Package 'miloR'

December 31, 2024

Type Package

Title Differential neighbourhood abundance testing on a graph

Version 2.3.0

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 either
a negative bionomial generalized linear model or negative binomial generalized linear mixed 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

LinkingTo Rcpp, RcppArmadillo, RcppEigen, RcppML

Depends R (>= 4.0.0), edgeR

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

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

RoxygenNote 7.3.2

NeedsCompilation no

Collate 'AllClasses.R' 'AllGenerics.R' 'buildFromAdjacency.R' 'buildGraph.R' 'calcNhoodExpression.R' 'calcNhoodDistance.R' 'checkSeparation.R' 'countCells.R' 'findNhoodMarkers.R' 'graphSpatialFDR.R' 'glmm.R' 'makeNhoods.R' 'milo.R'

'miloR-package.R' 'methods.R' 'plotNhoods.R' 'sim_discrete.R'
'sim_family.R' 'sim_nbglmm.R' 'sim_trajectory.R' 'testNhoods.R'
'testDiffExp.R' 'utils.R' 'buildNhoodGraph.R'
'annotateNhoods.R' 'groupNhoods.R' 'findNhoodGroupMarkers.R'
'RcppExports.R' 'miloR.R'

VignetteBuilder knitr

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

git_branch devel

git_last_commit 959804d

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2024-12-30

Author Mike Morgan [aut, cre] (ORCID: <https://orcid.org/0000-0003-0757-0711>),
Emma Dann [aut, ctb]

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

Contents

`miloR-package` *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")`.

Value

The miloR package

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, subset.nhoods = NULL)
```

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
subset.nhoods	A character, numeric or logical vector that will subset the annotation to the specific nhoods. If a character vector these should correspond to row names of nhoodCounts. If a logical vector then these should have the same length as nrow of nhoodCounts. If numeric, then these are assumed to correspond to indices of nhoodCounts - if the maximal index is greater than nrow(nhoodCounts(x)) an error will be produced. This is necessary if testNhoods was run using subset.nhoods=....

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 <- 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

```
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 overriden 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)
```

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 containg an `igraph` graph in the `nhoodGraph` slot.

Author(s)

Emma Dann

Examples

```
NULL
```

`calcNhoodDistance` *Calculate within neighbourhood distances*

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

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

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

```
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

```
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.

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))
```

checkSeparation *Check for separation of count distributions by variables*

Description

Check the count distributions for each nhood according to a test variable of interest. This is important for checking if there is separation in the GLMM to inform either nhood subsetting or recomputation of the NN-graph and refined nhoods.

Arguments

x	Milo object with a non-empty nhoodCounts slot.
design.df	A data.frame containing meta-data in which condition is a column variable. The rownames must be the same as, or a subset of, the colnames of nhoodCounts(x).
condition	A character scalar of the test variable contained in design.df. This should be a factor variable if it is numeric or character it will be cast to a factor variable.
min.val	A numeric scalar that sets the minimum number of counts across condition level samples, below which separation is defined.
factor.check	A logical scalar that sets the factor variable level checking. See *details* for more information.

Details

This function checks across nhoods for separation based on the separate levels of an input factor variable. It checks if *condition* is a factor variable, and if not it will cast it to a factor. Note that the function first checks for the number of unique values - if this exceeds > 50 error is generated. Users can override this behaviour with factor.check=FALSE.

Value

A logical vector of the same length as ncol(nhoodCounts(x)) where *TRUE* values represent nhoods where separation is detected. The output of this function can be used to subset nhood-based analyses e.g. testNhoods(..., subset.nhoods=checkSepartion(x, ...)).

Author(s)

Mike Morgan

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))

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"=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

check.sep <- checkSeparation(milo, design.df=test.meta, condition='Condition')
sum(check.sep)
```

`computePvalue` *Compute the p-value for the fixed effect parameters*

Description

Based on the asymptotic t-distribution, comptue the 2-tailed p-value that estimate != 0. This function is not intended to be used directly, but is included for reference or if an alternative estimate of the degrees of freedom is available.

Usage

```
computePvalue(Zscore, df)
```

Arguments

`Zscore`	A numeric vector containing the Z scores for each fixed effect parameter
`df`	A numeric vector containing the estimated degrees of freedom for each fixed effect parameter

Details

Based on sampling from a 2-tailed t-distribution with `df` degrees of freedom, compute the probability that the calculated `Zscore` is greater than or equal to what would be expected from random chance.

Value

Numeric vector of p-values, 1 per fixed effect parameter

Author(s)

Mike Morgan & Alice Kluzer

Examples

