Package 'cydar'

April 11, 2018

Version 1.2.1

Date 2017-11-08

Title Using Mass Cytometry for Differential Abundance Analyses

Author Aaron Lun <alun@wehi.edu.au>

Maintainer Aaron Lun <alun@wehi.edu.au>

Depends BiocParallel, SummarizedExperiment

Imports viridis, methods, shiny, graphics, stats, grDevices, S4Vectors, flowCore, Biobase, Rcpp

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

biocViews 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

R topics documented:

`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(x, tol=0.5, BPPARAM=SerialParam(), downsample=10, filter=10, naive=FALSE)
```

Arguments

`x`	A CyData object produced by `prepareCellData`.
`tol`	A numeric scalar proportional to the hypersphere radius.
`BPPARAM`	A BiocParallelParam object specifying how parallelization is to be performed.
`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.
`naive`	A logical scalar specifying whether a naive counting approach should be used.

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. This 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.

The default setting of `naive=FALSE` will use a method similar to that described by Samusik et al. (2016) to speed up the search. Here, the distance between the hypersphere and cluster centers is first computed to decide which, if any, cells in the cluster should be considered for counting. If `naive=TRUE`, a slower naive approach will be used where distances are computed between all pairs of cells. This is generally only useful for testing.

If `markerData(x)$used` is not all `TRUE`, only the specified markers will be used in the distance calculations. Median coordinates will not be calculated for any discarded markers, instead being reported as `NA` in the utput `intensities`. Note that the `used` field should not be set directly – `prepareCellData` must be run again to change the set of used markers.

Users can also increase speed by setting `BPPARAM` to use multiple cores. Neither this or `naive` will not affect the final results, only the speed with which they are obtained.

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), stored in the `intensities` slot.

`cellAssignments`: A list of integer vectors specifying the cells belonging in each group. This is stored in a compressed format, see `packIndices` for details.

`sample.id`: An integer vector containing an integer ID for each sample. This corresponds to values in `cellData(x)$sample.id`, to ensure that they can be properly interpreted regardless of column subsetting or combining.

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

`tol`: An integer scalar equal to `tol`, stored in the `metadata` slot.

`center.cell`: An integer vector specifying the column of `cellIntensities` containing the centre of each hypersphere, stored as a field in the `rowData` slot.

Any existing `rowData`, `intensities` and `assays` are discarded. All other slots are left unchanged from the values supplied in `x`.

Author(s)

Aaron Lun

References

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, packIndices

Examples

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

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

CyData-class *CyData class and methods*

Description

An overview of the CyData class and applicable methods.

Usage

```
CyData(markerData, intensities=NULL, cellAssignments=NULL,
    cellIntensities=NULL, cellData=NULL, assays=NULL, ...)
```

Arguments

markerData	A DataFrame where each row corresponds to a marker, and is named according to that marker.
intensities	A numeric matrix of median intensities for each group of cells (row) and each marker (column).
cellAssignments	A list of integer vectors specifying which cells are assigned to each group.
cellIntensities	A numeric matrix of intensities for each marker (row) and in each cell (column).
cellData	A DataFrame containing information about each cell in each row.
assays, ...	Arguments to be passed to the SummarizedExperiment constructor.

Details

The CyData class is designed to store the cell counts for each group of cells (e.g., hyperspheres, clusters), along with the median intensities for each group. It inherits from the SummarizedExperiment class and contains the additional slots:

markerData A DataFrame containing information about each marker in each row. The row names are used as the marker names.

intensities A numeric matrix where each row corresponds to a group of cells and each column corresponds to a marker. Each matrix entry contains the average (usually median) intensity across all cells in a group for a particular marker.

cellAssignments A list of integer vectors containing column indices of `cellIntensities`. Each vector corresponds to a group and specifies the cells that were assigned to that group.

cellIntensities A numeric matrix containing intensities for each marker in each individual cell in the experiment. Each column represents a cell while each row represents a marker. This column-major setup is more amenable to fast processing later.

cellData A DataFrame containing information about each cell in each row.

The above constructor will set `intensities` and `assays` to a matrix with no rows, if not specified. It will also set `cellIntensities` to a matrix with no columns and `cellData` to an empty list by default.

Value

A CyData object containing the specified information.

Author(s)

Aaron Lun

Examples

