Package 'cydar'

November 11, 2026

Version 1.36.0

Date 2024-08-11

Title Using Mass Cytometry for Differential Abundance Analyses

Depends SingleCellExperiment

Imports viridis, methods, shiny, graphics, stats, grDevices, utils, BiocGenerics, S4Vectors, BiocParallel, SummarizedExperiment, flowCore, Biobase, Rcpp, BiocNeighbors

Suggests ncdfFlow, testthat, rmarkdown, knitr, edgeR, limma, glmnet, BiocStyle, flowStats

biocViews ImmunoOncology, FlowCytometry, MultipleComparison, Proteomics, SingleCell

Description Identifies differentially abundant populations between samples and groups in mass cytometry data. Provides methods for counting cells into hyperspheres, controlling the spatial false discovery rate, and visualizing changes in abundance in the high-dimensional marker space.

License GPL-3

NeedsCompilation yes

VignetteBuilder knitr

LinkingTo Rcpp

SystemRequirements C++11

RoxygenNote 7.3.2

git_url https://git.bioconductor.org/packages/cydar

git_branch RELEASE_3_23

git_last_commit f118583

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Aaron Lun [aut, cre]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

Contents

countCells . . . 2
createColorBar . . . 4
CyData . . . 5
dnaGate . . . 7
expandRadius . . . 9
findFirstSphere . . . 10
intensityRanges . . . 12
interpretSpheres . . . 13
labelSpheres . . . 16
medIntensities . . . 17
multiIntHist . . . 19
neighborDistances . . . 20
normalizeBatch . . . 22
outlierGate . . . 26
pickBestMarkers . . . 27
plotSphereIntensity . . . 29
plotSphereLogFC . . . 30
poolCells . . . 31
prepareCellData . . . 33
spatialFDR . . . 34

Index **36**

`countCells` *Count cells in high-dimensional space*

Description

Count the number of cells from each sample lying inside hyperspheres in high-dimensional space.

Usage

```
countCells(
  prepared,
  tol = 0.5,
  num.threads = 1,
  BPPARAM = SerialParam(),
  downsample = 10,
  filter = 10
)
```

Arguments

prepared	A List object produced by prepareCellData.
tol	A numeric scalar to be used as the scaling factor for the hypersphere radius.
num.threads	Integer scalar specifying the number of threads to use.
BPPARAM	A BiocParallelParam object specifying how parallelization is to be performed in findNeighbors.
downsample	An integer scalar specifying the frequency with which cells are sampled to form hyperspheres.
filter	An integer scalar specifying the minimum count sum required to report a hypersphere.

Details

Consider that each cell defines a point in M-dimensional space (where M is the number of markers), based on its marker intensities. This function constructs hyperspheres and counts the number of cells from each sample lying within each hypersphere. In this manner, the distribution of cells across the space can be quantified. For each hypersphere, cell counts for all samples are reported along with the median intensity across the counted cells for each marker.

Each hypersphere is centered on a cell to ensure that only occupied spaces are counted. However, for high-density spaces, this can result in many redundant hyperspheres. To reduce computational work, only a subset of cells are used to define hyperspheres. The downsampling frequency is specified by `downsample`, e.g., only every 10th cell is used to make a hypersphere by default.

Each hypersphere also has a radius of `tol*sqrt(M)` (this relationship avoids loss of counts as M increases). `tol` can be interpreted as the acceptable amount of deviation in the intensity of a single marker for a given subpopulation. The default value of 0.5 means that, for any one marker, cells with +0.5 or -0.5 intensity will be counted into the same subpopulation. This value is sensible as intensities are usually on a log-10 scale, such that a total of 10-fold variability in marker intensities is tolerated.

The coordinates are reported as (weighted) medians across all cells in each hypersphere. Compared to the center, the median better reflects the location of the hypersphere if the cells are not distributed around the centre. Each cell is weighted inversely proportional to the total number of cells in the corresponding sample. This ensures that large samples do not dominate the median calculation.

All hyperspheres with count sums below `filter` are removed by default. Such hyperspheres do not have enough counts (and thus, information) for downstream analyses. Removing them reduces the amount of memory required to form the output matrix.

Value

A CyData object containing the following information:

`counts` An integer matrix of counts for each hypersphere (row) and sample (column) in the `assays` slot.

`intensities`: A numeric matrix of median intensities for each hypersphere (row) and marker (column), accessible with the `intensities` function.

`cellAssignments`: A list of integer vectors specifying the cells contained within each hypersphere, accessible with the `cellAssignments` function.

`totals`: An integer vector specifying the total number of cells in each sample, stored as a field in the `colData`.

`center.cell`: An integer vector specifying the cell that is used as the centre of each hypersphere, accessible with the `getCenterCell` function.

Contents of `prepared` are also stored in the `int_metadata` of the output object.

Author(s)

Aaron Lun

References

Lun ATL, Richard AC, Marioni JC (2017). Testing for differential abundance in mass cytometry data. *Nat. Methods*, 14, 7:707-709.

Samusik N, Good Z, Spitzer MH et al. (2016). Automated mapping of phenotype space with single-cell data. *Nat. Methods* 13:493-496

See Also

`prepareCellData`, to generate the input object `prepared`.

Examples

```
example(prepareCellData, echo=FALSE)
downsample <- 10L
tol <- 0.5

cnt <- countCells(cd, filter=1, downsample=downsample, tol=tol)
cnt
```

`createColorBar` *Create a color bar*

Description

A convenience function to create a color bar on the current graphics device.

Usage

```
createColorBar(colors, top.text=NULL, bottom.text=NULL,
    lower=-0.5, upper=0.5, x.pos=0, width=1, cex=1.5)
```

Arguments

colors	A character vector of colors to be used in successive steps of the color bar, from bottom to top.
top.text	String containing the label to be placed at the top of the color bar.
bottom.text	String containing the label to be placed at the bottom of the color bar.
lower	Numeric scalar specifying the bottom y-coordinate of the color bar.
upper	Numeric scalar specifying the top y-coordinate of the color bar.
x.pos	Numeric scalar specifying the x-coordinate of the middle of the color bar.
width	Numeric scalar specifying the width of the color bar.
cex	Numeric scalar specifying the size of the text.

Details

If `bottom.text` or `top.text` are not specified, they will be take from the first and last names of `colors`, respectively. If no names are available, no labels will be created.

Value

A color bar is created on the current graphics device. `NULL` is invisibly returned.

Author(s)

Aaron Lun

Examples

```
plot(0,0, axes=FALSE, xlab="", ylab="")
createColorBar(heat.colors(100), "High", "Low")
```

CyData *CyData class and methods*

Description

The CyData class is derived from the SingleCellExperiment class. It is intended to store the cell counts for each group of cells (rows) for each sample (columns). Groups are intended to be hyperspheres (see `countCells`) but could also be arbitrary clusters of cells. It also stores the median intensities for each group and the identity of cells in the groups, parallel to the rows.

Details

CyData objects should not be created directly by users. The class has some strict validity conditions that are not easily satisfied by manual construction. Users should rely on functions like `prepareCellData` and `countCells` to create the objects. An overview of the CyData class and the available methods.

Getter functions for group-level data

In the following code chunks, `x` or `object` are CyData objects. `mode` is a string specifying the types of markers that should be returned; this defaults to only those markers that are used in `prepareCellData`, but can also return the unused markers or all of them.

