Package ‘miloR’

April 2, 2024

Type Package

Title Differential neighbourhood abundance testing on a graph

Version 1.10.0

Description Milo performs single-cell differential abundance testing. Cell states are modelled as representative neighbourhoods on a nearest neighbour graph. Hypothesis testing is performed using a negative binomial generalized linear model.

License GPL-3 + file LICENSE

Encoding UTF-8

URL https://marionilab.github.io/miloR

BugReports https://github.com/MarioniLab/miloR/issues

biocViews SingleCell, MultipleComparison, FunctionalGenomics, Software

Depends R (>= 4.0.0), edgeR

Imports BiocNeighbors, BiocGenerics, SingleCellExperiment, Matrix (>= 1.3-0), S4Vectors, stats, stringr, methods, igraph, irlba, cowplot, BiocParallel, BiocSingular, limma, ggplot2, tibble, matrixStats, ggraph, gtools, SummarizedExperiment, patchwork, tidyr, dplyr, ggrepel, ggbeeswarm, RColorBrewer, grDevices

Suggests testthat, MASS, mvtnorm, scater, scran, covr, knitr, rmarkdown, uwot, scuttle, BiocStyle, MouseGastrulationData, MouseThymusAgeing, magick, RCurl, curl, graphics

RoxygenNote 7.2.3

NeedsCompilation no

Collate 'AllClasses.R' 'AllGenerics.R' 'buildFromAdjacency.R'
 'buildGraph.R' 'calcNhoodExpression.R' 'calcNhoodDistance.R'
 'countCells.R' 'findNhoodMarkers.R' 'graphSpatialFDR.R'
 'makeNhoods.R' 'milo.R' 'miloR-package.R' 'methods.R'
 'plotNhoods.R' 'sim_discrete.R' 'sim_trajectory.R'
 'testNhoods.R' 'testDiffExp.R' 'utils.R' 'buildNhoodGraph.R'
 'annotateNhoods.R' 'groupNhoods.R' 'findNhoodGroupMarkers.R'

VignetteBuilder knitr
git_url https://git.bioconductor.org/packages/miloR

git_branch RELEASE_3_18

git_last_commit 7a29982

git_last_commit_date 2023-10-24

Repository Bioconductor 3.18

Date/Publication 2024-04-01

Author Mike Morgan [aut, cre],
    Emma Dann [aut, ctb]

Maintainer Mike Morgan <michael.morgan@abdn.ac.uk>

R topics documented:

miloR-package .................................................. 3
annotateNhoods ................................................. 3
buildFromAdjacency ............................................. 4
buildGraph ........................................................ 5
buildNhoodGraph ............................................... 6
calcNhoodDistance ............................................. 7
calcNhoodExpression ........................................... 8
countCells ....................................................... 9
findNhoodGroupMarkers ........................................ 10
findNhoodMarkers ............................................... 12
graphSpatialFDR ................................................ 14
groupNhoods ..................................................... 15
makeNhoods ..................................................... 17
matrixORMatrix-class .......................................... 18
Milo-class ....................................................... 19
Milo-methods .................................................... 20
plotDAbeeswarm ................................................ 22
plotNhoodCounts ............................................... 23
plotNhoodExpressionDA .......................................... 24
plotNhoodGraph ................................................ 26
plotNhoodGraphDA ............................................... 27
plotNhoodMA ..................................................... 28
plotNhoodSizeHist ............................................ 29
sim_discrete ..................................................... 30
sim_trajectory .................................................. 30
testDiffExp ..................................................... 31
testNhoods ..................................................... 33

Index 36
**The miloR package**

**Description**

The miloR package provides modular functions to perform differential abundance testing on replicated single-cell experiments. For details please see the vignettes vignette("milo_demo", package="miloR") and vignette("milo_gastrulation", package="miloR").

**Author(s)**

Mike Morgan & Emma Dann

---

**annotateNhoods**

**Add annotations from colData to DA testing results**

**Description**

This function assigns a categorical label to neighbourhoods in the differential abundance results data.frame (output of testNhoods), based on the most frequent label among cells in each neighbourhood. This can be useful to stratify DA testing results by cell types or samples. Also the fraction of cells carrying that label is stored.

**Usage**

annotateNhoods(x, da.res, coldata_col)

**Arguments**

- **x**  
  A Milo object containing single-cell gene expression and neighbourhoods.
- **da.res**  
  A data.frame containing DA results, as expected from running testNhoods.
- **coldata_col**  
  A character scalar determining which column of colData(x) stores the annotation to be added to the neighbourhoods

**Details**

For each neighbourhood, this calculates the most frequent value of colData(x)[coldata_col] among cells in the neighbourhood and assigns that value as annotation for the neighbourhood, adding a column in the da.res data.frame. In addition, a coldata_col_fraction column will be added, storing the fraction of cells carrying the assigned label. While in practice neighbourhoods are often homogeneous, one might choose to remove an annotation label when the fraction of cells with the label is too low (e.g. below 0.6).
Value

A data.frame of model results (as da.res input) with two new columns: (1) coldata_col storing the assigned label for each neighbourhood; (2) coldata_col_fraction storing the fraction of cells in the neighbourhood with the assigned label.

Author(s)

Emma Dann

Examples

NULL

---

buildFromAdjacency

Build a graph from an input adjacency matrix

Description

Construct a kNN-graph from an input adjacency matrix - either binary or distances between NNs.

Arguments

- **x**: An n X n matrix of single-cells, where values represent edges between cells; 0 values are taken to mean no edge between cells. If the matrix is not binary, then it is assumed the values are distances; 0 retain the same meaning. This behaviour can be toggled using is.binary=TRUE.
- **k**: (optional) Scalar value that represents the number of nearest neighbours in the original graph. This can also be inferred directly from the adjacency matrix x.
- **is.binary**: Logical scalar indicating if the input matrix is binary or not.

Details

This function will take a matrix as input and construct the kNN graph that it describes. If the matrix is not symmetric then the graph is assumed to be directed, whereas if the matrix is not binary, i.e. all 0’s and 1’s then the input values are taken to be distances between graph vertices; 0 values are assumed to represent a lack of edge between vertices.

Value

A Milo with the graph slot populated.

Author(s)

Mike Morgan
Examples

```r
r <- 1000
c <- 1000
k <- 35
m <- floor(matrix(runif(r*c), r, c))
for(i in seq_along(1:r)){
  m[i, sample(1:c, size=k)] <- 1
}
milo <- buildFromAdjacency(m)
```

---

**buildGraph**  
*Build a k-nearest neighbour graph*

**Description**

This function is borrowed from the old buildKNNGraph function in scran. Instead of returning an igraph object it populates the graph and distance slots in a Milo object. If the input is a SingleCellExperiment object or a matrix then it will return a de novo Milo object with the same slots filled.

**Usage**

```r
buildGraph(
x,
k = 10,
d = 50,
transposed = FALSE,
get.distance = FALSE,
reduced.dim = "PCA",
BNPARAM = KmknnParam(),
BSPARAM = bsparam(),
BPPARAM = SerialParam()
)
```

**Arguments**

- **x**: A matrix, SingleCellExperiment or Milo object containing feature X cell gene expression data.
- **k**: An integer scalar that specifies the number of nearest-neighbours to consider for the graph building.
- **d**: The number of dimensions to use if the input is a matrix of cells X reduced dimensions. If this is provided, transposed should also be set=TRUE.
- **transposed**: Logical if the input x is transposed with rows as cells.
- **get.distance**: A logical scalar whether to compute distances during graph construction.
reduced.dim  A character scalar that refers to a specific entry in the reduceDim slot of the Milo object.

