character string of the x-axis; default is the variable name
`xlim`	vector of length 2 denoting the minimum and the maximum value of the break-point values, x-axis; default is the minimum and the maximum of the break-points for both stimulated and unstimulated samples
`main`	character string of the title of the file (ie, individual id number)
`labels`	Boolean; if TRUE, then the number/precentage in each bin is printed on the histogram, otherwise it is not; default is FALS
`freq`	Boolean; if TRUE, then the histogram is in terms of counts; if FALSE, then the histogram is in terms of relative frequencies/precentages; if TRUE and the areas in plot are wrong is output as a warning.
`plots.made`	character string denoting which histogram plot should be displayed; default is "both"
`...`	plotting options such as 'ylab' and 'ylim' to pass to hist

Value

Two histograms (one of the stimulated sample, and the other of the unstimulated sample) are displayed or only one histogram plot specified by the user will be displayed.

Author(s)

A.J. Rossini & J.Y. Wan

References

Mario Roederer, et al. "Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences" Cytometry 45:37-46 (2001).

See Also

`hist`, `ProbBin.FCS`

Examples

```
if (require(rfcdmin)){

if (!( is.element("st.1829", objects()) & is.element("unst.1829",
objects()) )){
## obtaining the FCS objects from VRC data
data(VRCmin)
}

## This only serves as an example.
## Gating/subsetting should precede this analysis
IFN.gamma.1<-unst.1829@data[1:2000,4]
IFN.gamma.2<-st.1829@data[1:2000,4]

#Probability binning using the control dataset to determine the breaks
PB1<-ProbBin.FCS(IFN.gamma.1, IFN.gamma.2, 200,
varname=colnames(unst.1829@data)[4], PBspec="by.control",MY.DEBUG=FALSE)

## Probability Binning using the combined dataset (control and stimulated)
## to determing the breaks
PB2<-ProbBin.FCS(IFN.gamma.1, IFN.gamma.2, 200,
varname=colnames(unst.1829@data)[4], PBspec="combined",MY.DEBUG=FALSE)

if (interactive()){
par(mfrow=c(2,2))
## plots both plots
plot(PB1, ylim=c(0,500),main="Prob Binning using the Control dataset")

## plots only the unstimulated
plot(PB2, main="Prob Binning using the Combined Dataset", plots.made="unstimulated")

## plots only the stimulated
plot(PB2, main="Prob Binning using the Combined Dataset", plots.made="stimulated")
}

}
```

`plot2sets.FCS` *Create a scatterplot to summaryze and compare two series of FCS objects*

Description

Create a scatterplot to summaryze and compare 1 paraneter from two series of FCS objects stored in 2 different plates. The points are colored according to their position in the plate (row or column number.)

Usage

```
plot2sets.FCS(data1,data2,varpos=c(1),FUN,nrow=8,ncol=12,ind=c(1:96),col="row",l
```

Arguments

`data1`	a list of fluorescent data from one (or more) FCS object(s) or a cytoset
`data2`	a list of fluorescent data from one (or more) FCS object(s) or a cytoset
`varpos`	the numerical column variable position of the FCS objects
`FUN`	function to summaryze the distribution of the data, e.g. mean, median, IQR, MODE
`col`	character vector either "row" or "col"
`nrow`	numeric, number of rows per plate
`ncol`	numeric, number of columns per plate
`ind`	numeric vector, index of the wells to be plotted
`labeling`	logical, draw plate position (default= TRUE)
`...`	any other arguments are passed to the `plot` function

Value

None.

Author(s)

Nolwenn Le Meur

See Also

`plot`

Examples

```
 ##Example I:
 ##data(flowcyt.data)

##Draw a scatterplot of the median values
##of the Foward scatter and the Side scatter parameters
##of each FCS file. The files correspond to samples store in a 96 well plate.
##plot2sets.FCS(flowcyt.data,varpos=c(1,2),FUN1=median,nrow=8,ncol=10,ind=c(1:80),col="r
```

`plotECDF.FCS` *Create a empirical cumulative distribution plot for one (or more) parameter(s) of one (or more) FCS object(s)*

Description

Create a empirical cumulative distribution plot for one parameter of one (or more) FCS object(s).

Usage

```
plotECDF.FCS(data, varpos, var.list, group.list, xlab,
ylab,alternating=TRUE, legend.title=NULL,...)
```

Arguments

`data`	a list of fluorescent data from one (or more) FCS object(s)
`varpos`	the numerical column variable position of the data of the FCS object
`var.list`	conditioning variables
`group.list`	a variable or expression to be evaluated in the data frame specified by 'data', expected to act as a grouping variable within each panel, typically used to distinguish different groups by varying graphical parameters like color and line type
`xlab`	a title for the x axis
`ylab`	a title for the y axis
`alternating`	logical specifying whether axis labels should alternate from one side of the group of panels to the other (for more details see `xyplot`)
`legend.title`	a title for the legend
`...`	any other arguments are passed to the `xyplot` function

Details

Other options from the functions xyplot from the lattice library.

Value

None.

Author(s)

N. Le Meur

See Also

ecdf,lattice, xyplot

Examples

```
require(rfcdmin)
require(lattice)

##Example I:
data(flowcyt.data)

##Draw an empirical cumulative density plot for the Foward scatter
##parameter of the different stains at a particular different time point
##(one panel per time point).
plotECDF.FCS(flowcyt.data,varpos=c(1),var.list=c(paste("time",1:12,sep="")),group.list=p

##Example II:
if (require(rfcdmin)) {
 ##Obtain the location of the fcs files
 pathFiles<-system.file("bccrc", package="rfcdmin")
 drugFiles<-dir(pathFiles)

 ##Read a serie of FCS files
 drugData<-read.series.FCS(drugFiles,path=pathFiles,MY.DEBUG=FALSE)
 }

 ##Draw a empirical cumulative density plot for the Foward scatter
##parameter for the differents aliquots (of the same cell line)
##treated with different compounds.
plotECDF.FCS(drugData,varpos=c(1),var.list=c("Serie"),group.list=paste("compound",c(1:8)
```

`plotQA.FCS` *Create a scatterplot summaryzing one (or two) parameter(s) for several FCS objects stored in a plate*

Description

Create a scatterplot summaryzing one (or two) parameter(s) for several FCS objects stored in a plate. The points are colored according to their position in the plate (row or column number.)

Usage

```
plotQA.FCS(data,varpos=c(1,2),FUN1=IQR,FUN2=NULL,col="row",nrow=8,ncol=12,ind=c(
```

Arguments

`data`	a list of fluorescent data from one (or more) FCS object(s) or a cytoset
`varpos`	the numerical column variable position of the data of the FCS object
`FUN1`	function to summaryze the distribution of the data, e.g. mean, median, IQR, MODE
`FUN2`	function to summaryze the distribution of the data e.g. mean, median, IQR, MODE
`col`	character vector either "row" or "col"
`nrow`	numeric, number of rows per plate
`ncol`	numeric, number of columns per plate
`ind`	numeric vector, index of the wells to be plotted
`labeling`	logical, draw plate position (default=TRUE)
`...`	any other arguments are passed to the `plot` function

Value

None.