- `intensities(x, mode=c("used", "all", "unused")` returns a numeric matrix of intensities for each group of cells (rows) and markers (columns). Rows of the output matrix correspond to rows of `x`. Values are returned for the markers specified by `mode` (see above).
- `cellAssignments(x)` returns a list of integer vectors, where each vector corresponds to a row of `x` and contains the indices of the cells in that group. Indices refer to columns of `cellIntensities(x)`.
- `markernames(object, mode=c("used", "all", "unused"))` returns a character vector of the marker names, depending on `mode` (see above).
- `getCenterCell(x)` returns the index of the cell used at the center of each hypersphere.

Getter functions for cell-level data

In the following code chunks, `x` is a CyData object and `mode` is as previously described.

- `cellIntensities(x, mode=c("used", "all", "unused"))` returns a numeric matrix of intensities for each marker (row) and cell (column).
- `cellInformation(x)` returns a DataFrame with one row per cell. The `sample` field specifies the sample of origin for each cell, while the `cell` field specifies the original row index of that cell in its original sample.

Subsetting

The subsetting and combining behaviour of CyData objects is mostly the same as that of SingleCellExperiment objects. The only difference is that subsetting or combining CyData objects by column is not advisable. Indeed, attempting to do so will result in a warning from the associated methods. This is because the columns are usually not independent in contexts involving clustering cells across multiple samples. If a sample is to be removed, it is more appropriate to do so in the function that generates the CyData object (usually `prepareCellData`).

Author(s)

Aaron Lun

Examples

```
example(countCells, echo=FALSE)

markernames(cnt)
head(intensities(cnt))
head(cellAssignments(cnt))
```

dnaGate *Gate events based on DNA channels*

Description

Construct a gate to remove debris and doublets, based on the two DNA (iridium) channels used in most mass cytometry experiments.

Usage

```
dnaGate(x, name1, name2, tol=0.5, nmads=3, type=c("both", "lower"),
    shoulder=FALSE, rank=1, ...)
```

Arguments

x	A flowFrame object like that constructed by poolCells.
name1, name2	Strings containing the names of the two DNA channels.
tol	A numeric scalar quantifying the maximum distance from the equality line.
nmads	A numeric scalar specifying the number of median absolute deviations (MADs) beyond which an event can be considered an outlier.
type	A string specifying the type of gating to be performed.
shoulder	A logical scalar indicating whether the function should attempt to detect shoulders.
rank	An integer scalar specifying the peak corresponding to singlets. By default, the largest mode is treated as the singlet peak.
...	Additional arguments to pass to density, to fine-tune identification of local minima.

Details

For each DNA channel, the rankth-largest local mode is identified and is assumed to correspond to singlets. Local minima of density that neighbour the chosen mode are identified. To remove debris, the lower bound is set to the largest local minima that is smaller than the chosen mode. To remove doublets, the upper bound is set to the smallest local minima that is larger than the chosen mode.

We also consider an alternative lower bound at nmads MADs below the chosen mode. (Here, the MAD is computed using only values below the mode, to avoid potential inflation due to a doublet peak.) If this alternative is larger than the largest local minima below the mode, it is used as the lower bound instead. This avoids using a poor lower bound when there are no obvious minima in the distribution. Similarly, an alternative upper bound is defined at nmads MADs above the median, and is used if it is smaller than the smallest local minima above the mode.

For some data sets, there may not be any clear bimodality in the intensity distribution, e.g., if the mean shift is dominated by noise. If shoulder=TRUE, the function will attempt to identify the doublet peak as a "shoulder" off the singlet peak. Alternatively, if there is no evidence for separate singlet/doublet peaks, it may not be feasible (or desirable) to try to distinguish them. In such

cases, users can set `type="lower"`, whereby the upper bound is set to an arbitrarily large value and effectively ignored during gating.

To simultaneously gate on both DNA channels, we fit a line to the paired intensities for all events, i.e., the "equality line". Two perpendicular lines passing through the paired lower/upper bounds are constructed. Two parallel lines that are `tol` away from the equality line are also defined. The box defined by these four lines is used to construct a polygonGate object, within which all events are retained.

The value of `tol` represents the maximum Euclidean distance of any event from the equality line in the two-dimensional space. Any event more the `tol` from the line is removed as the two iridium isotopes have not been evenly captured. This may be indicative of a problem with the TOF detector for this event.

Value

A polygonGate object, defined to retain singlet events.

Author(s)

Aaron Lun

See Also

polygonGate, poolCells, density

Examples

```
set.seed(200)

### Mocking up some data with clear bimodality: ###
library(flowCore)
singlets <- rnorm(20000, 2, 0.2)
dna1 <- matrix(rnorm(40000, singlets, 0.1), ncol=2)
doublets <- rnorm(10000, 3, 0.2)
dna2 <- matrix(rnorm(20000, doublets, 0.1), ncol=2)
dna.int <- rbind(dna1, dna2)
colnames(dna.int) <- c("Ir191", "Ir193")
ff <- flowFrame(dna.int)

### Defining the gate: ###
dgate <- dnaGate(ff, "Ir191", "Ir193")
smoothScatter(dna.int[,1], dna.int[,2])
polygon(dgate@boundaries[,1], dgate@boundaries[,2], border="red")

### Mocking up some data with no obvious bimodality: ###
singlets <- rnorm(20000, 2, 0.2)
dna1 <- matrix(rnorm(40000, singlets, 0.1), ncol=2)
doublets <- rnorm(10000, 2.5, 0.2) # <- less separation between modes
dna2 <- matrix(rnorm(20000, doublets, 0.1), ncol=2)
dna.int <- rbind(dna1, dna2)
colnames(dna.int) <- c("Ir191", "Ir193")
ff <- flowFrame(dna.int)
```

```
### Defining the gate: ###
dgate <- dnaGate(ff, "Ir191", "Ir193", shoulder=TRUE)
smoothScatter(dna.int[,1], dna.int[,2])
polygon(dgate@boundaries[,1], dgate@boundaries[,2], border="red")
```

---

expandRadius *Expand the hypersphere radius*

Description

Expands the hypersphere radius to account for intensity shifting between non-barcoded samples.

Usage

```
expandRadius(prepared, design = NULL, tol = 0.5)
```

Arguments

prepared	A List object produced by prepareCellData.
design	A numeric matrix specifying the experimental design.
tol	A numeric scalar proportional to the hypersphere radius, see countCells.

Details

This function increases the hypersphere radius to account for random shifts in marker intensity between non-barcoded samples. The required increase is estimated by taking the mean of all intensities for each marker in each sample; computing the variance of the mean intensities across samples for each marker; and taking the mean variance across all markers. This is equivalent to the square of the extra distance between cells caused by intensity shifts between samples.

The estimated increase is added onto tol, and the returned value can be directly used in the tol argument of countCells. This expands the hyperspheres to ensure that corresponding subpopulations in different samples are still counted together. Otherwise, an intensity shift in one sample may move the cells in a subpopulation out of a hypersphere. This will inflate the variability if it occurs between replicate samples, and introduce spurious differences if it occurs between samples in different conditions.

Value

A numeric scalar specifying a modified tol to use in countCells.

Author(s)

Aaron Lun

See Also

prepareCellData, to generate the required input.

countCells, where the tol can be set to the output of this function.

