Package ‘celda’

February 2, 2024

Title  CElullar Latent Dirichlet Allocation

Version 1.18.1

Description Celda is a suite of Bayesian hierarchical models for
clustering single-cell RNA-sequencing (scRNA-seq) data. It is able to
perform "bi-clustering" and simultaneously cluster genes into gene modules
and cells into cell subpopulations. It also contains DecontX, a novel
Bayesian method to computationally estimate and remove RNA contamination in
individual cells without empty droplet information. A variety of scRNA-seq
data visualization functions is also included.

Depends R (>= 4.0), SingleCellExperiment, Matrix

VignetteBuilder knitr

Imports plyr, foreach, ggplot2, RColorBrewer, grid, scales, tibble,
grDevices, graphics, matrixStats, doParallel, digest, methods,
reshape2, S4Vectors, data.table, Rcpp, RcppEigen, uwot,
enrichR, SummarizedExperiment, MCMCprecision, ggrepel, Rtsne,
withr, scater (>= 1.14.4), scran, dbscan, DelayedArray,
stringr, ComplexHeatmap, gridExtra, circlize

Suggests testthat, knitr, roxygen2, rmarkdown, bioMaRt, covr,
BiocManager, BiocStyle, TENxPBMCData, singleCellTK,
M3DExampleData

LinkingTo Rcpp, RcppEigen

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

BugReports https://github.com/campbio/celda/issues

biocViews SingleCell, GeneExpression, Clustering, Sequencing,
          Bayesian, ImmunoOncology, DataImport

NeedsCompilation yes

Git URL https://git.bioconductor.org/packages/celda

Git Branch RELEASE_3_18
git_last_commit 50211da1

git_last_commit_date 2023-11-04

Repository Bioconductor 3.18

Date/Publication 2024-02-02

Author Joshua Campbell [aut, cre], Shiyi Yang [aut], Zhe Wang [aut], Sean Corbett [aut], Yusuke Koga [aut]

Maintainer Joshua Campbell <camp@bu.edu>

R topics documented:

appendCeldaList ................................................. 4
availableModels ..................................................... 4
bestLogLikelihood .................................................. 5
celda ............................................................. 5
celdaCGGridSearchRes ........................................... 6
celdaCGMod ........................................................ 6
celdaCGSim ........................................................ 7
celdaClusters ....................................................... 7
celdaCMod .......................................................... 8
celdaCSim .......................................................... 9
celdaGMod .......................................................... 9
celdaGridSearch .................................................... 9
celdaGSim ........................................................... 12
celdaHeatmap ....................................................... 12
celdaModel .......................................................... 13
celdaModules ........................................................ 14
celdaPerplexity ...................................................... 15
celdaPerplexity,celdaList-method .............................. 15
celdaProbabilityMap .............................................. 16
celdatosce ........................................................... 18
celdaTsned .......................................................... 20
celdaUmap ............................................................ 22
celda_C .............................................................. 24
celda_CG ............................................................. 27
celda_G .............................................................. 31
clusterProbability ................................................. 34
compareCountMatrix ............................................... 35
contaminationSim ................................................. 36
countChecksum ..................................................... 36
countChecksum,celdaList-method ............................... 37
decontX .............................................................. 37
decontXcounts ......................................................... 41
distinctColors ......................................................... 41
eigenMatMultInt .................................................. 42
eigenMatMultNumeric ............................................. 43
factorizeMatrix .................................................. 43
fastNormProp .................................................... 45
fastNormPropLog .................................................. 45
fastNormPropSqrt .................................................. 46
featureModuleLookup ............................................. 46
featureModuleTable ............................................. 48
geneSetEnrich .................................................... 49
logLikelihood .................................................... 50
logLikelihoodHistory ............................................ 51
matrixNames ...................................................... 52
moduleHeatmap .................................................... 53
nonzero .......................................................... 56
normalizeCounts ................................................... 57
params .......................................................... 58
perplexity ......................................................... 59
plotCeldaViolin .................................................... 60
plotDecontXContamination ...................................... 62
plotDecontXMarkerExpression .................................. 63
plotDecontXMarkerPercentage .................................. 64
plotDimReduceCluster .......................................... 66
plotDimReduceFeature .......................................... 68
plotDimReduceGrid .............................................. 71
plotDimReduceModule .......................................... 74
plotGridSearchPerplexity ...................................... 77
plotHeatmap ..................................................... 78
plotRPC .......................................................... 81
recodeClusterY .................................................... 82
recodeClusterZ .................................................... 82
recursiveSplitCell ............................................... 83
recursiveSplitModule ........................................... 87
reorderCelda ........................................................ 90
reportceldaCG .................................................... 92
resamplePerplexity ............................................... 95
resList .......................................................... 96
retrieveFeatureIndex ............................................ 97
runParams ........................................................ 98
sampleCells ...................................................... 99
sampleLabel ....................................................... 99
sceCeldaC .......................................................... 100
sceCeldaCG ........................................................ 101
sceCeldaCGGridSearch .......................................... 102
sceCeldaG .......................................................... 102
selectBestModel ................................................... 103
selectFeatures .................................................... 104
semiPheatmap ..................................................... 105
simulateCells ..................................................... 110
appendCeldaList

Append two celdaList objects

Description

Returns a single celdaList representing the combination of two provided celdaList objects.

Usage

appendCeldaList(list1, list2)

Arguments

list1 A celda_list object
list2 A celda_list object to be joined with list_1

Value

A celdaList object. This object contains all resList entries and runParam records from both lists.

Examples

data(celdaCGGridSearchRes)
appendedList <- appendCeldaList(
  celdaCGGridSearchRes,
  celdaCGGridSearchRes
)

availableModels

available models

Description

available models

Usage

availableModels

Format

An object of class character of length 3.
**bestLogLikelihood**

*Get the log-likelihood*

---

**Description**

Retrieves the final log-likelihood from all iterations of Gibbs sampling used to generate a celdaModel.

**Usage**

```r
bestLogLikelihood(x, altExpName = "featureSubset")
```

```r
## S4 method for signature 'SingleCellExperiment'
bestLogLikelihood(x, altExpName = "featureSubset")
```

```r
## S4 method for signature 'celdaModel'
bestLogLikelihood(x)
```

**Arguments**

- `x`: A SingleCellExperiment object returned by celda_C, celda_G, or celda.CG, or a celda model object.
- `altExpName`: The name for the altExp slot to use. Default "featureSubset".

**Value**

Numeric. The log-likelihood at the final step of Gibbs sampling used to generate the model.

**Examples**

```r
data(sceCeldaCG)
bestLogLikelihood(sceCeldaCG)
data(celdaCGMod)
bestLogLikelihood(celdaCGMod)
```

---

**celda**

*Celda models*

---

**Description**

List of available Celda models with corresponding descriptions.

**Usage**

```r
celda()
```
**Value**

None

**Examples**

```
celda()
```

---

**celdaCGGridSearchRes  celdaCGGridSearchRes**

**Description**

Example results of old celdaGridSearch on celdaCGSim

**Usage**

```
celdaCGGridSearchRes
```

**Format**

An object as returned from old celdaGridSearch()

---

**celdaCGMod  celdaCGmod**

**Description**

celda_CG model object generated from celdaCGSim using old celda_CG function.

**Usage**

```
celdaCGMod
```

**Format**

A celda_CG object
Description

An deprecated example of simulated count matrix from the celda_CG model.

Usage

celdaCGSim

Format

A list of counts and properties as returned from old simulateCells().

celdaClusters

Get or set the cell cluster labels from a celda SingleCellExperiment object or celda model object.

Description

Return or set the cell cluster labels determined by celda_C or celda_CG models.

Usage

celdaClusters(x, altExpName = "featureSubset")

## S4 method for signature 'SingleCellExperiment'
celdaClusters(x, altExpName = "featureSubset")

## S4 method for signature 'celdaModel'
celdaClusters(x)

celdaClusters(x, altExpName = "featureSubset") <- value

## S4 replacement method for signature 'SingleCellExperiment'
celdaClusters(x, altExpName = "featureSubset") <- value

Arguments

x

Can be one of

- A SingleCellExperiment object returned by celda_C, or celda_CG, with the matrix located in the useAssay assay slot. The a altExp slot with name altExpName will be used. Rows represent features and columns represent cells.
celdaCMod

- Celda model object.

altExpName  The name for the altExp slot to use. Default "featureSubset".

value  Character vector of cell cluster labels for replacements. Works only if x is a SingleCellExperiment object.

Value

One of

- Character vector if x is a SingleCellExperiment object. Contains cell cluster labels for each cell in x.

- List if x is a celda model object. Contains cell cluster labels (for celda_C and celdaCG Models) and/or feature module labels (for celda_G and celdaCG Models).

Examples

```r
data(sceCeldaCG)
celdaClusters(sceCeldaCG)
data(celdaCGMod)
celdaClusters(celdaCGMod)
```

celdaCMod  celdaCMod

Description

Old celda_C results generated from celdaCSim

Usage

celdaCMod

Format

A celda_C object
**celdaCSim**

**Description**

An old example simulated count matrix from the celda_C model.