BNPARAM refer to buildKNNGraph for details.

BSPARAM refer to buildKNNGraph for details.

BPPARAM refer to buildKNNGraph for details.

Details

This function computes a k-nearest neighbour graph. Each graph vertex is a single-cell connected by the edges between its neighbours. Whilst a kNN-graph is strictly directed, we remove directionality by forcing all edge weights to 1; this behaviour can be overridden by providing directed=TRUE.

If you wish to use an alternative graph structure, such as a shared-NN graph I recommend you construct this separately and add to the relevant slot in the Milo object.

Value

A Milo object with the graph and distance slots populated.

Author(s)

Mike Morgan, with KNN code written by Aaron Lun & Jonathan Griffiths.

Examples

library(SingleCellExperiment)
ux <- matrix(rpois(12000, 5), ncol=200)
vx <- log2(ux + 1)
pca <- prcomp(t(vx))
sce <- SingleCellExperiment(assays=list(counts=ux, logcounts=vx),
                           reducedDims=SimpleList(PCA=pca$x))

milo <- Milo(sce)
milo <- buildGraph(milo, d=30, transposed=TRUE)
milo

buildNhoodGraph  Build an abstracted graph of neighbourhoods for visualization

Description

Build an abstracted graph of neighbourhoods for visualization

Usage

buildNhoodGraph(x, overlap = 1)
calcNhoodDistance

Arguments

x A Milo object with a non-empty nhoods slot.
overlap A numeric scalar that thresholds graph edges based on the number of overlapping cells between neighbourhoods.

Details

This constructs a weighted graph where nodes represent neighbourhoods and edges represent the number of overlapping cells between two neighbourhoods.

Value

A Milo object containing an igraph graph in the nhoodGraph slot.

Author(s)

Emma Dann

Examples

NULL

calcNhoodDistance(x, d, reduced.dim = NULL, use.assay = "logcounts")

Description

This function will calculate Euclidean distances between single-cells in a neighbourhood using the same dimensionality as was used to construct the graph. This step follows the makeNhoods call to limit the number of distance calculations required.

Usage

Arguments

x A Milo object with a valid graph slot. If reduced.dims is not provided and there is no valid populated reducedDim slot in x, then this is computed first with d + 1 principal components.

d The number of dimensions to use for computing within-neighbourhood distances. This should be the same value used construct the graph.

reduced.dim If x is an Milo object, a character indicating the name of the reducedDim slot in the Milo object to use as (default: 'PCA'). Otherwise this should be an N X P matrix with rows in the same order as the columns of the input Milo object x.

use.assay A character scalar defining which assay slot in the Milo to use
Value
A **Milo** object with the distance slots populated.

Author(s)
Mike Morgan, Emma Dann

Examples
```r
library(SingleCellExperiment)
ux <- matrix(rpois(12000, 5), ncol=200)
vx <- log2(ux + 1)
pca <- prcomp(t(vx))

sce <- SingleCellExperiment(assays=list(counts=ux, logcounts=vx),
                           reducedDims=SimpleList(PCA=pca$x))
milo <- Milo(sce)
milo <- buildGraph(milo, d=30, transposed=TRUE)
milo <- makeNhoods(milo)
milo <- calcNhoodDistance(milo, d=30)
milo
```

---

calcNhoodExpression  *Average expression within neighbourhoods*

Description
This function calculates the mean expression of each feature in the Milo object stored in the assays slot. Neighbourhood expression data are stored in a new slot nhoodExpression.

Usage
```r
calcNhoodExpression(x, assay = "logcounts", subset.row = NULL, exprs = NULL)
```

Arguments
- **x**: A Milo object with nhoods slot populated, alternatively a NxM indicator matrix of N cells and M nhoods.
- **assay**: A character scalar that describes the assay slot to use for calculating neighbourhood expression.
- **subset.row**: A logical, integer or character vector indicating the rows of x to use for sumamrizing over cells in neighbourhoods.
- **exprs**: If x is a list of neighbourhoods, exprs is a matrix of genes X cells to use for calculating neighbourhood expression.
countCells

Details
This function computes the mean expression of each gene, subset by subset.rows where present, across the cells contained within each neighbourhood.

Value
A Milo object with the nhoodExpression slot populated.

Author(s)
Mike Morgan

Examples
require(SingleCellExperiment)
m <- matrix(rnorm(100000), ncol=100)
milo <- Milo(SingleCellExperiment(assays=list(logcounts=m)))
milo <- buildGraph(m, k=20, d=30)
milo <- makeNhoods(milo)
milo <- calcNhoodExpression(milo)
dim(nhoodExpression(milo))

Description
This function quantifies the number of cells in each neighbourhood according to an input experimental design. This forms the basis for the differential neighbourhood abundance testing.

Usage
countCells(x, samples, meta.data = NULL)

Arguments
x An Milo object with non-empty graph and nhoods slots.
samples Either a string specifying which column of data should be used to identify the experimental samples for counting, or a named vector of sample ids mapping each single cell to it’s respective sample.
meta.data A cell X variable data.frame containing study meta-data including experimental sample IDs. Assumed to be in the same order as the cells in the input Milo object.
findNhoodGroupMarkers

Details

This function generates a counts matrix of nhoods X samples, and populates the nhoodCounts slot of the input Milo object. This matrix is used downstream for differential abundance testing.

Value

A Milo object containing a counts matrix in the nhoodCounts slot.

Author(s)

Mike Morgan, Emma Dann

Examples

library(igraph)
m <- matrix(rnorm(100000), ncol=100)
milo <- buildGraph(t(m), k=20, d=10)
milo <- makeNhoods(milo, k=20, d=10, prop=0.3)

cond <- rep("A", nrow(m))
cond.a <- sample(seq_len(nrow(m)), size=floor(nrow(m)*0.25))
cond.b <- setdiff(seq_len(nrow(m)), cond.a)
cond[cond.b] <- "B"
meta.df <- data.frame(Condition=cond, Replicate=c(rep("R1", 330), rep("R2", 330), rep("R3", 340)))
meta.df$SampID <- paste(meta.df$Condition, meta.df$Replicate, sep="_")
milo <- countCells(milo, meta.data=meta.df, samples="SampID")
milo

findNhoodGroupMarkers  Identify post-hoc neighbourhood marker genes

Description

This function will perform differential gene expression analysis on groups of neighbourhoods. Adjacent and concordantly DA neighbourhoods can be defined using groupNhoods or by the user. Cells between these aggregated groups are compared. For differential gene expression based on an input design within DA neighbourhoods see testDiffExp.