Examples

```
### Mocking up some data: ###
nmarkers <- 20
marker.names <- paste0("X", seq_len(nmarkers))
nsamples <- 8
sample.names <- paste0("Y", seq_len(nsamples))

x <- list()
for (i in sample.names) {
    ex <- matrix(rgamma(nmarkers*1000, 2, 2), ncol=nmarkers, nrow=1000)
    ex <- t(t(ex) + rnorm(nmarkers, 0, 0.25)) # Adding a shift per marker
    colnames(ex) <- marker.names
    x[[i]] <- ex
}

### Running the function: ###
cd <- prepareCellData(x)
expandRadius(cd)
```

---

findFirstSphere *Identifies the first non-redundant hyperspheres*

Description

Tests whether each hypersphere is not redundant to (i.e., lies more than a threshold distance away from) another hypersphere with a lower p-value.

Usage

```
findFirstSphere(x, pvalues, threshold=1, block=NULL, num.threads=1)
```

Arguments

x	A numeric matrix of hypersphere coordinates (median locations for all markers), where rows correspond to hyperspheres and columns correspond to markers. Alternatively, a CyData object containing median intensities for groups of cells, such as that produced by countCells.
pvalues	A numeric vector of p-values, one for each row (i.e., hypersphere) of x.
threshold	A numeric scalar specifying the maximum distance between the locations of two redundant hyperspheres.
block	A factor specifying which hyperspheres belong to which block, where non-redundant hyperspheres are identified within each block.
num.threads	Integer scalar specifying the number of threads to use.

Details

This function iterates across the set of hyperspheres, typically ordered by decreasing significance. It will tag a hypersphere as being redundant if its location lies within `threshold` of the location of a higher-ranking hypersphere in all dimensions. In this manner, the set of all DA hyperspheres can be filtered down to a non-redundant subset that is easier to interpret.

Note that the criterion for redundancy mentioned above is equivalent to a Chebyshev distance, rather than Euclidean. This is easier to interpret, especially given that the median intensity is defined separately for each marker. Unlike in `countCells`, the threshold is not scaled by the number of markers because each hypersphere location is computed as an average across cells. This means that there is generally no need to account for extra distance due to noise between cells.

The default `threshold` of unity assumes that the intensities have been transformed to or near a log10 scale. It means that one hypersphere must vary from another by at least one log10-unit (i.e., a 10-fold change in intensity) in at least one marker to be considered non-redundant. This avoids reporting many hyperspheres that differ from each other by relatively small, uninteresting shifts in intensity. Greater resolution can be obtained by decreasing this value, e.g., to 0.5.

If `block` is set, non-redundant hyperspheres are only identified within each block (i.e., a hypersphere cannot be redundant to hyperspheres in different blocks). For example, one can set `block` to the sign of the log-fold change. This ensures that hyperspheres changing in one direction are not considered redundant to those changing in another direction. By default, all hyperspheres are considered to be part of the same block.

Value

A logical vector indicating whether each of the hyperspheres in `x` is non-redundant.

Author(s)

Aaron Lun

Examples

```
# Mocking up some data.
coords <- matrix(rnorm(10000, 2, sd=0.3), nrow=1000)
pval <- runif(1000)
logfc <- rnorm(1000)

# Keep most significant non-redundant ("first") hyperspheres.
findFirstSphere(coords, pval)

# Block on the sign of the log-fold change.
findFirstSphere(coords, pval, block=sign(logfc))
```

`intensityRanges` *Define intensity ranges*

Description

Set the ranges of the marker intensities, to direct construction of the colour bar for plotting.

Usage

```
intensityRanges(x, p=0.01)
```

Arguments

`x`	A CyData object produced by `prepareCellData`.
`p`	A numeric scalar specifying the quantile at which intensities should be bounded.

Details

For each marker, intensities across all cells are used to calculate the `p` and `1-p` quantiles. This defines the lower and upper bound, respectively, to use as the `irange` argument in `plotSphereIntensity`. The aim is to prevent extreme outliers from skewing the distribution of colours. This would result in loss of resolution at non-outlier values.

Note that, while the bounds are defined at the quantiles `p` and `1-p`, the colour gradient will not be computed across the percentiles. That is, the "middle" of the gradient will not represent the median cell intensity. Rather, the colour gradient is computed from the lower and upper bounds, so the middle with represent the average of the bounds. Users should label the colour bar with the bounded intensities, rather than with the values of `p` or `1-p`.

Value

A matrix specifying the lower and upper bounds (rows) on the intensity for each marker (columns). Markers used in distance calculations (see `markers` in ?`prepareCellData`) are listed first, followed by the unused markers.

Author(s)

Aaron Lun

See Also

`prepareCellData`, `plotSphereIntensity`

Examples

```
example(countCells, echo=FALSE) # Using the mocked-up data set.
bounds <- intensityRanges(cnt)

# Plotting example (using a subset for fast PCA).
cd.subset <- t(cellIntensities(cnt)[,1:1000])
coords <- prcomp(cd.subset)

chosen.marker <- 5
plotSphereIntensity(coords$x[,1], coords$x[,2],
    intensity=cd.subset[chosen.marker,],
    irange=bounds[,chosen.marker])
```

`interpretSpheres` *Interactive interpretation of hyperspheres*

Description

Launches a Shiny app to assist interpretation of hyperspheres.

Usage

```
interpretSpheres(x, markers=NULL, labels=NULL, select=NULL, metrics=NULL,
    num.per.row=6, plot.height=100, xlim=NULL, p=0.01,
    red.coords=NULL, red.highlight=NULL, red.plot.height=500,
    add.plot=NULL, add.plot.height=500, run=TRUE, ...)
```

Arguments

`x`	A CyData object generated by `countCells`, containing counts and coordinates for each hypersphere.
`markers`	A character vector indicating the markers to use, and the order they should be plotted in. If `NULL`, all markers are used in the order corresponding to the columns of `intensities(x)`.
`labels`	A character vector containing existing labels for the hyperspheres. This should be of length equal to the number of rows in `x`.
`select`	A logical or integer vector indicating which rows of `x` should be inspected. Defaults to all rows.
`metrics`	A dataframe containing metrics to be reported for each hypersphere, with number of rows equal to `x`.
`num.per.row`	An integer scalar specifying the number of plots per row.
`plot.height`	An integer scalar specifying the height of each plot in pixels.
`xlim`	A numeric vector of length two specifying the x-axis limits for all plots. Otherwise, `intensityRanges` is used to define limits for each marker.
`p`	A numeric scalar to be passed to `intensityRanges`.

red.coords — A numeric matrix with two columns and number of rows equal to nrow(x), containing a reduced-dimension representation of hypersphere coordinates. The first and second columns should contain the x- and y-coordinates, respectively.

red.highlight — A logical or integer vector specifying which rows of x should be highlighed on the reduced dimensionality plot.

red.plot.height — An integer scalar specifying the height of the reduced-dimension plot.

add.plot — A function taking two arguments (see below) to create additional plots in the app.

add.plot.height — An integer scalar specifying the height of the additional plots.

run — A logical scalar specifying whether the Shiny app should be run.

... — Additional arguments to be passed to density.

Details

This function creates a Shiny app in which density plots are constructed for intensities across all cells, one for each marker. For a given hypersphere, the median intensity is plotted as a red circle on top of the density plot for each marker. This allows users to quickly determine the biological meaning of each hypersphere, based on its median marker expression (and other statistics in metrics).

For each marker, the area under the curve is highlighted using the viridis colour scheme. This is based on whether the median is relatively high (yellow) or low (purple) compared to all of the cells. An interval around the median is also displayed, representing the range of intensities across a given percentage (default 95%) of cells in the hypersphere. This provides more information about the spread of intensities within each hypersphere.

Each hypersphere can be labelled with some meaningful term, e.g., the cell type that corresponds to the suite of expressed markers. For each hypersphere, the closest hyperspheres that have already been labelled are shown, along with the Euclidean distances to their locations. This is designed to assist with the labelling process by identifying pre-labelled hyperspheres in the neighbouring space.