**Usage**

celdaCSim

**Format**

A list of counts and properties as returned from old simulateCells().

---

**celdaGMod**

**Description**

Old celda_G results generated from celdaGsim

**Usage**

celdaGMod

**Format**

A celda_G object

---

**celdaGridSearch**

Run Celda in parallel with multiple parameters

**Description**

Run Celda with different combinations of parameters and multiple chains in parallel. The variable `availableModels` contains the potential models that can be utilized. Different parameters to be tested should be stored in a list and passed to the argument `paramsTest`. Fixed parameters to be used in all models, such as `sampleLabel`, can be passed as a list to the argument `paramsFixed`. When `verbose = TRUE`, output from each chain will be sent to a log file but not be displayed in stdout.
Usage

celdaGridSearch(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  model,
  paramsTest,
  paramsFixed = NULL,
  maxIter = 200,
  nchains = 3,
  cores = 1,
  bestOnly = TRUE,
  seed = 12345,
  perplexity = TRUE,
  verbose = TRUE,
  logfilePrefix = "Celda"
)

## S4 method for signature 'SingleCellExperiment'
celdaGridSearch(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  model,
  paramsTest,
  paramsFixed = NULL,
  maxIter = 200,
  nchains = 3,
  cores = 1,
  bestOnly = TRUE,
  seed = 12345,
  perplexity = TRUE,
  verbose = TRUE,
  logfilePrefix = "Celda"
)

## S4 method for signature 'matrix'
celdaGridSearch(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  model,
  paramsTest,
  paramsFixed = NULL,
  maxIter = 200,
  nchains = 3,
  cores = 1,
  bestOnly = TRUE,
seed = 12345,
perplexity = TRUE,
verbose = TRUE,
logfilePrefix = "Celda"
)

Arguments

x A numeric matrix of counts or a SingleCellExperiment with the matrix located in
the assay slot under useAssay. Rows represent features and columns represent
cells.

useAssay A string specifying the name of the assay slot to use. Default "counts".

altExpName The name for the altExp slot to use. Default "featureSubset".

model Celda model. Options available in availableModels.

paramsTest List. A list denoting the combinations of parameters to run in a celda model. For
example, list(K = seq(5, 10), L = seq(15, 20)) will run all combinations of
K from 5 to 10 and L from 15 to 20 in model celda_CG.

paramsFixed List. A list denoting additional parameters to use in each celda model. Default
NULL.

maxIter Integer. Maximum number of iterations of sampling to perform. Default 200.

nchains Integer. Number of random cluster initializations. Default 3.

cores Integer. The number of cores to use for parallel estimation of chains. Default 1.

bestOnly Logical. Whether to return only the chain with the highest log likelihood per
combination of parameters or return all chains. Default TRUE.

seed Integer. Passed to with_seed. For reproducibility, a default value of 12345 is
used. Seed values seq(seed, (seed + nchains - 1)) will be supplied to each
chain in nchains. If NULL, no calls to with_seed are made.

perplexity Logical. Whether to calculate perplexity for each model. If FALSE, then per-
plexity can be calculated later with resamplePerplexity. Default TRUE.

verbose Logical. Whether to print log messages during celda chain execution. Default
TRUE.

logfilePrefix Character. Prefix for log files from worker threads and main process. Default
"Celda".

Value

A SingleCellExperiment object. Function parameter settings and celda model results are stored in
the metadata "celda_grid_search" slot.

See Also

celda_G for feature clustering, celda_C for clustering of cells, and celda(CG) for simultaneous
clustering of features and cells. subsetCeldaList can subset the celdaList object. selectBestModel
can get the best model for each combination of parameters.
Examples

```r
## Not run:
data(celdaCGSim)
## Run various combinations of parameters with 'celdaGridSearch'
celdaCGGridSearchRes <- celdaGridSearch(celdaCGSim$counts,
    model = "celda_CG",
    paramsTest = list(K = seq(4, 6), L = seq(9, 11)),
    paramsFixed = list(sampleLabel = celdaCGSim$sampleLabel),
    bestOnly = TRUE,
    nchains = 1,
    cores = 1)

## End(Not run)
```

---

celdaGSim

**Description**

An old example simulated count matrix from the celda_G model.

**Usage**

`celdaGSim`

**Format**

A list of counts and properties as returned from old `simulateCells()`

---

celdaHeatmap

**Description**

Render a stylable heatmap of count data based on celda clustering results.

**Usage**

```r
celdaHeatmap(
    sce,  
    useAssay = "counts",  
    altExpName = "featureSubset",  
    featureIx = NULL,  
    nfeatures = 25,  
    ...
)
```

celdaHeatmap(
  sce,
  useAssay = "counts",
  altExpName = "featureSubset",
  featureIx = NULL,
  nfeatures = 25,
  ...
)

Arguments

sce A SingleCellExperiment object returned by celda_C, celda_G, or celda.CG.
useAssay A string specifying which assay slot to use. Default "counts".
altExpName The name for the altExp slot to use. Default "featureSubset".
featureIx Integer vector. Select features for display in heatmap. If NULL, no subsetting will be performed. Default NULL. Only used for sce containing celda_C model result returned by celda_C.
nfeatures Integer. Maximum number of features to select for each gene module. Default 25. Only used for sce containing celda.CG or celda.G model results returned by celda.CG or celda.G.
...

Value

A list containing dendrogram information and the heatmap grob

See Also

‘celdaTsne()’ for generating 2-dimensional tSNE coordinates

Examples

data(sceCeldaCG)
celdaHeatmap(sceCeldaCG)

---

celdaModel Get celda model from a celda SingleCellExperiment object

Description

Return the celda model for sce returned by celda_C, celda_G or celda.CG.
Usage

celdaModel(sce, altExpName = "featureSubset")

## S4 method for signature 'SingleCellExperiment'
celdaModel(sce, altExpName = "featureSubset")

Arguments

sce A SingleCellExperiment object returned by celda_C, celda_G, or celda_CG.
altExpName The name for the altExp slot to use. Default "featureSubset".

Value

Character. The celda model. Can be one of "celda_C", "celda_G", or "celda_CG".

Examples

data(sceCeldaCG)
celdaModel(sceCeldaCG)

celdaModules

Get or set the feature module labels from a celda SingleCellExperiment object.

Description

Return or set the feature module cluster labels determined by celda_G or celda_CG models.

Usage

celdaModules(sce, altExpName = "featureSubset")

## S4 method for signature 'SingleCellExperiment'
celdaModules(sce, altExpName = "featureSubset")

celdaModules(sce, altExpName = "featureSubset") <- value

## S4 replacement method for signature 'SingleCellExperiment'
celdaModules(sce, altExpName = "featureSubset") <- value

Arguments

sce A SingleCellExperiment object returned by celda_G, or celda_CG, with the matrix located in the useAssay assay slot. Rows represent features and columns represent cells.
altExpName The name for the altExp slot to use. Default "featureSubset".
value Character vector of feature module labels for replacements. Works only if x is a SingleCellExperiment object.
Value

Character vector. Contains feature module labels for each feature in x.

Examples

data(sceCeldaCG)
celdaModules(sceCeldaCG)

Description

Returns perplexity for each model in a celdaList as calculated by ‘perplexity()’.

Usage

celdaPerplexity(celdaList)

Arguments

celdaList An object of class celdaList.

Value

List. Contains one celdaModel object for each of the parameters specified in the ‘runParams()’ of the provided celda list.

Examples

data(celdaCGGridSearchRes)
celdaCGGridModelPerplexities <- celdaPerplexity(celdaCGGridSearchRes)

celdaPerplexity, celdaList-method

Description

Returns perplexity for each model in a celdaList as calculated by ‘perplexity()’.

Usage

## S4 method for signature 'celdaList'
celdaPerplexity(celdaList)
Arguments

celdaList An object of class celdaList.

Value

List. Contains one celdaModel object for each of the parameters specified in the ‘runParams()’ of the provided celda list.

Examples

data(celdaCGGridSearchRes)
celdaCGGridModelPerplexities <- celdaPerplexity(celdaCGGridSearchRes)

celdaProbabilityMap(celdaCGGridModelPerplexities)

Description

Renders probability and relative expression heatmaps to visualize the relationship between features and cell populations (or cell populations and samples).