```
NULL
```

`countCells` *Count cells in neighbourhoods*

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`	A `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.

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 down-stream 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 experession 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,
  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 wheather 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 sumamrizing 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.

Examples

```
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 experession 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 concorantly 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 wheather 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 considerd '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 sumamrizing 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
```

`fitGeneticPLGlmm` *GLMM parameter estimation using pseudo-likelihood with a custom covariance matrix*

Description

Iteratively estimate GLMM fixed and random effect parameters, and variance component parameters using Fisher scoring based on the Pseudo-likelihood approximation to a Normal loglihood. This function incorporates a user-defined covariance matrix, e.g. a kinship matrix for genetic analyses.

Usage

```
fitGeneticPLGlmm(
  Z,
  X,
  K,
  muvec,
  offsets,
  curr_beta,
  curr_theta,
  curr_u,
  curr_sigma,
  curr_G,
  y,
  u_indices,
  theta_conv,
  rlevels,
  curr_disp,
  REML,
  maxit,
  solver,
  vardist
)
```

Arguments

Z	mat - sparse matrix that maps random effect variable levels to observations
X	mat - sparse matrix that maps fixed effect variables to observations
K	mat - sparse matrix that defines the known covariance patterns between individual observations. For example, a kinship matrix will then adjust for the known/estimated genetic relationships between observations.
muvec	vec vector of estimated phenotype means
offsets	vec vector of model offsets
curr_beta	vec vector of initial beta estimates
curr_theta	vec vector of initial parameter estimates
curr_u	vec of initial u estimates
curr_sigma	vec of initial sigma estimates
curr_G	mat c X c matrix of variance components
y	vec of observed counts
u_indices	List a List, each element contains the indices of Z relevant to each RE and all its levels
theta_conv	double Convergence tolerance for paramter estimates
rlevels	List containing mapping of RE variables to individual levels
curr_disp	double Dispersion parameter estimate
REML	bool - use REML for variance component estimation
maxit	int maximum number of iterations if theta_conv is FALSE
solver	string which solver to use - either HE (Haseman-Elston regression) or Fisher scoring
vardist	string which variance form to use NB = negative binomial, P=Poisson [not yet implemented]/

Details

Fit a NB-GLMM to the counts provided in *y*. The model uses an iterative approach that switches between the joint fixed and random effect parameter inference, and the variance component estimation. A pseudo-likelihood approach is adopted to minimise the log-likelihood of the model given the parameter estimates. The fixed and random effect parameters are estimated using Hendersons mixed model equations, and the variance component parameters are then estimated with the specified solver, i.e. Fisher scoring, Haseman-Elston or constrained Haseman-Elston regression. As the domain of the variance components is [0, +Inf], any negative variance component estimates will trigger the switch to the HE-NNLS solver until the model converges.

Value

A `list` containing the following elements (note: return types are dictated by Rcpp, so the R types are described here):

`FE`: `numeric` vector of fixed effect parameter estimates.

RE: list of the same length as the number of random effect variables. Each slot contains the best linear unbiased predictors (BLUPs) for the levels of the corresponding RE variable.

Sigma: numeric vector of variance component estimates, 1 per random effect variable. For this model the last variance component corresponds to the input *K* matrix.

converged: logical scalar of whether the model has reached the convergence tolerance or not.

Iters: numeric scalar with the number of iterations that the model ran for. Is strictly <= max.iter.

Dispersion: numeric scalar of the dispersion estimate computed off-line

Hessian: matrix of 2nd derivative elements from the fixed and random effect parameter inference.

SE: matrix of standard error estimates, derived from the hessian, i.e. the square roots of the diagonal elements.

t: numeric vector containing the compute t-score for each fixed effect variable.

COEFF: matrix containing the coefficient matrix from the mixed model equations.

P: matrix containing the elements of the REML projection matrix.

Vpartial: list containing the partial derivatives of the (pseudo)variance matrix with respect to each variance component.

Ginv: matrix of the inverse variance components broadcast to the full Z matrix.

Vsinv: matrix of the inverse pseudovariance.

Winv: matrix of the inverse elements of W = D^-1 V D^-1

VCOV: matrix of the variance-covariance for all model fixed and random effect variable parameter estimates. This is required to compute the degrees of freedom for the fixed effect parameter inference.

CONVLIST: list of list containing the parameter estimates and differences between current and previous iteration estimates at each model iteration. These are included for each fixed effect, random effect and variance component parameter. The list elements for each iteration are: *ThetaDiff*, *SigmaDiff*, *beta*, *u*, *sigma*.

Author(s)

Mike Morgan

Examples