Finally, the labels can be saved to R using the "Save to R" button. This stops the app and returns a character vector of labels in the R session. Existing labels can also be re-used by supplying them to labels, to allow users to label parts of the data set at a time.

Value

If run=FALSE, a Shiny app is returned that can be run with runApp. This passes control to a browser window in which labels can be entered for each hypersphere. Upon stopping the app, a character vector of length equal to the number of rows in x is returned.

If run=TRUE, a Shiny app is opened directly in a browser window. This returns a character vector upon stopping, as previously described.

Navigation

Users can navigate through the data set using the "Previous" or "Next" buttons. This moves across hyperspheres specified by select, i.e., pressing "Next" will jump to the next hypersphere in

select. By default, select=NULL which means that the app will progress through all hyperspheres in x. It is often worth setting select, e.g., to non-redundant significant hyperspheres in order to reduce the number of elements that need to be inspected.

Users can also jump to a particular row/hypersphere by providing an integer scalar in the "Go to sphere" field. This specifies the row index for the hypersphere of interest, and works for either selected or unselected hyperspheres. However, pressing "Previous" or "Next" will jump to the nearest index of the selected hyperspheres. The navigation history at any given time is shown in the side bar.

A reduced-dimensionality plot is also constructed using specified coordinates in two-dimensional space for each hypersphere. The current hypersphere is marked on this plot with a red dot. Previously visited and labelled hyperspheres are marked in black. Users can also highlight particular hyperspheres in orange with red.highlight, e.g., if they are significantly differential or not.

Putting in additional plots

Users can define add.plot as a function taking two arguments:

1. An integer scalar, specifying the row index of the second argument. This corresponds to the hypersphere currently being inspected.
2. A CyData object, which is set internally to the value of x used in interpretSpheres. This should contain information about all hyperspheres.

add.plot should generate a plot on the current graphics device. This is usually done in a hypersphere-specific manner, where the first argument is used to extract the relevant information from the second argument. For example, the abundances of all samples can be visualized directly for each hypersphere in the app.

Inspecting label propagation

If red.dim is supplied and at least one cell is labelled, the "Update labels" button can be used to propagate the labels to surrounding cells. Specifically, for each unlabelled cell, the closest labelled cell is identified and its label is assigned to the unlabelled cell. A plot is then created showing the distribution of cells for each label in the low-dimensional space.

This functionality is useful for determining how the labels would be automatically assigned by labelSpheres. If many distinct clusters have the same label, it suggests that more manual labelling is required to distinguish clusters. Note that the automatically assigned labels are *not* recorded, they are only used here for visualization purposes.

Author(s)

Aaron Lun

See Also

density, intensityRanges, runApp

Examples

```
# Mocking up some data.
example(prepareCellData, echo=FALSE)
cnt <- countCells(cd, filter=1)

# Constructing the app
app <- interpretSpheres(cnt, run=FALSE)

## Not run: # Running the app from the object.
labels <- shiny::runApp(app)

#Or directly running the app from the function.
labels <- interpretSpheres(cnt)

## End(Not run)

# Doing it with metrics and coordinates.
N <- nrow(cnt)
metrics <- data.frame(logFC=rnorm(N), PValue=runif(N))
coords <- matrix(rnorm(N*2), ncol=2)
app <- interpretSpheres(cnt, red.coord=coords, metrics=metrics, run=FALSE)

# Doing it with an extra plot.
app <- interpretSpheres(cnt, run=FALSE, add.plot=function(i, x) {
    barplot(assay(x)[i,]/x$totals*100, ylab="Percentage of cells")
})
```

`labelSpheres` *Label unannotated hyperspheres*

Description

Given a set of labels for annotated hyperspheres, propagate labels to the surrounding unannotated hyperspheres.

Usage

```
labelSpheres(x, labels)
```

Arguments

`x`	A numeric matrix of hypersphere coordinates, containing the median intensity of each marker (column) in each hypersphere (row). Alternatively, a CyData object containing median intensities for groups of cells, such as that produced by `countCells`.
`labels`	A character vector of labels for each hypersphere, set to an empty string for unannotated hyperspheres.

Details

After some hyperspheres have been labelled with `interpretSpheres`, the remainder can be automatically labelled with this function. Unlabelled hyperspheres are assigned the label of the closest labelled hypersphere. Obviously, this assumes that enough hyperspheres have been labelled so that the closest hypersphere is of a similar cell type/state.

Value

A character vector containing labels for all hyperspheres.

Author(s)

Aaron Lun

See Also

`interpretSpheres`

Examples

```
set.seed(1000)
coords <- matrix(rgamma(10000, 2, 2), nrow=1000)
labels <- character(nrow(coords))
labels[1:4] <- c("B", "CD4T", "CD8T", "Mono")

ref <- labelSpheres(coords, labels)
head(ref)
```

`medIntensities` *Compute median marker intensities*

Description

Calcalute the median intensity across cells in each group and sample for the specified markers.

Usage

```
medIntensities(x, markers)
```

Arguments

`x`	A CyData object where each row corresponds to a group of cells, such as that produced by `countCells`.
`markers`	A vector specifying the markers for which median intensities should be calculated.

Details

For each group of cells, the median intensity across all assigned cells in each sample is computed. This is returned as a matrix of median intensities, with one value per sample (column) and hypersphere (row). If a sample has no cells in a group, the corresponding entry of the matrix will be set to NA.

The groups in x should be defined using a different set of markers than in markers. Specifically, the markers used in prepareCellData should not be the same as the markers in this function. If the same markers are used for both functions, then a shift is unlikely to be observed. This is because, by definition, the groups will contain cells with similar intensities for those markers.

The idea is to use the median intensities for weighted linear regression to identify a shift in intensity within each hypersphere. The weight for each group/sample is defined as the number of cells, i.e., the "counts" assay in x. This accounts for the precision with which the median is estimated, under certain assumptions. See the Examples for how this data can be prepared for entry into analysis packages like limma.

The median intensity is used rather than the mean to ensure that shifts are interpreted correctly. For example, mean shifts can be driven by strong changes in a subset of cells that are not representative of the majority of cells in the group. This could lead to misinterpretation of the nature of the shift with respect to the group's overall identity.

Value

A CyData object is returned equivalent to x, but with numeric matrices of sample-specific median intensities as additional elements of the assays slot.

Choosing between counting strategies

In situations where markers can be separated into two sets (e.g., cell type and signalling markers), there are two options for analysis. The first is to define groups based on the "primary" set of markers, then use medIntensities to identify shifts in each group for each of the "secondary" markers. This is the best approach for detecting increases or decreases in marker intensity that affect a majority of cells in each group.

The second approach is to use all markers in prepareCellData and count cells accordingly in countCells. This provides more power to detect changes in marker intensity that only affect a subset of cells in each group. It is also more useful if one is interested in identifying cells with concomitant changes in multiple secondary markers. However, this tends to be less effective for studying changes in a specific marker, due to the loss of precision with increased dimensionality.