Usage

findNhoodGroupMarkers(
  x,
  da.res,
  assay = "logcounts",
  aggregate.samples = FALSE,
  sample_col = NULL,
  subset.row = NULL,
findNhoodGroupMarkers

gene.offset = TRUE,
subset.nhoods = NULL,
subset.groups = NULL,
na.function = "na.pass"
)

Arguments

x A Milo object containing single-cell gene expression and neighbourhoods.
da.res A data.frame containing DA results, as expected from running testNhoods, as a NhoodGroup column specifying the grouping of neighbourhoods, as expected from
assay A character scalar determining which assays slot to extract from the Milo object to use for DGE testing.
aggregate.samples logical indicating whether the expression values for cells in the same sample and neighbourhood group should be merged for DGE testing. This allows to perform testing exploiting the replication structure in the experimental design, rather than treating single-cells as independent replicates. The function used for aggregation depends on the selected gene expression assay: if assay="counts" the expression values are summed, otherwise we take the mean.
sample_col a character scalar indicating the column in the colData storing sample information (only relevant if aggregate.samples==TRUE)
subset.row A logical, integer or character vector indicating the rows of x to use for summarizing over cells in neighbourhoods.
gene.offset A logical scalar the determines whether a per-cell offset is provided in the DGE GLM to adjust for the number of detected genes with expression > 0.
subset.nhoods A logical, integer or character vector indicating which neighbourhoods to subset before aggregation and DGE testing (default: NULL).
subset.groups A character vector indicating which groups to test for markers (default: NULL)
na.function A valid NA action function to apply, should be one of na.fail, na.omit, na.exclude, na.pass.

Details

Using a one vs. all approach, each aggregated group of cells is compared to all others using the single-cell log normalized gene expression with a GLM (for details see limma-package), or the single-cell counts using a negative binomial GLM (for details see edgeR-package). When using the latter it is recommended to set gene.offset=TRUE as this behaviour adjusts the model offsets by the number of detected genes in each cell.

Value

A data.frame of DGE results containing a log fold change and adjusted p-value for each aggregated group of neighbourhoods. If return.groups then the return value is a list with the slots groups and dge containing the aggregated neighbourhood groups per single-cell and marker gene results, respectively.
Warning: If all neighbourhoods are grouped together, then it is impossible to run `findNhoodMarkers`. In this (hopefully rare) instance, this function will return a warning and return `NULL`.

**findNhoodMarkers**  
*Identify post-hoc neighbourhood marker genes*

**Description**

This function will perform differential gene expression analysis on differentially abundant neighbourhoods, by first aggregating adjacent and concordantly DA neighbourhoods, then comparing cells *between* these aggregated groups. For differential gene expression based on an input design *within* DA neighbourhoods see `testDiffExp`.

**Arguments**

- **x**: A `Milo` object containing single-cell gene expression and neighbourhoods.
- **da.res**: A `data.frame` containing DA results, as expected from running `testNhoods`.
- **da.fdr**: A numeric scalar that determines at what FDR neighbourhoods are declared DA for the purposes of aggregating across concordantly DA neighbourhoods.
- **assay**: A character scalar determining which `assays` slot to extract from the `Milo` object to use for DGE testing.
- **aggregate.samples**: logical indicating whether the expression values for cells in the same sample and neighbourhood group should be merged for DGE testing. This allows to perform testing exploiting the replication structure in the experimental design, rather than treating single-cells as independent replicates. The function used for aggregation depends on the selected gene expression assay: if `assay="counts"` the expression values are summed, otherwise we take the mean.
- **sample_col**: a character scalar indicating the column in the colData storing sample information (only relevant if `aggregate.samples==TRUE`)
- **overlap**: A scalar integer that determines the number of cells that must overlap between adjacent neighbourhoods for merging.
- **lfc.threshold**: A scalar that determines the absolute log fold change above which neighbourhoods should be considered 'DA' for merging. Default=NULL
- **merge.discord**: A logical scalar that overrides the default behaviour and allows adjacent neighbourhoods to be merged if they have discordant log fold change signs. Using this argument is generally discouraged, but may be useful for constructing an empirical null group of cells, regardless of DA sign.
- **subset.row**: A logical, integer or character vector indicating the rows of `x` to use for summarizing over cells in neighbourhoods.
- **gene.offset**: A logical scalar the determines whether a per-cell offset is provided in the DGE GLM to adjust for the number of detected genes with expression > 0.
- **return.groups**: A logical scalar that returns a `data.frame` of the aggregated groups per single-cell. Cells that are members of non-DA neighbourhoods contain NA values.
subset.nhoods  A logical, integer or character vector indicating which neighbourhoods to subset before aggregation and DGE testing.

na.function  A valid NA action function to apply, should be one of na.fail, na.omit, na.exclude, na.pass.

compute.new  A logical scalar indicating whether to force computing a new neighbourhood adjacency matrix if already present.

Details

Louvain clustering is applied to the neighbourhood graph. This graph is first modified based on two criteria: 1) neighbourhoods share at least overlap number of cells, and 2) the DA log fold change sign is concordant. This behaviour can be modulated by setting overlap to be more or less stringent. Additionally, a threshold on the log fold-changes can be set, such that lfc.threshold is required to retain edges between adjacent neighbourhoods. Note: adjacent neighbourhoods will never be merged with opposite signs.

Using a one vs. all approach, each aggregated group of cells is compared to all others using the single-cell log normalized gene expression with a GLM (for details see limma-package), or the single-cell counts using a negative binomial GLM (for details see edgeR-package). When using the latter it is recommended to set gene.offset=TRUE as this behaviour adjusts the model offsets by the number of detected genes in each cell.

Value

A data.frame of DGE results containing a log fold change and adjusted p-value for each aggregated group of neighbourhoods. If return.groups then the return value is a list with the slots groups and dge containing the aggregated neighbourhood groups per single-cell and marker gene results, respectively.

Warning: If all neighbourhoods are grouped together, then it is impossible to run findNhoodMarkers. In this (hopefully rare) instance, this function will return a warning and return NULL.

Author(s)

Mike Morgan & Emma Dann

Examples

library(SingleCellExperiment)
ux.1 <- matrix(rpois(12000, 5), ncol=400)
ux.2 <- matrix(rpois(12000, 4), ncol=400)
ux <- rbind(ux.1, ux.2)
vx <- log2(ux + 1)
pca <- prcomp(t(vx))
sce <- SingleCellExperiment(assays=list(counts=ux, logcounts=vx),
 reducedDims=SimpleList(PCA=pca$x))
colnames(sce) <- paste0("Cell", seq_len(ncol(sce)))
milo <- Milo(sce)
milo <- buildGraph(milo, k=20, d=10, transposed=TRUE)
milo <- makeNhoods(milo, k=20, d=10, prop=0.3)
milo <- calcNhoodDistance(milo, d=10)

cond <- rep("A", ncol(milo))
cond.a <- sample(seq_len(ncol(milo)), size=floor(ncol(milo)*0.25))
cond.b <- setdiff(seq_len(ncol(milo)), cond.a)
cond[cond.b] <- "B"
meta.df <- data.frame(Condition=cond, Replicate=c(rep("R1", 132), rep("R2", 132), rep("R3", 136)))
meta.df$SampID <- paste(meta.df$Condition, meta.df$Replicate, sep="_")
milo <- countCells(milo, meta.data=meta.df, samples="SampID")

test.meta <- data.frame("Condition"=c(rep("A", 3), rep("B", 3)), "Replicate"=rep(c("R1", "R2", "R3"), 2))
test.meta$Sample <- paste(test.meta$Condition, test.meta$Replicate, sep="_")
rownames(test.meta) <- test.meta$Sample
da.res <- testNhoods(milo, design=~0 + Condition, design.df=test.meta[colnames(nhoodCounts(milo)), ])

nhood.dge <- findNhoodMarkers(milo, da.res, overlap=1, compute.new=TRUE)
nhood.dge

---

**graphSpatialFDR**

**Control the spatial FDR**

**Description**

Borrowing heavily from cydar which corrects for multiple-testing using a weighting scheme based on the volumetric overlap over hyperspheres. In the instance of graph neighbourhoods this weighting scheme can use graph connectivity or incorporate different within-neighbourhood distances for the weighted FDR calculation.