Author(s)

Nolwenn Le Meur

See Also

`plot`

Examples

```
 ##Example I:
data(flowcyt.data)

##Draw a scatterplot of the median values
##of the Foward scatter and the Side scatter parameters
##of each FCS file. The files correspond to samples store in a 96 well plate.
plotQA.FCS(flowcyt.data,varpos=c(1,2),FUN1=median,nrow=8,ncol=10,ind=c(1:80),col="row",p

##Example II:
##Draw a a scatterplot of the mode and IQR values for the Foward scatter
##of each FCS file.
plotQA.FCS(flowcyt.data,varpos=c(1),FUN1=IQR,FUN2=MODE,nrow=8,ncol=10,ind=c(1:80),col="c
```

plotdensity.FCS *Create density plots one parameter of one (or more) FCS object(s)*

Description

Produce density plot(s) using the density.lf function of the locfit library. a single column variable specified from the data of one (or more) FCS object(s).

Usage

```
plotdensity.FCS(data,varpos, groups, xlab, ylab, col, xlim = NULL, ylim =
NULL, main=NULL,...)
```

Arguments

`data`	a list of one (or more) FCS object(s) or a cytoSet object
`varpos`	the numerical column variable position of the data of the FCS object
`groups`	a variable or expression to be evaluated in the data frame specified by 'data', expected to act as a grouping variable within each panel, typically used to distinguish different groups by varying graphical parameters like color and line type
`xlab`	a title for the x axis
`ylab`	a title for the y axis
`col`	The colors for lines and points. Multiple colors can be specified so that each point can be given its own color. If there are fewer colors than points they are recycled in the standard fashion. Lines will all be plotted in the first colour specified.
`xlim`	limits for the x axis
`ylim`	limits for the y axis
`main`	title of the plot
`...`	any other arguments are passed to the plot function

Details

Produce density plot(s) using the density.lf function of the locfit library. Other options from the functions plot.

Value

None.

Author(s)

N. Le Meur

See Also

density.lf

Examples

```
 if (require(rfcdmin)) {
  ##Obtain the location of the fcs files
  pathFiles<-system.file("bccrc", package="rfcdmin")
  drugFiles<-dir(pathFiles)

  ## Read a serie of FCS files
  drugData<-read.series.FCS(drugFiles,path=pathFiles,MY.DEBUG=FALSE)

  }

 ##Draw a density plot for the Foward SCatter parameter for the
 ##differents aliquots (of the same cell line) tested with different
 ##compounds.
 plotdensity.FCS(drugData,varpos=c(1),main="FSC for the aliquots treated
with different compounds", ylim=c(0,0.005), ylab="Density of cells")
```

`plotvar.FCS` *Making Univariate/Bivariate plots of the column variables of a FCS object*

Description

A univariate histogram or scatterplot will be made for a single column variable specified from the data of the FCS object, or a bivariate scatterplot or contour-image scatter plot will be shown for any two variables specified in the FCS object.

Usage

```
plotvar.FCS(x, varpos, type = c("uni", "bi"),
            plotType = c("hist", "ContourScatterPlot", "plot"),
            names.var = NULL, title.pl = "",
            xlimit = NULL, ylimit = NULL, plot.freq = TRUE,
            color.hist.plot = "white", CSPlot = TRUE,
            hexbin.CSPlot=TRUE,
            hexbin.style.CSPlot=c("colorscale", "lattice", "centroids",
                 "nested.lattice", "nested.centroids"),
            n.hexbins.CSPlot=100,
            x.grid.CSPlot = seq(0, 1025, by = 25),
            y.grid.CSPlot = seq(0, 1025, by = 25),
            image.col.CSPlot = heat.colors(2),
            numlev.CSPlot = 25,
            xaxt="s", yaxt="s",
            MY.DEBUG = FALSE,...)
```

Arguments

`x`	FCS object
`varpos`	the numerical column variable position of the data of the FCS object

`type`	character string specifying the type of plot; either "uni" for univariate or "bi" for bivariate; currently this option need not be specified because of automatic detection within the function
`plotType`	the type of plot to be used; either `plot`, `hist`, `ContourScatterPlot`; currently this option need not be specified because of automatic detection within the function; a univariate histogram plot is default when varpos is a single numeric value, and a default contour-image scatter plot with hexagonal binning or rectangular binning is displayed for a bivariate plot.
`names.var`	(optional) character string or vector of characte strings of the variable or variables to be plotted; default is NULL and will be changed to the names specified in the data of the FCS object
`title.pl`	character string of the plot title (main)
`xlimit`	numerical vector of the range of the x variable (horizontal axis)
`ylimit`	numerical vector of the range of the y variable (vertical axis)
`plot.freq`	boolean; if TRUE, then the frequencies instead of the relative frequencies are plotted (only if plotType=hist)
`color.hist.plot`	character string or numerical value indicating the color of the histogram plot
`CSPlot`	a boolean of whether or not this is a ContourScatterPlot; if FALSE then an ordinary scatterplot is produced
`hexbin.CSPlot`	boolean; if TRUE then the grid cells/compartments are hexagons; otherwise the grid cells are rectangular; default value is TRUE
`hexbin.style.CSPlot`	the style of hexbin plot; default is "colorscale" (for ContourScatterPlot hexagonal binning ONLY!)
`n.hexbins.CSPlot`	number of xbins for hexagon binning; default is 100 (for ContourScatterPlot hexagonal binning ONLY!)
`x.grid.CSPlot`	a numeical sequence denoting the grid marks for the x coordinate (for ContourScatterPlot rectanglar binning ONLY!)
`y.grid.CSPlot`	a numerical sequence denoting the grid marks for the y coordinate (for ContourScatterPlot rectanglar binning ONLY!)
`image.col.CSPlot`	a color map for the image (for ContourScatterPlot rectanglar binning ONLY!)
`numlev.CSPlot`	number of levels for the contours in a ContourScatterPlot (for ContourScatterPlot rectanglar binning ONLY!)
`xaxt`	if "s", then the x-axis is plotted, if "n" then there is no x-axis plotted (for ContourScatterPlot rectanglar binning ONLY!)
`yaxt`	if "s", then the y-axis is plotted, if "n" then there is no y-axis plotted (for ContourScatterPlot rectanglar binning ONLY!)
`MY.DEBUG`	boolean; if TRUE then the variable check statements are printed; default is FALSE
`...`	`plot` options (for histograms and ContourScatterPlot hexagonal binning) or `contour` options for ContourScatterPlot rectangular binning

Details

Other options from the functions `plot`, `hist`, `ContourScatterPlot` may be used in the signature of this function to define the plot further.

