Package 'flowSpy'

October 17, 2020

Type Package

Title A Toolkit for Flow And Mass Cytometry Data

Version 1.2.3

Date 2020-07-09

Description A trajectory inference and visualization toolkit for flow and mass cytometry data. flowSpy offers complete analyzing workflow for flow and mass cytometry data. flowSpy can be a valuable tool for application ranging from clustering and dimensionality reduction to trajectory reconstruction and pseudotime estimation for flow and mass cytometry data.

Depends R (>= 3.6), igraph

Imports FlowSOM, Rtsne, ggplot2, destiny, gmodels, flowUtils, Biobase, Matrix, flowCore, sva, matrixStats, methods, mclust, prettydoc, RANN(>= 2.5), Rcpp (>= 0.12.0), BiocNeighbors, cluster, pheatmap, scatterpie, umap, scatterplot3d, limma, stringr, grDevices, grid, stats

Suggests BiocGenerics, knitr, RColorBrewer, rmarkdown, testthat, BiocStyle

biocViews CellBiology, Clustering, Visualization, Software, CellBasedAssays, FlowCytometry, NetworkInference, Network

VignetteBuilder knitr

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

BugReports https://github.com/JhuangLab/flowSpy/issues

URL http://www.r-project.org, https://github.com/JhuangLab/flowSpy

LinkingTo Rcpp

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

git_branch RELEASE_3_11

git_last_commit a24c324

git_last_commit_date 2020-07-09

Date/Publication 2020-10-16

Author Yuting Dai [aut, cre]

Maintainer Yuting Dai <forlynna@sjtu.edu.cn>

R topics documented:

`flowSpy-package` *Visualization and analyzation for flow cytometry data*

Description

Functions and methods to visualize and analyze flow cytometry data.

Details

Package:	flowSpy
Type:	Package
Version:	1.2.3
Date:	2020-07-09
License:	GPL-3.0

While high-dimensional single-cell based flow and mass cytometry data has demonstrated increased applications in microenvironment composition and stem-cell research, integrated analyzing workflow design for experimental cytometry data has been challenging. Here, we present flowSpy, an R package designed for the analysis and interpretation of flow and mass cytometry data. We have applied flowSpy to mass cytometry and time course flow cytometry data to validate the usage and practical utility of its computational modules. These use cases introduce flowSpy as a reliable tool for high-dimensional cytometry data workflow and reveal good performance on trajectory reconstruction and pseudotime estimation.

Author(s)

Maintainer: Yuting Dai <forlynna@sjtu.edu.cn> Authors: Yuting Dai

Examples

```
if (FALSE) {
## examples go here
## See vignette tutorials
vignette(package = "flowSpy")
vignette("Quick_start", package = "flowSpy")
}
```

`buildTree` *buildTree*

Description

buildTree

Usage

```
buildTree(
  object,
  method = "euclidean",
  dim.type = c("raw", "pca", "tsne", "dc", "umap"),
  dim.use = 1:2,
  verbose = FALSE
)
```

Arguments

object	an FSPY object
method	character. Mehtod to build MST.
dim.type	character. Type of dimensions that will be used to build the tree. Five dim.type are provided, 'raw', 'pca', 'tsne', 'dc' and 'umap'. By default is 'raw'.
dim.use	numeric. Number of dimensions that will be used to build the tree. For example. If dim.use is 'raw', there is no limit for dim.type. And if the dim.use is 'tsne' or 'umap', the default dim.use is 1:2.
verbose	logical. Whether to print calculation progress.

Value

An FSPY object with tree

Examples

```
if (FALSE) {
# build minimum spanning tree (MST) based on raw expression matrix
fspy <- buildTree(fspy, dim.type = "raw")

# build minimum spanning tree (MST) based on tsne
fspy <- buildTree(fspy, dim.type = "tsne", dim.use = 1:2)

# Using PCA
fspy <- buildTree(fspy, dim.type = "pca", dim.use = 1:4)

# Using UMAP
fspy <- buildTree(fspy, dim.type = "umap", dim.use = 1:2)

# Using Diffusion Maps
fspy <- buildTree(fspy, dim.type = "dc", dim.use = 1:3)
}
```

constraintMatrix *constraintMatrix*

Description

constraint FCS data by a provid cutoff

Usage

```
constraintMatrix(x, cutoff = 0.99, markers = NULL, method = "euclidean")
```

Arguments

x	matrix
cutoff	numeric. Cutoff of the constraint value
markers	character. Markers used in the calculation of constraint model.
method	character. the distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".

Value

a matrix

Examples

```
mat <- matrix(runif(10000), nrow = 1000, ncol = 10)
colnames(mat) <- LETTERS[1:10]
dim(mat)

mat <- constraintMatrix(mat)
dim(mat)
```

correctBatchFSPY *correctBatchFSPY*

Description

Remove batch effect in FSPY object

Usage

```
correctBatchFSPY(
  object,
  batch = NULL,
  par.prior = TRUE,
  mean.only = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

object	An FSPY object
batch	vector. Batch covariate (only one batch allowed)
par.prior	logical. TRUE indicates parametric adjustments will be used, FALSE indicates non-parametric adjustments will be used.
mean.only	logical. FALSE If TRUE ComBat only corrects the mean of the batch effect (no scale adjustment)
verbose	logical. Whether to show log information
...	Parameters passing to ComBat function

Value

An FSPY object after removing batch effect

An FSPY object with corrected batch effects

See Also

findKNN

Examples

```
if (FALSE) {
  plot.meta <- fetchPlotMeta(fspy)
  batch <- as.numeric(plot.meta$stage)
  fspy <- correctBatchFSPY(object, batch = batch)
}
```

---

createFSPY *create an FSPY object*

Description

This function is about how to build an FSPY object. An FSPY object is the base for the whole analysizing workflow of flow and mass cytometry data.

Usage

```
createFSPY(
  raw.data,
  markers,
  meta.data,
  batch = NULL,
  batch.correct = FALSE,
  normalization.method = "none",
  verbose = FALSE,
  ...
)
```

Arguments

raw.data	matrix. Raw data read from FCS file after perform preprocessing.
markers	vector. Detailed marker information in the gate of flow cytometer.
meta.data	data.frame. Raw metadata of each cell. Columns "cell" and "stage" are required.
batch	vector. Batch covariate (only one batch allowed). Method to correct batch effect function is refered to ComBat.
batch.correct	logical. Whether to correct batch effect. If TRUE, batch must be provided.
normalization.method	character. Normalization and transformation method. Whether to normalize and log transformed of raw.data. In flowSpy workflow, it's better to perform transformation of FCS data using runExprsExtract or runExprsMerge before creating an FSPY object. flowSpy only provide log transforma method. If you need to using truncateTransform, scaleTransform, linearTransform, quadraticTransform and lnTransform, see flowCore for more information. And runExprsExtract in flowSpy, autoLgcl, cytofAsinh, logicle, arcsinh, and logAbs can be used to perform transformation of FCS data.
verbose	logical. Whether to print calculation progress.
...	paramters pass to correctBatchFSPY function.

Value

An FSPY object with raw.data and markers and meta.data

Examples

```
if (FALSE) {
## See vignette tutorials
vignette(package = "flowSpy")
vignette("Quick_start", package = "flowSpy")

## Build using test data
markers <- c("CD43", "CD34", "CD90", "CD45RA",
             "CD31", "CD49f", "CD73", "FLK1", "CD38")

fspy <- createFSPY(raw.data = test.fcs.data,
                   markers = markers,
                   meta.data = test.meta.data,
                   normalization.method = "log",
                   verbose = TRUE)

fspy

}
```

defLeafCells *definition of leaf cells*

Description

definition of root cells

Usage