```
# A minimal example.
my.markers <- DataFrame(row.names=LETTERS)
cyd.minimal <- CyData(markerData=my.markers)

# Adding extra detail.
counts <- matrix(rpois(1000, 10), ncol=10)
medians <- matrix(rgamma(1000, 1, 1), ncol=10)
cell.int <- matrix(rgamma(10000, 1, 1), nrow=10)
marker.data <- DataFrame(row.names=LETTERS[1:10])
cell.data <- DataFrame(sample.id=sample(10, 1000, replace=TRUE))
cyd <- CyData(assay=counts, markerData=marker.data, cellData=cell.data,
    intensities=medians, cellIntensities=cell.int)
```

CyData-getset *CyData getters and setters*

Description

Methods to get and set data slots in the CyData object.

Usage

```
## S4 method for signature 'CyData'
markerData(x)
## S4 replacement method for signature 'CyData'
markerData(x) <- value

## S4 method for signature 'CyData'
```

```
intensities(x)
## S4 replacement method for signature 'CyData'
intensities(x) <- value

## S4 method for signature 'CyData'
cellAssignments(x)
## S4 replacement method for signature 'CyData'
cellAssignments(x) <- value

## S4 method for signature 'CyData'
cellIntensities(x)
## S4 replacement method for signature 'CyData'
cellIntensities(x) <- value

## S4 method for signature 'CyData'
cellData(x)
## S4 replacement method for signature 'CyData'
cellData(x) <- value

## S4 method for signature 'CyData'
nmarkers(x)
## S4 method for signature 'CyData'
ncells(x)

## S4 method for signature 'CyData'
markernames(object)
## S4 replacement method for signature 'CyData'
markernames(object) <- value

## S4 method for signature 'CyData'
sampleNames(object)
## S4 replacement method for signature 'CyData'
sampleNames(object) <- value
```

Arguments

`x, object`	A CyData object.
`value`	For `markernames` and `sampleNames`, a character vector specifying the names of the markers and samples, respectively. Otherwise, an appropriate object to use to replace the corresponding slot of `x`.

Details

`markerData`, `intensities`, `cellAssignments`, `cellIntensities` and `cellData` are methods to get or set the slots in a CyData object. general, setting should not be performed unless adding annotation for particular markers or cells (or if you really know what you're doing). Caution is especially advised when using `cellIntensities<-` and/or `cellData<-`, as several methods in this package rely on a particular ordering of cells.

Several convenience functions are also provided. `nmarkers` will return the number of markers, while `ncells` will return the number of cells. `markernames` will return the marker names (i.e., the row names of `markerData`), while `sampleNames` will return the sample names (i.e., the column names of `object`). The respective replacement functions will modify these names.

Value

For the setters, a new CyData object will be generated with modified entries for the appropriate slots. For the getters, an object of the class corresponding to each slot will be returned. `nmarkers` and `ncells` will return integer scalars, while `markerNames` and `sampleNames` will return character vectors.

Author(s)

Aaron Lun

Examples

```
example(CyData) # Mocking up an object.

nmarkers(cyd)
markernames(cyd)
sampleNames(cyd)

markerData(cyd)
markerData(cyd)$stuff <- runif(nmarkers(cyd))
markerData(cyd)

head(intensities(cyd))
cellIntensities(cyd)[,1:10]
```

CyData-subset *Subsetting and combining CyData*

Description

Methods to subset and combine objects of the CyData class.

Usage

```
## S4 method for signature 'CyData,ANY,ANY'
x[i, j, ..., drop=TRUE]
## S4 replacement method for signature 'CyData,ANY,ANY,CyData'
x[i, j] <- value
## S4 method for signature 'CyData'
subset(x, i, j)