```
NULL
```

---

fitGLMM — *Perform differential abundance testing using a NB-generalised linear mixed model*

Description

This function will perform DA testing per-nhood using a negative binomial generalised linear mixed model

Usage

```
fitGLMM(
  X,
  Z,
  y,
  offsets,
  init.theta = NULL,
  Kin = NULL,
  random.levels = NULL,
  REML = FALSE,
 glmm.control = list(theta.tol = 1e-06, max.iter = 100, init.sigma = NULL, init.beta =
    NULL, init.u = NULL, solver = NULL),
  dispersion = 1,
  geno.only = FALSE,
  intercept.type = "fixed",
  solver = NULL
)
```

Arguments

X	A matrix containing the fixed effects of the model.
Z	A matrix containing the random effects of the model.
y	A matrix containing the observed phenotype over each neighborhood.
offsets	A vector containing the (log) offsets to apply normalisation for different numbers of cells across samples.
init.theta	A column vector (m X 1 matrix) of initial estimates of fixed and random effect coefficients
Kin	A n x n covariance matrix to explicitly model variation between observations
random.levels	A list describing the random effects of the model, and for each, the different unique levels.
REML	A logical value denoting whether REML (Restricted Maximum Likelihood) should be run. Default is TRUE.
glmm.control	A list containing parameter values specifying the theta tolerance of the model, the maximum number of iterations to be run, initial parameter values for the fixed (init.beta) and random effects (init.u), and glmm solver (see details).
dispersion	A scalar value for the initial dispersion of the negative binomial.
geno.only	A logical value that flags the model to use either just the `matrix` 'Kin' or the supplied random effects.
intercept.type	A character scalar, either *fixed* or *random* that sets the type of the global intercept variable in the model. This only applies to the GLMM case where additional random effects variables are already included. Setting `intercept.type="fixed"` or `intercept.type="random"` will require the user to test their model for failures with each. In the case of using a kinship matrix, `intercept.type="fixed"` is set automatically.

`solver` a character value that determines which optimisation algorithm is used for the variance components. Must be either HE (Haseman-Elston regression) or Fisher (Fisher scoring).

Details

This function runs a negative binomial generalised linear mixed effects model. If mixed effects are detected in testNhoods, this function is run to solve the model. The solver defaults to the *Fisher* optimiser, and in the case of negative variance estimates it will switch to the non-negative least squares (NNLS) Haseman-Elston solver. This behaviour can be pre-set by passing `glmm.control$solver="HE"` for Haseman-Elston regression, which is the recommended solver when a covariance matrix is provided, or `glmm.control$solver="HE-NNLS"` which is the constrained HE optimisation algorithm.

Value

A list containing the GLMM output, including inference results. The list elements are as follows:

`FE:` `numeric` vector of fixed effect parameter estimates.

`RE:` `list` of the same length as the number of random effect variables. Each slot contains the best linear unbiased predictors (BLUPs) for the levels of the corresponding RE variable.

`Sigma:` `numeric` vector of variance component estimates, 1 per random effect variable.

`converged:` `logical` scalar of whether the model has reached the convergence tolerance or not.

`Iters:` `numeric` scalar with the number of iterations that the model ran for. Is strictly <= `max.iter`.

`Dispersion:` `numeric` scalar of the dispersion estimate computed off-line

`Hessian:` `matrix` of 2nd derivative elements from the fixed and random effect parameter inference.

`SE:` `matrix` of standard error estimates, derived from the hessian, i.e. the square roots of the diagonal elements.

`t:` `numeric` vector containing the compute t-score for each fixed effect variable.

`COEFF:` `matrix` containing the coefficient matrix from the mixed model equations.

`P:` `matrix` containing the elements of the REML projection matrix.

`Vpartial:` `list` containing the partial derivatives of the (pseudo)variance matrix with respect to each variance component.

`Ginv:` `matrix` of the inverse variance components broadcast to the full Z matrix.

`Vsinv:` `matrix` of the inverse pseudovariance.

`Winv:` `matrix` of the inverse elements of W = D^-1 V D^-1

`VCOV:` `matrix` of the variance-covariance for all model fixed and random effect variable parameter estimates. This is required to compute the degrees of freedom for the fixed effect parameter inference.

`DF:` `numeric` vector of the number of inferred degrees of freedom. For details see Satterthwaite_df.

`PVALS:` `numeric` vector of the compute p-values from a t-distribution with the inferred number of degrees of freedom.

`ERROR:` `list` containing Rcpp error messages - used for internal checking.

Author(s)

Mike Morgan

Examples