Value

Either a univariate or a bivariate plot of the specified variable(s) of the FCS object. A `hist` plot will output the breaks and bins of the histogram.

WARNING

Please read the warning for `ContourScatterPlot`.

Note

For a description of colors please look up `colors`, `palette`, and `heat.colors`

Author(s)

A.J. Rossini and J.Y. Wan

See Also

`ContourScatterPlot`, `plot`, `hist`

Examples

```
### to identify all the colors available on your system
colors()
if (interactive()) {
  if (require(rfcdmin)) {

    if (!is.element("unst.1829", objects())) {
      ## obtaining the FCS objects from VRC data
      data(VRCmin)
    }

    ## univariate plot
    plotvar.FCS(unst.1829, varpos=1)

    ## bivariate plot :hexagonal binning
    plotvar.FCS(unst.1829, varpos=c(1,2))

    ## bivariate plot :rectangonal binning
    plotvar.FCS(unst.1829, varpos=c(1,2), hexbin.CSPlot=FALSE)
  }
}
```

`"print-methods"` *Printing an object*

Description

An object is displayed in a concise manner.

Methods

x = "ANY" Displays all the contents of the object

x = "FCSmetadata" displays the original status, the objectname and the filename with the current size and nparam slot information; details can be viewed by 'x@slotName' where slotName is one of the following: "mode", "size", "nparam", "longnames", "shortnames", "paramranges", "filename", "objectname", "fcsinfo", "original"

x = "FCS" displays the original status, the objectname and the filename with the current size and nparam slot information; Note that the long and gory details can be viewed by 'x@data' or 'x@metadata'

x = "FCSsummary" Displays the statistics of the data and information about the metadata

x="PRIM.step" Displays the 'step.name', size of the starting data, the decision for the box, the percent change for each iteration, the number of iterations, and the chosen box's ranges within the data X.

x="PRIM.step.set", y="missing" Displays the "PRIM.step" information for the peeling and expansion steps.

x="PRIM.crossval.step", y="missing" Displays the "PRIM.step" information for the peeling and expansion steps for each testdata set.

x="PRIM.rule", y="missing" displays the "PRIM.step" information for all 3 steps is displayed.

`read.FCS` *Reading in a raw binary Flow Cytometry Standard (FCS) file*

Description

Reads in a Flow Cytometry Standard (FCS) file and outputs an "FCS" R object.

Usage

```
read.FCS(fileName, FCSobj.name="", fcs.type=NULL,
         fcs.byte.size =2, fcs.signed=TRUE,
         use.FCS.shortnames = FALSE, no.names = FALSE,
         UseS3 = FALSE,
         MY.DEBUG = TRUE)
```

Arguments

`fileName`	string of the FCS file location
`FCSobj.name`	character string of the FCS object name given; default is ""
`fcs.type`	a list of information (version, byte.size, signed, endian) about the FCS file; see `fcs.type`
`fcs.byte.size`	numeric indicating the fcs file byte size, default is 2
`fcs.signed`	TRUE if signed binary data, FALSE if unsigned
`use.FCS.shortnames`	boolean indicating whether or not to use the short or longnames for the dataframe in the FCS object output, default is TRUE/to use the short names
`no.names`	boolean indicating whether or not to use the names in the fcs file for the FCS object output, default is FALSE/to use the names in the FCS file
`UseS3`	If true, save in old S3 class structure, else save in new S4 class strucuture
`MY.DEBUG`	boolean indicating whether or not to print the debugging statements, default is TRUE/to print

Details

This function also checks if there are discrepancies between the data and the metadata in terms of range and size. If there is, then the data is re-read with different fcs.byte.size (1,2,4,8) and fcs.signed (TRUE, FALSE) combinations until there is no discrepancy between the data and the metadata. If there is still a discrepancy, then the routine is halted. Note: For FCS version 3.0 files, only the range of the data is checked against what is stated in the metadata because FCS version 3.0 files have extra elements that are read into the data.

Value

`a "FCS" object`	has the following slots:
`data`	a dataframe of the cells as rows and the variables for each cell as the columns
`metadata`	a list of the variable names and comments as in the FCS file which may include the following (for FCS file version 3.2.19):
$PAR	the number of columns/parameters
$TOT	the total number of cells/rows
$MODE	the mode of the FCS file
$BEGINANALYSIS	part of FCS file heading indicating the position of the beginning of the analysis portion
$BEGINDATA	part of FCS file heading indicating the beginning of the data portion
$BYTEORD	part of FCS file heading indicating byte order/endian
$BEGINSTEXT	part of FCS file heading indicating beginning of text
$DATATYPE	part of FCS file heading indicating the type of data
$ENDANALYSIS	part of FCS file heading indicating the end of the analysis portion
$ENDDATA	part of FCS file heading indicating the end of the data portion
$ENDSTEXT	part of FCS file heading indicating the end of the text portion

$NEXTDATA	part of FCS file heading indicating the next data
$PnB	Number of bits reserved for parameter number n
$PnE	Amplification type for parameter n
$PnR	Range for parameter number n
$ABRT	Events lost due to data acquisition electronic coincidence
$BTIM	Clock time at beginning of data acquisition
$CELLS	Description of objects measured.
$COM	Comment
$COMP	Fluorescence compensation matrix.
$CSMODE	Cell subset mode, number of subsets to which an object may belong
$CSVBITS	Number of bits used to encode a cell subset identifier
$CSVnFLAG	The bit set as a flag for subset n.
$CYT	Type of flow cytometer
$CYTSN	Flow cytometer serial number
$DATE	Date of data set acquisition
$ETIM	Clock time at end of data acquisition
$EXP	Name of investigator initiating the experiment
$FIL	Name of the data file containing the data set
$GATE	Number of gating parameters
$GATING	Specifies region combinations used for gating
$GnE	Amplification type for gating parameter number n
$GnF	Optical filter used for gating parameter number n
$GnN	Name of gating parameter number n
$GnP	Percent of emitted light collected by gating parameter n
$GnR	Range of gating parameter n
$GnS	Name used for gating parameter n
$GnT	Detector type for gating parameter n
$GnV	Detector voltage for gating parameter n
$INST	Institution at which data acquired
$LOST	Number of events lost due to computer busy
$OP	Name of flow cytometry operator
$Pkn	Peak channel number of univariate histogram for parameter n
$PKNn	Count in peak channel of univariate histogram for parameter n
$PnF	Name of optical filter for parameter n
$PnG	Amplifier gain used for acquisition of parameter n
$PnL	Excitation wavelength for parameter n
$PnN	Short name for parameter n
$PnO	Excitation power for parameter n
$PnP	Percent of emitted light collected by parameter n
$PnS	Long name/Name used for parameter n in the dataset

$PnT	Detector type for parameter n
$PnV	Detector voltage for parameter n
$PROJ	Name of the experiment project
$RnI	Gating region for parameter number n
$RnW	Window settings for gating region n
$SMNO	Specimen (tube or well) label
$SRC	Source of the specimen (patient name,cell types)
$SYS	Type of computer and its operating system
$TIMESTEP	Time step for time parameter
$TR	Trigger parameter and its threshold
$UNICODE	UNICODE code page for string type keyword values
RFACSadd> > ...	metadata information added using rflowcyt package via `addParameter`, `extractGatedData`

WARNING

The following scenerios may happen in which read.FCS has failed:

Problem 1 A number of names assigned to the columns of the data is different from the number of columns.