Usage

celdaProbabilityMap(
sce,  
useAssay = "counts",  
altExpName = "featureSubset",  
level = c("cellPopulation", "sample"),  
ncols = 100,  
col2 = circlize::colorRamp2(c(-2, 0, 2), c("#1E90FF", "#FFFFFF", "#CD2626")),  
title1 = "Absolute probability",  
title2 = "Relative expression",  
showColumnNames = TRUE,  
showRowNames = TRUE,  
rowNamesgp = grid::gpar(fontsize = 8),  
colNamesgp = grid::gpar(fontsize = 12),  
clusterRows = FALSE,  
clusterColumns = FALSE,  
showHeatmapLegend = TRUE,  
heatmapLegendParam = list(title = NULL, legend_height = grid::unit(6, "cm")),  
...  
)

## S4 method for signature 'SingleCellExperiment'
celdaProbabilityMap(
sce,  
useAssay = "counts",  
...  
)
celdaProbabilityMap

altExpName = "featureSubset",
level = c("cellPopulation", "sample"),
col2 = circlize::colorRamp2(c(-2, 0, 2), c("#1E90FF", "#FFFFFF", "#CD2626")),
title1 = "Absolute probability",
title2 = "Relative expression",
showColumnNames = TRUE,
showRowNames = TRUE,
rowNamesgp = grid::gpar(fontsize = 8),
colNamesgp = grid::gpar(fontsize = 12),
clusterRows = FALSE,
clusterColumns = FALSE,
showHeatmapLegend = TRUE,
heatmapLegendParam = list(title = NULL, legend_height = grid::unit(6, "cm")),
...
)

Arguments

cse A SingleCellExperiment object returned by celda_C, celda_G, or celda_CG.
useAssay A string specifying which assay slot to use. Default "counts".
altExpName The name for the altExp slot to use. Default "featureSubset".
level Character. One of "cellPopulation" or "Sample". "cellPopulation" will display the absolute probabilities and relative normalized expression of each module in each cell population. level = "cellPopulation" only works for celda_CG sce objects. "sample" will display the absolute probabilities and relative normalized abundance of each cell population in each sample. Default "cellPopulation".

ncols The number of colors (>1) to be in the color palette of the absolute probability heatmap.

col2 Passed to col argument of Heatmap. Set color boundaries and colors for the relative expression heatmap.
title1 Passed to column_title argument of Heatmap. Figure title for the absolute probability heatmap.
title2 Passed to column_title argument of Heatmap. Figure title for the relative expression heatmap.

Arguments

showColumnNames Passed to show_column_names argument of Heatmap. Show column names.
showRowNames Passed to show_row_names argument of Heatmap. Show row names.
rowNamesgp Passed to row_names_gp argument of Heatmap. Set row name font.
colNamesgp Passed to column_names_gp argument of Heatmap. Set column name font.
clusterRows Passed to cluster_rows argument of Heatmap. Cluster rows.
clusterColumns Passed to cluster_columns argument of Heatmap. Cluster columns.
showHeatmapLegend Passed to show_heatmap_legend argument of Heatmap. Show heatmap legend.
Passed to heatmap_legend_param argument of Heatmap. Heatmap legend parameters.

... Additional parameters passed to Heatmap.

Value

A HeatmapList object containing 2 Heatmap-class objects

See Also

celda_C for clustering cells. celda.CG for clustering features and cells

Examples

data(sceCeldaCG)
celdaProbabilityMap(sceCeldaCG)

celdatosce  Convert old celda model object to SCE object

Description

Convert a old celda model object (celda_C, celda_G, or celda.CG object) to a SingleCellExperiment object containing celda model information in metadata slot. Counts matrix is stored in the "counts" assay slot in assays.

Usage

celdatosce(
  celdaModel,
  counts,
  useAssay = "counts",
  altExpName = "featureSubset"
)

## S4 method for signature 'celda_C'
celdatosce(  
  celdaModel,
  counts,
  useAssay = "counts",
  altExpName = "featureSubset"
)

## S4 method for signature 'celda_G'
celdatosce(  
  celdaModel,
  counts,

## S4 method for signature 'celda_CG'
celdatosce(
  celdaModel,
  counts,
  useAssay = "counts",
  altExpName = "featureSubset"
)

## S4 method for signature 'celdaList'
celdatosce(
  celdaModel,
  counts,
  useAssay = "counts",
  altExpName = "featureSubset"
)

### Arguments

celdaModel  A celdaModel or celdaList object generated using older versions of celda.

counts      A numeric matrix of counts used to generate celdaModel. Dimensions and MD5 checksum will be checked by compareCountMatrix.

useAssay    A string specifying the name of the assay slot to use. Default "counts".

altExpName  The name for the altExp slot to use. Default "featureSubset".

### Value

A SingleCellExperiment object. Function parameter settings are stored in the metadata "celda_parameters" slot. Columns celda_sample_label and celda_cell_cluster in colData contain sample labels and celda cell population clusters. Column celda_feature_module in rowData contain feature modules.

### Examples

data(celdaCMod, celdaCSim)
sce <- celdatosce(celdaCMod, celdaCSim$counts)
data(celdaGMod, celdaGSim)
sce <- celdatosce(celdaGMod, celdaGSim$counts)
data(celdaCGMod, celdaCGSim)
sce <- celdatosce(celdaCGMod, celdaCGSim$counts)
data(celdaCGGridSearchRes, celdaCGSim)
sce <- celdatosce(celdaCGGridSearchRes, celdaCGSim$counts)
celdaTsne

t-Distributed Stochastic Neighbor Embedding (t-SNE) dimension reduction for celda sce object

Description

Embeds cells in two dimensions using Rtsne based on a celda model. For celda_C sce objects, PCA on the normalized counts is used to reduce the number of features before applying t-SNE. For celda_CG and celda_G sce objects, tSNE is run on module probabilities to reduce the number of features instead of using PCA. Module probabilities are square-root transformed before applying tSNE.

Usage

celdaTsne(
  sce,
  useAssay = "counts",
  altExpName = "featureSubset",
  maxCells = NULL,
  minClusterSize = 100,
  initialDims = 20,
  modules = NULL,
  perplexity = 20,
  maxIter = 2500,
  normalize = "proportion",
  scaleFactor = NULL,
  transformationFun = sqrt,
  seed = 12345
)

## S4 method for signature 'SingleCellExperiment'
celdaTsne(
  sce,
  useAssay = "counts",
  altExpName = "featureSubset",
  maxCells = NULL,
  minClusterSize = 100,
  initialDims = 20,
  modules = NULL,
  perplexity = 20,
  maxIter = 2500,
  normalize = "proportion",
  scaleFactor = NULL,
  transformationFun = sqrt,
  seed = 12345
)
Arguments

- **sce**
  A SingleCellExperiment object returned by `celda_C`, `celda_G`, or `celda_CG`.

- **useAssay**
  A string specifying which assay slot to use. Default "counts".

- **altExpName**
  The name for the altExp slot to use. Default "featureSubset".

- **maxCells**
  Integer. Maximum number of cells to plot. Cells will be randomly subsampled if `ncol(counts)` > `maxCells`. Larger numbers of cells requires more memory. If `NULL`, no subsampling will be performed. Default `NULL`.

- **minClusterSize**
  Integer. Do not subsample cell clusters below this threshold. Default 100.

- **initialDims**
  Integer. PCA will be used to reduce the dimensionality of the dataset. The top `initialDims` principal components will be used for tSNE. Default 20.

- **modules**
  Integer vector. Determines which feature modules to use for tSNE. If `NULL`, all modules will be used. Default `NULL`.

- **perplexity**
  Numeric. Perplexity parameter for tSNE. Default 20.

- **maxIter**
  Integer. Maximum number of iterations in tSNE generation. Default 2500.

- **normalize**
  Character. Passed to `normalizeCounts` in normalization step. Divides counts by the library sizes for each cell. One of 'proportion', 'cpm', 'median', or 'mean'. 'proportion' uses the total counts for each cell as the library size. 'cpm' divides the library size of each cell by one million to produce counts per million. 'median' divides the library size of each cell by the median library size across all cells. 'mean' divides the library size of each cell by the mean library size across all cells.

- **scaleFactor**
  Numeric. Sets the scale factor for cell-level normalization. This scale factor is multiplied to each cell after the library size of each cell had been adjusted in `normalize`. Default `NULL` which means no scale factor is applied.

- **transformationFun**
  Function. Applies a transformation such as 'sqrt', 'log', 'log2', 'log10', or 'log1p'. If `NULL`, no transformation will be applied. Occurs after applying normalization and scale factor. Default `NULL`.

- **seed**
  Integer. Passed to `with_seed`. For reproducibility, a default value of 12345 is used. If `NULL`, no calls to `with_seed` are made.

Value

`sce` with t-SNE coordinates (columns "celda_tSNE1" & "celda_tSNE2") added to `reducedDim(sce, "celda_tSNE")`.

Examples

```r
data(sceCeldaCG)
tsneRes <- celdaTsne(sceCeldaCG)
```
celdaUmap

*Uniform Manifold Approximation and Projection (UMAP) dimension reduction for celda sce object*

**Description**

Embeds cells in two dimensions using umap based on a celda model. For celda_C sce objects, PCA on the normalized counts is used to reduce the number of features before applying UMAP. For celda.CG sce object, UMAP is run on module probabilities to reduce the number of features instead of using PCA. Module probabilities are square-root transformed before applying UMAP.