**Arguments**

- **x.nhoods**: A list of vertices and the constituent vertices of their neighbourhoods.
- **graph**: The kNN graph used to define the neighbourhoods.
- **pvalues**: A vector of p-values calculated from a GLM or other appropriate statistical test for differential neighbourhood abundance.
- **k**: A numeric integer that determines the kth nearest neighbour distance to use for the weighted FDR. Only applicable when using weighting="k-distance".
- **weighting**: A string scalar defining which weighting scheme to use. Choices are: max, k-distance, neighbour-distance or graph-overlap.
- **reduced.dimensions** *(optional)*: A matrix of cells X reduced dimensions used to calculate the kNN graph. Only necessary if this function is being used outside of testNhoods where the Milo object is not available.
- **distances** *(optional)*: A matrix of cell-to-cell distances or a list of distance matrices, 1 per neighbourhood. Only necessary if this function is being used outside of testNhoods where the Milo object is not available.
- **indices** *(optional)*: A list of neighbourhood index vertices in the same order as the input neighbourhoods. Only used for the k-distance weighting.
Each neighbourhood is weighted according to the weighting scheme defined. k-distance uses the distance to the kth nearest neighbour of the index vertex, neighbour-distance uses the average within-neighbourhood Euclidean distance in reduced dimensional space, max uses the largest within-neighbourhood distance from the index vertex, and graph-overlap uses the total number of cells overlapping between neighborhoods (distance-independent measure). The frequency-weighted version of the BH method is then applied to the p-values, as in cydar.

Value

A vector of adjusted p-values

Author(s)

Adapted by Mike Morgan, original function by Aaron Lun

Examples

NULL

Usage

groupNhoods(
x,      
da.res,  
    da.fdr = 0.1,  
    overlap = 1,  
    max.lfc.delta = NULL,  
    mergediscord = FALSE,  
    subset.nhoods = NULL,  
    compute.new = FALSE,  
    na.function = “na.pass”  
)
Arguments

- **x**: A *Milo* object containing single-cell gene expression and neighbourhoods.
- **da.res**: A *data.frame* containing DA results, as expected from running `testNhoods`.
- **da.fdr**: A numeric scalar that determines at what FDR neighbourhoods are declared DA for the purposes of aggregating across concordantly DA neighbourhoods.
- **overlap**: A scalar integer that determines the number of cells that must overlap between adjacent neighbourhoods for merging.
- **max.lfc.delta**: A scalar that determines the absolute difference in log fold change below which neighbourhoods should not be considered adjacent. Default=NULL
- **merge.discord**: A logical scalar that overrides the default behaviour and allows adjacent neighbourhoods to be merged if they have discordant log fold change signs. Using this argument is generally discouraged, but may be useful for constructing an empirical null group of cells, regardless of DA sign.
- **subset.nhoods**: A logical, integer or character vector indicating which neighbourhoods to subset before grouping. All other neighbourhoods will be assigned NA.
- **compute.new**: A logical scalar indicating whether to force computing a new neighbourhood adjacency matrix if already present.
- **na.function**: A valid NA action function to apply, should be one of `na.fail`, `na.omit`, `na.exclude`, `na.pass` (default='na.pass').

Details

Louvain clustering is applied to the neighbourhood graph. This graph is first modified based on two criteria: 1) neighbourhoods share at least `overlap` number of cells, and 2) the DA log fold change sign is concordant. This behaviour can be modulated by setting `overlap` to be more or less stringent. Additionally, a threshold on the log fold-changes can be set, such that `max.lfc.delta` is required to retain edges between adjacent neighbourhoods. Note: adjacent neighbourhoods will never be merged with opposite signs.

Value

A *data.frame* of model results (as `da.res` input) with a new column storing the assigned group label for each neighbourhood (`NhoodGroup` column)

Author(s)

Emma Dann & Mike Morgan
**Description**

This function randomly samples vertices on a graph to define neighbourhoods. These are then refined by either computing the median profile for the neighbourhood in reduced dimensional space and selecting the nearest vertex to this position (refinement_scheme = "reduced_dim"), or by computing the vertex with the highest number of triangles within the neighborhood (refinement_scheme = "graph"). Thus, multiple neighbourhoods may be collapsed down together to prevent oversampling the graph space.

**Usage**

```r
makeNhoods(
  x,
  prop = 0.1,
  k = 21,
  d = 30,
  refined = TRUE,
  reduced_dims = "PCA",
  refinement_scheme = "reduced_dim"
)
```

**Arguments**

- **x**: A `Milo` object with a non-empty graph slot. Alternatively an igraph object on which neighbourhoods will be defined.
- **prop**: A double scalar that defines what proportion of graph vertices to randomly sample. Must be $0 < \text{prop} < 1$.
- **k**: An integer scalar - the same $k$ used to construct the input graph.
- **d**: The number of dimensions to use if the input is a matrix of cells X reduced dimensions.
- **refined**: A logical scalar that determines the sampling behavior, default=TRUE implements a refined sampling scheme, specified by the refinement_scheme argument.
- **reduced_dims**: If `x` is an Milo object, a character indicating the name of the reducedDim slot in the Milo object to use as (default: 'PCA'). If `x` is an igraph object, a matrix of vertices X reduced dimensions with rownames() set to correspond to the cellIDs.
- **refinement_scheme**: A character scalar that defines the sampling scheme, either "reduced_dim" or "graph". Default is "reduced_dim".
matrixORMatrix-class

Details
This function randomly samples graph vertices, then refines them to collapse down the number of
neighbourhoods to be tested. The refinement behaviour can be turned off by setting refine=FALSE,
however, we do not recommend this as neighbourhoods will contain a lot of redundancy and lead
to an unnecessarily larger multiple-testing burden.

Value
A Milo object containing a list of vertices and the indices of vertices that constitute the neighbour-
hoods in the nhoods slot. If the input is a igraph object then the output is a matrix containing a list
of vertices and the indices of vertices that constitute the neighbourhoods.

Author(s)
Emma Dann, Mike Morgan

Examples
require(igraph)
m <- matrix(rnorm(100000), ncol=100)
milo <- buildGraph(m, d=10)
milo <- makeNhoods(milo, prop=0.1)
milo

matrixORMatrix-class  The Milo container class

Description
The Milo container class

Slots
- graph An igraph object that represents the kNN graph
- nhoods A CxN binary sparse matrix mapping cells to the neighbourhoods they belong to
- nhoodDistances An list of PxN sparse matrices of Euclidean distances between vertices in each
neighbourhood, one matrix per neighbourhood
- nhoodCounts An NxM sparse matrix of cells counts in each neighbourhood across M samples
- nhoodIndex A list of the index vertices for each neighbourhood
- nhoodExpression An GxN matrix of genes X neighbourhoods containing average gene expression
levels across cells in each neighbourhood
- nhoodReducedDim a list of reduced dimensional representations of neighbourhoods, including pro-
ductions into lower dimension space
- nhoodGraph an igraph object that represents the graph of neighbourhoods
- .k A hidden slot that stores the value of k used for graph building
The Milo class extends the SingleCellExperiment class and is designed to work with neighbourhoods of cells. Therefore, it inherits from the SingleCellExperiment class and follows the same usage conventions. There is additional support for cell-to-cell distances via distance, and the KNN-graph used to define the neighbourhoods.