Possible Solution Use read.FCS again and choose a different fcs.byte.size value (such as 1, 2, 4, 8, 12, 16, etc.)

Problem 2 The file has been read properly by read.FCS, but the range of the resulting FCS R-object is wrong (ie, there are negative values when all values should be positive).

Possible Solutions Use read.FCS again, and choose a different fcs.signed value (either TRUE or FALSE).

Note

Thanks to Peter Rabinovitch for informaton and Julie McElrath lab for the example data.

Author(s)

A.J. Rossini, J.Y. Wan and N. Le Meur

See Also

`summary`, `print`, `extractGatedData`, `addParameter`, `"[-methods"`, `"[[-methods"`, `fcs.type`

Examples

```
if (require(rfcdmin)) {
  ## obtaining the location of the fcs files in the data

  FACSCAN256<- paste(system.file("fcs", package="rfcdmin"),
                          "facscan256.fcs",
                          sep="/")
```

```
    ## reading in the FCS files
    FCSobj1<-read.FCS(FACSCAN256)


  }
```

read.series.FCS *Reading a serie of raw binary Flow Cytometry Standard (FCS) files*

Description

Reads a serie of raw Flow Cytometry Standard (FCS) files and outputs several "FCS" R object.

Usage

```
read.series.FCS(fcsfiles,path=NULL,ext=NULL,...)
```

Arguments

`fcsfiles`	names of the FCS files without any extension
`path`	a character vector of full path names; the default corresponds to the working directory `getwd`
`ext`	character string giving optional extension to be added to each file name
`...`	any other arguments are passed to `read.FCS`

Details

This function read several FCS files by the means of the `read.FCS` function. Thus,this function can also checks if there are discrepancies between the data and the metadata in terms of range and size (MY.DEBUG=TRUE). If there is, then the data is re-read with different fcs.byte.size (1,2,4,8) and fcs.signed (TRUE, FALSE) combinations until there is no discrepancy between the data and the metadata. If there is still a discrepancy, then the routine is halted. Note: For FCS version 3.0 files, only the range of the data is checked against what is stated in the metadata because FCS version 3.0 files have extra elements that are read into the data.

Value

No value is returned. However a series of "FCS" object are created on the current environment with names of the form filename. The files names are given by the elements of slides. Each object is composed of the same data and metadata return by the `read.FCS` function.

Author(s)

N. Le Meur

See Also

`read.FCS`, `summary`, `print`, `extractGatedData`, `addParameter`, `"[-methods"`, `"[[-methods"`, `fcs.type` `readCytoSet`

Examples

```
if (require(rfcdmin)) {

##obtaining the location of the fcs files in the data
 pathFiles<-system.file("bccrc", package="rfcdmin")
 drugFiles<-dir(pathFiles)

## reading in the FCS files
 drugData<-read.series.FCS(drugFiles,path=pathFiles,MY.DEBUG=FALSE)
 }
```

`rect.box.idx` *Superimposes a rectangle on an existing plot given positional indicies*

Description

The boundaries of a rectangle are determined from a vector of positional indicies 'box.idx' and the given variables, 'x1' and 'x2'. This box is then displayed on the existing plot.

Usage

```
rect.box.idx(x1, x2, box.idx = NULL,
    original.data.idx = 1:length(x1),
    border = "black", lwd = 3, ...)
```

Arguments

`x1`	vector of values for variable 1
`x2`	vector of values for variable 2
`box.idx`	vector of positional indicies that indicate the box to be shown
`original.data.idx`	positional values of the current 'x1' and 'x2' observations
`border`	the color of the outline of the box or rectangle
`lwd`	the width of the lines of the box
`...`	other options in rect

Details

This function would be coupled with the use of ContourScatterPlot to show the boxes obtained by 'do.PRIM' (Patient Rule Induction Method) from the **rfcprim** package. PRIM is a semi-automated bump-hunting program.

Author(s)

A.J. Rossini and J.Y. Wan

References

See details in **rfcprim**

See Also

ContourScatterPlot, **rfcprim** library

Examples

```
if (require(rfcdmin)){

data(PRIM.example.data)


if (require(rfcprim)){

## only the peeling step is implemented
out.peel <- peel.step(X.PRIM, Y.PRIM)

if (interactive()){
ContourScatterPlot(X.PRIM[,1], X.PRIM[,2],  status=Y.PRIM,
   main="z statistic",
   xlab=col.nm[4],
   ylab=col.nm[5], image.col=heat.colors(20),plot.legend.CSP=TRUE)
## the Green box is the initial estimate of the first rule
## after the peeling step
rect.box.idx(out.peel@best.box.idx, X.PRIM[,1], X.PRIM[,2], border="green")
}
}
}
```

`rflowcyt-defunct` *Defunct Functions in rflowcyt package*

Description

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

Usage

```
parallel.coordinates()
add.parallel.coordinates()
```

Details

'parallel.coordinates' and 'add.parallel.coordinates' have been replaced by 'parallelCoordinates'and 'add.parallelCoordinates' respectively because a conflict with S3 method names.

See Also

.Defunct

runflowcytests — *Tests the equivalence of two univariate sample distributions by using four different methods*

Description

Runs the following flowcytests:

1. `WLR.flowcytest` weighted log rank test (by default when rho=0) and a the plot of survival curves for both samples is also output
2. `KS.flowcytest` Kolmogorov-Smirnoff test for the difference in distributions for the control and the stimulated
3. `ProbBin.flowcytest` Statistics proposed by Keith A. Baggerly and Mario Roederer which include Chi-squared and Normal tests for the PB metric via probability binning (both based on the control data only ("by.control") and based on the combined dataset of both the stimulated and the control samples ("combined")
4. `pkci2.flowcytest` Tests the difference of the upper tails of the two distributions

Usage

```
runflowcytests(controldata, stimuldata, flowcytests = c("WLR", "KS",
                  "ProbBin.by.control", "ProbBin.combined", "pkci2"),
                  N.in.bin = 100, varname = "", title = " ", output.all
                  = FALSE, graph.outlay = c(3, 2), crit.pkci2 = 0.999,
                  alpha.pkci2 = 0.05, na.action.WLR =
                  options()$na.action, rho.WLR = 0, WLR.plotted=TRUE, alternative
                  "two.sided", ..., KS.plotted=TRUE,
                             PBobj.plotted=TRUE,
                 PBobj.plots.made=c("both", "stimulated", "unstimulated"))
```