**Usage**

```r
celdaUmap(
  sce,
  useAssay = "counts",
  altExpName = "featureSubset",
  maxCells = NULL,
  minClusterSize = 100,
  modules = NULL,
  seed = 12345,
  nNeighbors = 30,
  minDist = 0.75,
  spread = 1,
  pca = TRUE,
  initialDims = 50,
  normalize = "proportion",
  scaleFactor = NULL,
  transformationFun = sqrt,
  cores = 1,
  ...)
```

## S4 method for signature 'SingleCellExperiment'
celdaUmap(
  sce,
  useAssay = "counts",
  altExpName = "featureSubset",
  maxCells = NULL,
  minClusterSize = 100,
  modules = NULL,
  seed = 12345,
  nNeighbors = 30,
  minDist = 0.75,
  spread = 1,
  pca = TRUE,
  initialDims = 50,
)
normalize = "proportion",
scaleFactor = NULL,
transformationFun = sqrt,
cores = 1,
...
)

Arguments

sce A SingleCellExperiment object returned by celda_C, celda_G, or celda_CG.
useAssay A string specifying which assay slot to use. Default "counts".
altExpName The name for the altExp slot to use. Default "featureSubset".
maxCells Integer. Maximum number of cells to plot. Cells will be randomly subsampled if ncol(sce) > maxCells. Larger numbers of cells requires more memory. If NULL, no subsampling will be performed. Default NULL.
minClusterSize Integer. Do not subsample cell clusters below this threshold. Default 100.
modules Integer vector. Determines which features modules to use for UMAP. If NULL, all modules will be used. Default NULL.
seed Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
nNeighbors The size of local neighborhood used for manifold approximation. Larger values result in more global views of the manifold, while smaller values result in more local data being preserved. Default 30. See umap for more information.
minDist The effective minimum distance between embedded points. Smaller values will result in a more clustered/clumped embedding where nearby points on the manifold are drawn closer together, while larger values will result on a more even dispersal of points. Default 0.75. See umap for more information.
spread The effective scale of embedded points. In combination with min_dist, this determines how clustered/clumped the embedded points are. Default 1. See umap for more information.
pca Logical. Whether to perform dimensionality reduction with PCA before UMAP. Only works for celda_C sce objects.
initialDims Integer. Number of dimensions from PCA to use as input in UMAP. Default 50. Only works for celda_C sce objects.
normalize Character. Passed to normalizeCounts in normalization step. Divides counts by the library sizes for each cell. One of 'proportion', 'cpm', 'median', or 'mean'. 'proportion' uses the total counts for each cell as the library size. 'cpm' divides the library size of each cell by one million to produce counts per million. 'median' divides the library size of each cell by the median library size across all cells. 'mean' divides the library size of each cell by the mean library size across all cells.
scaleFactor Numeric. Sets the scale factor for cell-level normalization. This scale factor is multiplied to each cell after the library size of each cell had been adjusted in normalize. Default NULL which means no scale factor is applied.
transformationFun Function. Applies a transformation such as 'sqrt', 'log', 'log2', 'log10', or 'log1p'. If NULL, no transformation will be applied. Occurs after applying normalization and scale factor. Default NULL.

cores Number of threads to use. Default 1.

... Additional parameters to pass to umap.

Value

c with UMAP coordinates (columns "celda_UMAP1" & "celda_UMAP2") added to reducedDim(sce, "celda_UMAP").

Examples

data(sceCeldaCG)
umapRes <- celdaUmap(sceCeldaCG)

---

**celda_C**  
*Cell clustering with Celda*

**Description**

Clusters the columns of a count matrix containing single-cell data into K subpopulations. The useAssay assay slot in altExpName altExp slot will be used if it exists. Otherwise, the useAssay assay slot in x will be used if x is a SingleCellExperiment object.

**Usage**

celda_C(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  sampleLabel = NULL,
  K,
  alpha = 1,
  beta = 1,
  algorithm = c("EM", "Gibbs"),
  stopIter = 10,
  maxIter = 200,
  splitOnIter = 10,
  splitOnLast = TRUE,
  seed = 12345,
  nchains = 3,
  zInitialize = c("split", "random", "predefined"),
  countChecksum = NULL,
  zInit = NULL,
  logfile = NULL,
)
## S4 method for signature 'SingleCellExperiment'
celda_C(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  sampleLabel = NULL,
  K,
  alpha = 1,
  beta = 1,
  algorithm = c("EM", "Gibbs"),
  stopIter = 10,
  maxIter = 200,
  splitOnIter = 10,
  splitOnLast = TRUE,
  seed = 12345,
  nchains = 3,
  zInitialize = c("split", "random", "predefined"),
  countChecksum = NULL,
  zInit = NULL,
  logfile = NULL,
  verbose = TRUE
)

## S4 method for signature 'ANY'
celda_C(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  sampleLabel = NULL,
  K,
  alpha = 1,
  beta = 1,
  algorithm = c("EM", "Gibbs"),
  stopIter = 10,
  maxIter = 200,
  splitOnIter = 10,
  splitOnLast = TRUE,
  seed = 12345,
  nchains = 3,
  zInitialize = c("split", "random", "predefined"),
  countChecksum = NULL,
  zInit = NULL,
  logfile = NULL,
  verbose = TRUE
)
Arguments

x
A SingleCellExperiment with the matrix located in the assay slot under useAssay. Rows represent features and columns represent cells. Alternatively, any matrix-like object that can be coerced to a sparse matrix of class "dgCMatrix" can be directly used as input. The matrix will automatically be converted to a SingleCellExperiment object.

useAssay
A string specifying the name of the assay slot to use. Default "counts".

altExpName
The name for the altExp slot to use. Default "featureSubset".

sampleLabel
Vector or factor. Denotes the sample label for each cell (column) in the count matrix.

K
Integer. Number of cell populations.

alpha
Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.

beta
Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature in each cell population. Default 1.

algorithm
String. Algorithm to use for clustering cell subpopulations. One of 'EM' or 'Gibbs'. The EM algorithm is faster, especially for larger numbers of cells. However, more chains may be required to ensure a good solution is found. If 'EM' is selected, then 'stopIter' will be automatically set to 1. Default 'EM'.

stopIter
Integer. Number of iterations without improvement in the log likelihood to stop inference. Default 10.

maxIter
Integer. Maximum number of iterations of Gibbs sampling or EM to perform. Default 200.

splitOnIter
Integer. On every 'splitOnIter' iteration, a heuristic will be applied to determine if a cell population should be reassigned and another cell population should be split into two clusters. To disable splitting, set to -1. Default 10.

splitOnLast
Integer. After 'stopIter' iterations have been performed without improvement, a heuristic will be applied to determine if a cell population should be reassigned and another cell population should be split into two clusters. If a split occurs, then 'stopIter' will be reset. Default TRUE.

seed
Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

nchains
Integer. Number of random cluster initializations. Default 3.

zInitialize
Character. One of 'random', 'split', or 'predefined'. With 'random', cells are randomly assigned to a populations. With 'split', cells will be split into sqrt(K) populations and then each population will be subsequently split into another sqrt(K) populations. With 'predefined', values in 'zInit' will be used to initialize 'z'. Default 'split'.

countChecksum
Character. An MD5 checksum for the 'counts' matrix. Default NULL.

zInit
Integer vector. Sets initial starting values of z. 'zInit' is only used when 'zInitialize = 'predefined". Default NULL.

logfile
Character. Messages will be redirected to a file named 'logfile'. If NULL, messages will be printed to stdout. Default NULL.

verbose
Logical. Whether to print log messages. Default TRUE.
Value

A SingleCellExperiment object. Function parameter settings are stored in the metadata "celda_parameters" slot. Columns celda_sample_label and celda_cell_cluster in colData contain sample labels and celda cell population clusters.

See Also

celda_G for feature clustering and celda_CG for simultaneous clustering of features and cells. celdaGridSearch can be used to run multiple values of K and multiple chains in parallel.

Examples

data(celdaCSim)
sce <- celda_C(celdaCSim$counts,
  K = celdaCSim$K,
  sampleLabel = celdaCSim$sampleLabel,
  nchains = 1)

---

celda_CG

Cell and feature clustering with Celda

Description

Clusters the rows and columns of a count matrix containing single-cell data into L modules and K subpopulations, respectively. The useAssay assay slot in altExpName altExp slot will be used if it exists. Otherwise, the useAssay assay slot in x will be used if x is a SingleCellExperiment object.