Author(s)

Aaron Lun

See Also

prepareCellData, countCells

Examples

```
### Mocking up some data: ###
nmarkers <- 21
marker.names <- paste0("X", seq_len(nmarkers))
nsamples <- 5
sample.names <- paste0("Y", seq_len(nsamples))

x <- list()
for (i in sample.names) {
    ex <- matrix(rgamma(nmarkers*1000, 2, 2), ncol=nmarkers, nrow=1000)
    colnames(ex) <- marker.names
    x[[i]] <- ex
}

### Processing it beforehand with one set of markers: ###
cd <- prepareCellData(x, markers=marker.names[1:10])
cnt <- countCells(cd, filter=5)

## Computing the median intensity for one marker: ###
cnt2 <- medIntensities(cnt, markers=marker.names[21])
library(limma)
median.int.21 <- assay(cnt2, "med.X21")
cell.count <- assay(cnt2, "counts")
el <- new("EList", list(E=median.int.21, weights=cell.count))
```

`multiIntHist` *multiIntHist*

Description

Generate intensity histograms from multiple batches.

Usage

```
multiIntHist(collected, cols=NULL, xlab="Intensity", ylab="Density",
    lwd=2, lty=1, pch=16, cex=2, ...)
```

Arguments

`collected`	A list of numeric vectors, where each vector contains intensities for a given marker from all cells of a single batch.
`cols`	A vector of R colours of the same length as `collected`, to be used in colouring the histograms.
`xlab`, `ylab`	Strings specifying the x- and y-axis labels.
`lwd`, `lty`	Parameters for plotting the histogram traces.
`pch`, `cex`	Parameters for plotting the frequency of zeroes.
`...`	Other arguments to pass to `plot`.

Details

A histogram is constructed for the set of intensities from each batch, and the histogram outline is plotted with the specified parameters. The frequency of intensities at zero (or negative values) is indicated with a single point at an intensity of zero. This ensures that the number of events at zero and small non-zero intensities can be distinguished.

The process is repeated for all batches so that intensity distributions can be compared between batches. If `cols=NULL`, the `rainbow` colour palette is automatically used to generate the colour for each batch. Some small jitter is added to the zero points so that they do not completely overlap each other.

Value

Histogram traces representing the intensity distributions are produced on the current graphics device.

Author(s)

Aaron Lun

See Also

`normalizeBatch`

Examples

```
multiIntHist(list(rgamma(1000, 1, 1), rgamma(1000, 2, 1), rgamma(1000, 1, 2)))
```

---

neighborDistances *Compute distances to neighbors*

Description

Calculate the distances in high-dimensional space to the neighboring cells.

Usage

```
neighborDistances(
  prepared,
  neighbors = 50,
  downsample = 50,
  as.tol = TRUE,
  num.threads = 1
)
```

Arguments

prepared	A List object produced by prepareCellData.
neighbors	An integer scalar specifying the number of neighbours.
downsample	An integer scalar specifying the frequency with which cells are examined.
as.tol	A logical scalar specifying if the distances should be reported as tolerance values.
num.threads	Integer scalar specifying the number of threads to use.

Details

This function examines each cell at the specified downsampling frequency, and computes the Euclidean distances to its nearest neighbors. If as.tol=TRUE, these distances are reported on the same scale as tol in countCells. This allows users to choose a value for tol based on the output of this function. Otherwise, the distances are reported without modification.

To visualize the distances/tolerances, one option is to use boxplots, as shown below. Each boxplot represents the distribution of tolerances required for hyperspheres to contain a certain number of cells. For example, assume that at least 20 cells in each hypersphere are needed to have sufficient power for hypothesis testing. Now, consider all hyperspheres that are large enough to include the 19th nearest neighbour. The average distance required to do so would be the median of the boxplot generated from the 19th column of the output.

Another option is to examine the distribution of counts at a given tolerance/distance. This is done by counting the number of hyperspheres with a particular number of nearest neighbors closer than the specified tolerance. In this manner, the expected count distribution from setting a particular tolerance can be determined. Note that the histogram is capped at neighbors to save time.

Note that, for each examined cell, its neighbors are identified from the full set of cells. Downsampling only changes the rate at which cells are examined, for the sake of computational efficiency. Neighbors are not identified from the downsampled set as this will inflate the reported distances.

Value

A numeric matrix of distances where each row corresponds to an examined cell and each column i corresponds to the ith closest neighbor.

Author(s)

Aaron Lun

See Also

prepareCellData, to generate the prepared object.

countCells, where the choice of tol can be guided by the distance distributions.

Examples

```
example(prepareCellData, echo=FALSE)

distances <- neighborDistances(cd, as.tol=FALSE)
boxplot(distances, xlab="Neighbor", ylab="Distance")

# Making a plot to choose 'tol' in countCells().
distances <- neighborDistances(cd, as.tol=TRUE)
boxplot(distances, xlab="Neighbor", ylab="Tolerance")

required.count <- 20 # 20 cells per hypersphere
med <- median(distances[,required.count-1])
segments(-10, med, required.count-1, col="dodgerblue")
segments(required.count-1, med, y1=0, col="dodgerblue")

# Examining the distribution of counts at a given 'tol' of 0.7.
# (Adding 1 to account for the cell at the centre of the hypersphere.)
counts <- rowSums(distances <= 0.7) + 1
hist(counts, xlab="Count per hypersphere")
```

`normalizeBatch` *Normalize intensities across batches*

Description

Perform normalization to correct intensities across batches with at least one common level.

Usage

```
normalizeBatch(batch.x, batch.comp, mode="range", p=0.01,
    fix.zero=FALSE, target=NULL, markers=NULL, ...)
```

Arguments

`batch.x`	A list of length equal to the number of batches. Each element of the list should be of the same type as `x` used in `prepareCellData`.
`batch.comp`	A list of length equal to the number of batches. Each element should be a factor (or coercible to a factor) specifying the composition of each batch, i.e., which samples belong to which groups. Also can be `NULL`, see below.
`mode`	A string or character vector of length equal to the number of markers, specifying whether range-based or warping normalization should be performed for each marker. This can take values of `"range"`, `"warp"`, `"quantile"` or `"none"` (in which case no normalization is performed).
`p`	A numeric scalar between 0 and 0.5, specifying the percentile used to define the range of the distribution for range-based normalization.

fix.zero	A logical scalar indicating whether zero intensities should remain at zero when mode="range".
target	An integer scalar indicating the reference batch.
markers	A character vector specifying the markers to be normalized and returned.
...	Additional arguments to be passed to warpSet for mode="warp".

Details

Consider an experiment containing several batches of barcoded samples, in which the barcoding was performed within but not between batches. This function normalizes the intensities for each marker such that they are comparable between samples in different batches. The process for each marker is as follows:

1. Weight each batch by composition and number of cells.
2. Compute a transformation function for each batch to a reference intensity distribution.
3. Apply the batch-specific functions to that batch's samples to obtain normalized intensities.

Each element of batch.x should contain data from all samples of a single batch. This can be a ncdfFlowSet constructed from all samples in that batch, or a list of intensity matrices from all samples. In short, each element should be equivalent to the x argument in ?prepareCellData.

Each element of batch.comp should correspond to the same batch as the entry at the same position of batch.x. Each element should be a factor of length equal to the number of samples in the associated batch. Each value of the factor specifies the group identity of a sample in the batch, and should follow the ordering of samples within the corresponding element of batch.x.

All markers are used by default when markers=NULL. If markers is specified, only the specified markers will be normalized and returned in the output expression matrices. This is usually more convenient than subsetting the inputs or outputs manually.

To convert the output into a format appropriate for prepareCellData, apply unlist with recursive=FALSE. This will generate a list of intensity matrices for all samples in all batches, rather than a list of list of matrices. Note that a batch effect should still be included in the design matrix when modelling abundances, as only the intensities are corrected here.

Value

A list of lists, where each internal list corresponds to a batch and contains intensity matrices corresponding to all samples in that batch. This matches the format of batch.x.