```
defLeafCells(object, leaf.cells = NULL, pseudotime.cutoff = 0, verbose = FALSE)
```

Arguments

object	an FSPY object
leaf.cells	character or numeric. Cell name of the root cells or cluster.id of root.cells
pseudotime.cutoff	numeric. Cutoff of pseudotime. Cells with pseudotime over pseudotime.cutoff will be set to be leaf cells
verbose	logical. Whether to print calculation progress.

Value

An FSPY object

Examples

```
if (FALSE) {
# Define leaf cells by cluster
fspy <- defLeafCells(fspy, leaf.cells = 1, verbose = TRUE)
fspy <- defLeafCells(fspy, leaf.cells = c(1,3), verbose = TRUE)

# Define root cells by cell names
cells <- test.meta.data$cell[which(test.meta.data$stage == "D10")]
cells <- as.character(cells)
fspy <- defLeafCells(fspy, leaf.cells = cells, verbose = TRUE)
}
```

defRootCells *definition of root cells*

Description

definition of root cells

Usage

```
defRootCells(object, root.cells = NULL, verbose = FALSE)
```

Arguments

object	an FSPY object
root.cells	vector. Cell name of the root cells
verbose	logical. Whether to print calculation progress.

Value

An FSPY object

Examples

```
if (FALSE) {
# Define root cells by cluster
fspy <- defRootCells(fspy, root.cells = 6, verbose = TRUE)
fspy <- defRootCells(fspy, root.cells = c(6,8), verbose = TRUE)

# Define root cells by cell names
cells <- test.meta.data$cell[which(test.meta.data$stage == "D0")]
cells <- as.character(cells)
fspy <- defRootCells(fspy, root.cells = cells, verbose = TRUE)
}
```

---

fetchCell *Fetching cellls of FSPY*

Description

Fetching cellls of FSPY

Usage

```
fetchCell(object, logical.connect = "or", verbose = FALSE, ...)
```

Arguments

object	An FSPY object
logical.connect	character. "and" or "or"
verbose	logical. Whether to print calculation progress.
...	Paramters to pass to limitation

Value

a vector containing cell names

Examples

```
if (FALSE) {
cell.fetch <- fetchCell(fspy, traj.value.log = 0.01)
cell.fetch <- fetchCell(fspy, stage = c("D0", "D10"))
cell.fetch <- fetchCell(fspy, stage = c("D0", "D10"), traj.value.log = 0.01,
                        logical.connect = "or")
}
```

`fetchClustMeta` *Fetching clusters' metadata of FSPY*

Description

Fetching clusters' metadata of FSPY

Usage

```
fetchClustMeta(object, verbose = FALSE)
```

Arguments

`object`	An FSPY object
`verbose`	logical. Whether to print calculation progress.

Value

a data.frame containing clustering information for visualization

Examples

```
if (FALSE) {
clust.data <- fetchClustMeta(fspy)
head(clust.data)
}
```

`fetchPlotMeta` *Fetching plot metadata of FSPY*

Description

Fetching plot metadata of FSPY

Usage

```
fetchPlotMeta(object, markers = NULL, verbose = FALSE)
```

Arguments

object	An FSPY object
markers	vector. Makers fetched from expression matrix
verbose	logical. Whether to print calculation progress.

Value

a data.frame containing meta information for visualization

Examples

```
if (FALSE) {
plot.data <- fetchPlotMeta(fspy)
head(plot.data)

plot.data <- fetchPlotMeta(fspy, markers = c("CD43", "CD34"))
head(plot.data)
}
```

---

find_neighbors *K Nearest Neighbour Search*

Description

Uses a kd-tree to find the p number of near neighbours for each point in an input/output dataset. Use the nn2 function from the RANN package, utilizes the Approximate Near Neighbor (ANN) C++ library, which can give the exact near neighbours or (as the name suggests) approximate near neighbours to within a specified error bound. For more information on the ANN library please visit http://www.cs.umd.edu/~mount/ANN/.

Usage

```
find_neighbors(data, k)
```

Arguments

data	matrix; input data matrix
k	integer; number of nearest neighbours

Value

a n-by-k matrix of neighbor indices

Author(s)

Hao Chen <chen_hao@immunol.a-star.edu.sg>

Examples

```
iris_unique <- unique(iris) # Remove duplicates
data <- as.matrix(iris_unique[,1:4])
neighbors <- find_neighbors(data, k=10)
```

FSPY-class *Class* FSPY

Description

All information stored in FSPY object. You can use `creatFSPY` to create an FSPY object. In this package, most of the functions will use FSPY object as input, and return a modified FSPY obejct as well.

Slots

`raw.data` matrix. Raw signal data captured in flow or mass cytometry.

`log.data` matrix. Log-transfromed dataset of raw.data.

`meta.data` data.frame. Meta data information, and colnames of "stage" and "cell" are required.

`markers` vector. Markers used in the calculation of PCA, tSNE, diffusion map and UMAP.

`markers.idx` vector. Index of markers used in the calculation of PCA, tSNE, destiny and umap.

`cell.name` vector. Cell names after performing downsampling.

`knn` numeric. Numbers of nearest neighbors

`knn.index,knn.distance` matrix. Each row of the `knn.index` matrix corresponds to a point in `log.data` and contains the row indices in `log.data` that are its nearest neighbors. And each row of the `knn.distance` contains the distance of its nearest neighbors.

`som` list. Store som network information calculated using `FlowSOM`.

`cluster` data.frame. Cluster information

`pca.sdev,pca.value,pca.scores` PCA information of FSPY object which are generated from `fast.prcomp`.

`tsne.value` matrix. tSNE coordinates information. See `Rtsne`.

`dm` DiffusionMap object. Diffusion map calculated by package `destiny`

`umap.value` matrix umap coordinates information calculated using `umap`.

`root.cells` vector, Names of root cells, which can be modified by `defRootCells`. An root cell is manually set to be the origin of all cells. Pseudotime in root cells are the lowest.

`leaf.cells` vector. Names of leaf cells, which can be modified by `defLeafCells`. An leaf cell is manually set to be the terminal state of all cells. Pseuodtime in leaf cells are the largest.

`network` list. Network stored in the calculation of trajectory and pseudotime.

`walk` list. Random forward and backward walk between `root.cells` and `leaf.cells`.

`diff.traj` list. Differentiation trajectory all cells.

`plot.meta` data.frame. Plot meta information for `plot2D` or `plot3D`.

`tree.meta` data.frame. Tree meta information of FSPY object.

gatingMatrix *Apply gating on the matrix data*

Description

Apply gating on the matrix data

Usage

```
gatingMatrix(x, lower.gate = NULL, upper.gate = NULL)
```

Arguments

x	matrix
lower.gate	vector. Gating parameter, the name of the vector is the marker name, and the value of the vector is the lower bound of gating cutoff.
upper.gate	vector. Gating parameter, the name of the vector is the marker name, and the value of the vector is the upper bound of gating cutoff.

Value

a matrix

Examples

```
par(mfrow=c(1,2))
x <- matrix(rnorm(200, 3, 1), nrow = 100, ncol = 2)
colnames(x) <- c("CD34", "CD43")
plot(x[, "CD34"], x[, "CD43"], main = "Before gating")

lower.gate = c(CD34 = 2, CD43 = 3)
upper.gate = c(CD34 = 4, CD43 = 5)

x <- gatingMatrix(x, lower.gate = lower.gate, upper.gate = upper.gate)
plot(x[, "CD34"], x[, "CD43"], main = "After gating")

par(mfrow=c(1,1))
```

plot2D *Visualization of 2D data of FSPY*

Description

Visualization of 2D data of FSPY

Usage