Arguments

`controldata`	a vector of values/fluoroescent measurements; a univariate control sample
`stimuldata`	a vector of values/fluoroescent measurements; a univariate stimulated sample
`flowcytests`	vector denoting the names of the tests that are implemented; default is a vector of all the test names
`N.in.bin`	a number which denotes the number per bin in used in probability binning
`varname`	character strong of the name of the variable under investigation (this is usually the gamma interferon variable)
`title`	character string of the title of the plots
`output.all`	boolean; if TRUE then all the statistics and p-values obtained are output in list form by test; if FALSE then only the names of the statistics, the statistics, the names of the p-values and the p-values are output in a data.frame; default is FALSE.
`graph.outlay`	a vector of length 2, describing the number of graphs on each row and the number of graphs on each column, respectively

`crit.pkci2`	the percent of control sample to above the meaningful percentile (usually 99.9th percentile) (for pkci2.flowcytest)
`alpha.pkci2`	Type I error rate for construction of the (1-alpha)% Confidence Interval (for pkci2.flowcytest)
`na.action.WLR`	a missing-data filter function. This is applied to the `model.frame` after any subset argument has been used. Default is `options()$na.action` (as quoted from the `survdiff` documentation)
`rho.WLR`	the exponent in $S(t)\hat{}\rho$, where S is the Kaplan-Meier estimate of survival; A value of 0 specifies using the weighted log-rank test, and a value of 1 specifies using the Peto and Peto modification of the Gehan-Wilcoxon test.
`WLR.plotted`	boolean; if TRUE, then plot is made; otherwise if FALSE, plotting is surpressed; default=TRUE
`alternative.KS`	character string of the alternative hypothesis:
"two-sided"	Two sided alternative hypoothesis
"less"	One-sided alternative hypothesis: controldata distribution is less than the stimuldata distribution
"greater"	One-sided alternative hypothesis: controldata distribution is greater than the stimuldata distribution
`...`	other options in `KS.flowcytest`
`KS.plotted`	boolean to display the corresponding plot; default is TRUE and the plot will be displayed
`PBobj.plotted`	boolean; if TRUE then histograms of the ProbBin.FCS object will be plotted; if FALSE, then these plots are surpressed; default is TRUE
`PBobj.plots.made`	character string denoting which histogram plot should be displayed; default is "both"

Value

A dataframe consisting of 4 columns and 20 rows. The labels on the columns are "statistics.names", "statistics", "pvalues.names", and "pvalues" or if 'output.all' is TRUE, a list of statistics and tesing output by test name will be produced. Also 6 to 0 plots are produced.

WARNING

Usually the FCS object is gated and subset prior to this testing and analysis. Also this function requires the library `survival`.

Note

For more information about the output, please see the other flowcytests in the "See Also" Section.

Author(s)

Zoe Moodie, A.J. Rossini, J.Y. Wan

References

Keith A. Baggerly "Probability Binning and Test Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared test" Cytometry 45: 141:150 (2001).

Harrington, D. P. and Fleming, T. R. (1982). "A class of rank test procedures for censored survival data". Biometrika 69, 553-566.

Zoe Moodie, PhD Statistical Center for HIV/AIDS Research and Prevention (SCHARP) Fred Hutchison Cancer Research Center Seattle, WA 98109-1024

Mario Roederer, et al. "Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences" Cytometry 45:37-46 (2001).

See Also

`pkci2.flowcytest`, `ProbBin.flowcytest`, `KS.flowcytest`, `WLR.flowcytest`

Examples

```
if (require(rfcdmin)){
## obtaining the FCS objects from VRC data
if ( !(is.element("unst.1829", objects()) & is.element("st.1829",
objects()) & is.element("unst.DRT", objects()) & is.element("st.DRT",
objects())) ){
data(VRCmin)
}

## This only serves as an example.  Usually the FCS object is
## gated and then subset

## HIV negative individual 1829
  IFN.control<-unst.1829@data[1:2000,4]
  IFN.stimul<-st.1829@data[1:2000,4]

if (interactive()){

## running all the tests
output1.runall<-runflowcytests(IFN.control, IFN.stimul,
varname="Interferon Gamma",
title="HIV negative individual 1829", crit.pkci2=0.9999)
}

## HIV positive individual DRT
  IFN.control2<-unst.DRT@data[1:2000,4]
  IFN.stimul2<-st.DRT@data[1:2000,4]

if (interactive()){
## running only WLR.flowcytest and pkci2.flowcytest
output2.runall<-runflowcytests(IFN.control2, IFN.stimul2,
flowcytests=c("WLR","pkci2"), varname="Interferon Gamma",
title="HIV negative individual 1829", crit.pkci2=0.9999)
}
## This is an artifical example, but one would expect the
## distributions of the stimulated and control samples
## to be the same in the HIV negative individual 1829
## and to be different in the HIV positive individual DRT
## The test in this example is a bit contrived but
## the bigger picture is achieved.
}
```

"show-methods" *Showing an object*

Description

An object is displayed in a concise manner.

Methods

object = "ANY" Displays all the contents of the object

object = "traceable" Displays the contents of the object

object = "ObjectsWithPackage" Displays the contents of the object

object = "MethodDefinition" Displays the contents of the object

object = "MethodWithNext" Displays the contents of the object

object = "genericFunction" Displays the contents of the object

object = "classRepresentation" Displays the contents of the object

object = "FCSmetadata" displays the original status, the objectname and the filename with the current size and nparam slot information; details can be viewed by 'x@slotName' where slotName is one of the following: "mode", "size", "nparam", "longnames", "shortnames"

object = "FCS" displays the original status, the objectname and the filename with the current size and nparam slot information; Note that the long and gory details can be viewed by 'x@data' or 'x@metadata'

object = "FCSsummary" Displays the statistics of the data and information about the metadata

x="PRIM.step" Displays the 'step.name', size of the starting data, the decision for the box, the percent change for each iteration, the number of iterations, and the chosen box's ranges within the data X.

x="PRIM.step.set", y="missing" Displays the "PRIM.step" information for the peeling and expansion steps.

x="PRIM.crossval.step", y="missing" Displays the "PRIM.step" information for the peeling and expansion steps for each testdata set.

x="PRIM.rule", y="missing" displays the "PRIM.step" information for all 3 steps is displayed.

`showgate.FCS` *Showing the gate and the datapoints within the gate on a prevous plot*

Description

On an exisiting plot, the gate specified will be plotted and the datapoints lying within the gating range will be colored (default is the color purple).