```
data(sim_nbglmm)
random.levels <- list("RE1"=paste("RE1", levels(as.factor(sim_nbglmm$RE1)), sep="_"),
                      "RE2"=paste("RE2", levels(as.factor(sim_nbglmm$RE2)), sep="_"))
X <- as.matrix(data.frame("Intercept"=rep(1, nrow(sim_nbglmm)), "FE2"=as.numeric(sim_nbglmm$FE2)))
Z <- as.matrix(data.frame("RE1"=paste("RE1", as.numeric(sim_nbglmm$RE1), sep="_"),
                          "RE2"=paste("RE2", as.numeric(sim_nbglmm$RE2), sep="_")))
y <- sim_nbglmm$Mean.Count
dispersion <- 0.5

glmm.control <- glmmControl.defaults()
glmm.control$theta.tol <- 1e-6
glmm.control$max.iter <- 15
model.list <- fitGLMM(X=X, Z=Z, y=y, offsets=rep(0, nrow(X)), random.levels=random.levels,
               REML = TRUE, glmm.control=glmm.control, dispersion=dispersion, solver="Fisher")
model.list
```

`fitPLGlmm` *GLMM parameter estimation using pseudo-likelihood*

Description

Iteratively estimate GLMM fixed and random effect parameters, and variance component parameters using Fisher scoring based on the Pseudo-likelihood approximation to a Normal loglihood.

Usage

```
fitPLGlmm(
  Z,
  X,
  muvec,
  offsets,
  curr_beta,
  curr_theta,
  curr_u,
  curr_sigma,
  curr_G,
  y,
  u_indices,
  theta_conv,
  rlevels,
  curr_disp,
  REML,
```

```
  maxit,
  solver,
  vardist
)
```

Arguments

Z	mat - sparse matrix that maps random effect variable levels to observations
X	mat - sparse matrix that maps fixed effect variables to observations
muvec	vec vector of estimated phenotype means
offsets	vec vector of model offsets
curr_beta	vec vector of initial beta estimates
curr_theta	vec vector of initial parameter estimates
curr_u	vec of initial u estimates
curr_sigma	vec of initial sigma estimates
curr_G	mat c X c matrix of variance components
y	vec of observed counts
u_indices	List a List, each element contains the indices of Z relevant to each RE and all its levels
theta_conv	double Convergence tolerance for paramter estimates
rlevels	List containing mapping of RE variables to individual levels
curr_disp	double Dispersion parameter estimate
REML	bool - use REML for variance component estimation
maxit	int maximum number of iterations if theta_conv is FALSE
solver	string which solver to use - either HE (Haseman-Elston regression) or Fisher scoring
vardist	string which variance form to use NB = negative binomial, P=Poisson [not yet implemented.]

Details

Fit a NB-GLMM to the counts provided in *y*. The model uses an iterative approach that switches between the joint fixed and random effect parameter inference, and the variance component estimation. A pseudo-likelihood approach is adopted to minimise the log-likelihood of the model given the parameter estimates. The fixed and random effect parameters are estimated using Hendersons mixed model equations, and the variance component parameters are then estimated with the specified solver, i.e. Fisher scoring, Haseman-Elston or constrained Haseman-Elston regression. As the domain of the variance components is [0, +Inf], any negative variance component estimates will trigger the switch to the HE-NNLS solver until the model converges.

Value

A `list` containing the following elements (note: return types are dictated by Rcpp, so the R types are described here):

`FE`: `numeric` vector of fixed effect parameter estimates.

`RE`: `list` of the same length as the number of random effect variables. Each slot contains the best linear unbiased predictors (BLUPs) for the levels of the corresponding RE variable.

`Sigma`: `numeric` vector of variance component estimates, 1 per random effect variable.

`converged`: `logical` scalar of whether the model has reached the convergence tolerance or not.

`Iters`: `numeric` scalar with the number of iterations that the model ran for. Is strictly <= `max.iter`.

`Dispersion`: `numeric` scalar of the dispersion estimate computed off-line

`Hessian`: `matrix` of 2nd derivative elements from the fixed and random effect parameter inference.

`SE`: `matrix` of standard error estimates, derived from the hessian, i.e. the square roots of the diagonal elements.

`t`: `numeric` vector containing the compute t-score for each fixed effect variable.

`COEFF`: `matrix` containing the coefficient matrix from the mixed model equations.

`P`: `matrix` containing the elements of the REML projection matrix.

`Vpartial`: `list` containing the partial derivatives of the (pseudo)variance matrix with respect to each variance component.

`Ginv`: `matrix` of the inverse variance components broadcast to the full Z matrix.

`Vsinv`: `matrix` of the inverse pseudovariance.

`Winv`: `matrix` of the inverse elements of W = D^-1 V D^-1

`VCOV`: `matrix` of the variance-covariance for all model fixed and random effect variable parameter estimates. This is required to compute the degrees of freedom for the fixed effect parameter inference.

`CONVLIST`: `list` of `list` containing the parameter estimates and differences between current and previous iteration estimates at each model iteration. These are included for each fixed effect, random effect and variance component parameter. The list elements for each iteration are: *ThetaDiff*, *SigmaDiff*, *beta*, *u*, *sigma*.

Author(s)

Mike Morgan

Examples