Weighting within each batch

Weighting is performed to downweight the contribution of larger samples within each batch, as well as to match the composition of samples across different batches. The composition of each batch can be specified by batch.comp, see below for more details. The weighted intensities for each batch represents the pooled distribution of intensities from all samples in that batch.

Groupings can be specified as batch-specific factors in batch.comp, with at least one common group required across all batches. This composition is used to weight the contribution of each sample to the reference distribution. For example, a batch with more samples in group A and fewer samples in group B would get lower weights assigned to the former and larger weights to the latter.

Ideally, all batches would contain samples from all groups, with similar total numbers of cells across batches for each group. This maximizes the number of samples that are used to construct the reference distribution (and thus the stability of the reference). Samples in groups that are not present in all batches will be ignored as no weighting can be applied to an absent group.

If the composition of each batch is the same, `batch.comp` can be set to `NULL` rather than being manually specified.

Transforming to a reference

If `mode="range"`, a quantile function is constructed for the pooled distribution of each batch. These batch-specific functions are used to contruct a reference quantile function, representing a reference distribution. A batch-specific scaling function is defined to equalize the range of the weighted distribution of intensities from each batch to the range of this reference distribution. The range of each distribution is computed at percentiles `p` and `1-p` to avoid distortions due to outliers. If `fix.zero=TRUE`, the lower bound of the range is always set to zero to avoid turning a zero intensity into a non-zero value after normalization.

If `mode="quantile"`, a quantile function is constructed as described above, along with a reference quantile function. A batch-specific transformation function is defined to convert the quantiles for each batch to the corresponding quantiles of the reference. This is equivalent to coercing all batch-specific intensity distributions to the reference distribution.

If `mode="warp"`, a mock set of intensities is generated is generated that accounts for the differential weighting of events in each batch. This is used to construct a flowSet for use in warping normalization - see `?normalization` and `?warpSet` for details. A warping function is computed for each batch that equalizes the locations of landmarks (i.e., peaks) in both the batch-specific and reference intensity distributions.

If `target=NULL`, the reference distribution for each marker is defined as an average of the relevant statistic across batches. For `mode="range"` or `mode="quantile"`, this means that the reference quantile function is defined as the average of the functions across all batches. For `mode="warp"`, the average location of each landmark across all batches is used.

If `target` is not `NULL`, the specified batch will be used as the reference distribution. For `mode="range"` or `mode="quantile"`, this means that the reference quantile function will be defined as the quantile function of the chosen batch. Similarly, if `mode="warp"`, `warpSet` will align all other batches to the locations of the peaks in `target`.

Applying the transformation function

The transformation function is applied to the intensities of *all* samples in that batch, yielding corrected intensities for direct comparisons between samples. This is possible provided that there is at least one group that is present across all batches, in order to construct a common reference distribution. The assumption is that the batch effect in those shared groups is the same as that in the batch-specific groups. However, note that the adjustment may not be accurate if the to-be-corrected intensities lie outside the range of values used to construct the function.

Choosing between normalization methods

Warping normalization can be more powerful than range-based normalization, as the former can eliminate non-linear changes to the intensities whereas the latter cannot. However, it requires that landmarks in the intensity distribution (i.e., peaks) be easily identifiable and consistent across

batches. Large differences (e.g., a peak present in one batch and absent in another) may lead to incorrect adjustments. Such differences may be present when batches are confounded with uninteresting biological factors (e.g., individual, mouse of origin) that affect cell abundance. In such cases, range-based normalization with `mode="range"` is recommended as it is more constrained in how the intensities are adjusted. This reduces the risk of distorting the intensities, albeit at the cost of "under-normalizing" the data.

Quantile normalization is the most powerful of these methods but also requires the strongest assumptions. In particular, it requires that the composition of samples in shared groups is *exactly* the same across all batches. This is only practically applicable to experimental designs where the same control sample has been used in multiple batches. In such cases, users should set all control samples to the same "group" in `batch.comp`, while all other samples should be set to batch-specific groups (and are thus ignored during calculation of the transformation functions). Note that this approach also requires the control samples to cover the range of intensities in the other samples, otherwise the transformation function will need to extrapolate - often badly.

It is advisable to inspect the intensity distributions before and after normalization, to ensure that the methods have behaved appropriately. This can be done by constructing histograms for each marker with `multiIntHist`.

Author(s)

Aaron Lun

See Also

`prepareCellData`, `multiIntHist`, `normalization`, `warpSet`

Examples

```
### Mocking up some data: ###
nmarkers <- 10
marker.names <- paste0("X", seq_len(nmarkers))
all.x <- list()

for (b in paste0("Batch", 1:3)) { # 3 batches
    nsamples <- 10
    sample.names <- paste0("Y", seq_len(nsamples))
    trans.shift <- runif(nmarkers, 0, 1)
    trans.grad <- runif(nmarkers, 1, 2)
    x <- list()
    for (i in sample.names) {
        ex <- matrix(rgamma(nmarkers*1000, 2, 2), nrow=nmarkers)
        ex <- t(ex*trans.grad + trans.shift)
        colnames(ex) <- marker.names
        x[[i]] <- ex
    }
    all.x[[b]] <- x
}

batch.comp <- list( # Each batch contains different composition/ordering of groups
    factor(rep(1:2, c(3,7))),
```

```
    factor(rep(1:2, c(7,3))),
    factor(rep(1:2, 5))
)

### Running the function: ###
corrected <- normalizeBatch(all.x, batch.comp, mode="range")
par(mfrow=c(1,2))
plot(ecdf(all.x[[1]][[3]][,1]), col="blue", main="Before")
plot(ecdf(all.x[[2]][[3]][,1]), add=TRUE, col="red")
plot(ecdf(corrected[[1]][[3]][,1]), col="blue", main="After")
plot(ecdf(corrected[[2]][[3]][,1]), add=TRUE, col="red")
```

`outlierGate` *Create an outlier gate*

Description

Define gating thresholds to remove outlier events for a particular channel.

Usage

```
outlierGate(x, name, nmads=3, type=c("both", "upper", "lower"))
```

Arguments

`x`	A flowFrame object like that constructed by `poolCells`.
`name`	A string specifying the name of the channel in `x` from which intensities are to be extracted.
`nmads`	A numeric scalar specifying the number of median absolute deviations (MADs) beyond which an event can be considered an outlier.
`type`	A string specifying the type of outliers to be removed.

Details

Outliers are defined as events with intensities that are more than `nmads` median absolute deviations from the median of the intensity distribution. The lower gate threshold is defined as the median minus `nmads` MADs, while the upper gate threshold is defined as the median plus `nmads` MADs. If `type="upper"`, only large outliers are removed (e.g., dead/alive stains), so the lower threshold is set to `-Inf`. If `type="lower"`, only small outliers are removed (e.g., DNA), so the upper threshold is set to `Inf`.

Value

A rectangleGate object with lower and upper thresholds defined from `x`.

Author(s)

Aaron Lun

See Also

poolCells, rectangleGate

Examples

```
example(poolCells)
ogate <- outlierGate(ff, "X1")
ogate

ogate <- outlierGate(ff, "X2", type="upper")
ogate

ogate <- outlierGate(ff, "X3", type="lower")
ogate

sff <- Subset(ff, ogate) # for actual gating.
```

---

pickBestMarkers *Pick best markers*

Description

Pick the best markers that distinguish between cells in and outside of a set of hyperspheres.