Usage

celda_CG(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  sampleLabel = NULL,
  K,
  L,
  alpha = 1,
  beta = 1,
  delta = 1,
  gamma = 1,
  algorithm = c("EM", "Gibbs"),
  stopIter = 10,
  maxIter = 200,
  splitOnIter = 10,
  splitOnLast = TRUE,
  seed = 12345,
  nchains = 3,
zInitialize = c("split", "random", "predefined"),
yInitialize = c("split", "random", "predefined"),
countChecksum = NULL,
zInit = NULL,
yInit = NULL,
logfile = NULL,
verbose = TRUE)

## S4 method for signature 'SingleCellExperiment'
celda_CG(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  sampleLabel = NULL,
  K,
  L,
  alpha = 1,
  beta = 1,
  delta = 1,
  gamma = 1,
  algorithm = c("EM", "Gibbs"),
  stopIter = 10,
  maxIter = 200,
  splitOnIter = 10,
  splitOnLast = TRUE,
  seed = 12345,
  nchains = 3,
  zInitialize = c("split", "random", "predefined"),
  yInitialize = c("split", "random", "predefined"),
countChecksum = NULL,
zInit = NULL,
yInit = NULL,
logfile = NULL,
verbose = TRUE)

## S4 method for signature 'ANY'
celda_CG(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  sampleLabel = NULL,
  K,
  L,
  alpha = 1,
  beta = 1,
  delta = 1,
gamma = 1,
algorithm = c("EM", "Gibbs"),
stopIter = 10,
maxIter = 200,
splitOnIter = 10,
splitOnLast = TRUE,
seed = 12345,
nchains = 3,
zInitialize = c("split", "random", "predefined"),
yInitialize = c("split", "random", "predefined"),
countChecksum = NULL,
zInit = NULL,
yInit = NULL,
logfile = NULL,
verbose = TRUE
)

Arguments

x A SingleCellExperiment with the matrix located in the assay slot under useAssay. Rows represent features and columns represent cells. Alternatively, any matrix-like object that can be coerced to a sparse matrix of class "dgCMatrix" can be directly used as input. The matrix will automatically be converted to a SingleCellExperiment object.

useAssay A string specifying the name of the assay slot to use. Default "counts".

altExpName The name for the altExp slot to use. Default "featureSubset".

sampleLabel Vector or factor. Denotes the sample label for each cell (column) in the count matrix.

K Integer. Number of cell populations.

L Integer. Number of feature modules.

alpha Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1.

beta Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell population. Default 1.

delta Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.

gamma Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 1.

algorithm String. Algorithm to use for clustering cell subpopulations. One of 'EM' or 'Gibbs'. The EM algorithm for cell clustering is faster, especially for larger numbers of cells. However, more chains may be required to ensure a good solution is found. Default 'EM'.

stopIter Integer. Number of iterations without improvement in the log likelihood to stop inference. Default 10.

maxIter Integer. Maximum number of iterations of Gibbs sampling to perform. Default 200.
splitOnIter Integer. On every splitOnIter iteration, a heuristic will be applied to determine if a cell population or feature module should be reassigned and another cell population or feature module should be split into two clusters. To disable splitting, set to -1. Default 10.

splitOnLast Integer. After stopIter iterations have been performed without improvement, a heuristic will be applied to determine if a cell population or feature module should be reassigned and another cell population or feature module should be split into two clusters. If a split occurs, then 'stopIter' will be reset. Default TRUE.

seed Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

nchains Integer. Number of random cluster initializations. Default 3.

zInitialize Character. One of 'random', 'split', or 'predefined'. With 'random', cells are randomly assigned to a populations. With 'split', cells will be split into sqrt(K) populations and then each population will be subsequently split into another sqrt(K) populations. With 'predefined', values in zInit will be used to initialize z. Default 'split'.

yInitialize Character. One of 'random', 'split', or 'predefined'. With 'random', features are randomly assigned to a modules. With 'split', features will be split into sqrt(L) modules and then each module will be subsequently split into another sqrt(L) modules. With 'predefined', values in yInit will be used to initialize y. Default 'split'.

countChecksum Character. An MD5 checksum for the counts matrix. Default NULL.

zInit Integer vector. Sets initial starting values of z. 'zInit' is only used when 'zInitialize = 'predefined''. Default NULL.

yInit Integer vector. Sets initial starting values of y. 'yInit' is only be used when 'yInitialize = "predefined"'. Default NULL.

logfile Character. Messages will be redirected to a file named 'logfile'. If NULL, messages will be printed to stdout. Default NULL.

verbose Logical. Whether to print log messages. Default TRUE.

Value

A SingleCellExperiment object. Function parameter settings are stored in metadata "celda_parameters" in altExp slot. In altExp slot, columns celda_sample_label and celda_cell_cluster in colData contain sample labels and celda cell population clusters. Column celda_feature_module in rowData contains feature modules.

See Also

celda_G for feature clustering and celda_C for clustering cells. celdaGridSearch can be used to run multiple values of K/L and multiple chains in parallel.
celda_G

Examples

data(celdaCGSim)
sce <- celda_CG(celdaCGSim$counts,
    K = celdaCGSim$K,
    L = celdaCGSim$L,
    sampleLabel = celdaCGSim$sampleLabel,
    nchains = 1)

Description

Clusters the rows of a count matrix containing single-cell data into L modules. The useAssay assay slot in altExpName altExp slot will be used if it exists. Otherwise, the useAssay assay slot in x will be used if x is a SingleCellExperiment object.

Usage

celda_G(
    x,
    useAssay = "counts",
    altExpName = "featureSubset",
    L,
    beta = 1,
    delta = 1,
    gamma = 1,
    stopIter = 10,
    maxIter = 200,
    splitOnIter = 10,
    splitOnLast = TRUE,
    seed = 12345,
    nchains = 3,
    yInitialize = c("split", "random", "predefined"),
    countChecksum = NULL,
    yInit = NULL,
    logfile = NULL,
    verbose = TRUE
)

## S4 method for signature 'SingleCellExperiment'
celda_G(
    x,
    useAssay = "counts",
    altExpName = "featureSubset",
    L,
    beta = 1,
delta = 1,
gamma = 1,
stopIter = 10,
maxIter = 200,
splitOnIter = 10,
splitOnLast = TRUE,
seed = 12345,
nchains = 3,
yInitialize = c("split", "random", "predefined"),
countChecksum = NULL,
yInit = NULL,
logfile = NULL,
verbose = TRUE
)

## S4 method for signature 'ANY'
celda_G(
x,
useAssay = "counts",
altExpName = "featureSubset",
L,
beta = 1,
delta = 1,
gamma = 1,
stopIter = 10,
maxIter = 200,
splitOnIter = 10,
splitOnLast = TRUE,
seed = 12345,
nchains = 3,
yInitialize = c("split", "random", "predefined"),
countChecksum = NULL,
yInit = NULL,
logfile = NULL,
verbose = TRUE
)

Arguments

x A SingleCellExperiment with the matrix located in the assay slot under useAssay. Rows represent features and columns represent cells. Alternatively, any matrix-like object that can be coerced to a sparse matrix of class "dgCMatrix" can be directly used as input. The matrix will automatically be converted to a SingleCellExperiment object.

useAssay A string specifying the name of the assay slot to use. Default "counts".

altExpName The name for the altExp slot to use. Default "featureSubset".

L Integer. Number of feature modules.
beta Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell. Default 1.
delta Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.
gamma Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 1.
stopIter Integer. Number of iterations without improvement in the log likelihood to stop inference. Default 10.
maxIter Integer. Maximum number of iterations of Gibbs sampling to perform. Default 200.
splitOnIter Integer. On every ‘splitOnIter’ iteration, a heuristic will be applied to determine if a feature module should be reassigned and another feature module should be split into two clusters. To disable splitting, set to -1. Default 10.
splitOnLast Integer. After ‘stopIter’ iterations have been performed without improvement, a heuristic will be applied to determine if a cell population should be reassigned and another cell population should be split into two clusters. If a split occurs, then ‘stopIter’ will be reset. Default TRUE.
seed Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.
nchains Integer. Number of random cluster initializations. Default 3.
yInitialize Character. One of 'random', 'split', or 'predefined'. With 'random', features are randomly assigned to modules. With 'split', features will be split into sqrt(L) modules and then each module will be subsequently split into another sqrt(L) modules. With 'predefined', values in 'yInit' will be used to initialize 'y'. Default 'split'.
countChecksum Character. An MD5 checksum for the 'counts' matrix. Default NULL.
yInit Integer vector. Sets initial starting values of y. ‘yInit’ can only be used when ‘yInitialize = 'predefined''. Default NULL.
logfile Character. Messages will be redirected to a file named logfile. If NULL, messages will be printed to stdout. Default NULL.
verbose Logical. Whether to print log messages. Default TRUE.

Value

A SingleCellExperiment object. Function parameter settings are stored in the metadata "celda_parameters" slot. Column celda_feature_module in rowData contains feature modules.

See Also

celda_C for cell clustering and celda.CG for simultaneous clustering of features and cells. celda-GridSearch can be used to run multiple values of L and multiple chains in parallel.

Examples

data(celdaGSim)
sce <- celda_G(celdaGSim$counts, L = celdaGSim$L, nchains = 1)
clusterProbability

*Get the conditional probabilities of cell in subpopulations from celda model*

**Description**

Calculate the conditional probability of each cell belonging to each subpopulation given all other cell cluster assignments and/or each feature belonging to each module given all other feature cluster assignments in a celda model.

**Usage**

```r
clusterProbability(
  sce,
  useAssay = "counts",
  altExpName = "featureSubset",
  log = FALSE
)
```

```r
## S4 method for signature 'SingleCellExperiment'
clusterProbability(
  sce,
  useAssay = "counts",
  altExpName = "featureSubset",
  log = FALSE
)
```

**Arguments**

- **sce**: A `SingleCellExperiment` object returned by `celda_C`, `celda_G`, or `celda_CG`, with the matrix located in the `useAssay` assay slot. Rows represent features and columns represent cells.
- **useAssay**: A string specifying which `assay` slot to use. Default "counts".
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".
- **log**: Logical. If FALSE, then the normalized conditional probabilities will be returned. If TRUE, then the unnormalized log probabilities will be returned. Default FALSE.