Usage

```r
Milo(
  ..., 
  graph = list(),
  nhoodDistances = Matrix(0L, sparse = TRUE),
  nhoods = Matrix(0L, sparse = TRUE),
  nhoodCounts = Matrix(0L, sparse = TRUE),
  nhoodIndex = list(),
  nhoodExpression = Matrix(0L, sparse = TRUE),
  .k = NULL
)
```

Arguments

- `...` Arguments passed to the Milo constructor to fill the slots of the base class. This should be either a `SingleCellExperiment` or matrix of features X cells.
- `graph` An igraph object or list of adjacent vertices that represents the KNN-graph.
- `nhoodDistances` A list containing sparse matrices of cell-to-cell distances for cells in the same neighbourhoods, one list entry per neighbourhood.
- `nhoods` A list of graph vertices, each containing the indices of the constituent graph vertices in the respective neighbourhood.
- `nhoodCounts` A matrix of neighbourhood X sample counts of the number of cells in each neighbourhood derived from the respective samples.
- `nhoodIndex` A list of cells that are the neighborhood index cells.
- `nhoodExpression` A matrix of gene X neighbourhood expression.
- `.k` An integer value. The same value used to build the k-NN graph if already computed.

Details

In this class the underlying structure is the gene/feature X cell expression data. The additional slots provide a link between these single cells and the neighbourhood representation. This can be
further extended by the use of an abstracted graph for visualisation that preserves the structure of the single-cell KNN-graph

A Milo object can also be constructed by inputting a feature X cell gene expression matrix. In this case it simply constructs a SingleCellExperiment and fills the relevant slots, such as reducedDims.

Value

a Milo object

Author(s)

Mike Morgan

Examples

```r
library(SingleCellExperiment)
ux <- matrix(rpois(12000, 5), ncol=200)
vx <- log2(ux + 1)
pca <- prcomp(t(vx))

sce <- SingleCellExperiment(assays=list(counts=ux, logcounts=vx),
                         reducedDims=SimpleList(PCA=pca$x))

milo <- Milo(sce)
milo
```

---

### Milo-methods

*Get and set methods for Milo objects*

---

Description

Get and set methods for Milo object slots. Generally speaking these methods are used internally, but they allow the user to assign their own externally computed values - should be used *with caution*.

Getters

In the following descriptions `x` is always a Milo object.

`graph(x)`: Returns an igraph object representation of the KNN-graph, with number of vertices equal to the number of single-cells.

`nhoodDistances(x)`: Returns a list of sparse matrix of cell-to-cell distances between nearest neighbours, one list entry per neighbourhood. Largely used internally for computing the k-distance weighting in `graphSpatialFDR`.

`nhoodCounts(x)`: Returns a NxM sparse matrix of cell counts in each of N neighbourhoods with respect to the M experimental samples defined.

`nhoodExpression(x)`: Returns a GxN matrix of gene expression values.
Milo-methods

nhoodIndex(x): Returns a list of the single-cells that are the neighbourhood indices.
nhoodReducedDim(x): Returns an N\times P matrix of reduced dimension positions. Either generated by projectNhoodExpression(x) or by providing an N\times P matrix (see setter method below).
nhoods(x): Returns a sparse matrix of \(C\times N\) mapping of \(C\) single-cells to \(N\) neighbourhoods.
nhoodGraph(x): Returns an igraph object representation of the graph of neighbourhoods, with number of vertices equal to the number of neighbourhoods.
nhoodAdjacency(x): Returns a matrix of \(N\) by \(N\) neighbourhoods with entries of 1 where neighbourhoods share cells, and 0 elsewhere.

Setters

In the following descriptions \(x\) is always a Milo object.

\texttt{graph(x) <- value}: Populates the graph slot with \(value\) - this should be a valid graph representation in either igraph or list format.

\texttt{nhoodDistances(x) <- value}: Replaces the internally computed neighbourhood distances. This is normally computed internally during graph building, but can be defined externally. Must be a list with one entry per neighbourhood containing the cell-to-cell distances for the cells within that neighbourhood.

\texttt{nhoodCounts(x) <- value}: Replaces the neighbourhood counts matrix. This is normally computed and assigned by \texttt{countCells}, however, it can also be user-defined.

\texttt{nhoodExpression(x) <- value}: Replaces the \texttt{nhoodExpression} slot. This is calculated internally by \texttt{calcNhoodExpression}, which calculates the mean expression. An alternative summary function can be used to assign an alternative in this way.

\texttt{nhoodIndex(x) <- value}: Replaces the list of neighbourhood indices. This is provided purely for completeness, and is usually only set internally in \texttt{makeNhoods}.

\texttt{nhoodReducedDim(x) <- value}: Replaces the reduced dimensional representation or projection of neighbourhoods. This can be useful for externally computed projections or representations.

\texttt{nhoods(x) <- value}: Replaces the neighbourhood matrix. Generally use of this function is discouraged, however, it may be useful for users to define their own bespoke neighbourhoods by some means.

\texttt{nhoodGraph(x) <- value}: Populates the \texttt{nhoodGraph} slot with \(value\) - this should be a valid graph representation in either igraph or list format.

\texttt{nhoodAdjacency(x) <- value}: Populates the \texttt{nhoodAdjacency} slot with \(value\) - this should be a \(N\) by \(N\) matrix with elements denoting which neighbourhoods share cells

Miscellaneous

A collection of non-getter and setter methods that operate on Milo objects.

\texttt{show(x)}: Prints information to the console regarding the Milo object.

Author(s)

Mike Morgan
Examples

```r
eexample(Milo, echo=FALSE)
show(milo)
```

---

### plotDAbeeswarm

**Visualize DA results as a beeswarm plot**

**Description**

Visualize DA results as a beeswarm plot.

**Usage**

```r
plotDAbeeswarm(da.res, group.by = NULL, alpha = 0.1, subset.nhoods = NULL)
```

**Arguments**

- `da.res`: A data.frame of DA testing results.
- `group.by`: A character scalar determining which column of `da.res` to use for grouping. This can be a column added to the DA testing results using the `annotateNhoods` function. If `da.res[, group.by]` is a character or a numeric, the function will coerce it to a factor (see details) (default: NULL, no grouping).
- `alpha`: Significance level for Spatial FDR (default: 0.1).
- `subset.nhoods`: A logical, integer or character vector indicating a subset of nhoods to show in plot (default: NULL, no subsetting).

**Details**

The `group.by` variable will be coerced to a factor. If you want the variables in `group.by` to be in a given order make sure you set the column to a factor with the levels in the right order before running the function.

**Value**

A `ggplot` object.