```
NULL
```

`glmmControl.defaults` *glmm control default values*

Description

This will give the default values for the GLMM solver

Usage

```
glmmControl.defaults(...)
```

Arguments

`...`	see `fitGLMM` for details

Details

The default values for the parameter estimation convergence is 1e-6, and the maximum number of iterations is 100. In practise if the solver converges it generally does so fairly quickly on moderately well conditioned problems. The default solver is Fisher scoring, but this will switch (with a warning produced) to the NNLS Haseman-Elston solver if negative variance estimates are found.

Value

`list` containing the default values GLMM solver. This can be saved in the user environment and then passed to testNhoods directly to modify the convergence criteria of the solver that is used.

`theta.tol`: `numeric` scalar that sets the convergence threshold for the parameter inference - this is applied globally to fixed and random effect parameters, and to the variance estimates.

`max.iter`: `numeric` scalar that sets the maximum number of iterations that the NB-GLMM will run for.

`solver`: `character` scalar that sets the solver to use. Valid values are *Fisher*, *HE* or *HE-NNLS*. See fitGLMM for details.

Author(s)

Mike Morgan

Examples

```
mmcontrol <- glmmControl.defaults()
mmcontrol
mmcontrol$solver <- "HE-NNLS"
mmcontrol
```

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 incorpate different within-neighbourhood distances for the weighted FDR calculation.

Arguments

x.nhoods	A list of vertices and the constituent vertices of their neighbourhood
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 applicaple 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.

Details

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
```

`groupNhoods` *Group neighbourhoods*

Description

This function groups overlapping and concordantly DA neighbourhoods, using the louvain community detection algorithm.

Usage

```
groupNhoods(
  x,
  da.res,
  da.fdr = 0.1,
  overlap = 1,
  max.lfc.delta = NULL,
  merge.discord = 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 concorantly 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

`initialiseG` *Construct the initial G matrix*

Description

This function maps the variance estimates onto the full `c` x `q` levels for each random effect. This ensures that the matrices commute in the NB-GLMM solver. This function is included for reference, and should not be used directly

Usage

```
initialiseG(cluster_levels, sigmas, Kin = NULL)
```

Arguments

`cluster_levels`	A `list` containing the random effect levels for each variable
`sigmas`	A `matrix` of c X 1, i.e. a column vector, containing the variance component estimates
`Kin`	A `matrix` containing a user-supplied covariance matrix

Details

Broadcast the variance component estimates to the full `c\*q` x `c\*q` matrix.

Value

`matrix` of the full broadcast variance component estimates.

Author(s)

Mike Morgan & Alice Kluzer

Examples

```
data(sim_nbglmm)
random.levels <- list("RE1"=paste("RE1", levels(as.factor(sim_nbglmm$RE1)), sep="_"),
                      "RE2"=paste("RE2", levels(as.factor(sim_nbglmm$RE2)), sep="_"))
rand.sigma <- matrix(runif(2), ncol=1)
rownames(rand.sigma) <- names(random.levels)
big.G <- initialiseG(random.levels, rand.sigma)
dim(big.G)
```

`initializeFullZ` *Construct the full Z matrix*

Description

Using a simplified version of the `n x c` Z matrix, with one column per variable, construct the fully broadcast `n x (c*q)` binary matrix that maps each individual onto the random effect variable levels. It is not intended for this function to be called by the user directly, but it can be useful to debug mappings between random effect levels and input variables.

Usage

```
initializeFullZ(Z, cluster_levels, stand.cols = FALSE)
```

Arguments

`Z`	A `n x c` matrix containing the numeric or character levels
`cluster_levels`	A `list` that maps the column names of Z onto the individual levels
`stand.cols`	A logical scalar that determines if Z* should be computed which is the row-centered and scaled version of the full Z matrix

Details

To make sure that matrices commute it is necessary to construct the full `n x c*q` matrix. This is a binary matrix where each level of each random effect occupies a column, and the samples/observations are mapped onto the correct levels based on the input Z.

Value

`matrix` Fully broadcast Z matrix with one column per random effect level for all random effect variables in the model.

Author(s)

Mike Morgan & Alice Kluzer

Examples