**Value**

A list containing a matrix for the conditional cell subpopulation cluster and/or feature module probabilities.

**See Also**

`celda_C()` for clustering cells
**compareCountMatrix**

**Examples**

```r
data(sceCeldaCG)
clusterProb <- clusterProbability(sceCeldaCG, log = TRUE)
data(sceCeldaC)
clusterProb <- clusterProbability(sceCeldaC)
```

---

**Description**

Checks if the counts matrix is the same one used to generate the celda model object by comparing dimensions and MD5 checksum.

**Usage**

```r
compareCountMatrix(counts, celdaMod, errorOnMismatch = TRUE)
```

```
## S4 method for signature 'ANY,celdaModel'
compareCountMatrix(counts, celdaMod, errorOnMismatch = TRUE)
```

```
## S4 method for signature 'ANY,celdaList'
compareCountMatrix(counts, celdaMod, errorOnMismatch = TRUE)
```

**Arguments**

- `counts`: Integer, Numeric, or Sparse matrix. Rows represent features and columns represent cells.
- `celdaMod`: A `celdaModel` or `celdaList` object.
- `errorOnMismatch`: Logical. Whether to throw an error in the event of a mismatch. Default TRUE.

**Value**

Returns TRUE if provided count matrix matches the one used in the celda object and/or `errorOnMismatch` = FALSE, FALSE otherwise.

**Examples**

```r
data(celdaCGSim, celdaCGMod)
compareCountMatrix(celdaCGSim$counts, celdaCGMod, errorOnMismatch = FALSE)
data(celdaCGSim, celdaCGGridSearchRes)
compareCountMatrix(celdaCGSim$counts, celdaCGGridSearchRes, errorOnMismatch = FALSE)
```
contaminationSim

Description

A toy contamination data generated by `simulateContamination`.

Usage

`contaminationSim`

Format

A list

countChecksum

Get the MD5 hash of the count matrix from the celdaList.

Description

Returns the MD5 hash of the count matrix used to generate the celdaList.

Usage

`countChecksum(celdaList)`

Arguments

celdaList    An object of class celdaList.

Value

A character string of length 32 containing the MD5 digest of the count matrix.

Examples

```r
data(celdaCGGridSearchRes)
countChecksum <- countChecksum(celdaCGGridSearchRes)
```
**countChecksum, celdaList-method**

*Get the MD5 hash of the count matrix from the celdaList*

---

**Description**

Returns the MD5 hash of the count matrix used to generate the celdaList.

**Usage**

```r
## S4 method for signature 'celdaList'
countChecksum(celdaList)
```

**Arguments**

- `celdaList` An object of class celdaList.

**Value**

A character string of length 32 containing the MD5 digest of the count matrix.

**Examples**

```r
data(celdaCGGridSearchRes)
countChecksum <- countChecksum(celdaCGGridSearchRes)
```

---

**decontX**

*Contamination estimation with decontX*

---

**Description**

Identifies contamination from factors such as ambient RNA in single cell genomic datasets.

**Usage**

```r
decontX(x, ...)
```

```r
## S4 method for signature 'SingleCellExperiment'
decontX(
  x,
  assayName = "counts",
  z = NULL,
  batch = NULL,
  background = NULL,
  bgAssayName = NULL,
  bgBatch = NULL,
)```
maxIter = 500,
delta = c(10, 10),
estimateDelta = TRUE,
convergence = 0.001,
iterLogLik = 10,
varGenes = 5000,
dbscanEps = 1,
seed = 12345,
logfile = NULL,
verbose = TRUE
)

## S4 method for signature 'ANY'
decontX(
  x,
  z = NULL,
  batch = NULL,
  background = NULL,
  bgBatch = NULL,
  maxIter = 500,
  delta = c(10, 10),
  estimateDelta = TRUE,
  convergence = 0.001,
  iterLogLik = 10,
  varGenes = 5000,
  dbscanEps = 1,
  seed = 12345,
  logfile = NULL,
  verbose = TRUE
)

**Arguments**

- **x**  
  A numeric matrix of counts or a SingleCellExperiment with the matrix located in the assay slot under assayName. Cells in each batch will be subsetted and converted to a sparse matrix of class dgCMatrix from package Matrix before analysis. This object should only contain filtered cells after cell calling. Empty cell barcodes (low expression droplets before cell calling) are not needed to run DecontX.

- **z**  
  Numeric or character vector. Cell cluster labels. If NULL, PCA will be used to reduce the dimensionality of the dataset initially, `umap` from the `uwot` package will be used to further reduce the dataset to 2 dimensions and the `dbscan` function from the `dbscan` package will be used to identify clusters of broad cell types. Default NULL.

- **batch**  
  Numeric or character vector. Batch labels for cells. If batch labels are supplied, DecontX is run on cells from each batch separately. Cells run in different
channels or assays should be considered different batches. Default NULL.

background
A numeric matrix of counts or a SingleCellExperiment with the matrix located in the assay slot under assayName. It should have the same data format as x except it contains the empty droplets instead of cells. When supplied, empirical distribution of transcripts from these empty droplets will be used as the contamination distribution. Default NULL.

bgAssayName
Character. Name of the assay to use if background is a SingleCellExperiment. Default to same as assayName.

gBatch
Numeric or character vector. Batch labels for background. Its unique values should be the same as those in batch, such that each batch of cells have their corresponding batch of empty droplets as background, pointed by this parameter. Default to NULL.

maxIter

delta
Numeric Vector of length 2. Concentration parameters for the Dirichlet prior for the contamination in each cell. The first element is the prior for the native counts while the second element is the prior for the contamination counts. These essentially act as pseudocounts for the native and contamination in each cell. If estimateDelta = TRUE, this is only used to produce a random sample of proportions for an initial value of contamination in each cell. Then fit_dirichlet is used to update delta in each iteration. If estimateDelta = FALSE, then delta is fixed with these values for the entire inference procedure. Fixing delta and setting a high number in the second element will force decontX to be more aggressive and estimate higher levels of contamination at the expense of potentially removing native expression. Default c(10, 10).

estimateDelta
Boolean. Whether to update delta at each iteration.

convergence
Numeric. The EM algorithm will be stopped if the maximum difference in the contamination estimates between the previous and current iterations is less than this. Default 0.001.

iterLogLik

varGenes
Integer. The number of variable genes to use in dimensionality reduction before clustering. Variability is calculated using modelGeneVar function from the 'scran' package. Used only when z is not provided. Default 5000.

dbscanEps
Numeric. The clustering resolution parameter used in 'dbscan' to estimate broad cell clusters. Used only when z is not provided. Default 1.

seed
Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

logfile
Character. Messages will be redirected to a file named 'logfile'. If NULL, messages will be printed to stdout. Default NULL.

verbose
Logical. Whether to print log messages. Default TRUE.

Value

If x is a matrix-like object, a list will be returned with the following items:

decontXcounts: The decontaminated matrix. Values obtained from the variational inference procedure may be non-integer. However, integer counts can be obtained by rounding, e.g. round(decontXcounts).
contamination: Percentage of contamination in each cell.
estimates: List of estimated parameters for each batch. If z was not supplied, then the UMAP coordinates used to generated cell cluster labels will also be stored here.z: Cell population/cluster labels used for analysis.runParams: List of arguments used in the function call.

If x is a SingleCellExperiment, then the decontaminated counts will be stored as an assay and can be accessed with decontXcounts(x). The contamination values and cluster labels will be stored in colData(x). estimates and runParams will be stored in metadata(x)$decontX. The UMAPs used to generated cell cluster labels will be stored in reducedDims slot in x.

Author(s)
Shiyi Yang, Yuan Yin, Joshua Campbell

Examples
# Generate matrix with contamination
s <- simulateContamination(seed = 12345)

library(SingleCellExperiment)
sce <- SingleCellExperiment(list(counts = s$observedCounts))
sce <- decontX(sce)

# Plot contamination on UMAP
plotDecontXContamination(sce)

# Plot decontX cluster labels
umap <- reducedDim(sce)
plotDimReduceCluster(x = sce$decontX_clusters, 
dim1 = umap[, 1], dim2 = umap[, 2],
)

# Plot percentage of marker genes detected
# in each cell cluster before decontamination
s$markers
plotDecontXMarkerPercentage(sce, markers = s$markers, assayName = "counts")

# Plot percentage of marker genes detected
# in each cell cluster after contamination
plotDecontXMarkerPercentage(sce, markers = s$markers, 
assayName = "decontXcounts")

# Plot percentage of marker genes detected in each cell
# comparing original and decontaminated counts side-by-side
plotDecontXMarkerPercentage(sce, markers = s$markers, 
assayName = c("counts", "decontXcounts"))

# Plot raw counts of individual markers genes before
# and after decontamination
plotDecontXMarkerExpression(sce, unlist(s$markers))
decontXcounts

Get or set decontaminated counts matrix

Description

Gets or sets the decontaminated counts matrix from a SingleCellExperiment object.

Usage

decountXcounts(object, ...)
decountXcounts(object, ...) <- value

## S4 method for signature 'SingleCellExperiment'
decountXcounts(object, ...)

## S4 replacement method for signature 'SingleCellExperiment'
decountXcounts(object, ...) <- value

Arguments

object
A SingleCellExperiment object.

... For the generic, further arguments to pass to each method.

value A matrix to save as an assay called decontXcounts

Value

If getting, the assay from object with the name decontXcounts will be returned. If setting, a SingleCellExperiment object will be returned with decontXcounts listed in the assay slot.

See Also

assay and assay<-

distinctColors

Create a color palette

Description

Generate a palette of ‘n’ distinct colors.
Usage

distinctColors(
  n,
  hues = c("red", "cyan", "orange", "blue", "yellow", "purple", "green", "magenta"),
  saturationRange = c(0.7, 1),
  valueRange = c(0.7, 1)
)

Arguments

- **n**: Integer. Number of colors to generate.
- **hues**: Character vector. Colors available from `colors()`. These will be used as the base colors for the clustering scheme in HSV. Different saturations and values will be generated for each hue. Default `c("red", "cyan", "orange", "blue", "yellow", "purple", "green", "magenta")`.
- **saturationRange**: Numeric vector. A vector of length 2 denoting the saturation for HSV. Values must be in \([0, 1]\). Default: `c(0.25, 1)`.
- **valueRange**: Numeric vector. A vector of length 2 denoting the range of values for HSV. Values must be in \([0, 1]\). Default: `c(0.5, 1)`.

Value

A vector of distinct colors that have been converted to HEX from HSV.

Examples

colorPal <- distinctColors(6) # can be used in plotting functions

decription

Fast matrix multiplication for double x int

Usage

eigenMatMultInt(A, B)

Arguments

- **A**: a double matrix
- **B**: an integer matrix

Value

An integer matrix representing the product of A and B
**eigenMatMultNumeric**  
*Fast matrix multiplication for double x double*

**Description**
Fast matrix multiplication for double x double

**Usage**
eigenMatMultNumeric(A, B)

**Arguments**
- **A**: a double matrix
- **B**: an integer matrix

**Value**
An integer matrix representing the product of A and B

---

**factorizeMatrix**  
*Generate factorized matrices showing each feature’s influence on cell / gene clustering*

**Description**
Generates factorized matrices showing the contribution of each feature in each cell population or each cell population in each sample.

**Usage**
factorizeMatrix(  
  x,  
  celdaMod,  
  useAssay = "counts",  
  altExpName = "featureSubset",  
  type = c("counts", "proportion", "posterior")
)