```
plot2D(
  object,
  item.use = c("PC_1", "PC_2"),
  color.by = "stage",
  order.by = NULL,
  size = 1,
  alpha = 1,
  category = "categorical",
  show.cluser.id = FALSE,
  show.cluser.id.size = 4,
  main = "2D plot of FSPY",
  plot.theme = theme_bw()
)
```

Arguments

object	An FSPY object
item.use	character. Items use to 2D plot, axes x and y must be numeric.
color.by	character. Dot or mesh color by which character. It can be one of the column of plot.meta, or it can be just "density" (the default value).
order.by	vector. Order of color theme.
size	numeric. Size of the dot
alpha	numberic. Transparency (0-1) of the dot, default is 1.
category	character. numeric or categorical
show.cluser.id	logical. Whether to show cluster id in the plot.
show.cluser.id.size	numeric. Size of the cluster id.
main	character. Title of the plot.
plot.theme	themes from ggplot2

Value

ggplot2 figure

Examples

```
if (FALSE) {

# Default plot
plot2D(fspy)

# PCA plot
plot2D(fspy, item.use = c("PC_1", "PC_2"))
plot2D(fspy, item.use = c("PC_1", "PC_2"), color.by = "cluster.id")
plot2D(fspy, item.use = c("PC_1", "PC_2"), color.by = "stage")
plot2D(fspy, item.use = c("PC_2", "PC_3"), color.by = "stage") +
  scale_color_manual(values = c("#00599F","#009900","#FF9933",
                                "#FF99FF","#7A06A0","#FF3222"))
plot2D(fspy, item.use = c("PC_2", "PC_3"), color.by = "CD43",
```

```
       category = "numeric")
plot2D(fspy, item.use = c("PC_2", "PC_3"), color.by = "CD43",
       category = "numeric") +
       scale_colour_gradientn(colors = c("blue","white","red"))

# tSNE plot
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"))
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "stage")
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "cluster.id",
       alpha = 0.5, main = "tSNE Plot")
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "cluster.id",
       alpha = 1, main = "tSNE Plot", show.cluser.id = T)
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "CD43",
       category = "numeric", size = 3) +
       scale_colour_gradientn(colors = c("blue","white","red"))
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), color.by = "stage") +
  scale_color_manual(values = c("#00599F","#009900","#FF9933",
                                "#FF99FF","#7A06A0","#FF3222"))

# Diffusion Map plot
plot2D(fspy, item.use = c("DC_1", "DC_2"))
plot2D(fspy, item.use = c("DC_1", "DC_2"), color.by = "stage")
plot2D(fspy, item.use = c("DC_2", "DC_3"), color.by = "cluster.id",
       alpha = 0.5, main = "Diffusion Map Plot")
plot2D(fspy, item.use = c("DC_2", "DC_3"), color.by = "cluster.id",
       alpha = 1, main = "Diffusion Map Plot", show.cluser.id = T)
plot2D(fspy, item.use = c("DC_1", "DC_2"), color.by = "CD43",
       category = "numeric", size = 3) +
       scale_colour_gradientn(colors = c("blue","white","red"))

# UMAP plot
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"))
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "stage")
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "cluster.id",
       alpha = 0.5, main = "UMAP Plot")
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "cluster.id",
       alpha = 1, main = "UMAP Plot", show.cluser.id = T)
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "CD43",
       category = "numeric", size = 3) +
       scale_colour_gradientn(colors = c("blue","white","red"))
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), color.by = "stage") +
  scale_color_manual(values = c("#00599F","#009900","#FF9933",
                                "#FF99FF","#7A06A0","#FF3222"))

# Marker Plot
plot2D(fspy, item.use = c("CD43", "CD90"), color.by = "cluster.id")
plot2D(fspy, item.use = c("CD34", "CD90"), color.by = "CD43",
       category = "numeric", size = 3) +
       scale_colour_gradientn(colors = c("blue","white","red"))

# Pseudotime
plot2D(fspy, item.use = c("pseudotime", "CD43"), color.by = "stage")

}
```

plot3D *Visualization of 3D data of FSPY*

Description

Visualization of 3D data of FSPY

Usage

```
plot3D(
  object,
  item.use = c("PC1", "PC2", "PC3"),
  color.by = "stage",
  order.by = NULL,
  size = 1,
  angle = 60,
  scale.y = 0.8,
  category = "categorical",
  main = "3D plot of FSPY",
  color.theme = NULL,
  ...
)
```

Arguments

object	An FSPY object
item.use	character. Items use to 3D plot, axes x and y and z must be numeric.
color.by	character. Dot or mesh color by which character. It can be one of the column of plot.meta, or it can be just "density" (the default value).
order.by	character. Order of color theme.
size	numeric. size of the dot
angle	numberic. angle of the plot
scale.y	numeric. scale of y axis related to x- and z axis
category	character. numeric or categorical
main	character. title of the plot
color.theme	vector. Color themes use in the plot.
...	options to pass on to the scatterplot3d function.

Value

gplots figure

Examples

```
if (FALSE) {

 plot3D(fspy, item.use = c("DC_2","DC_1","DC_3"), color.by = "stage",
        size = 0.5, angle = 60, color.theme = c("#FF99FF","#7A06A0","#FF3222"))
```

```
}
```

`plotBranchHeatmap` *Visualization heatmap of branch data of FSPY*

Description

Visualization heatmap of branch data of FSPY

Usage

```
plotBranchHeatmap(
  object,
  color = colorRampPalette(c("blue", "white", "red"))(100),
  scale = "row",
  ...
)
```

Arguments

`object`	An FSPY object
`color`	vector. Colors used in heatmap.
`scale`	character. Whether the values should be centered and scaled in either the row direction or the column direction, or none. Corresponding values are "row", "column" and "none"
`...`	options to pass on to the `pheatmap` function.

Value

ggplot2 figure

Examples

```
if (FALSE) {

plotBranchHeatmap(fspy)
plotBranchHeatmap(fspy, color = colorRampPalette(c("purple","white","yellow"))(100))
plotBranchHeatmap(fspy, cluster_row = FALSE)
plotBranchHeatmap(fspy, cluster_row = FALSE, cluster_col = FALSE)

}
```

plotCluster *Visualization of cluster data of FSPY*

Description

Visualization of cluster data of FSPY

Usage

```
plotCluster(
  object,
  item.use = c("PC_1", "PC_2"),
  color.by = "cluster",
  size.by = "cell.number.percent",
  order.by = NULL,
  size = 1,
  alpha = 1,
  category = "categorical",
  show.cluser.id = FALSE,
  show.cluser.id.size = 4,
  main = "2D plot of cluster in FSPY",
  plot.theme = theme_bw()
)
```

Arguments

object	An FSPY object
item.use	character. Items use to 2D plot, axes x and y must be numeric.
color.by	character. Dot or mesh color by which character. It can be one of the column of plot.meta, or it can be just "density" (the default value).
size.by	character. Size of the dot
order.by	vector. Order of color theme.
size	numeric. Size of the dot
alpha	numberic. Transparency (0-1) of the dot, default is 1.
category	character. numeric or categorical
show.cluser.id	logical. Whether to show cluster id in the plot.
show.cluser.id.size	numeric. Size of the cluster id.
main	character. Title of the plot.
plot.theme	themes from ggplot2

Value

ggplot2 figure

Examples

```
if (FALSE) {
plotCluster(fspy)

plotCluster(fspy, item.use = c("PC_1", "PC_2"))
plotCluster(fspy, item.use = c("PC_2", "PC_3"))
plotCluster(fspy, item.use = c("PC_2", "PC_3"), color.by = "CD43", category = "numeric")
plotCluster(fspy, item.use = c("PC_2", "PC_3"), color.by = "CD43", category = "numeric") +
    scale_colour_gradientn(colors = c("blue", "white", "red"))

plotCluster(fspy, item.use = c("tSNE_1", "tSNE_2"))
plotCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), show.cluser.id = T)

plotCluster(fspy, item.use = c("DC_1", "DC_2"))

plotCluster(fspy, item.use = c("UMAP_1", "UMAP_2"))
}
```

`plotClusterHeatmap` *Visualization heatmap of cluster data of FSPY*

Description

Visualization heatmap of cluster data of FSPY

Usage