Usage

```
pickBestMarkers(x, chosen, downsample=10, p=0.05)
```

Arguments

x	A CyData object, constructed using countCells.
chosen	A vector specifying the rows of x corresponding to the hyperspheres of interest.
downsample	A numeric scalar specifying the cell downsampling interval.
p	A numeric scalar defining the quantiles for gating.

Details

A putative subpopulation is defined by a user-supplied set of hyperspheres in chosen. Cells in cellIntensities(x) are downsampled according to downsample. Then, this function identifies all cells in the downsampled set that were counted into any of the hyperspheres specified by chosen at the tolerance tol. We recommend that downsample also be set to the same value as that used in countCells to construct x. (This ensures that the identified cells are consistent with those that were originally counted. It also avoids situations where no cells are counted into hyperspheres for rare subpopulations, which prevents GLM fitting as the response will only have one level.)

Relevant markers are identified by fitting a binomial GLM with LASSO regression to the downsampled cells, using the glmnet function. The response is whether or not the cell was counted into the hyperspheres (and thus, the subpopulation). The covariates are the marker intensities of each cell,

used in a simple additive model with an intercept. Upon fitting, the markers can be ranked from most to least important in terms of their ability to separate counted from uncounted cells. This is done based on the LASSO iteration at which each marker's coefficient becomes non-zero - smaller values indicate more importance, while equal values indicate tied importance. A panel of useful markers can subsequently be constructed by taking the top set from this ranking.

To evaluate the performance of each extra marker, we consider a progressive gating scheme. For each marker, we define the gating boundaries as the interval between the `p` and `1-p` quantiles. For a top set of markers, we calculate the number of cells from the subpopulation that fall inside the gating boundaries for each marker (i.e., true positives). We repeat this for the number of cells not in the subpopulation (false positives). This allows us to compute the recovery (i.e., sensitivity) of the gating scheme as the proportion of true positives out of the total number of cells in the subpopulation; and the contamination (i.e., non-specificity), as the proportion of false positives out of the total number of gated cells.

Value

A data frame is returned, where each row is a marker ordered in terms of decreasing importance. The combined contamination and recovery proportions of the top `n` markers are reported at row `n`, along with the LASSO iteration to denote ties. The lower and upper gating boundaries are also reported for each marker.

Author(s)

Aaron Lun

See Also

`countCells`, `prepareCellData`, `glmnet`

Examples

```
# Mocking up some data with two clear subpopulations.
nmarkers <- 10L
ex1 <- matrix(rgamma(nmarkers*1000, 2, 2), ncol=nmarkers, nrow=1000)
ex2 <- ex1; ex2[,1:4] <- ex2[,1:4] + 1
ex <- rbind(ex1, ex2)
colnames(ex) <- paste0("X", seq_len(nmarkers))
cd <- prepareCellData(list(A=ex))
cnt <- countCells(cd, filter=1L)

# Selecting all hyperspheres from one population.
second.pop <- cellInformation(cnt)$row > nrow(ex1)
selected <- second.pop[getCenterCell(cnt)]
pickBestMarkers(cnt, chosen=selected)
```

plotSphereIntensity *Plot cell or hypersphere data*

Description

Visualize cells or hyperspheres in low-dimensional space, coloured by marker intensities or log-fold changes.

Usage

```
plotSphereIntensity(x, y, intensity, irange=NULL,
    col.range=viridis(100), pch=16, ...)
```

Arguments

x, y	A numeric vector of coordinates for each hypersphere.
intensity	A numeric vector specifying the marker intensities for each hypersphere.
irange	A numeric vector of length 2, specifying the upper and lower bound for the intensities.
col.range	A vector of colours specifying the colour scale to be used for increasing intensity. More values represent a higher-resolution scale.
pch, ...	Additional arguments to pass to plot.

Details

Each hypersphere is represented by a point in the two-dimensional embedding, colored using the viridis colour scheme, i.e., purple (low intensity) to green (medium) to yellow (high). If irange is not NULL, extreme values in intensity will be winsorized to lie within irange. This preserves the resolution of colours for smaller changes at low intensities. Users should consider using intensityRanges to define appropriate values of irange for each marker.

Value

A plot of the low-dimensional embedding of the hypersphere locations is made on the current graphics device. A vector of colours equal to col.range is returned, containing the colour gradient used for the intensities. The vector names contain the numeric values associated with each colour. This can be used to construct a colour bar with createColorBar.

Author(s)

Aaron Lun

See Also

viridis, intensityRanges, createColorBar

Examples

```
# Making up some coordinates.
x <- rnorm(100)
y <- rnorm(100)

# Intensity plot and colour bar.
intensities <- rgamma(100, 2, 2)
out <- plotSphereIntensity(x, y, intensities)

plot(0,0, type="n", axes=FALSE, ylab="", xlab="", ylim=c(-1, 1), xlim=c(-1, 0.5))
createColorBar(out)
text(-0.6, 0, srt=90, "Intensity", cex=1.2)
```

---

plotSphereLogFC *Plot changes in hypersphere abundance*

Description

Visualize hyperspheres in low-dimensional space, coloured by log-fold change in abundance for each hypersphere.

Usage

```
plotSphereLogFC(x, y, logFC, max.logFC=NULL, zero.col="grey80",
    left.col="blue", right.col="red",length.out=100, pch=16, ...)
```

Arguments

x, y	A numeric vector of coordinates for each hypersphere.
logFC	A numeric vector of log-fold changes for each hypersphere.
max.logFC	A numeric scalar specifying the maximum absolute log-fold change.
zero.col	A string specifying the colour to use at a log-fold change of zero.
left.col	A string specifying the colour to use at the most negative log-fold change.
right.col	A string specifying the colour to use at the most positive log-fold change.
length.out	An integer scalar specifying the resolution of the colour bar.
pch, ...	Additional arguments to pass to plot.

Details

Each hypersphere is represented by a point in the two-dimensional embedding, coloured from blue (negative log-FC) to grey (zero log-FC) to red (positive log-FC). The darkness of the grey colour is set with zero.col.

If max.logFC is not NULL, extreme values in logFC are winsorized to lie within [-max.logFC, max.logFC]. This preserves the resolution of colours for smaller log-fold changes.

Value

A plot of the low-dimensional embedding of the hypersphere locations is made on the current graphics device. A vector of colours of length `length.out` is returned, containing the colour gradient used for the log-fold changes. The vector name contains the numeric values associated with each colour. This can be used to construct a colour bar with `createColorBar`.

Author(s)

Aaron Lun

See Also

`createColorBar`

Examples

```
# Making up some coordinates.
x <- rnorm(100)
y <- rnorm(100)

# Log-FC plot and colour bar.
logFC <- rnorm(100)
out <- plotSphereLogFC(x, y, logFC)

out <- plotSphereLogFC(x, y, logFC, max.logFC=0.5)
plot(0,0, type="n", axes=FALSE, ylab="", xlab="", ylim=c(-1, 1), xlim=c(-1, 0.5))
createColorBar(out)
text(-0.6, 0, srt=90, "Log-FC", cex=1.2)
```

---

poolCells *Pool cells for pre-processing*

Description

Construct a flowFrame object by pooling cells from multiple (barcoded) samples, for use in common transformation and gating.

Usage

```
poolCells(x, equalize=TRUE, n=NULL)
```

Arguments

`x`	A named list of numeric matrices, where each matrix corresponds to a sample and contains expression intensities for each cell (row) and each marker (column). Alternatively, a ncdfFlowSet object containing the same information.
`equalize`	A logical scalar specifying whether the same number of cells should be taken from each sample for pooling. If `FALSE`, all cells are used from all samples.

n — A numeric scalar specifying the number of cells to be used from each sample if equalize=TRUE. If NULL, this is set to the number of cells in the smallest sample.