```
data(sim_nbglmm)
random.levels <- list("RE1"=paste("RE1", levels(as.factor(sim_nbglmm$RE1)), sep="_"),
                      "RE2"=paste("RE2", levels(as.factor(sim_nbglmm$RE2)), sep="_"))
Z <- as.matrix(data.frame("RE1"=paste("RE1", as.numeric(sim_nbglmm$RE1), sep="_"),
                          "RE2"=paste("RE2", as.numeric(sim_nbglmm$RE2), sep="_")))
fullZ <- initializeFullZ(Z, random.levels)
dim(Z)
dim(fullZ)
```

---

makeNhoods *Define neighbourhoods on a graph (fast)*

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

```
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 < 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".

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 neighbourhoods 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
```

---

matrix.trace *Compute the trace of a matrix*

Description

Exactly what it says on the tin - compute the sum of the matrix diagonal

Usage

```
matrix.trace(x)
```

Arguments

x A matrix

Details

It computes the matrix trace of a square matrix.

Value

`numeric` scalar of the matrix trace.

Author(s)

Mike Morgan

Examples

```
matrix.trace(matrix(runif(9), ncol=3, nrow=3))
```

`Milo-class` *The Milo constructor*

Description

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

```
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 constiuent 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

```
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*.

Value

See individual methods for return values

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.

`nhoodIndex(x)`: Returns a list of the single-cells that are the neighbourhood indices.

`nhoodReducedDim(x)`: Returns an NxP matrix of reduced dimension positions. Either generated by `projectNhoodExpression(x)` or by providing an NxP matrix (see setter method below).

`nhoods(x)`: Returns a sparse matrix of `CxN` 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 neighbourhods share cells, and 0 elsewhere.

Setters

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

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

`nhoodDistances(x) <- value`: Replaces the internally comptued 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.

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

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

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

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

`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.

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

`nhoodAdjacency(x) <- value`: Populates the 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.

`show(x)`: Prints information to the console regarding the `Milo` object.

Author(s)

Mike Morgan

Examples

```
example(Milo, echo=FALSE)
show(milo)
```

`miloR` *miloR*

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 bionomial generalized linear model.

`plotDAbeeswarm` *Visualize DA results as a beeswarm plot*

Description

This function constructs a beeswarm plot using the ggplot engine to visualise the distribution of log fold changes across neighbourhood annotations.

Usage

```
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

```
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

```
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

```
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)
```

```
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,
```

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

Arguments

x	A Milo 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 facent to number of neighbourhoods in group)

Value

a ggplot object

a ggplot object

Author(s)

Emma Dann

Examples

```
NULL
```

```
NULL
```

`plotNhoodGraph` *Plot graph of neighbourhood*

Description

Visualize graph of neighbourhoods

Usage

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

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). This is necessary if `testNhoods` was run using `subset.nhoods=....`
`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
`is.da`	logical scalar that tells plotNhoodGraph to order nhoods by |LFC| which can help to visually emphasise which nhoods are DA.

highlight.da	logical or numeric scalar that emphasises the DA nhoods in the layout by adjusting the transparency of the non-DA nhoods. Can only be used if is.da=TRUE, otherwise will give a warning. If highlight.da is a numeric then it explicitly sets the transparency level (must be between 0 and 1). If highlight.da is logical then the transparency is set to 0.1
...	arguments to pass to ggraph

Value

a ggplot-class object

Author(s)

Emma Dann

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.

Usage

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

Arguments

x	A Milo object
milo_res	a data.frame of milo results
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

Value

a ggplot object

Author(s)

Emma Dann

Examples

```
NULL
```

`plotNhoodGroups` *Plot graph of neighbourhoods coloring by nhoodGroups*

Description

Visualize grouping of neighbourhoods obtained with `groupNhoods`

Usage

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

Arguments

`x`	A `Milo` object
`milo_res`	a data.frame of milo results containing the `nhoodGroup` column
`show_groups`	a character vector indicating which groups to plot all other neighbourhoods will be gray
`...`	arguments to pass to `plotNhoodGraph`

Value

a `ggplot` object

Author(s)

Emma Dann

Examples

```
NULL
```

plotNhoodMA *Visualize DA results as an MAplot*

Description

Make an MAplot to visualise the relationship between DA log fold changes and neighbourhood abundance. This is a useful way to diagnose issues with the DA testing, such as large compositional biases and/or issues relating to large imbalances in numbers of cells between condition labels/levels.