**Author(s)**

Emma Dann

**Examples**

```r
NULL
```
plotNhoodCounts

Plot the number of cells in a neighbourhood per sample and condition

**Description**

Plot the number of cells in a neighbourhood per sample and condition

**Usage**

```r
plotNhoodCounts(x, subset.nhoods, design.df, condition, n_col = 3)
```

**Arguments**

- `x`: A Milo object with a non-empty nhoodCounts slot.
- `subset.nhoods`: A logical, integer or character vector indicating the rows of nhoodCounts(x) to use for plotting. If you use a logical vector, make sure the length matches nrow(nhoodCounts(x)).
- `design.df`: A data.frame which matches samples to a condition of interest. The row names should correspond to the samples. You can use the same design.df that you already used in the testNhoods function.
- `condition`: String specifying the condition of interest. Has to be a column in the design.
- `n_col`: Number of columns in the output ggplot.

**Value**

A ggplot-class object

**Author(s)**

Nick Hirschmüller

**Examples**

```r
require(SingleCellExperiment)
ux.1 <- matrix(rpois(12000, 5), ncol=300)
ux.2 <- matrix(rpois(12000, 4), ncol=300)
ux <- rbind(ux.1, ux.2)
vx <- log2(ux + 1)
pca <- prcomp(t(vx))
sce <- SingleCellExperiment(assays=list(counts=ux, logcounts=vx),
                           reducedDims=SimpleList(PCA=pca$x))
milo <- Milo(sce)
milo <- buildGraph(milo, k=20, d=10, transposed=TRUE)
milo <- makeNhoods(milo, k=20, d=10, prop=0.3)
milo <- calcNhoodDistance(milo, d=10)
cond <- sample(c("A","B","C"),300,replace=TRUE)
```
plotNhoodExpressionDA

meta.df <- data.frame(Condition=cond, Replicate=c(rep("R1", 100), rep("R2", 100), rep("R3", 100)))
meta.df$SampID <- paste(meta.df$Condition, meta.df$Replicate, sep="_")

milo <- countCells(milo, meta.data=meta.df, samples="SampID")

design.mtx <- data.frame("Condition"=c(rep("A", 3), rep("B", 3), rep("C",3)),
                         "Replicate"=rep(c("R1", "R2", "R3"), 3))
design.mtx$SampID <- paste(design.mtx$Condition, design.mtx$Replicate, sep="_")
rownames(design.mtx) <- design.mtx$SampID

plotNhoodCounts(x = milo,
                subset.nhoods = c(1,2),
                design.df = design.mtx,
                condition = "Condition")

plotNhoodExpressionDA  Visualize gene expression in neighbourhoods

Description
Plots the average gene expression in neighbourhoods, sorted by DA fold-change
Plots the average gene expression in neighbourhood groups

Usage

plotNhoodExpressionDA(
  x,
  da.res,
  features,
  alpha = 0.1,
  subset.nhoods = NULL,
  cluster_features = FALSE,
  assay = "logcounts",
  scale_to_1 = FALSE,
  show_rownames = TRUE,
  highlight_features = NULL)

plotNhoodExpressionGroups(
  x,
  da.res,
  features,
  alpha = 0.1,
  subset.nhoods = NULL,
  cluster_features = FALSE,
  assay = "logcounts",
  scale_to_1 = FALSE,
plotNhoodExpressionDA

```r
  show_rownames = TRUE,
  highlight_features = NULL,
  grid.space = "free"
)
```

**Arguments**

- `x`  
  A [Milo](https://example.com) object

- `da.res`  
  a data.frame of DA testing results

- `features`  
  a character vector of features to plot (they must be in rownames(x))

- `alpha`  
  significance level for Spatial FDR (default: 0.1)

- `subset.nhoods`  
  A logical, integer or character vector indicating a subset of nhoods to show in plot (default: NULL, no subsetting)

- `cluster_features`  
  logical indicating whether features should be clustered with hierarchical clustering. If FALSE then the order in features is maintained (default: FALSE)

- `assay`  
  A character scalar that describes the assay slot to use for calculating neighbourhood expression. (default: logcounts) Of note: neighbourhood expression will be computed only if the requested features are not in the nhoodExpression slot of the milo object. If you wish to plot average neighbourhood expression from a different assay, you should run calcNhoodExpression(x) with the desired assay.

- `scale_to_1`  
  A logical scalar to re-scale gene expression values between 0 and 1 for visualisation.

- `show_rownames`  
  A logical scalar whether to plot rownames or not. Generally useful to set this to show_rownames=FALSE when plotting many genes.

- `highlight_features`  
  A character vector of feature names that should be highlighted on the right side of the heatmap. Generally useful in conjunction to show_rownames=FALSE, if you are interested in only a few features

- `grid.space`  
  a character setting the space parameter for facet.grid (’fixed’ for equally sized facets, ’free’ to adapt the size of facet to number of neighbourhoods in group)

**Value**

- a ggplot object
- a ggplot object

**Author(s)**

Emma Dann
Examples

```r
NULL
NULL
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotNhoodGraph</td>
<td>Plot graph of neighbourhood</td>
</tr>
</tbody>
</table>

**Description**

Visualize graph of neighbourhoods

**Usage**

```r
plotNhoodGraph(
  x,layout = "UMAP",
  colour_by = NA,
  subset.nhoods = NULL,
  size_range = c(0.5, 3),
  node_stroke = 0.3,
  ...
)
```

**Arguments**

- `x` A `Milo` object
- `layout` this can be (a) a character indicating the name of the `reducedDim` slot in the `Milo` object to use for layout (default: 'UMAP') (b) an igraph layout object
- `colour_by` this can be a data.frame of milo results or a character corresponding to a column in colData
- `subset.nhoods` A logical, integer or character vector indicating a subset of nhoods to show in plot (default: NULL, no subsetting)
- `size_range` a numeric vector indicating the range of node sizes to use for plotting (to avoid overplotting in the graph)
- `node_stroke` a numeric indicating the desired thickness of the border around each node
- `...` arguments to pass to `ggraph`

**Value**

A `ggplot`-class object

**Author(s)**

Emma Dann
plotNhoodGraphDA

Examples

NULL

plotNhoodGraphDA  
Plot Milo results on graph of neighbourhood

Description

Visualize log-FC estimated with differential nhood abundance testing on embedding of original single-cell dataset.
Visualize grouping of neighbourhoods obtained with groupNhoods

Usage

plotNhoodGraphDA(x, milo_res, alpha = 0.05, res_column = "logFC", ...)

plotNhoodGroups(x, milo_res, show_groups = NULL, ...)

Arguments

x  
A Milo object

milo_res  
a data.frame of milo results containing the nhoodGroup column

alpha  
significance level for Spatial FDR (default: 0.05)

res_column  
which column of milo_res object to use for color (default: logFC)

...  
arguments to pass to plotNhoodGraph

show_groups  
a character vector indicating which groups to plot all other neighbourhoods will be gray

Value

a ggplot object

Author(s)

Emma Dann

Examples

NULL

NULL
plotNhoodMA

Visualize DA results as an MAplot

Description

Visualize DA results as an MAplot

Usage

plotNhoodMA(da.res, alpha = 0.05, null.mean = 0)

Arguments

da.res A data.frame of DA testing results
alpha A numeric scalar that represents the Spatial FDR threshold for statistical significance.
null.mean A numeric scalar determining the expected value of the log fold change under the null hypothesis. default=0.

Details

MA plots provide a useful means to evaluate the distribution of log fold changes after differential abundance testing. In particular, they can be used to diagnose global shifts that occur in the presence of confounding between the number of cells acquired and the experimental variable of interest. The expected null value for the log FC distribution (grey dashed line), along with the mean observed log fold change for non-DA neighbourhoods (purple dashed line) are plotted for reference. The deviation between these two lines can give an indication of biases in the results, such as in the presence of a single strong region of DA leading to an increase in false positive DA neighbourhoods in the opposite direction.

Value

a ggplot object

Author(s)

Mike Morgan

Examples

NULL
**plotNhoodSizeHist**

Plot histogram of neighbourhood sizes

**Description**

This function plots the histogram of the number of cells belonging to each neighbourhood.

**Usage**