## S4 method for signature 'CyData'
rbind(..., deparse.level=1)
## S4 method for signature 'CyData'
c(x, ..., recursive=FALSE)
## S4 method for signature 'CyData'
cbind(..., deparse.level=1)
```

Arguments

x	A CyData object.
i, j	A vector of subscripts, indicating the rows and columns to be subsetted for i and j, respectively. Rows correspond to cell-groups while columns correspond to samples.
...	For cbind, ... contains CyData objects to be combined column-wise. For rbind and c, ... contains CyData objects to be combined row-wise. For c, any objects are additional to that already specified in x. For other methods, this argument is ignored.
drop	A logical scalar, ignored.
value	A CyData object to replace the entries in the specified i and j.
deparse.level, recursive	Addition arguments, ignored.

Details

Subsetting of CyData objects consider groups of cells (e.g., clusters, hyperspheres) as the rows, and samples as the columns. Only the rows of intensities are affected during row-wise subsetting. Marker and cell information in markerData, cellData and cellIntensities is not modified.

A similar principle applies when combining different CyData objects, i.e., rows are groups of cells and columns are samples. Values of markerData and cellIntensities should be identical in the objects to be combined. Furthermore, for cbind, values of intensities should also be identical. Note that c is a synonym for rbind.

When replacing entries with [<-, markerData, cellData and cellIntensities should be identical in x and value. Column replacement also requires intensities and cellAssignments to be the same. Simultaneous row and column replacement is not allowed as this will introduce inconsistencies into the sample ID annotation.

Value

A CyData object with different rows or columns, depending on whether subsetting or merging was performed.

Author(s)

Aaron Lun

Examples

```
example(CyData) # Mocking up an object.

cyd[1:5,]
cyd[,6:10]
rbind(cyd, cyd)
cbind(cyd, cyd)
```

`diffIntDistr` *Compute differences in intensity distributions*

Description

Calculate the difference in the intensity distributions between samples across multiple batches.

Usage

```
diffIntDistr(..., markers=NULL, npts=200)
```

Arguments

`...`	One or more lists of intensity matrices or ncdfFlowSet objects, like `x` in ?`prepareCellData`. These can be named, otherwise default names will be assigned.
`markers`	A character vector specifying which markers should have their differences computed. All markers that are present in all objects will be used by default.
`npts`	An integer scalar specifying the number of points for computing differences.

Details

The difference between two intensity distributions is defined as half of the area between their probability density functions. For each intensity distribution, the density function is approximated by binning events into a grid of length `npts`. The absolute difference in the proportion of events is computed for each bin; summed across all bins; multiplied by the width of each bin; and halved. The final value ranges from 0 to 1 and represents the proportion of probability mass that differs between the distributions.

The differences for each marker are returned in the form of a symmetric matrix. Each row/column is a sample, named as a combination of the batch and sample names. Samples are ordered according to the specified order of batches in `...` and the order of samples in each batch. Each entry of the matrix contains the difference between the corresponding samples.

If multiple objects are specified in `...`, they are assumed to represent different batches of samples. The differences between samples in the same or different batches are useful for evaluating the effects of inter-batch normalization, as shown in the example below. If normalization was successful, differences between batches should be similar to the differences within batches. However, this diagnostic is only applicable if the batches are not confounded by biological factors - otherwise, if such factors were present, inter-batch differences *should* be larger.

Value

A list is returned with two objects:

`index`**:** An integer matrix specifying which entries of each matrix correspond to differences between samples in the same batch. These are numbered according to the specified order of the batches in `...`. Between-batch differences have values of 0.

`difference`**:** A named list of symmetric numeric matrices, with one for each marker. Each row and column represents a sample and is named accordingly. Each entry of the matrix contains the difference between intensity distributions for the two samples corresponding to the row and column.

Author(s)

Aaron Lun

See Also

normalizeBatch

Examples