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

```
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

```
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)
```

`Satterthwaite_df` *Compute degrees of freedom using Satterthwaite method*

Description

This function is not intended to be called by the user, and is included for reference

Usage

```
Satterthwaite_df(
  coeff.mat,
  mint,
  cint,
  SE,
  curr_sigma,
  curr_beta,
  V_partial,
  V_a,
  G_inv,
  random.levels
)
```

Arguments

`coeff.mat`	A `matrix` class object containing the coefficient matrix from the mixed model equations
`mint`	A numeric scalar of the number of fixed effect variables in the model
`cint`	A numeric scalar of the number of random effect variables in the model
`SE`	A `1` x `mint` `matrix`, i.e. column vector, containing the standard errors of the fixed effect parameter estimates
`curr_sigma`	A `1` x `cint` `matrix`, i.e. column vector, of the variance component parameter estimates
`curr_beta`	A `1` x `mint` `matrix`, i.e. column vector, of the fixed effect parameter estimates
`V_partial`	A `list` of the partial derivatives for each fixed and random effect variable in the model
`V_a`	A `c+m` x `c+m` variance-covariance matrix of the fixed and random effect variable parameter estimates
`G_inv`	A `nxc` `X` `nxc` inverse matrix containing the variance component estimates
`random.levels`	A `list` containing the mapping between the random effect variables and each respective set of levels for said variable.

Details

The Satterthwaite degrees of freedom are computed, which estimates the numbers of degrees of freedom in the NB-GLMM based on ratio of the squared standard errors and the product of the Jacobians of the variance-covariance matrix from the fixed effect variable parameter estimation with full variance-covariance matrix. For more details see Satterthwaite FE, Biometrics Bulletin (1946) Vol 2 No 6, pp110-114.

Value

`matrix` containing the inferred number of degrees of freedom for the specific model.

Author(s)

Mike Morgan & Alice Kluzer

Examples

```
NULL
```

`sim_discrete` *sim_discrete*

Description

Simulated discrete groups data

Usage

```
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

```
NULL
```

sim_family *sim_family*

Description

Simulated counts data from a series of simulated family trees

Usage

```
data(sim_family)
```

Format

A list containing a `data.frame` in the "DF" slot containing the mean counts and meta-data, and a `matrix` containing the kinship matrix across all families in the "IBD" slot.

Details

Data are simulated counts from 30 families and includes X and Z design matrices, as well as a single large kinship matrix. Kinships between family members are dictated by the simulated family, i.e. sibs=0.5, parent-sib=0.5, sib-grandparent=0.25, etc. These kinships, along with 2 other random effects, are used to induce a defined covariance between simulated obserations as such:

Z:= random effect design matrix, n X q G:= matrix of variance components, including kinship matrix

LL^T = Chol(ZGZ^T) := the Cholesky decomposition of the random effect contribution to the sample covariance Ysim:= simulated means based on exp(offset + Xbeta + Zb) Y = LYsim := simulated means with defined covariance

Examples

```
NULL
```

sim_nbglmm *sim_nbglmm*

Description

Simulated counts data from a NB-GLMM for a single trait

Usage

```
data(sim_nbglmm)
```

Format

A `data.frame` *sim_nbglmm* containing the following columns:

`Mean:` `numeric` containing the base mean computed as the linear combination of the simulated fixed and random effect weights multiplied by their respective weight matrices.

`Mean.Count:` `numeric` containing the integer count values randomly sampled from a negative binomail distribution with mean = *Mean* and dispersion = *r*

`r:` `numeric` containing the dispersion value used to simulate the integer counts in *Mean.Count*.

`Intercept:` `numeric` of all 1s which can be used to set the intercept term in the X design matrix.

`FE1:` `numeric` a binary fixed effect variable taking on values [0, 1]

`FE2:` `numeric` a continuous fixed effect variables

`RE1:` `numeric` a random effect variable with 10 levels

`RE2:` `numeric` a random effect variable with 7 levels

Details

Data are simulated counts from 50 samples in a single data frame, from which the X and Z design matrices, can be constructed (see examples). There are 2 random effects and 2 fixed effect variables used to simulate the count trait.

Examples

```
data(sim_nbglmm)
head(sim_nbglmm)
```

---

sim_trajectory *Simulated linear trajectory data*

Description

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 in the `simulate_linear_trajectory` function in the `dyntoy` package.