```
plotClusterHeatmap(
  object,
  color = colorRampPalette(c("blue", "white", "red"))(100),
  scale = "row",
  ...
)
```

Arguments

`object`	An FSPY object
`color`	vector. Colors used in heatmap.
`scale`	character. Whether the values should be centered and scaled in either the row direction or the column direction, or none. Corresponding values are "row", "column" and "none"
`...`	options to pass on to the `pheatmap` function.

Value

ggplot2 figure

Examples

```
if (FALSE) {

plotClusterHeatmap(fspy)
plotClusterHeatmap(fspy, color = colorRampPalette(c("purple","white","yellow"))(100))
plotClusterHeatmap(fspy, cluster_row = F)
plotClusterHeatmap(fspy, cluster_row = F, cluster_col = F)

}
```

`plotHeatmap` *Visualization heatmap of data of FSPY*

Description

Visualization heatmap of data of FSPY

Usage

```
plotHeatmap(
  object,
  markers = NULL,
  color = colorRampPalette(c("blue", "white", "red"))(100),
  scale = "row",
  downsize = 1000,
  cluster_rows = FALSE,
  cluster_cols = FALSE,
  ...
)
```

Arguments

`object`	An FSPY object
`markers`	vector. markers to plot on the heatmap
`color`	vector. Colors used in heatmap.
`scale`	character. Whether the values should be centered and scaled in either the row direction or the column direction, or none. Corresponding values are "row", "column" and "none"
`downsize`	numeric. Cells size used to plot heatmap
`cluster_rows`	logical. Whether rows should be clustered
`cluster_cols`	logical. Whether columns should be clustered
`...`	options to pass on to the `pheatmap` function.

Value

ggplot2 figure

Examples

```
if (FALSE) {

plotHeatmap(fspy)
plotHeatmap(fspy, cluster_rows = T)
plotHeatmap(fspy, cluster_rows = T, clustering_method = "ward.D")
plotHeatmap(fspy, cluster_rows = T, cluster_cols = T)

}
```

`plotMarkerDensity` *plotMarkerDensity*

Description

plotMarkerDensity

Usage

```
plotMarkerDensity(
  object,
  cutoff = -1,
  markers = NULL,
  adjust = 0.5,
  plot.theme = theme_bw()
)
```

Arguments

`object`	An FSPY object
`cutoff`	numeric. Cutoff of trajectory value
`markers`	character. Markers used in the calculation progress
`adjust`	numeric. Transparency (0-1) of the dot, default is 1.
`plot.theme`	themes from `ggplot2`

Value

ggplot2 figure

Examples

```
if (FALSE) {

plotMarkerDensity(fspy)
plotMarkerDensity(fspy, adjust = 1)

}
```

`plotPieCluster` *Visualization pie plot of cluster data of FSPY*

Description

Visualization pie plot of cluster data of FSPY

Usage

```
plotPieCluster(
  object,
  item.use = c("PC_1", "PC_2"),
  cex.size = 1,
  size.by.cell.number = TRUE,
  main = "2D pie plot of FSPY",
  plot.theme = theme_bw()
)
```

Arguments

`object`	An FSPY object
`item.use`	character. Items use to 2D plot, axes x and y must be numeric.
`cex.size`	numeric. Size of the dot
`size.by.cell.number`	logical. Whether to show size of cell number.
`main`	character. Title of the plot.
`plot.theme`	themes from `ggplot2`

Value

ggplot2 figure

Examples

```
if (FALSE) {
# Runs only have more than two stages
plotPieCluster(fspy, cex.size = 0.5)

plotPieCluster(fspy, item.use = c("PC_1", "PC_2"), cex.size = 0.5)
plotPieCluster(fspy, item.use = c("PC_2", "PC_3"), cex.size = 0.5)

plotPieCluster(fspy, item.use = c("tSNE_1", "tSNE_2"), cex.size = 20)

plotPieCluster(fspy, item.use = c("DC_1", "DC_2"), cex.size = 0.5)

plotPieCluster(fspy, item.use = c("UMAP_1", "UMAP_2"), cex.size = 1)
plotPieCluster(fspy, item.use = c("UMAP_1", "UMAP_2"), cex.size = 1) +
   scale_fill_manual(values = c("#00599F","#FF3222","#009900",
                                "#FF9933","#FF99FF","#7A06A0"))
}
```

plotPieTree *plot MST pie of FSPY*

Description

plot MST pie of FSPY

Usage

```
plotPieTree(
  object,
  cex.size = 2,
  size.by.cell.number = TRUE,
  as.tree = FALSE,
  root.id = NULL,
  show.node.name = FALSE
)
```

Arguments

object	an FSPY object
cex.size	numeric. size cex of the dot
size.by.cell.number	logical. Whether to size node by cell number
as.tree	logical. Whether to show node as tree
root.id	numeric. Root id of the tree, if as.tree is TRUE
show.node.name	logical. whether to show node name

Value

ggplot2 figure

Examples

```
if (FALSE) {

# Runs only have two or more stages
plotPieTree(fspy, cex.size = 1, size.by.cell.number = T) +
   scale_fill_manual(values = c("#00599F","#FF3222","#009900",
                                "#FF9933","#FF99FF","#7A06A0"))
}
```

`plotPseudotimeDensity` *plot Pseudotime density of FSPY*

Description

plot Pseudotime density of FSPY

Usage

```
plotPseudotimeDensity(
  object,
  color.by = "stage",
  main = "Density of pseudotime",
  adjust = 0.5,
  plot.theme = theme_bw()
)
```

Arguments

`object`	an FSPY object
`color.by`	character.
`main`	character. Title of the plot
`adjust`	numeric. A multiplicate bandwidth adjustment.
`plot.theme`	themes from `ggplot2`

Value

ggplot2 figure

Examples

```
if (FALSE) {

plotPseudotimeDensity(fspy)

plotPseudotimeDensity(fspy, adjust = 1)
plotPseudotimeDensity(fspy, adjust = 2)

plotPseudotimeDensity(fspy, adjust = 2) +
  scale_color_manual(values = c("#00599F","#FF3222","#009900",
                                "#FF9933","#FF99FF","#7A06A0"))

}
```

plotPseudotimeTraj *plotPseudotimeTraj*

Description

plotPseudotimeTraj

Usage

```
plotPseudotimeTraj(
  object,
  cutoff = -1,
  markers = NULL,
  size = 0.5,
  alpha = 0.6,
  print.curve = TRUE,
  var.cols = FALSE,
  plot.theme = theme_bw()
)
```

Arguments

object	An FSPY object
cutoff	numeric. Cutoff of trajectory value
markers	character. Markers used in the calculation progress
size	numeric. Size of the dot
alpha	numeric. Transparency (0-1) of the dot, default is 1.
print.curve	logical. Whether to perform curve fitting
var.cols	logical. Whether to plot stage
plot.theme	themes from ggplot2

Value

ggplot2 figure

Examples

```
if (FALSE) {

plotPseudotimeTraj(fspy)
plotPseudotimeTraj(fspy, print.curve = F)
plotPseudotimeTraj(fspy, var.cols = T)

plotPseudotimeTraj(fspy) +
   scale_colour_gradientn(colors = c("#F4D31D", "#FF3222","#7A06A0"))

plotPseudotimeTraj(fspy, markers = c("CD43", "CD34")) +
scale_colour_gradientn(colors = c("#F4D31D", "#FF3222","#7A06A0"))