Usage

```
showgate.FCS(data.mat, gatingrange, Index,
        type = c("uniscut", "biscut", "bidcut", "bipcut"),
        IndexValue.In = 1,
        coltype = 12, pchtype = 8,
        biscut.quadrant = c("+/-","-/-", "+/+", "-/+"))
```

Arguments

`data.mat`	the data to be gated:
univariate case	single column of values: a (m X 1) data vector where m is the number of cells/rows
bivariate case	matrix of two column variables: a (m X 2) data matrix where m is the number or cells/rows
`gatingrange`	gating threshold range in one of the following formats for each type of gating:
"uniscut"	univariate single cut; gatingrange = x1 will select/include all points $>$ = x1 , x1 is numeric value
"bidcut"	bivariate double cut: gatingrange = c(x1,x2, y1,y2), a numeric vector of lowerbound, upperbound cutoffs for x and y variables
"biscut"	bivariate single cut:gatingrange=c(x1,y1), a numeric vector of the cutoffs for x and y variables
"bipcut"	bivariate polygonal cut: polygonal thresholds for an n-sided polygon with gatingrange = cbind(c(x1, x2, ...,xn, x1), c(y1, y2, ...,yn, y1)), a vector of vectors which denote the outer points of the polygonal vertices)
`Index`	a vector of 0's and 1's denoting the selection of row observations of 'data.mat'
`type`	character string of the type of cut/gating:
"uniscut"	univariate single cut: selects datapoints that are greater than or equal to the cutoff value denoted in gatingrange
"bidcut"	bivariate double cut: selects datapoints in the central rectangle formed by two vertical lines (x variable cutoffs) and two horizontal lines (y variable cutoffs)
"biscut"	bivariate single cut: cuts graph into quadrants (selects datapoints in the quadrant denoted by biscut.quadrant)
"bipcut"	bivariate polygonal cut: selects the datapoints in a polygon
`IndexValue.In`	The value of 'Index' to be selected; default is 1
`coltype`	a character string or a numerical value describing the option for the color of the data point inside the gating range

`pchtype`	a character string or a numerical value describing the option for the point size and type of data point inside the gating range
`biscut.quadrant`	character string value denoting the (x,y) quadrant that is to be selected; Values are one of the following:
"+/+"	selects the upper right quadrant, where x is positive and y is positive
"−/+"	selects the upper left quadrant, where x is negative and y is positive
"+/−"	selects the lower right quadrant, where x is positive and y is negative
"−/−"	selects the lower left quadrant, where x is negative and y is negative

Value

The gating range or gate will be displayed and the data points within the gating range will be colored.

Note

The coloring in of data points may take a while to process. The gate selection can only be shown using rectangular binning of the image plots using `ContourScatterPlot`. The `showgate.FCS` does not work with hexagonal binning.

Author(s)

A.J. Rossini and J.Y. Wan

See Also

`FHCRC.HVTNFCS`,`VRC.HVTNFCS`, `plotvar.FCS`,`createGate`, `icreateGate`

Examples

```
if (interactive()){
if (require(rfcdmin)){
  ## obtaining the FCS objects from VRC data
  if ( !(is.element("unst.1829", objects())
                & is.element("st.1829", objects())) ){
    data(VRCmin)
  }

  ## univariate plot
  plotvar.FCS(unst.1829, type="uni", varpos=1, plotType=hist)
  ## show cut off at 350
  showgate.FCS(unst.1829@data[,1], type="uniscut", gatingrange=350)
  ## show different cutoff at 500
  showgate.FCS(unst.1829@data[,1], type="uniscut", gatingrange=500,
               coltype="green")

  ## bivariate plot : rectanglar bins in which the gate can be shown
  plotvar.FCS(unst.1829, type="bi", varpos=c(1,2), hexbin.CSPlot=FALSE)
  ## show cutoff at 275 to 600 for both variables
  ## may take a while
  ## create the gate index as the first column entry of the "gate" matrix
  unst.1829.gt<-createGate(unst.1829, varpos=1:2, type="bidcut",
        gatingrange=c(275, 600, 275, 600))

  ## show the gate
  showgate.FCS(unst.1829.gt@data[,c(1,2)], unst.1829.gt@gate[,1],
     type="bidcut", gatingrange=c(275, 600, 275, 600))

  }
}
```

`standard` *Estimate the critical bandwidth for specific number of modes*

Description

Standardize a numeric vector by its median and median absolute deviation (MAD).

Usage

```
standard(x)
```

Arguments

`x`	the data vector to be standardized

Value

returns the standardized version of x

Author(s)

Kevin Rader

References

Silverman, B.W. (1981). Using Kernel Density Estimates to Investigate Multimodality. J. Royal Statistical Society B, 43, 97-99.

See Also

`get.h`, `get.p`, `emp.f`, `get.num.modes`

Examples

```
set.seed(12345)
x<-rnorm(50,2,3)
x1<-standard(x)
c(median(x1),mad(x1))
```

`"[-methods"` *Extraction of slot information using "["*

Description

Specifically this method is able to extract components or slots.

ANY.object[1] retrieves the first element or slot

FCSmetadata.object["fcsinfo"] obtains the "fcsinfo" slot which is a list

FCSmetadata.object["$P1R"] obtains the first parameter range/max

FCSmetadata.object[1:10] obtains first 10 elements of the "fcsinfo" slot of the metadata

FCS.object[1,2:3] extracts/reduces the data of the "FCS-class" object

Methods

x = "ANY" extracts elements

x = "FCSmetadata" Extracts slot information.

If using a single character string index such as the slotNames ("mode" or "$MODE"; "size" or "$TOT"; "nparam" or "$PAR"; "longnames" or "$PnS" or "$P1S" or "$P2S" etc...; "shortnames"or "$PnN" or "$P1N" or "$P2N" etc...; "paramranges" or "$PnR" or "$P1R" or "$P2R" etc...;"fcsinfo";"objectname", "original", "filename") as well as the "fcsinfo" slotNames can be retrieved.

If using a numeric single-valued or numeric vector index, only the "fcsinfo" slots are numerically indexed and can be retreived.

x = "FCS" extracts or reduces the data portion of the object and returns a "FCS-class" object

x="PRIM.step" extracts the object via a character slot name and/or a numeric iteration ID

x="PRIM.step.set", y="missing" extracts the object via a character slot name for the step (ie, "peel.step" or "expand.step") and with an optional slot name for the "PRIM.step" object.

x="PRIM.crossval.step", y="missing" extracts the object via a character slot name and/or a numeric testdata ID

`"[[-methods"` *Extraction of slot information using "[["*

Description

Specifically this method is able to extract components or slots.

ANY.object[1] retrieves the first element or slot

FCSmetadata.object["fcsinfo"] obtains the "fcsinfo" slot which is a list

FCSmetadata.object["$P1R"] obtains the first parameter range/max

FCSmetadata.object[1:10] obtains first 10 elements of the "fcsinfo" slot of the metadata

FCS.object[1,2:3] extracts/reduces the data of the "FCS-class" object

Methods

x = "ANY" extracts elements

x = "FCSmetadata" Extracts slot information.