```r
## S4 method for signature 'SingleCellExperiment,ANY'
factorizeMatrix(  
  x,  
  useAssay = "counts",  
  altExpName = "featureSubset",  
  type = c("counts", "proportion", "posterior")
)
```
## factorizeMatrix

```r
## S4 method for signature 'ANY,celda_CG'
factorizeMatrix(x, celdaMod, type = c("counts", "proportion", "posterior"))

## S4 method for signature 'ANY,celda_C'
factorizeMatrix(x, celdaMod, type = c("counts", "proportion", "posterior"))

## S4 method for signature 'ANY,celda_G'
factorizeMatrix(x, celdaMod, type = c("counts", "proportion", "posterior"))
```

### Arguments

- **x**: Can be one of
  - A `SingleCellExperiment` object returned by `celda_C`, `celda_G` or `celda_CG`, with the matrix located in the `useAssay` assay slot in `altExp(x, altExpName)`. Rows represent features and columns represent cells.
  - Integer counts matrix. Rows represent features and columns represent cells.
    - This matrix should be the same as the one used to generate `celdaMod`.
- **celdaMod**: Celda model object. Only works if `x` is an integer counts matrix.
- **useAssay**: A string specifying which assay slot to use if `x` is a `SingleCellExperiment` object. Default "counts".
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".
- **type**: Character vector. A vector containing one or more of "counts", "proportion", or "posterior". "counts" returns the raw number of counts for each factorized matrix. "proportions" returns the normalized probabilities for each factorized matrix, which are calculated by dividing the raw counts in each factorized matrix by the total counts in each column. "posterior" returns the posterior estimates which include the addition of the Dirichlet concentration parameter (essentially as a pseudocount). Default "counts".

### Value

For `celda_CG` model, a list with elements for "counts", "proportions", or "posterior" probabilities. Each element will be a list containing factorized matrices for "module", "cellPopulation", and "sample". Additionally, the contribution of each module in each individual cell will be included in the "cell" element of "counts" and "proportions" elements.

For `celda_C` model, a list with elements for "counts", "proportions", or "posterior" probabilities. Each element will be a list containing factorized matrices for "module" and "sample".

For `celda_G` model, a list with elements for "counts", "proportions", or "posterior" probabilities. Each element will be a list containing factorized matrices for "module" and "cell".

### Examples

```r
data(sceCeldaCG)
factorizedMatrices <- factorizeMatrix(sceCeldaCG, type = "posterior")
data(celdaCGSim, celdaCGMod)
```
factorizedMatrices <- factorizeMatrix(
celdaGSim$counts, 
celdaGMod, 
"posterior")
data(celdaCSim, celdaCMod)
factorizedMatrices <- factorizeMatrix(
celdaCSim$counts, 
celdaCMod, "posterior"
)
data(celdaGSim, celdaGMod)
factorizedMatrices <- factorizeMatrix(
celdaGSim$counts, 
celdaGMod, "posterior"
)

---

**fastNormProp**

*Fast normalization for numeric matrix*

**Description**

Fast normalization for numeric matrix

**Usage**

```r
fastNormProp(R_counts, R_alpha)
```

**Arguments**

- **R_counts**: An integer matrix
- **R_alpha**: A double value to be added to the matrix as a pseudocount

**Value**

A numeric matrix where the columns have been normalized to proportions

---

**fastNormPropLog**

*Fast normalization for numeric matrix*

**Description**

Fast normalization for numeric matrix

**Usage**

```r
fastNormPropLog(R_counts, R_alpha)
```
Arguments

- **R_counts**: An integer matrix
- **R_alpha**: A double value to be added to the matrix as a pseudocount

Value

A numeric matrix where the columns have been normalized to proportions

---

**fastNormPropSqrt**  
*Fast normalization for numeric matrix*

Description

Fast normalization for numeric matrix

Usage