}
```

`plotTrajHeatmap` *Visualization heatmap of intermediate cells of FSPY*

Description

Visualization heatmap of intermediate cells of FSPY

Usage

```
plotTrajHeatmap(
  object,
  cutoff = 0,
  markers = NULL,
  color = colorRampPalette(c("blue", "white", "red"))(100),
  scale = "row",
  ...
)
```

Arguments

`object`	An FSPY object
`cutoff`	numeric. value to identify intermediate state cells
`markers`	markers to plot on the heatmap
`color`	vector. Colors used in heatmap.
`scale`	character. Whether the values should be centered and scaled in either the row direction or the column direction, or none. Corresponding values are "row", "column" and "none"
`...`	options to pass on to the `pheatmap` function.

Value

ggplot2 figure

Examples

```
if (FALSE) {

plotTrajHeatmap(fspy)
plotBranchHeatmap(fspy, color = colorRampPalette(c("purple","white","yellow"))(100))
plotBranchHeatmap(fspy, cluster_row = FALSE)
plotBranchHeatmap(fspy, cluster_row = FALSE, cluster_col = FALSE)

}
```

plotTree *plot MST of FSPY*

Description

plot MST of FSPY

Usage

```
plotTree(
  object,
  cex.size = 1,
  color.by = "cell.number",
  size.by = "cell.number",
  as.tree = FALSE,
  root.id = NULL,
  show.node.name = FALSE
)
```

Arguments

object	an FSPY object
cex.size	numeric. size cex of the dot
color.by	numeric. size color theme of the dot
size.by	numeric. size theme of the dot
as.tree	logical. Whether to show node as tree
root.id	numeric. Root id of the tree, if as.tree is TRUE
show.node.name	logical. whether to show node name

Value

ggplot2 figure

Examples

```
if (FALSE) {

plotTree(fspy)

plotTree(fspy, show.node.name = T)

plotTree(fspy, color.by = "CD43", show.node.name = T, cex.size = 1) +
    scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))

plotTree(fspy, color.by = "D0.percent", show.node.name = T, cex.size = 1) +
    scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))

plotTree(fspy, color.by = "D2.percent", show.node.name = T, cex.size = 1) +
    scale_colour_gradientn(colors = c("#00599F", "#EEEEEE", "#FF3222"))
```

```
plotTree(fspy, color.by = "pseudotime", cex.size = 1) +
    scale_colour_gradientn(colors = c("#F4D31D", "#FF3222","#7A06A0"))

}
```

---

plotViolin *Visualization violin plot of FSPY*

Description

Visualization violin plot of FSPY

Usage

```
plotViolin(
  object,
  marker,
  color.by = "cluster.id",
  order.by = NULL,
  size = 1,
  text.angle = 0,
  main = "Violin plot FSPY",
  plot.theme = theme_bw()
)
```

Arguments

object	An FSPY object
marker	character. Markers used to plot
color.by	character. Dot or mesh color by which character. It can be one of the column of plot.meta, or it can be just "density" (the default value).
order.by	vector. Order of color theme.
size	numeric. Size of the dot
text.angle	numberic. Text angle of the violin plot
main	character. Title of the plot.
plot.theme	themes from ggplot2

Value

ggplot2 figure

Examples

```
if (FALSE) {
plotViolin(fspy, marker = "CD34")
plotViolin(fspy, marker = "CD34", order.by = "pseudotime")
}
```

processingCluster *processingCluster*

Description

Calculate Principal Components Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (tSNE), Diffusion Map and Uniform Manifold Approximation and Projection (UMAP) of clusters calculated by runCluster.

Usage

```
processingCluster(
  object,
  perplexity = 5,
  k = 5,
  downsampling.size = 1,
  force.resample = TRUE,
  random.cluster = FALSE,
  umap.config = umap.defaults,
  verbose = FALSE,
  ...
)
```

Arguments

object	an FSPY object
perplexity	numeric. Perplexity parameter (should not be bigger than 3 * perplexity < nrow(X) - 1, see details for interpretation). See Rtsne for more information.
k	numeric. The parameter k in k-Nearest Neighbor.
downsampling.size	numeric. Percentage of sample size of downsampling. This parameter is from 0 to 1. by default is 1.
force.resample	logical. Whether to do resample if downsampling.size < 1
random.cluster	logical. Whether to perfrom random downsampling. If FALSE, an uniform downsampling will be processed.
umap.config	object of class umap.config. See umap.
verbose	logic. Whether to print calculation progress.
...	options to pass on to the dimensionality reduction functions.

Value

An FSPY object with cluster.id in meta.data

An FSPY object with dimensionality reduction of clusters

See Also

umap, fast.prcomp, Rtsne, destiny

Examples

```
if (FALSE) {

# After running clustering
set.seed(1)
fspy <- runCluster(fspy, cluster.method = "som", xdim = 3, ydim = 3, verbose = T)

# Do not perfrom downsampling
fspy <- processingCluster(fspy, perplexity = 2)

# Perform cluster based downsampling
# Only keep 50% cells
fspy <- processingCluster(fspy, perplexity = 2, downsampling.size = 0.5)

# Processing clusters without downsampling step
fspy <- processingCluster(fspy, perplexity = 2, force.resample = FALSE)

}
```

Rphenograph *RphenoGraph clustering*

Description

R implementation of the PhenoGraph algorithm

A simple R implementation of the [PhenoGraph](http://www.cell.com/cell/abstract/S0092-8674(15)00637-6) algorithm, which is a clustering method designed for high-dimensional single-cell data analysis. It works by creating a graph ("network") representing phenotypic similarities between cells by calclating the Jaccard coefficient between nearest-neighbor sets, and then identifying communities using the well known [Louvain method](https://sites.google.com/site/findcommunities/) in this graph.

This function is developed by Hao Chen and updated by Yuting Dai.

Usage

```
Rphenograph(data, k = 30)
```

Arguments

data	matrix; input data matrix
k	integer; number of nearest neighbours (default:30)

Value

a list contains an igraph graph object for `graph_from_data_frame` and a communities object, the operations of this class contains:

print	returns the communities object itself, invisibly.
length	returns an integer scalar.

sizes	returns a numeric vector.
membership	returns a numeric vector, one number for each vertex in the graph that was the input of the community detection.
modularity	returns a numeric scalar.
algorithm	returns a character scalar.
crossing	returns a logical vector.
is_hierarchical	returns a logical scalar.
merges	returns a two-column numeric matrix.
cut_at	returns a numeric vector, the membership vector of the vertices.
as.dendrogram	returns a dendrogram object.
show_trace	returns a character vector.
code_len	returns a numeric scalar for communities found with the InfoMAP method and NULL for other methods.
plot	for communities objects returns NULL, invisibly.

cluster information

Author(s)

Hao Chen <chen_hao@immunol.a-star.edu.sg>

References

Jacob H. Levine and et.al. Data-Driven Phenotypic Dissection of AML Reveals Progenitor-like Cells that Correlate with Prognosis. Cell, 2015.

Examples

```
iris_unique <- unique(iris) # Remove duplicates
data <- as.matrix(iris_unique[,1:4])
Rphenograph_out <- Rphenograph(data, k = 45)
modularity(Rphenograph_out[[2]])
membership(Rphenograph_out[[2]])
iris_unique$phenograph_cluster <- factor(membership(Rphenograph_out[[2]]))
```

---

runClara *runClara*

Description

Clustering a data matrix into k clusters

Usage