```
example(normalizeBatch)

original <- do.call(diffIntDistr, all.x)
par(mfrow=c(5,4), mar=c(2.1, 2.1, 2.1, 1.1))
for (m in names(original$difference)) {
    current <- original$difference[[m]] * 100
    is.inter <- original$index==0L
    is.intra <- !is.inter & lower.tri(original$index)
    boxplot(list(Inter=current[is.inter], Intra=current[is.intra]), main=m)
}

par(mfrow=c(5,4), mar=c(2.1, 2.1, 2.1, 1.1))
fixed <- do.call(diffIntDistr, corrected)
for (m in names(fixed$difference)) {
    current <- fixed$difference[[m]] * 100
    is.inter <- fixed$index==0L
    is.intra <- !is.inter & lower.tri(fixed$index)
    boxplot(list(Inter=current[is.inter], Intra=current[is.intra]), main=m)
}
```

---

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(x, design=NULL, markers=NULL, tol=0.5)
```

Arguments

`x`	A CyData object produced by `prepareCellData`.
`design`	A numeric matrix specifying the experimental design.
`markers`	A vector specifying the markers to use.
`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.

If `markers` is specified, only the indicated markers are used to calculate the mean variance. Otherwise, the markers to use are extracted from `markerData(x)$used`, which is set by `prepareCellData`. By default, all markers are used unless otherwise requested.

Value

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

Author(s)

Aaron Lun

See Also

`prepareCellData`, `countCells`

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(coords, pvalues, threshold=1, block=NULL, naive=FALSE)
```

Arguments

coords	A numeric matrix of hypersphere coordinates (median locations for all markers), where rows correspond to hyperspheres and columns correspond to markers.
pvalues	A numeric vector of p-values, one for each hypersphere/row of coords.
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.
naive	A logical scalar specifying whether naive counting should be performed.

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.

Note that setting naive=TRUE will only change the speed of the algorithm, not the results.

Value

A logical vector indicating whether each of the hyperspheres in coords 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 `plotCellIntensity`. 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. In short, 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).

Author(s)

Aaron Lun

See Also

`prepareCellData`, `plotCellIntensity`

Examples

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

# Plotting example (using a subset for speed).
cd.subset <- t(cellIntensities(cd)[,1:1000])
coords <- prcomp(cd.subset)
chosen.marker <- 5
```

```
plotCellIntensity(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(coords, labels, naive=FALSE)
```

Arguments

`coords`	A numeric matrix of hypersphere coordinates, containing the median intensity of each marker (column) in each hypersphere (row).
`labels`	A character vector of labels for each hypersphere, set to an empty string for unannotated hyperspheres.
`naive`	A logical scalar specifying whether a naive search should be performed.

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)
naive <- labelSpheres(coords, labels, naive=TRUE)
```

`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`. If the same markers were 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 the markers used.

The idea is to use these values 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 recount cells into new groups using recountCells to focus on each secondary marker. This provides more power to detect changes in marker intensity that only affect a subset of cells in each group. Such changes are also easier to interpret as any correlation with respect to the primary markers for the affected subset can be studied.

The second approach is also more useful if one is interested in identifying cells with concomitant changes in multiple secondary markers. Indeed, if we were interested in studying changes in all combinations of second markers, we would effectively revert to the obvious approach of just using all markers for counting. 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

countCells, recountCells

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(x, neighbors=50, downsample=50, as.tol=TRUE, naive=FALSE)
```

Arguments

x	A CyData 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.
naive	A logical scalar specifying whether a naive counting approach should be used.

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. If markerData(x)$used is not all TRUE, only the specified markers will be used to calculate the distance.

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, as a greater number of neighbors is not considered.

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. Similarly, setting naive=TRUE will only affect speed and will not change the results.