```r
plotNhoodSizeHist(milo, bins = 50)
```

**Arguments**

- `milo`: A Milo object with a non-empty `nhoods` slot.
- `bins`: number of bins for `geom_histogram`

**Value**

A `ggplot-class` object

**Author(s)**

Emma Dann

**Examples**

```r
require(igraph)
require(SingleCellExperiment)
ux.1 <- matrix(rpois(12000, 5), ncol=400)
ux.2 <- matrix(rpois(12000, 4), ncol=400)
ux <- rbind(ux.1, ux.2)
vx <- log2(ux + 1)
 pca <- prcomp(t(vx))

sce <- SingleCellExperiment(assays=list(counts=ux, logcounts=vx),
 reducedDims=SimpleList(PCA=pca$x))

colnames(sce) <- paste0("Cell", seq_len(ncol(sce)))
milo <- Milo(sce)
milo <- buildGraph(milo, k=20, d=10, transposed=TRUE)
milo <- makeNhoods(milo, d=10, prop=0.1)
plotNhoodSizeHist(milo)
```
**Description**

Simulated discrete groups data

**Usage**

```r
data(sim_discrete)
```

**Format**

A list containing a *Milo* object in the "mylo" slot, and a *data.frame* containing experimental metadata in the "meta" slot.

**Details**

Data are simulated single-cells in 4 distinct groups of cells. Cells in each group are assigned to 1 of 2 conditions: *A* or *B*. Specifically, the cells in block 1 are highly abundant in the *A* condition, whilst cells in block 4 are most abundant in condition *B*.

**Examples**

```r
NULL
```

---

**Description**

Simulated linear trajectory data

**Usage**

```r
data(sim_trajectory)
```

**Format**

A list containing a *Milo* object in the "mylo" slot, and a *data.frame* containing experimental metadata in the "meta" slot.

**Details**

Data are simulated single-cells along a single linear trajectory. Cells are simulated from 5 groups, and assigned to 1 of 2 conditions: *A* or *B*. Data were generated using the `simulate_linear_trajectory` function in the *dyntoy* package.
**testDiffExp**

**References**

https://github.com/dynverse/dyntoy

**Examples**

NULL

---

**testDiffExp** *Perform post-hoc differential gene expression analysis*

**Description**

This function will perform differential gene expression analysis within differentially abundant neighbourhoods, by first aggregating adjacent and concordantly DA neighbourhoods, then comparing cells *within* these aggregated groups for differential gene expression using the input design. For comparing *between* DA neighbourhoods see `findNhoodMarkers`.

**Usage**

```r
testDiffExp(
  x,
  da.res,
  design,
  meta.data,
  model.contrasts = NULL,
  assay = "logcounts",
  subset.nhoods = NULL,
  subset.row = NULL,
  gene.offset = TRUE,
  n.coef = NULL,
  na.function = "na.pass"
)
```

**Arguments**

- `x` A *Milo* object containing single-cell gene expression and neighbourhoods.
- `da.res` A *data.frame* containing DA results, as expected from running `testNhoods`.
- `design` A formula or *model.matrix* object describing the experimental design for differential gene expression testing. The last component of the formula or last column of the model matrix are by default the test variable. This behaviour can be overridden by setting the `model.contrasts` argument. This should be the same as was used for DA testing.
- `meta.data` A cell X variable *data.frame* containing single-cell meta-data to which `design` refers. The order of rows (cells) must be the same as the *Milo* object columns.
model.contrasts  A string vector that defines the contrasts used to perform DA testing. This should be the same as was used for DA testing.

assay  A character scalar determining which assays slot to extract from the Milo object to use for DGE testing.

subset.nhoods  A logical, integer or character vector indicating which neighbourhoods to subset before aggregation and DGE testing (default: NULL).

subset.row  A logical, integer or character vector indicating the rows of x to use for summarizing over cells in neighbourhoods.

gene.offset  A logical scalar the determines whether a per-cell offset is provided in the DGE GLM to adjust for the number of detected genes with expression > 0.

n.coef  A numeric scalar refering to the coefficient to select from the DGE model. This is especially pertinent when passing an ordered variable and only one specific type of effects are to be tested.

na.function  A valid NA action function to apply, should be one of na.fail, na.omit, na.exclude, na.pass.

Details

Adjacent neighbourhoods are first merged based on two criteria: 1) they share at least overlap number of cells, and 2) the DA log fold change sign is concordant. This behaviour can be modulated by setting overlap to be more or less stringent. Additionally, a threshold on the log fold-changes can be set, such that lfc.threshold is required to merge adjacent neighbourhoods. Note: adjacent neighbourhoods will never be merged with opposite signs unless merge.discard=TRUE.

Within each aggregated group of cells differential gene expression testing is performed using the single-cell log normalized gene expression with a GLM (for details see limma-package), or the single-cell counts using a negative binomial GLM (for details see edgeR-package). When using single-cell data for DGE it is recommended to set gene.offset=TRUE as this behaviour adjusts the model by the number of detected genes in each cell as a proxy for differences in capture efficiency and cellular RNA content.

Value

A list containing a data.frame of DGE results for each aggregated group of neighbourhoods.

Author(s)

Mike Morgan & Emma Dann

Examples

data(sim_discrete)
milo <- Milo(sim_discrete$SCE)
milo <- buildGraph(milo, k=20, d=10, transposed=TRUE)
milo <- makeNhoods(milo, k=20, d=10, prop=0.3)
meta.df <- sim_discrete$meta
```
meta.df$SampID <- paste(meta.df$Condition, meta.df$Replicate, sep="_")
milo <- countCells(milo, meta.data=meta.df, samples="SampID")