```
runClara(
  object,
  k = 25,
  metric = c("euclidean", "manhattan", "jaccard"),
  stand = FALSE,
  samples = 5,
  scale = TRUE,
  trace = 0,
  verbose = FALSE,
  ...
)
```

Arguments

object	an FSPY object
k	numeric. The number of clusters. It is required that 0 < k < n where n is the number of observations (i.e., n = nrow(x)).
metric	character. string specifying the metric to be used for calculating dissimilarities between observations.
stand	logical. Indicating if the measurements in x are standardized before calculating the dissimilarities.
samples	numeric. Say N, the number of samples to be drawn from the dataset. The default is N = 5,
scale	logical. Whether to use scaled data in kmeans.
trace	numberic. Indicating a trace level for diagnostic output during the algorithm
verbose	logical. Whether to print calculation progress.
...	Parameters passing to clara function

Value

an FSPY object with clara.id in meta.data

See Also

clara

Examples

```
if (FALSE) {
fspy <- runClara(fspy, k = 25, verbose = TRUE)
}
```

runCluster *Specific Clustering Method Toolkits*

Description

Compute a specific clustering using the combined flow cytometry data. "som" SOM, "hclust" hclust, "clara" clara, "phenograph", "kmeans" kmeans are provided.

Usage

```
runCluster(
  object,
  cluster.method = c("som", "kmeans", "clara", "phenograph", "hclust", "mclust"),
  verbose = FALSE,
  ...
)
```

Arguments

object	an FSPY object
cluster.method	character. Four clustering method are provided: som, clara, kmeans and phenograph. Clustering method "hclust" and "mclust" are not recommended because of long computing time.
verbose	logic. Whether to print calculation progress.
...	options to pass on to the clustering functions.

Value

An FSPY object with cluster

See Also

SOM, hclust, clara, kmeans. You can use runSOM, runClara, runPhenotype, runKmeans, runMclust and runHclust to run clustering respectively.

Examples

```
if (FALSE) {
# After building an FSPY object
# Set random seed to make results reproducible

set.seed(1)
fspy <- runCluster(fspy, cluster.method = "som", xdim = 3, ydim = 3, verbose = TURE)

# K-means clustering
fspy <- runCluster(fspy, cluster.method = "kmeans", k = 9, verbose = TRUE)

# Clara clustering
fspy <- runCluster(fspy, cluster.method = "clara", k = 9, verbose = TRUE)

# phenoGraph clustering
```

```
fspy <- runCluster(fspy, cluster.method = "phenograph", verbose = TRUE)

# hclust clustering
# not recommended for large cell size
fspy <- runCluster(fspy, cluster.method = "hclust", k = 9, verbose = TRUE)

# mclust clustering
# not recommended for large cell size
fspy <- runCluster(fspy, cluster.method = "mclust", verbose = TRUE)
}
```

`runDiff` *Calculate differential expression markers*

Description

Calculating differentially expressed markers

Usage

```
runDiff(object, branch.id = NULL, branch.id.2 = NULL, verbose = FALSE)
```

Arguments

`object`	an FSPY object
`branch.id`	vector. Branch ids use to run differentially expressed markers
`branch.id.2`	vector. Branch ids use to run differentially expressed markers in compare with branch.id
`verbose`	logic. Whether to print calculation progress.

Value

An FSPY object with cluster.id in meta.data

a data.frame with differential expressed markers

See Also

`bulidTree`

Examples

```
if (FALSE) {

DEG.table <- runDiff(fspy)

}
```

runDiffusionMap *Calculate diffusion map in FSPY*

Description

Calculate diffusion map in FSPY

Usage

```
runDiffusionMap(
  object,
  sigma.use = NULL,
  distance = c("euclidean", "cosine", "rankcor"),
  k = 30,
  density.norm = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

object	an FSPY object
sigma.use	numeric. Diffusion scale parameter of the Gaussian kernel. One of 'local', 'global', a numeric global sigma or a Sigmas object. When choosing 'global', a global sigma will be calculated using find_sigmas (See destiny). A larger sigma might be necessary if the eigenvalues can not be found because of a singularity in the matrix. See destiny.
distance	Distance measurement method applied to data or a distance matrix/dist. For the allowed values, see destiny
k	numeric. By default is 30. destiny can be used to specify k.
density.norm	logical. If TRUE, use density normalisation. See destiny
verbose	logical. Whether to print calculation progress.
...	options to pass on to the destiny.

Value

An FSPY object

See Also

destiny

Examples

```
if (FALSE) {
fspy <- runDiffusionMap(fspy, verbose = TRUE)
}
```

runExprsExtract *Extract the expression data from a FCS file with preprocessing*

Description

Extract the FCS expresssion data with preprocessing of compensation (for FCM data only) and transformation. Transformtion methods includes `autoLgcl`, `cytofAsinh`, `logicle` (customizable) and `arcsinh` (customizable).

Usage

```
runExprsExtract(
  fcsFile,
  verbose = FALSE,
  comp = FALSE,
 transformMethod = c("autoLgcl", "cytofAsinh", "logicle", "arcsinh", "logAbs", "none"),
  scaleTo = NULL,
  showDesc = TRUE,
  keepRaw = TRUE,
  q = 0.05,
  l_w = 0.1,
  l_t = 4000,
  l_m = 4.5,
  l_a = 0,
  a_a = 1,
  a_b = 1,
  a_c = 0
)
```

Arguments

`fcsFile`	The name of the FCS file.
`verbose`	If TRUE, print the message details of FCS loading.
`comp`	If TRUE, does compensation by compensation matrix contained in FCS. Agrument also accepts a compensation matrix to be applied. Otherwise FALSE.
`transformMethod`	Data Transformation method, including `autoLgcl`, `cytofAsinh`, `logicle` and `arcsinh`, or `none` to avoid transformation.
`scaleTo`	Scale the expression to a specified range c(a, b), default is NULL.
`showDesc`	logical. Whether to show `desc` name in the output matrix.
`keepRaw`	logical. Whether to keep raw data for FSC and SSC.
`q`	Quantile of negative values removed for auto w estimation, default is 0.05, parameter for autoLgcl transformation.
`l_w`	Linearization width in asymptotic decades, parameter for logicle transformation.
`l_t`	Top of the scale data value, parameter for logicle transformation.
`l_m`	Full width of the transformed display in asymptotic decades, parameter for logicle transformation.

l_a	Additional negative range to be included in the display in asymptotic decades, parameter for logicle transformation.
a_a	Positive double that corresponds to the base of the arcsinh transformation, arcsinh = asinh(a + b * x) + c).
a_b	Positive double that corresponds to a scale factor of the arcsinh transformation, arcsinh = asinh(a + b * x) + c).
a_c	Positive double that corresponds to another scale factor of the arcsinh transformation, arcsinh = asinh(a + b * x) + c).

Value

A transformed expression data matrix

Author(s)

Chen Hao

References

Hao Chen, Mai Chan Lau, Michael Thomas Wong, Evan W. Newell, Michael Poidinger, Jinmiao Chen. Cytofkit: A Bioconductor Package for an Integrated Mass Cytometry Data Analysis Pipeline. PLoS Comput Biol, 2016.