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

countCells

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,
    target=NULL, markers=NULL, ...)
```

Arguments

batch.x	A list, where each element is of the same type as x used in prepareCellData (i.e., a ncdfFlowSet or a list of intensity matrices across all samples).
batch.comp	A list of factors (or elements coercible to factors) 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" 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.
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. 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.
2. If mode="range", a quantile function is constructed for the pooled distribution of each batch. These functions are averaged across batches to obtain a reference quantile function, representing a reference distribution. The range of the reference distribution is computed at percentiles p and 1-p (to avoid distortions due to outliers). A batch-specific scaling function is defined to equalize the range of the weighted distribution of intensities from each batch to the reference range.
3. If mode="warp", weighted sampling from each pooled distribution is performed to generate a pseudo-sample for 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 adjust the intensity distribution to be more similar to the reference (constructed by averaging across batches).
4. The scaling or warping function is applied to the intensities of all samples in that batch, yielding corrected intensities for direct comparisons between samples.

Groupings can be specified as batch-specific factors in batch.comp, with at least one common group required across all batches. If the composition of each batch is the same, batch.comp can be set to NULL rather than being manually specified. 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.

Construction of the adjustment function relies on the presence of samples from the same group across the different batches. Ideally, all batches would contain samples from all groups, with similar total numbers of cells across batches for each group. The adjustment function will still be applied to intensities for samples from non-shared groups that do not contribute to the reference distribution. 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.

By default, the reference distribution for each marker is defined as an average of the relevant statistic across batches. If target is not NULL, the specified batch will be used as the reference distribution. This means that if mode="range", 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`.

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`.

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.

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`. See also `diffIntDistr` for a quantitative measure of similarity between distributions.

Author(s)

Aaron Lun

See Also

`prepareCellData`, `diffIntDistr`, `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")

# Similar effects with warping normalization.
if (.Platform$OS.type!="windows") {
    wcorrected <- normalizeBatch(all.x, batch.comp, mode="warp")
}
```

`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.
```

---

packIndices *Pack or unpack indices*

Description

Compress or decompress a vector of indices, usually specifying cell assignments to groups.

Usage

```
packIndices(assignments)
unpackIndices(assignments)
```

Arguments

assignments	A list of integer vectors containing indices that specify cell assignments to groups. Each vector corresponds to a group and should be uncompressed for packIndices, or already compressed for unpackIndices.

Details

Indices are stored in a compressed format whereby a negative number indicates that all consecutive integers from the preceding index should be used. For example, a sequence of `c(1, 3, -6, 8, -10, 12)` would be unpacked as `c(1, 3:6, 8:10, 12)`. This saves a lot of memory for storing cell assignments in hyperspheres, where there are likely to be many consecutive indices.

The `countCells` function will automatically fill in the `cellAssignments` slot of the output object with compressed index vectors. The `unpackIndices` function can be used to construct the full vector from some or all of these vectors, for manual use elsewhere. Conversely, users manually constructing CyData objects can use `packIndices` to compress the vectors and save space.

Value

A list of integer vectors containing compressed or uncompressed index vectors, for `packIndices` and `unpackIndices` respectively.

Author(s)

Aaron Lun

See Also

`countCells`

Examples

```
a <- c(1L, 3:6, 8:10, 12L)
packIndices(list(a))

b <- c(1L, 3L, -6L, 8L, -10L, 12L)
unpackIndices(list(b))
```

`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, markers=NULL, downsample=10, p=0.05, naive=FALSE)
```

Arguments

`x`	A CyData object, constructed using `countCells`.
`chosen`	A vector specifying the rows of `x` corresponding to the hyperspheres of interest.
`markers`	A vector specifying the markers to use in the LASSO regression.
`downsample`	A numeric scalar specifying the cell downsampling interval.
`p`	A numeric scalar defining the quantiles for gating.
`naive`	A logical scalar specifying whether a naive search should be used.

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.

If `markers` is specified, only the supplied markers are used in the regression. Otherwise, the markers to use are extracted from `markerData(x)$used`, which is set by `prepareCellData`. By default, all markers are used unless otherwise requested.