test.meta <- data.frame("Condition"=c(rep("A", 3), rep("B", 3)), "Replicate"=rep(c("R1", "R2", "R3"), 2));
rownames(test.meta) <- test.meta$Sample
da.res <- testNhoods(milo, design=~Condition, design.df=test.meta[colnames(nhoodCounts(milo))])
da.res <- groupNhoods(milo, da.res, da.fdr=0.1)
nhood.dge <- testDiffExp(milo, da.res, design=~Condition, meta.data=meta.df)
nhood.dge
```

---

**testNhoods**  
Perform differential neighbourhood abundance testing

**Description**

This will perform differential neighbourhood abundance testing after cell counting.

**Arguments**

- `x`  
  A **Milo** object with a non-empty `nhoodCounts` slot.

- `design`  
  A formula or `model.matrix` object describing the experimental design for differential abundance testing. The last component of the formula or last column of the model matrix are by default the test variable. This behaviour can be overridden by setting the `model.contrasts` argument.

- `design.df`  
  A `data.frame` containing meta-data to which `design` refers to.

- `min.mean`  
  A scalar used to threshold neighbourhoods on the minimum average cell counts across samples.

- `model.contrasts`  
  A string vector that defines the contrasts used to perform DA testing.

- `fdr.weighting`  
  The spatial FDR weighting scheme to use. Choice from max, neighbour-distance, graph-overlap or k-distance (default). If none is passed no spatial FDR correction is performed and returns a vector of NAs.

- `robust`  
  If `robust=TRUE` then this is passed to `edgeR` and limma which use a robust estimation for the global quasilikelihood dispersion distribution. See `edgeR` and Phipson et al, 2013 for details.

- `norm.method`  
  A character scalar, either "logMS", "TMM" or "RLE". The "logMS" method normalises the counts across samples using the log columns sums of the count matrix as a model offset. "TMM" uses the trimmed mean of M-values normalisation as described in Robinson & Oshlack, 2010, whilst "RLE" uses the relative log expression method by Anders & Huber, 2010, to compute normalisation factors relative to a reference computed from the geometric mean across samples. The latter methods provides a degree of robustness against false positives when there are very large compositional differences between samples.

- `reduced.dim`  
  A character scalar referring to the reduced dimensional slot used to compute distances for the spatial FDR. This should be the same as used for graph building.
Details

This function wraps up several steps of differential abundance testing using the edgeR functions. These could be performed separately for users who want to exercise more control over their DA testing. By default this function sets the lib.sizes to the colSums(x), and uses the Quasi-Likelihood F-test in glmQLFTest for DA testing. FDR correction is performed separately as the default multiple-testing correction is inappropriate for neighbourhoods with overlapping cells.

Value

A data.frame of model results, which contain:

- **logFC**: Numeric, the log fold change between conditions, or for an ordered/continuous variable the per-unit change in (normalized) cell counts per unit-change in experimental variable.
- **logCPM**: Numeric, the log counts per million (CPM), which equates to the average log normalized cell counts across all samples.
- **F**: Numeric, the F-test statistic from the quasi-likelihood F-test implemented in edgeR.
- **PValue**: Numeric, the unadjusted p-value from the quasi-likelihood F-test.
- **FDR**: Numeric, the Benjamini & Hochberg false discovery weight computed from p.adjust.
- **Nhood**: Numeric, a unique identifier corresponding to the specific graph neighbourhood.
- **SpatialFDR**: Numeric, the weighted FDR, computed to adjust for spatial graph overlaps between neighbourhoods. For details see graphSpatialFDR.

Author(s)

Mike Morgan

Examples

```r
library(SingleCellExperiment)
ux.1 <- matrix(rpois(12000, 5), ncol=400)
ux.2 <- matrix(rpois(12000, 4), ncol=400)
ux <- rbind(ux.1, ux.2)
vx <- log2(ux + 1)
pca <- prcomp(t(vx))
sce <- SingleCellExperiment(assays=list(counts=ux, logcounts=vx),
                         reducedDims=SimpleList(PCA=pca$x))

milo <- Milo(sce)
milo <- buildGraph(milo, k=20, d=10, transposed=TRUE)
milo <- makeNhoods(milo, k=20, d=10, prop=0.3)
milo <- calcNhoodDistance(milo, d=10)
cond <- rep("A", ncol(milo))
cond.a <- sample(1:ncol(milo), size=floor(ncol(milo)*0.25))
cond.b <- setdiff(1:ncol(milo), cond.a)
cond[cond.b] <- "B"
meta.df <- data.frame(Condition=cond, Replicate=c(rep("R1", 132), rep("R2", 132), rep("R3", 136)))
meta.df$SampID <- paste(meta.df$Condition, meta.df$Replicate, sep="_")
```
milo <- countCells(milo, meta.data=meta.df, samples="SampID")

test.meta <- data.frame("Condition"=rep("A", 3), rep("B", 3)), "Replicate"=rep(c("R1", "R2", "R3"), 2))
test.meta$Sample <- paste(test.meta$Condition, test.meta$Replicate, sep="_")
rownames(test.meta) <- test.meta$Sample
da.res <- testNhoods(milo, design=~Condition, design.df=test.meta[colnames(nhoodCounts(milo))], norm.method="TMM")
da.res
Index

* datasets
  sim_discrete, 30
  sim_trajectory, 30
annotateNhoods, 3
buildFromAdjacency, 4
buildGraph, 5
buildKNNGraph, 6
buildNhoodGraph, 6
calcNhoodDistance, 7
calcNhoodExpression, 8
countCells, 9
data.frame, 12
findNhoodGroupMarkers, 10
findNhoodMarkers, 12, 31
graph(Milo-methods), 20
graph,Milo-method (Milo-methods), 20
graph<-(Milo-methods), 20
graph<-,Milo-method (Milo-methods), 20
graphSpatialFDR, 14, 34
groupNhoods, 15
makeNhoods, 17
matrixORMatrix-class, 18
Milo, 3, 4, 6–12, 14, 16–18, 20, 21, 23, 25–27, 29–33
Milo (Milo-class), 19
Milo-class, 19
Milo-methods, 20
miloR-package, 3

nhoodAdjacency (Milo-methods), 20
nhoodAdjacency,Milo-method (Milo-methods), 20
nhoodAdjacency<-,Milo-method (Milo-methods), 20
nhoodCounts (Milo-methods), 20
nhoodCounts,Milo-method (Milo-methods), 20
nhoodCounts<-(Milo-methods), 20
nhoodCounts<-,Milo-method (Milo-methods), 20
nhoodDistances (Milo-methods), 20
nhoodDistances,Milo-method (Milo-methods), 20
nhoodDistances<-(Milo-methods), 20
nhoodDistances<-,Milo-method (Milo-methods), 20
nhoodExpression (Milo-methods), 20
nhoodExpression,Milo-method (Milo-methods), 20
nhoodExpression<-(Milo-methods), 20
nhoodExpression<-,Milo-method (Milo-methods), 20
nhoodGraph (Milo-methods), 20
nhoodGraph,Milo-method (Milo-methods), 20
nhoodGraph<-(Milo-methods), 20
nhoodGraph<-,Milo-method (Milo-methods), 20
nhoodIndex (Milo-methods), 20
nhoodIndex,Milo-method (Milo-methods), 20
nhoodIndex<-(Milo-methods), 20
nhoodIndex<-,Milo-method (Milo-methods), 20
nhoodReducedDim (Milo-methods), 20
nhoodReducedDim,Milo-method (Milo-methods), 20
nhoodReducedDim<-(Milo-methods), 20
nhoodReducedDim<-,Milo-method (Milo-methods), 20
nhoods (Milo-methods), 20
nhoods,Milo-method (Milo-methods), 20
nhoods<- (Milo-methods), 20
nhoods<-,Milo-method (Milo-methods), 20

plotDAbeeswarm, 22
plotNhoodCounts, 23
plotNhoodExpressionDA, 24
plotNhoodExpressionGroups
    (plotNhoodExpressionDA), 24
plotNhoodGraph, 26
plotNhoodGraphDA, 27
plotNhoodGroups (plotNhoodGraphDA), 27
plotNhoodMA, 28
plotNhoodSizeHist, 29

show (Milo-methods), 20
show,Milo-method (Milo-methods), 20
sim_discrete, 30
sim_trajectory, 30
SingleCellExperiment, 5, 19

testDiffExp, 10, 12, 31
testNhoods, 33