Examples

```
if (FALSE) {
# See vignette tutorials for more information
vignette(package = "flowSpy")
vignette("Quick_start", package = "flowSpy")

  # Path to your FCS files
  fcs.path <- "flowSpy-dataset/FCS/usecase1/"
  fcs.file <- paste0(fcs.path, "FR-FCM-ZY9R-Bone_Marrow_cytof.fcs")

  # Read FCS files
  exp.data <- runExprsExtract(fcs.file, showDesc = FALSE, transformMethod = "autoLgcl")
}
```

---

runExprsMerge *Merge the expression matrix from multiple FCS files with preprocessing*

Description

Apply preprocessing on each FCS file including compensation (for FCM data only) and transformation with selected markers, then expression matrix are extracted and merged using one of the methods, all, min, fixed or ceil

Usage

```
runExprsMerge(
  fcsFiles,
  comp = FALSE,
 transformMethod = c("autoLgcl", "cytofAsinh", "logicle", "arcsinh", "logAbs", "none"),
  scaleTo = NULL,
  mergeMethod = c("ceil", "all", "fixed", "min"),
  fixedNum = 2000,
  ...
)
```

Arguments

fcsFiles	A vector of FCS file names.
comp	If TRUE, does compensation by compensation matrix contained in FCS. Agrument also accepts a compensation matrix to be applied. Otherwise FALSE.
transformMethod	Data Transformation method, including autoLgcl, cytofAsinh, logicle and arcsinh, or none to avoid transformation.
scaleTo	Scale the expression to a specified range c(a, b), default is NULL.
mergeMethod	Merge method for mutiple FCS expression data. cells can be combined using one of the four different methods including ceil, all, min, fixed. The default option is ceil, up to a fixed number (specified by fixedNum) of cells are sampled without replacement from each fcs file and combined for analysis. all: all cells from each fcs file are combined for analysis. min: The minimum number of cells among all the selected fcs files are sampled from each fcs file and combined for analysis. fixed: a fixed num (specified by fixedNum) of cells are sampled (with replacement when the total number of cell is less than fixedNum) from each fcs file and combined for analysis.
fixedNum	The fixed number of cells to be extracted from each FCS file.
...	Other arguments passed to runExprsExtract

Value

A matrix containing the merged expression data, with selected markers.

Author(s)

Chen Hao

References

Hao Chen, Mai Chan Lau, Michael Thomas Wong, Evan W. Newell, Michael Poidinger, Jinmiao Chen. Cytofkit: A Bioconductor Package for an Integrated Mass Cytometry Data Analysis Pipeline. PLoS Comput Biol, 2016.

See Also

runExprsExtract

Examples

```
if (FALSE) {
  # See vignette tutorials for more information
  vignette("Quick_start", package = "flowSpy")

  # Path to your FCS files
  fcs.path <- "flowSpy-dataset/FCS/usecase2/"
  fcs.files <- paste0(fcs.path, "D", c(0,2,4,6,8,10), "-sub.fcs")

  # Merge FCS files, and each file contain 2000 cells
  set.seed(1)
 fcs.data <- runExprsMerge(fcs.files, comp = F, transformMethod = "none", fixedNum = 2000)
}
```

`runFastPCA` *Calculate principal components in FSPY*

Description

Calculate principal components in FSPY

Usage

```
runFastPCA(object, center = FALSE, scale. = TRUE, verbose = FALSE, ...)
```

Arguments

`object`	an FSPY object
`center`	logical, a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale. See `fast.prcomp`
`scale.`	logical, a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of x can be supplied. The value is passed to scale. See `fast.prcomp`
`verbose`	logical. Whether to print calculation progress.
`...`	Parameters passing to `fast.prcomp` function

Value

An FSPY object with PCA

See Also

`fast.prcomp`

Examples

```
if (FALSE) {
fspy <- runFastPCA(fspy, verbose = TRUE)
}
```

---

runHclust *runHclust*

Description

Hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it.

Usage

```
runHclust(
  object,
  k = 25,
  hclust.method = "complete",
  dist.method = "euclidean",
  verbose = FALSE
)
```

Arguments

object	an FSPY object
k	numeric. The number of clusters.
hclust.method	character or a function. The agglomeration method to be used. This should be one of "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" or "centroid". Or you can specify an equation as input, for example `function(x) hclust(x,method = 'ward.D2')`.
dist.method	character or a function. The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Or you can specify an equation as input, for example `function(x) as.dist((1-cor(t(x)))/2)`.
verbose	logical. Whether to print calculation progress.

Value

An FSPY object with cluster

if (FALSE) fspy <- runHclust(fspy, k = 9, verbose = TRUE)

See Also

hclust, dist

runKmeans *runKmeans*

Description

Perform k-means clustering on a data matrix.

Usage

```
runKmeans(
  object,
  k = 25,
  iter.max = 10,
  nstart = 1,
  algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"),
  trace = FALSE,
  scale = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

object	an FSPY object
k	numeric. The number of clusters.
iter.max	numeric. The maximum number of iterations allowed.
nstart	numeric. If k is a number, how many random sets should be chosen.
algorithm	character. Type of algorithm that will be choosen to calculate kmeans. Four algoritms are provided: Hartigan-Wong, Lloyd, Forgy, MacQueen.
trace	logical or integer number.
scale	logical. Whether to use scaled data in kmeans.
verbose	logical. Whether to print calculation progress.
...	Parameters passing to kmeans function

Value

an FSPY object with kmeans.id in meta.data

See Also

kmeans

Examples

```
if (FALSE) {
fspy <- runKmeans(fspy, k = 25, verbose = TRUE)
}
```

runKNN *Calculate k-nearest neighbors of FSPY*

Description

Calculates and stores a k-nearest neighbor graph based on Euclidean distance with (KMKNN) algorithm using log-transformed signaling matrix of flow cytometry data. The base function are base on findKNN.

Usage

```
runKNN(
  object,
  given.mat = NULL,
  knn = 30,
  knn.replace = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

object	an FSPY object
given.mat	matrix. Given matrix to run knn
knn	numeric. Number of k-nearest neighbors.
knn.replace	logic. Whether to replace knn in FSPY object
verbose	logical. Whether to print calculation progress.
...	Parameters passing to findKNN function

Value

An FSPY object with knn, knn.index and knn.distance information.

See Also

findKNN

Examples

```
if (FALSE) {

fspy <- runKNN(fspy)

}
```

runMclust *runMclust*

Description

Model-based clustering based on parameterized finite Gaussian mixture models. This function is based on Mclust.

Usage

```
runMclust(object, scale = FALSE, verbose = FALSE, ...)
```

Arguments

object	an FSPY object
scale	logical. Whether to use scaled data in Mclust.
verbose	logical. Whether to print calculation progress.
...	Parameters passing to Mclust function

Value

an FSPY object with mclust.id in meta.data

See Also

Mclust

Examples

```
if (FALSE) {
fspy <- runMclust(fspy, verbose = TRUE)
}
```

runPhenograph *RphenoGraph clustering*

Description

A simple R implementation of the phenograph [PhenoGraph](http://www.cell.com/cell/abstract/S0092-8674(15)00637-6) algorithm, which is a clustering method designed for high-dimensional single-cell data analysis. It works by creating a graph ("network") representing phenotypic similarities between cells by calculating the Jaccard coefficient between nearest-neighbor sets, and then identifying communities using the well known [Louvain method](https://sites.google.com/site/findcommunities/) in this graph.

Usage

```
runPhenograph(object, knn = 30, scale = FALSE, verbose = FALSE, ...)
```

Arguments

object	an FSPY object.
knn	numeric. Number of nearest neighbours, default is 30.
scale	logical. Whether to scale the expression matrix
verbose	logical. Whether to print calculation progress.
...	Parameters passing to igraph function

Value

An FSPY object with cluster

Examples

```
if (FALSE) {
fspy <- runPhenograph(fspy, knn = 30, verbose = TRUE)
}
```

---

runPseudotime *Calculation of Pseudotime*

Description

calculation of Pseudotime based on KNN

Usage

```
runPseudotime(
  object,
  mode = "undirected",
  dim.type = c("raw", "pca", "tsne", "dc", "umap"),
  dim.use = 1:2,
  verbose = FALSE,
  ...
)
```

Arguments

object	An FSPY object
mode	character. Specifies how igraph should interpret the supplied matrix. Possible values are: directed, undirected, upper, lower, max, min, plus.
dim.type	character. Type of dimensionality reduction method used to calculate pseudotime: raw, umap, tsne, dc and pca. By default is raw.
dim.use	numeric. Dimensions used to calculate pseudotime
verbose	logical. Whether to print calculation progress.
...	Parameters passing to calculation function.