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 hyperspheres centred on cells in the first mocked-up subpopulation.
selected <- cellData(cnt)$cell.id[rowData(cnt)$center.cell] > 1000
pickBestMarkers(cnt, selected)
```

`Plot Cells` *Plot cell or hypersphere data*

Description

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

Usage

```
plotCellLogFC(x, y, logFC, max.logFC=NULL, zero.col=0.8,
    length.out=100, pch=16, ...)

plotCellIntensity(x, y, intensity, irange=NULL,
    length.out=100, pch=16, ...)
```

Arguments

`x, y`	A numeric vector of coordinates for each feature (i.e., cell or hypersphere).
`logFC`	A numeric vector of log-fold changes for each feature.
`max.logFC`	A numeric scalar specifying the maximum absolute log-fold change.
`zero.col`	A numeric scalar between 0 and 1, specifying the greyscale intensity to represent a log-fold change of zero.
`intensity`	A numeric vector specifying the marker intensities for each feature.
`irange`	A numeric vector of length 2, specifying the upper and lower bound for the intensities.
`length.out`	An integer scalar specifying the resolution of the colour bar.
`pch, ...`	Additional arguments to pass to `plot`.

Details

`plotCellLogFC` will colour the points 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 for smaller log-fold changes.

`plotCellIntensity` will colour the points 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`. Like before, this preserves the resolution for smaller changes in intensity. Users should consider using `intensityRanges` to define appropriate values of `irange` for each marker.

Value

A vector of colours of length `length.out` is returned, containing the colour gradient used for plotting. The vector names contains the numeric values associated with each colour. This can be used to construct a colour bar.

Author(s)

Aaron Lun

See Also

viridis, intensityRanges

Examples

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

# Log-FC plot and colour bar.
logFC <- rnorm(100)
out <- plotCellLogFC(x, y, logFC)
out <- plotCellLogFC(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))
start.loc <- seq(-0.5, 0.5, length.out=length(out))
rect(-0.5, start.loc, 0.5, start.loc+diff(start.loc)[1], col=out, border=NA)
text(0, -0.5, pos=1, names(out)[1], cex=1.2)
text(0, 0.5,  pos=3, names(out)[length(out)], cex=1.2)
text(-0.6, 0, srt=90, "Log-FC", cex=1.2)

# Intensity plot and colour bar.
intensities <- rgamma(100, 2, 2)
out <- plotCellIntensity(x, y, intensities)
out <- plotCellIntensity(x, y, intensities, irange=c(0, 2))

plot(0,0, type="n", axes=FALSE, ylab="", xlab="", ylim=c(-1, 1), xlim=c(-1, 0.5))
start.loc <- seq(-0.5, 0.5, length.out=length(out))
rect(-0.5, start.loc, 0.5, start.loc+diff(start.loc)[1], col=out, border=NA)
text(0, -0.5, pos=1, names(out)[1], cex=1.2)
text(0, 0.5,  pos=3, names(out)[length(out)], cex=1.2)
text(-0.6, 0, srt=90, "Intensity", 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, naive=FALSE, 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.
`naive`	A logical scalar specifying whether k-means clustering should be performed.
`markers`	A vector specifying the markers to use in distance calculations.
`...`	Additional arguments to pass to `kmeans`.

Details

This function constructs a CyData object from the marker intensities of each cell in one or more samples.

If `naive=FALSE`, this function performs k-means clustering on all the cells based on their marker intensities. The number of clusters is set to the square-root of the total number of cells. The cluster centres and cell assignments are then stored for later use in speeding up high-dimensional searches. Intensity matrices from several samples are also merged into a single matrix for greater efficiency.

Note that `naive` does *not* change the results of downstream functions, only the computational algorithm with which they are obtained.

If `markers` is specified, only the specified markers will be used in the distance calculations. This also applies to calculations in downstream functions like `countCells` and `neighborDistances`. All other markers will be ignored unless their usage is explicitly requested. By default, `markers=NULL` which means that all supplied markers will be used in the calculations.

Value

A CyData object with marker intensities for each cell stored in the `cellIntensities` slot. In addition:

- Marker names are stored as the row names of the `markerData` slot. An additional `used` field also specifies whether the marker was used or not.
- Sample names are stored as the column names of the output object.
- `cellData` contains `sample.id`, an integer vector specifying the element of `x` that each cell was taken from; and `cell.id`, an integer vector specifying the row index of each cell in its original matrix.

- If `naive=FALSE`, the metadata contains `cluster.centers`, a numeric matrix containing the centre coordinates of each cluster (column) for each marker (row); and `cluster.info`, a list containing information for each cluster.

Each element of `cluster.info` is a list, containing the zero-indexed column index of the output matrix that specifies the first cell in the cluster; as well as a numeric vector of distances between each cell in the cluster and the cluster centre. Cells in `cellIntensities` are arranged in blocks corresponding to the clusters and ordered such that the distances are increasing.

Author(s)

Aaron Lun

See Also

`countCells`, `neighborDistances`

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
```