Details

The idea is to use the pooled set of cells to estimate common parameters such as transformation values and gating thresholds. Otherwise, if these parameters were estimated separately for each sample, they may distort the comparisons between samples. This function is typically used to generate an object for use in estimateLogicle or in various gating functions like outlierGate. This yields parameter values that can be applied to the full set of cells in the original x object.

Value

A flowFrame object containing cells pooled from all samples.

Author(s)

Aaron Lun

See Also

flowFrame, outlierGate, estimateLogicle

Examples

```
### Mocking up some data: ###
set.seed(100)
nmarkers <- 40
marker.names <- paste0("X", seq_len(nmarkers))
nsamples <- 10
sample.names <- paste0("Y", seq_len(nsamples))

x <- list()
for (i in sample.names) {
    ex <- matrix(rexp(nmarkers*1000, 0.01), ncol=nmarkers, nrow=1000)
    colnames(ex) <- marker.names
    x[[i]] <- ex
}

### Running the function: ###
ff <- poolCells(x)
ff

### Using for estimation: ###
library(flowCore)
trans <- estimateLogicle(ff, colnames(ff))
ff <- transform(ff, trans) # or, apply to original data.
```

prepareCellData *Prepare mass cytometry data*

Description

Convert single-cell marker intensities from a mass cytometry experiment into a format for efficient counting.

Usage

```
prepareCellData(x, markers = NULL, ...)
```

Arguments

x	A named list of numeric matrices, where each matrix corresponds to a sample and contains expression intensities for each cell (row) and each marker (column). Alternatively, a ncdfFlowSet object containing the same information.
markers	A character vector containing the names of the markers to use in downstream analyses.
...	Additional arguments to pass to buildIndex.

Details

This function constructs a neighbor search index from the marker intensities of each cell in one or more samples. The precomputed index is used to speed up downstream nearest-neighbour searching, avoiding redundant work from repeated calls to countCells (e.g., with different values of tol).

If markers is specified, only the selected markers will be used in the precomputation. This restricts the markers that are used in downstream functions - namely, countCells and neighborDistances. By default, markers=NULL which means that all supplied markers will be used.

Markers that are *not* in markers will be ignored in distance calculations. However, their intensities are still stored in the output object, for use in functions like medIntensities.

Value

A List containing precomputed values for use in countCells. This includes:

- precomputed, a prebuilt index for the neighbor search.
- sample.id, an integer vector specifying the sample of origin for each cell in precomputed.
- cell.id, an integer vector specifying the original index in the corresponding sample of x for each cell in precomputed.
- unused, a matrix of intensity values for markers *not* in markers.
- colData, a DataFrame containing per-sample statistics.

Author(s)

Aaron Lun

See Also

countCells, where the output of this function is used to obtain hypersphere counts.

Examples

```
### Mocking up some data: ###
nmarkers <- 20
marker.names <- paste0("X", seq_len(nmarkers))
nsamples <- 8
sample.names <- paste0("Y", seq_len(nsamples))

x <- list()
for (i in sample.names) {
    ex <- matrix(rgamma(nmarkers*1000, 2, 2), ncol=nmarkers, nrow=1000)
    colnames(ex) <- marker.names
    x[[i]] <- ex
}

### Running the function: ###
cd <- prepareCellData(x)
cd
```

---

spatialFDR *Compute the spatial FDR*

Description

Computed adjusted p-values for all hyperspheres, using a density-weighted version of the Benjamini-Hochberg method.

Usage

```
spatialFDR(x, pvalues, neighbors=50, bandwidth=NULL, num.threads=1)
```

Arguments

x	A numeric matrix of hypersphere coordinates, containing the median intensity of each marker (column) in each hypersphere (row). Alternatively, a CyData object containing median intensities for groups of cells, such as that produced by countCells.
pvalues	A numeric vector of p-values for each hypersphere.
neighbors	An integer scalar specifying the number of neighbors with which to compute the bandwidth.

bandwidth	A numeric scalar specifying the bandwidth for density estimation.
num.threads	Integer scalar specifying the number of threads to use.

Details

Consider the set of significant hyperspheres, distributed in some manner across the M-dimensional space (for M markers). The aim is to control the FDR across the subspaces containing significant hyperspheres. This is subtly different from controlling the FDR across the hypersphere themselves, which will skew the results for densely occupied subspaces.

Control of the spatial FDR is achieved by weighting the hyperspheres inversely proportional to their local densities. This downweights hyperspheres in dense subspaces while upweighting hyperspheres in sparse subspaces. The computed weights are then used as frequency weights in the Benjamini-Hochberg method, to control the FDR across subspaces.

The local density is calculated using a tricube kernel and the specified `bandwidth`. If unspecified, `bandwidth` is set to the median of the distances to the `neighbors`-closest neighbor for all hyperspheres. This usually provides stable density estimates while maintaining sensitivity to fine-scale structure.

Value

A numeric vector of adjusted p-values for all hyperspheres.

Author(s)

Aaron Lun

References

Lun ATL, Richard AC, Marioni JC (2017). Testing for differential abundance in mass cytometry data. *Nat. Methods*, 14, 7:707-709.

Examples

```
coords <- matrix(rgamma(10000, 2, 2), nrow=1000)
pvalues <- rbeta(nrow(coords), 1, 2)
out <- spatialFDR(coords, pvalues)
```

Index

[,CyData,ANY,ANY,ANY-method *(CyData)*, 5
[<-,CyData,ANY,ANY,CyData-method *(CyData)*, 5

BiocParallelParam, *3*
buildIndex, *33*

cbind,CyData-method *(CyData)*, 5
cellAssignments, *3*
cellAssignments *(CyData)*, 5
cellInformation *(CyData)*, 5
cellIntensities *(CyData)*, 5
countCells, 2, *5*, *9–11*, *13*, *16–18*, *21*, *27, 28*, *33, 34*
createColorBar, 4, *29*, *31*
CyData, *3*, 5
CyData-class *(CyData)*, 5

DataFrame, *6*, *33*
density, *7, 8*, *14, 15*
dnaGate, 7

estimateLogicle, *32*
expandRadius, 9

findFirstSphere, 10
findNeighbors, *3*
flowFrame, *32*

getCenterCell, *4*
getCenterCell *(CyData)*, 5
glmnet, *27, 28*

int_metadata, *4*
intensities, *3*
intensities *(CyData)*, 5
intensityRanges, 12, *13*, *15*, *29*
interpretSpheres, 13, *17*

labelSpheres, *15*, 16
List, *3*, *9*, *21*, *33*

markernames *(CyData)*, 5
markernames,CyData-method *(CyData)*, 5
medIntensities, 17, *33*
multiIntHist, 19, *25*

neighborDistances, 20, *33*
normalization, *24, 25*
normalizeBatch, *20*, 22

outlierGate, 26, *32*

pickBestMarkers, 27
plotSphereIntensity, *12*, 29
plotSphereLogFC, 30
polygonGate, *8*
poolCells, *7, 8*, *26, 27*, 31
prepareCellData, *3–6*, *9, 10*, *12*, *18*, *21–23*, *25*, *28*, 33

rectangleGate, *27*
runApp, *14, 15*

show,CyData-method *(CyData)*, 5
SingleCellExperiment, *5, 6*
spatialFDR, 34

unlist, *23*

viridis, *14*, *29*

warpSet, *23–25*