Value

An FSPY object

Examples

```
if (FALSE) {

fspy <- runPseudotime(fspy, verbose = TRUE, dim.type = "raw")
fspy <- runPseudotime(fspy, verbose = TRUE, dim.type = "umap", dim.use = 1:2)
fspy <- runPseudotime(fspy, verbose = TRUE, dim.type = "tsne", dim.use = 1:2)
fspy <- runPseudotime(fspy, verbose = TRUE, dim.type = "dc", dim.use = 1:3)
fspy <- runPseudotime(fspy, verbose = TRUE, dim.type = "pca", dim.use = 1:3)

# tSNE plot colored by pseudotime
plot2D(fspy, item.use = c("tSNE_1", "tSNE_2"), category = "numeric",
       size = 1, color.by = "pseudotime") +
       scale_colour_gradientn(colors = c("#F4D31D", "#FF3222","#7A06A0"))
# UMAP plot colored by pseudotime
plot2D(fspy, item.use = c("UMAP_1", "UMAP_2"), category = "numeric",
       size = 1, color.by = "pseudotime") +
       scale_colour_gradientn(colors = c("#F4D31D", "#FF3222","#7A06A0"))
}
```

---

runSOM *calculation SOM in FSPY object*

Description

Build a self-organizing map

Usage

```
runSOM(
  object,
  xdim = 6,
  ydim = 6,
  rlen = 8,
  mst = 1,
  alpha = c(0.05, 0.01),
  radius = 1,
  init = FALSE,
  distf = 2,
  codes = NULL,
  importance = NULL,
  method = "euclidean",
  verbose = FALSE,
  ...
)
```

Arguments

object	an FSPY object
xdim	Width of the grid.
ydim	Hight of the grid.

rlen	Number of times to loop over the training data for each MST
mst	Number of times to build an MST
alpha	Start and end learning rate
radius	Start and end radius
init	Initialize cluster centers in a non-random way
distf	Distance function (1=manhattan, 2=euclidean, 3=chebyshev, 4=cosine)
codes	Cluster centers to start with
importance	array with numeric values. Parameters will be scaled according to importance
method	the distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given. See dist
verbose	logical. Whether to print calculation progress.
...	Parameters passing to SOM function

Value

an FSPY object with som.id in FSPY object

References

This code is strongly based on the SOM function. Which is developed by Sofie Van Gassen, Britt Callebaut and Yvan Saeys (2018).

See Also

BuildSOM

SOM

Examples

```
if (FALSE) {
fspy <- runSOM(fspy, xdim = 10, ydim = 10, verbose = TRUE)
}
```

---

runTSNE *Calculate t-Distributed Stochastic Neighbor Embedding in FSPY*

Description

Calculate t-Distributed Stochastic Neighbor Embedding in FSPY

Usage

```
runTSNE(
  object,
  dims = 2,
  initial_dims = 50,
  perplexity = 30,
  theta = 0.5,
  check_duplicates = TRUE,
  pca = TRUE,
  max_iter = 1000,
  verbose = FALSE,
  is_distance = FALSE,
  Y_init = NULL,
  pca_center = TRUE,
  pca_scale = FALSE,
  ...
)
```

Arguments

object	an FSPY object
dims	integer, Output dimensionality (default: 2)
initial_dims	integer. the number of dimensions that should be retained in the initial PCA step (default: 50). See Rtsne
perplexity	numeric. Perplexity parameter. See Rtsne
theta	numeric. Speed/accuracy trade-off (increase for less accuracy), set to 0.0 for exact TSNE (default: 0.5). See Rtsne
check_duplicates	logical. Checks whether duplicates are present. It is best to make sure there are no duplicates present and set this option to FALSE, especially for large datasets (default: TRUE). See Rtsne
pca, max_iter, is_distance, Y_init, pca_center, pca_scale	See Rtsne
verbose	logical. Whether to print calculation progress.
...	Parameters passing to Rtsne function

Value

An FSPY object

References

Maaten, L. Van Der, 2014. Accelerating t-SNE using Tree-Based Algorithms. Journal of Machine Learning Research, 15, p.3221-3245.

van der Maaten, L.J.P. & Hinton, G.E., 2008. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research, 9, pp.2579-2605.

See Also

Rtsne

Examples

```
if (FALSE) {

fspy <- runTSNE(fspy, dims = 2, verbose = TRUE)
fspy <- runTSNE(fspy, dims = 2, perplexity = 20, verbose = TRUE)

}
```

`runUMAP` *Calculating UMAP*

Description

Calculate Uniform Manifold Approximation and Projection in FSPY

Usage

```
runUMAP(
  object,
  umap.config = umap.defaults,
  n_neighbors = 30,
  dims = 2,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	an FSPY object
`umap.config`	object of class umap.config. See `umap`.
`n_neighbors`	numeric. Number of neighbors
`dims`	numeric. Dim of umap, you can also change it in umap.config.
`verbose`	logical. Whether to print calculation progress.
`...`	Options to pass on to the `umap` function

Value

An FSPY object

See Also

`umap`

Examples

```
if (FALSE) {

fspy <- runUMAP(fspy, verbose = TRUE)
fspy <- runUMAP(fspy, n_neighbors = 20, verbose = TRUE)

}
```

`runWalk` *Walk between root cells and leaf cells*

Description

Walk between root cells and leaf cells

Usage

```
runWalk(
  object,
  mode = c("undirected", "directed", "max", "min", "upper", "lower", "plus"),
  max.run.forward = 20,
  backward.walk = FALSE,
  max.run.backward = 20,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	An FSPY object
`mode`	character. Specifies how igraph should interpret the supplied matrix. Possible values are: undirected, directed, upper, lower, max, min, plus. By default is undirected.
`max.run.forward`	numeric. Maximum cycles of forward walk.
`backward.walk`	logical. Whether to run backward walk.
`max.run.backward`	numeric. Maximum cycles of backward walk.
`verbose`	logical. Whether to print calculation progress.
`...`	Parameters passing to calculation function.

Value

An FSPY object

Examples

```
if (FALSE) {
  fspy <- runWalk(fspy, verbose = TRUE)
  fspy <- runWalk(fspy, backward.walk = FALSE, verbose = TRUE)
}
```

`subsetFSPY` *subset FSPY object*

Description

This subsets an FSPY object by given a list of cells or cluster id. This function will subset all results without recalculating them, such as knn, PCA, tSNE, umap and pseudotime. For instance, you can choose recalculate PCA and tSNE and destiny scores by paramter recalculate.

Usage

```
subsetFSPY(object, cells = NULL, knn = NA, verbose = FALSE)
```

Arguments

`object`	An FSPY object
`cells`	vector, Names of the cells to retain.
`knn`	numeric. If is NA, the KNN will be equal to the knn number in the input FSPY object.
`verbose`	logic. Whether to print calculation progress.

Value

An FSPY object

Examples

```
if (FALSE) {

cells <- test.meta.data$cell[which(test.meta.data$stage == "D0")]
sub.fspy <- subsetFSPY(fspy, cells = cells)
sub.fspy

}
```

updateClustMeta *Update clusters' meta information of FSPY*

Description

Update clusters' meta information of FSPY

Usage

```
updateClustMeta(object, verbose = TRUE)
```

Arguments

object	An FSPY object
verbose	logical. Whether to print calculation progress.

Value

An FSPY object

Examples

```
if (FALSE) {
fspy <- updateClustMeta(fspy)
}
```

updatePlotMeta *Update plot meta information of FSPY*

Description

Update plot meta information of FSPY

Usage

```
updatePlotMeta(object, verbose = TRUE)
```

Arguments

object	An FSPY object
verbose	logical. Whether to print calculation progress.

Value

An FSPY object

Examples

```
if (FALSE) {
fspy <- updatePlotMeta(fspy)
}
```

Index