---

recountCells *Recount cells in each group*

Description

Count the number of cells in hyperspheres across a specified marker space, nested within pre-defined groups of cells.

Usage

```
recountCells(x, markers, tol=0.5)
```

Arguments

`x`	A CyData object containing cell assignments into hyperspheres, such as that produced by `countCells`.
`markers`	A vector specifying the markers for which hyperspheres should be constructed.
`tol`	A numeric scalar proportional to the hypersphere radius.

Details

Each row of x corresponds to an existing hypersphere across some high-dimensional space, to which a set of cells are assigned. This function extends the hypersphere into the dimensions specified by markers. Thus, each new hypersphere is "nested" within the existing hypersphere in x. Only the cells in the latter are assigned into the former (though obviously, some cells will not be assigned if they are too distant from the centre in the new dimensions).

This function allows for fast recounting in situations where the markers have different purposes. For example, x could be constructed using cell type-specific markers to define cell types. recountCells can then be applied with markers that define, e.g., the activation status within each cell type. In general, it is most interesting to use markers that were *not* used to construct x. Otherwise, by definition, all the cells in each hypersphere would have similar marker intensities.

Also see medIntensities for a discussion of strategies to use when markers can be separated into two distinct sets.

Value

A CyData object containing counts and cell assignments for nested hyperspheres. This follows the same format as the output from countCells, i.e., each row is a hypersphere and each column is a sample. Some fields are modified:

- counts, intensities and cellAssignments contain the relevant values for the nested hyperspheres.
- The used field in markerData is set to all markers used in both the original and re-counting.
- The tol value in the metadata is set to the (effective) tolerance used in re-counting.

Note on the radius calculation

The output of this function is designed to be equivalent to directly running countCells with both new and old markers. However, to speed up the counting, only the cells already assigned to each hypersphere in x are considered for re-counting. This has some consequences for the results, as the radius scales with respect to tol and the number of markers.

Specifically, when new markers are specified in markers, the radius must increase to accommodate the increase in dimensions. However, the cells were originally counted with a radius proportional to the (square root of the) old number of markers and metadata(x)$tol. If the radius now increases, but only pre-assigned cells are used for re-counting, then there will be cells that are missed in the re-counts.

Thus, to preserve equivalence with countCells output, tol is decreased so that the radius does not change with new markers. This shows up as a warning specifying the effective tolerance that was used in during re-counting. Users can also avoid this problem by using a higher radius when constructing x, such that the radius calculated from the tol here will be smaller.

Author(s)

Aaron Lun

See Also

countCells, medIntensities

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
}

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

### Processing it afterwards with another set of markers: ###
rcnt <- recountCells(cnt, markers=marker.names[11:12])
rcnt
```

`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(coords, pvalues, neighbors=50, bandwidth=NULL, naive=FALSE)
```

Arguments

`coords`	A numeric matrix of hypersphere coordinates, containing the median intensity of each marker (column) in each hypersphere (row).
`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.
`naive`	A logical scalar specifying whether a naive search should be performed.

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.

Setting `naive=TRUE` will perform a naive search for nearest neighbors, rather than the more efficient convex method. However, this should only affect computational efficiency and should not change the final results.

Value

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

Author(s)

Aaron Lun

Examples

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

Index