Usage

```
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.

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

```
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 sumamrizing 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.discord=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))
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)), ])
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
`kinship`	(optional) An n X n `matrix` containing pair-wise relationships between observations, such as expected relationships or computed from SNPs/SNVs/other genetic variants. Row names and column names should correspond to the column names of `nhoods(x)` and rownames of `design.df`.
`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. For a specific comparison we recommend a single contrast be passed to `testNhoods`. More details can be found in the vignette `milo_contrasts`.
`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.

`cell.sizes` A named numeric vector of cell numbers per experimental samples. Names should correspond to the columns of `nhoodCounts`. This can be used to define the model normalisation factors based on a set of numbers instead of the `colSums(nhoodCounts(x))`. The example use-case is when performing an analysis of a subset of nhoods while retaining the need to normalisation based on the numbers of cells collected for each experimental sample to avoid compositional biases. Infinite or NA values will give an error.

`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.

`REML` A logical scalar that controls the variance component behaviour to use either restricted maximum likelihood (REML) or maximum likelihood (ML). The former is recommened to account for the bias in the ML variance estimates.

`glmm.solver` A character scalar that determines which GLMM solver is applied. Must be one of: Fisher, HE or HE-NNLS. HE or HE-NNLS are recommended when supplying a user-defined covariance matrix.

`max.iters` A scalar that determines the maximum number of iterations to run the GLMM solver if it does not reach the convergence tolerance threshold.

`max.tol` A scalar that deterimines the GLMM solver convergence tolerance. It is recommended to keep this number small to provide some confidence that the parameter estimates are at least in a feasible region and close to a *local* optimum

`subset.nhoods` A character, numeric or logical vector that will subset the analysis to the specific nhoods. If a character vector these should correspond to row names of `nhoodCounts`. If a logical vector then these should have the same `length` as `nrow` of `nhoodCounts`. If numeric, then these are assumed to correspond to indices of `nhoodCounts` - if the maximal index is greater than `nrow(nhoodCounts(x))` an error will be produced.

`intercept.type` A character scalar, either *fixed* or *random* that sets the type of the global intercept variable in the model. This only applies to the GLMM case where additional random effects variables are already included. Setting `intercept.type="fixed"` or `intercept.type="random"` will require the user to test their model for failures with each. In the case of using a kinship matrix, `intercept.type="fixed"` is set automatically.

`fail.on.error` A logical scalar the determines the behaviour of the error reporting. Used for debugging only.

`BPPARAM` A BiocParallelParam object specifying the arguments for parallelisation. By default this will evaluate using `SerialParam()`. See `detailson` how to use parallelisation in `testNhoods`.

`force` A logical scalar that overrides the default behaviour to nicely error when N < 50 and using a mixed effect model. This is because model parameter estimation may be unstable with these sample sizes, and hence the fixed effect GLM is recommended instead. If used with the LMM, a warning will be produced.

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 contol 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. The GLMM testing cannot be performed using `edgeR`, however, a separate function `fitGLMM` can be used to fit a mixed effect model to each nhood (see `fitGLMM` docs for details).

Parallelisation is currently only enabled for the NB-GLMM and uses the BiocParallel paradigm at the level of R, and OpenMP to allow multi-threading of RCpp code. In general the GLM implementation in `glmQLFit` is sufficiently fast that it does not require parallelisation. Parallelisation requires the user to pass a BiocParallelParam object with the parallelisation arguments contained therein. This relies on the user specifying how to parallelise - for details see the `BiocParallel` package.

`model.contrasts` are used to define specific comparisons for DA testing. Currently, `testNhoods` will take the last formula variable for comparisons, however, contrasts need this to be the first variable. A future update will harmonise these behaviours for consistency. While it is strictly feasible to compute multiple contrasts at once, the recommendation, for ease of interpretability, is to compute one at a time.

If using the GLMM option, i.e. including a random effect variable in the `design` formula, then `testNhoods` will check for the sample size of the analysis. If this is less than 60 it will stop and produce an error. It is *strongly* recommended that the GLMM is not used with relatively small sample sizes, i.e. N<60, and even up to N~100 may have unstable parameter estimates across nhoods. This behaviour can be overriden by setting `force=TRUE`, but also be aware that parameter estimates may not be accurate. A warning will be produced to alert you to this fact.

Value

A `data.frame` of model results, which contain:

`logFC`**:** Numeric, the log fold change between conditions, or for an ordered/continous 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 quali-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

```
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"=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=~Condition, design.df=test.meta[colnames(nhoodCounts(milo)), ], norm.method="T
da.res
```

Index