If using a single character string index such as the slotNames ("mode" or "$MODE"; "size" or "$TOT"; "nparam" or "$PAR"; "longnames" or "$PnS" or "$P1S" or "$P2S" etc...; "shortnames"or "$PnN" or "$P1N" or "$P2N" etc...; "paramranges" or "$PnR" or "$P1R" or "$P2R" etc...;"fcsinfo";"objectname", "original", "filename") as well as the "fcsinfo" slotNames can be retrieved.

If using a numeric single-valued or numeric vector index, only the "fcsinfo" slots are numerically indexed and can be retreived.

x = "FCS" extracts the slot information from the metadata portion of the object; see x="FCSmetadata" description (above) for specific indexing using "[["

x="PRIM.step" extracts the object via a character slot name and/or a numeric iteration ID

x="PRIM.step.set", y="missing" extracts the object via a character slot name for the step (ie, "peel.step" or "expand.step") and with an optional slot name for the "PRIM.step" object.

x="PRIM.crossval.step", y="missing" extracts the object via a character slot name and/or a numeric testdata ID

`"[[<-methods"` *Replacement and/or Addition of new slot or indexed elements using "[[<-"*

Description

This method replaces the slot with a value that is assigned. In circumstances mentioned below, a new slot can also be added.

Methods

x = "ANY" Replaces a slot with the assigned value.

x = "FCSmetadata" Replaces the slot with the assigned value.

If using a single character string index such as the slotNames ("mode" or "$MODE"; "size" or "$TOT"; "nparam" or "$PAR"; "longnames" or "$PnS" or "$P1S" or "$P2S" etc...; "shortnames"or "$PnN" or "$P1N" or "$P2N" etc...; "paramranges" or "$PnR" or "$P1R" or "$P2R" etc...;"fcsinfo";"objectname", "original", "filename") as well as the "fcsinfo" slotNames can be assigned a value. If no slot is found by the character index referring to the slotName, then a new slot will be made in the "fcsinfo" list with the particular character index as the slotName will be added along with the value that is assigned.

If using a numeric single-valued or numeric vector index, only the "fcsinfo" slots are numerically indexed and assigned a new value.

x = "FCS" Replaces the indexed slots of the metadata portion of the object; See x="FCSmetadata" (above) for details.

x="PRIM.step" replaces the object via a character slot name and/or a numeric iteration ID

x="PRIM.step.set", y="missing" replaces the object via a character slot name for the step (ie, "peel.step" or "expand.step") and with an optional slot name for the "PRIM.step" object.

x="PRIM.crossval.step", y="missing" replaces the object via a character slot name and/or a numeric testdata ID

`"[<-methods"` *Replacement and/or Addition of new slot or indexed elements using "[<-"*

Description

This method replaces the slot with a value that is assigned. In circumstances mentioned below, a new slot can also be added.

Methods

x = "ANY" Replaces a slot with the assigned value.

x = "FCSmetadata" Replaces the slot with the assigned value. If using a single character string index such as the slotNames ("mode" or "$MODE"; "size" or "$TOT"; "nparam" or "$PAR"; "longnames" or "$PnS" or "$P1S" or "$P2S" etc...; "shortnames"or "$PnN" or "$P1N" or "$P2N" etc...; "paramranges" or "$PnR" or "$P1R" or "$P2R" etc...;"fcsinfo";"objectname", "original", "filename") as well as the "fcsinfo" slotNames can be assigned a value. If no slot is found by the character index referring to the slotName, then a new slot will be made in the "fcsinfo" list with the particular character index as the slotName will be added along with the value that is assigned.

If using a numeric single-valued or numeric vector index, only the "fcsinfo" slots are numerically indexed and assigned a new value.

x = "FCS" Replaces the indexed data portion of the object

x="PRIM.step" replaces the object via a character slot name and/or a numeric iteration ID

x="PRIM.step.set", y="missing" replaces the object via a character slot name for the step (ie, "peel.step" or "expand.step") and with an optional slot name for the "PRIM.step" object.

x="PRIM.crossval.step", y="missing" replaces the object via a character slot name and/or a numeric testdata ID

`"summary-methods"` *Summary of object*

Description

A summary such as statistics or the names of the list items will be output depending on the class of object.

Methods

object = "ANY" usually a print-out of statistics and names

object = "FCSmetadata" Displays the structure of this object

object = "FCS" A "FCSsummary" object is returned; Displays five-number summary using Tukey's method and the standard deviation for each column variable in the data of the FCS object and a print-out of information about the metadata, showing the description of the slots, the column parameter descriptives, and the slotNames in metdata@fcsinfo.

object = "PRIM.step" A matrix summarizing the iterations for the step is output

object = "PRIM.step.set" A list of matrices summarizing the iterations for each step is output ; the names of the list components is 'peel.step' and 'expand.step'

object = "PRIM.crossval.step" A list of 'PRIM.step.set' summary outputs is output; the list is indexed by testdata set "TD*" where "*" is the numeric ID

`summary.ProbBin.FCS` *Chi-Squared/Standard Normal Approximation Summary Statistics for a ProbBin.FCS object*

Description

This function provides summary statistics for the test of distribution difference of two samples that have been probability-binned or in histogram form.

Given two probability-binned samples, of which one will be called the stimulated sample and the other the unstimulated/control sample, the null hypothesis is that both the unstimulated/Control Data Histogram/Bins are the statistically the same as the Stimulated Data Histogram/Bins. Thus, the two samples have the same distribution in the null hypothesis.

The alternative hypothesis is that the Unstimulated/Control Data Histogram/Bins are significantly different from the Stimulated Data Histogram/Bins. Thus, the two distributions have a different distribution.

Usage

```
summary.ProbBin.FCS(object, verbose=FALSE,...)
```

Arguments

`object`	ProbBin.FCS object
`verbose`	Boolean whether to output all the counts in each bin
`...`	not used

Details

There are four main test statistics involved which are the following:

1. Test1: T.chi.unadj=max(0,(PBmetric-mean(PBmetric)) / SD(PBmetric)) is approximately standard normal (by the Central Limit Theorem (CLT)). Thus, the test of significance used the standard normal test as proposed by Mario Roederer.

2. Test2: Adjusted PB metric statistic is distributed as a chi-squared statistics. Thus, the test of significance uses the chi-squared test as proposed by Keith A. Baggerly.

3. Test3: Adjusted T.chi.unadj statistic is approximately the standard normal (by CLT). Thus the test of significance uses the standard normal test as proposed by Keith A. Baggerly.

4. Test4: Pearson's statistic using the Chi-Squared Test. There has been a suggestion of using a different number of degrees of freedom

Please note that all four tests use different statistics to test the same null hypothesis against the same alternative hypothesis.

Test 2 and 3 are ajusted forms of the statistics mentioned in Test 1.

Different p-values both one and two-sided are given for those applicable statistics.

Value

A list consisting of:

`PBinType`	Type of Probability Binning:
"by.control"	uses the control dataset to obtain the breaks/cutoffs to bin the stimulated dataset given a certain number of observations in each bin of the control dataset
"combined"	uses the combined dataset (both control and stimulated datasets) to obtain the breaks/cutoffs for the bins given a certain number in each bin
`control.bins`	single column matrix of the counts in each bin of the control dataset
`stim.bins`	single column matrix of the counts in each bin of the stimulated dataset
`total.control`	numeric; total number in the control dataset
`total.stim`	numeric; total number in the stimulated dataset
`T.chi.unadj`	Roederer's unadjusted normalized PB metric statistic which is normalized by subtracting off the mean and then dividing by the standard deviation. This statistic is approximately standard normal.
`p.val.2tail.z.unadj`	Two-tailed standard normal p-value corresponding to the Roederer's unadjusted normalized PB metric statistic which is approximated as a standard normal
`p.val.1tail.z.unadj`	Upper standard normal one-tailed p-value corresponding to the Roederer's unadjusted PB metric statistic which is approximated as a standard normal
`PBmetric.unadj`	Roederer's unadjusted PB metric which is ((n.c + n.s)/(2*nc.*n.s))*Chi-squared or an unadjusted chi-squared statistic, where n.c is the number of control observations (unbinned) and n.s is the number of stimulated observations (unbinned)
`PBmetric.adj`	Baggerly's adjusted PB metric statistic which is a Chi-squared statistic
`PB.df`	The degrees of freedom of the PB metric (adjusted and unadjusted) which is B-1, where B is the number of bins in the eitherthe control or the stimulated binned data
`p.val.1tail.chi.adj`	Upper one-tailed chi-squared p-value corresponding to Baggerly's adjusted PB metric
`T.chi.adj`	Baggerly's PB metric which is normalized by subtracting off the mean and dividing by the standard deviation; This normalized statistic is approximately standard normal.
`p.val.1tail.z.adj`	Upper one-tailed standard normal p-value corresponding to the Baggerly's adjusted normalized PB metric statistic which is approximated as a standard normal
`p.val.2tail.z.adj`	Standard normal two-tailed p-value corresponding to the Baggerly's adjusted PB metric statistic which is approximated as a standard normal
`pearson.stat`	Pearson's Chi-Squared Statistic with degrees of freedom 2B-1, where B is the number of bins in either the control or the stimulated binned data
`pearson.df`	the degrees of freedom for the chi-squared statistic
`pearson.p.value`	The p-value corresponding to the chi-squared distribution

`pearson.method`
: string of the indicating the type of test and options performed

`pearson.dataname`
: string of the name(s) of the data

`pearson.observed`
: a vector of the observed counts

`pearson.expected`
: a vector of the expected counts under the null hypothesis

`pearson.p.val.PB.df`
: Fisher's Chi-squared statistic with degrees of freedom B-1, where B is the number of bins in either the control or the stimulated binned data

Author(s)

A.J. Rossini and J.Y. Wan

References

Keith A. Baggerly "Probability Binning and Test Agreement between Multivariate Immunofluorescence Histograms: Extending the Chi-Squared test" Cytometry 45: 141:150 (2001).

Mario Roederer, et al. "Probability Binning Comparison: A Metric for Quantitating Univariate Distribution Differences" Cytometry 45:37-46 (2001).

Documentation for `chisq.test`.

See Also

`ProbBin.FCS`, `ProbBin.flowcytest`, `chisq.test`

Examples

```
if (require(rfcdmin)){
  ## obtaining the FCS objects from VRC data
if ( !(is.element("unst.1829", objects()) & is.element("st.1829", objects())) ){
data(VRCmin)
}
IFN.gamma.1<-unst.1829@data[1:2000,4]
IFN.gamma.2<-st.1829@data[1:2000,4]

#Probability binning using the control dataset to determine the breaks
PB1<-ProbBin.FCS(IFN.gamma.1, IFN.gamma.2, 200,
varname=colnames(unst.1829@data)[4], PBspec="by.control",MY.DEBUG=FALSE)

sum.PB1.1<-summary(PB1)
sum.PB1.2<-summary.ProbBin.FCS(PB1)

}
```

xgobi.FCS *XGobi: Dynamic Graphics for Data Analysis on FCS R objects*

Description

This function allows for a multidimensional view/manipulation of the data of the FCS object. Each row is an observation/cell, and the columns are regarded as the different variable conditions.

Usage

```
xgobi.FCS(myFCSobj, subset.row = NULL, subset.col = NULL, ...)
```

Arguments

`myFCSobj`	FCS object
`subset.row`	a vector of the row positions to be displayed; by default the first 1/15th rows are chosen to be displayed
`subset.col`	a vector of the column positions to be displayed; by default the first 1/2 of the columns are displayed
`...`	additional 'xgobi' function parameters/options in 'xgobi' package

Value

A graphics window with user-enabled manipulations The UNIX 'status' upon completion, i.e. '0' if ok.

WARNING

Abuses/uses xgobi: XGobi cannot handle datasets that are too large. Therefore, use subset.col and subset.row options to reduce the data matrix of the FCS R-object. Please see 'xgobi' for other commands in the signature.

Note

By default only a subset of the data is shown in xgobi because of size limitations. The user may be able to view the whole FCS dataset by using xgobi, but only if the dataset is not too huge for xgobi capabilities. It may be advisable to `createGate` and `extractGatedData` before viewing with xgobi.

Author(s)

A.J. Rossini and J.Y. Wan

References

Please see 'xgobi' in 'xgobi' package.

websites <URL: http://www.research.att.com/areas/stat/xgobi/>, <URL: http://www.public.iastate.edu/~dicook/>

of R port Kurt Hornik and Martin Maechler maechler@stat.math.ethz.ch

See Also

'xgobi' in **xgobi** package, `plot-methods`,`plotvar.FCS`, `createGate`, `extractGatedData`, `icreateGate`

Examples

```
  if (require(xgobi)) {
   if (require(rfcdmin)){
    ## obtaining the FCS objects from VRC data
    if (!(is.element("unst.1829", objects()))) {
      data(VRCmin)
    }
    if (interactive()==TRUE) {
      ## plots first 1/15 rows
      ## plots first 1/2 columns
      xgobi.FCS(unst.1829, title="unst.1829 default subset")

      ## plots all the rows
      ## plots only the first 3 columns
      xgobi.FCS(unst.1829, subset.row=1:6000, subset.col=1:2,
                title="unst.1829 first 6000 rows/cells with 2 column params")
    }
  }
 }
```

Index