```r
fastNormPropSqrt(R_counts, R_alpha)
```

Arguments

- **R_counts**: An integer matrix
- **R_alpha**: A double value to be added to the matrix as a pseudocount

Value

A numeric matrix where the columns have been normalized to proportions

---

**featureModuleLookup**  
*Obtain the gene module of a gene of interest*

Description

This function will output the corresponding feature module for a specified vector of genes from a celda_CG or celda_G celdaModel. features must match the rownames of sce.
Usage

featureModuleLookup(
  sce,
  features,
  altExpName = "featureSubset",
  exactMatch = TRUE,
  by = "rownames"
)

## S4 method for signature 'SingleCellExperiment'
featureModuleLookup(
  sce,
  features,
  altExpName = "featureSubset",
  exactMatch = TRUE,
  by = "rownames"
)

Arguments

- **sce**: A SingleCellExperiment object returned by `celda_G`, or `celda_CG`, with the matrix located in the `useAssay` assay slot. Rows represent features and columns represent cells.
- **features**: Character vector. Identify feature modules for the specified feature names. `feature` must match the rownames of `sce`.
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".
- **exactMatch**: Logical. Whether to look for exactMatch of the gene name within counts matrix. Default TRUE.
- **by**: Character. Where to search for features in the sce object. If set to "rownames" then the features will be searched for among rownames(sce). This can also be set to one of the colnames of rowData(sce). Default "rownames".

Value

Numeric vector containing the module numbers for each feature. If the feature was not found, then an NA value will be returned in that position. If no features were found, then an error will be given.

Examples

data(sceCeldaCG)
module <- featureModuleLookup(sce = sceCeldaCG,
  features = c("Gene_1", "Gene_XXX"))
featureModuleTable  

Output a feature module table

Description

Creates a table that contains the list of features in each feature module.

Usage

\[
\text{featureModuleTable}( \\
\quad \text{sce,} \\
\quad \text{useAssay = "counts",} \\
\quad \text{altExpName = "featureSubset",} \\
\quad \text{displayName = NULL,} \\
\quad \text{outputFile = NULL} \\
\)
\]

Arguments

- **sce**: A `SingleCellExperiment` object returned by `celda_G`, or `celda.CG`, with the matrix located in the `useAssay` assay slot. Rows represent features and columns represent cells.
- **useAssay**: A string specifying which assay slot to use. Default "counts".
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".
- **displayName**: Character. The column name of `rowData(sce)` that specifies the display names for the features. Default NULL, which displays the row names.
- **outputFile**: File name for feature module table. If NULL, file will not be created. Default NULL.

Value

Matrix. Contains a list of features per each column (feature module)

Examples

\[
\text{data(sceCeldaCG)} \\
\text{featureModuleTable(sceCeldaCG)}
\]
geneSetEnrich

Gene set enrichment

Description

Identify and return significantly-enriched terms for each gene module in a Celda object or a SingleCellExperiment object. Performs gene set enrichment analysis for Celda identified modules using the enrichr.

Usage

geneSetEnrich(
  x,
  celdaModel, 
  useAssay = "counts", 
  altExpName = "featureSubset", 
  databases, 
  fdr = 0.05
)

## S4 method for signature 'SingleCellExperiment'
geneSetEnrich(
  x, 
  useAssay = "counts", 
  altExpName = "featureSubset", 
  databases, 
  fdr = 0.05
)

## S4 method for signature 'matrix'
geneSetEnrich(x, celdaModel, databases, fdr = 0.05)

Arguments

x A numeric matrix of counts or a SingleCellExperiment with the matrix located in the assay slot under useAssay. Rows represent features and columns represent cells. Rownames of the matrix or SingleCellExperiment object should be gene names.
celdaModel Celda object of class celda_G or celda_CG.
useAssay A string specifying which assay slot to use if x is a SingleCellExperiment object. Default "counts".
altExpName The name for the altExp slot to use. Default "featureSubset".
databases Character vector. Name of reference database. Available databases can be viewed by listEnrichrDbs.
fdr False discovery rate (FDR). Numeric. Cutoff value for adjusted p-value, terms with FDR below this value are considered significantly enriched.
logLikelihood

Value

List of length 'L' where each member contains the significantly enriched terms for the corresponding module.

Author(s)

Ahmed Youssef, Zhe Wang

Examples

library(M3DExampleData)
counts <- M3DExampleData::Mmus_example_list$data
# subset 500 genes for fast clustering
counts <- counts[seq(1501, 2000), ]
# cluster genes into 10 modules for quick demo
sce <- celda_G(x = as.matrix(counts), L = 10, verbose = FALSE)
gse <- geneSetEnrich(sce,
  databases = c("GO_Biological_Process_2018", "GO_Molecular_Function_2018"))

logLikelihood

Calculate the Log-likelihood of a celda model

Description

Calculate the log-likelihood for cell population and feature module cluster assignments on the count matrix, per celda model.

Usage

logLikelihood(x, celdaMod, useAssay = "counts", altExpName = "featureSubset")

## S4 method for signature 'SingleCellExperiment,ANY'
logLikelihood(x, useAssay = "counts", altExpName = "featureSubset")

## S4 method for signature 'matrix,celda_C'
logLikelihood(x, celdaMod)

## S4 method for signature 'matrix,celda_G'
logLikelihood(x, celdaMod)

## S4 method for signature 'matrix,celda_CG'
logLikelihood(x, celdaMod)
Arguments

- **x**: A `SingleCellExperiment` object returned by `celda_C`, `celda_G`, or `celda_CG`, with the matrix located in the `useAssay` assay slot. Rows represent features and columns represent cells.
- **celdaMod**: celda model object. Ignored if `x` is a `SingleCellExperiment` object.
- **useAssay**: A string specifying which `assay` slot to use. Default "counts".
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".

Value

The log-likelihood of the cluster assignment for the provided `SingleCellExperiment`.

See Also

- `celda_C()` for clustering cells

Examples

```r
data(sceCeldaC, sceCeldaCG)
loglikC <- logLikelihood(sceCeldaC)
loglikCG <- logLikelihood(sceCeldaCG)
```

Description

Retrieves the complete log-likelihood from all iterations of Gibbs sampling used to generate a celda model.

Usage

```r
logLikelihoodHistory(x, altExpName = "featureSubset")
```

```
## S4 method for signature 'SingleCellExperiment'
logLikelihoodHistory(x, altExpName = "featureSubset")

## S4 method for signature 'celdaModel'
logLikelihoodHistory(x)
```

Arguments

- **x**: A `SingleCellExperiment` object returned by `celda_C`, `celda_G`, or `celda_CG`, or a celda model object.
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".
matrixNames

Value

Numeric. The log-likelihood at each step of Gibbs sampling used to generate the model.

Examples

data(sceCeldaCG)
logLikelihoodHistory(sceCeldaCG)
data(celdaCGMod)
logLikelihoodHistory(celdaCGMod)

matrixNames Get feature, cell and sample names from a celdaModel

Description

Retrieves the row, column, and sample names used to generate a celdaModel.

Usage

matrixNames(celdaMod)

## S4 method for signature 'celdaModel'
matrixNames(celdaMod)

Arguments

celdaMod celdaModel. Options available in ‘celda::availableModels’.

Value

List. Contains row, column, and sample character vectors corresponding to the values provided when the celdaModel was generated.

Examples

data(celdaCGMod)
matrixNames(celdaCGMod)
Heatmap for featureModules

**Description**

Renders a heatmap for selected featureModule. Cells are ordered from those with the lowest probability of the module on the left to the highest probability on the right. Features are ordered from those with the highest probability in the module on the top to the lowest probability on the bottom.

**Usage**

```r
moduleHeatmap(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  modules = NULL,
  featureModule = NULL,
  col = circlize::colorRamp2(c(-2, 0, 2), c("#1E90FF", "#FFFFFF", "#CD2626")),
  topCells = 100,
  topFeatures = NULL,
  normalizedCounts = NA,
  normalize = "proportion",
  transformationFun = sqrt,
  scaleRow = scale,
  showFeatureNames = TRUE,
  displayName = NULL,
  trim = c(-2, 2),
  rowFontSize = NULL,
  showHeatmapLegend = FALSE,
  showTopAnnotationLegend = FALSE,
  showTopAnnotationName = FALSE,
  topAnnotationHeight = 5,
  showModuleLabel = TRUE,
  moduleLabel = "auto",
  moduleLabelSize = NULL,
  byrow = TRUE,
  top = NA,
  unit = "mm",
  ncol = NULL,
  useRaster = TRUE,
  returnAsList = FALSE,
  ...
)
```

```
## S4 method for signature 'SingleCellExperiment'
moduleHeatmap(
```
x,  
useAssay = "counts",  
altExpName = "featureSubset",  
modules = NULL,  
featureModule = NULL,  
transformationFun = sqrt,  
showFeatureNames = TRUE,  
displayName = NULL,  
trim = c(-2, 2),  
rowFontSize = NULL,  
showHeatmapLegend = FALSE,  
showTopAnnotationLegend = FALSE,  
showTopAnnotationName = FALSE,  
moduleLabel = "auto",  
moduleLabelSize = NULL,  
byrow = TRUE,  
unit = "mm",  
ncol = NULL,  
returnAsList = FALSE,  
...  
)

Arguments

x  A numeric matrix of counts or a SingleCellExperiment with the matrix located in the assay slot under useAssay. Rows represent features and columns represent cells. Celda results must be present under metadata(altExp(x, altExpName)).

useAssay A string specifying which assay slot to use if x is a SingleCellExperiment object. Default "counts".

altExpName The name for the altExp slot to use. Default "featureSubset".

modules Integer Vector. The featureModule(s) to display. Multiple modules can be included in a vector. Default NULL which plots all module heatmaps.

featureModule Same as modules. Either can be used to specify the modules to display.

col Passed to Heatmap. Set color boundaries and colors.

topCells Integer. Number of cells with the highest and lowest probabilities for each module to include in the heatmap. For example, if topCells = 50, the 50 cells with the lowest probabilities and the 50 cells with the highest probabilities for each featureModule will be included. If NULL, all cells will be plotted. Default 100.
topFeatures  Integer. Plot ‘topFeatures’ features with the highest probabilities in the module heatmap for each featureModule. If NULL, plot all features in the module. Default NULL.

normalizedCounts  Integer matrix. Rows represent features and columns represent cells. If you have a normalized matrix result from normalizeCounts, you can pass through the result here to skip the normalization step in this function. Make sure the col-names and rownames match the object in x. This matrix should correspond to one generated from this count matrix assay(altExp(x, altExpName), i = useAssay). If NA, normalization will be carried out in the following form normalizeCounts(assay(altExp(x, altExpName), i = useAssay), normalize = "proportion", transformationFun = sqrt). Use of this parameter is particularly useful for plotting many module heatmaps, where normalizing the counts matrix repeatedly would be too time consuming. Default NA.

normalize  Character. Passed to normalizeCounts if normalizedCounts is NA. Divides counts by the library sizes for each cell. One of ‘proportion’, ‘cpm’, ‘median’, or ‘mean’. ‘proportion’ uses the total counts for each cell as the library size. ‘cpm’ divides the library size of each cell by one million to produce counts per million. ‘median’ divides the library size of each cell by the median library size across all cells. ‘mean’ divides the library size of each cell by the mean library size across all cells. Default "proportion".

transformationFun  Function. Passed to normalizeCounts if normalizedCounts is NA. Applies a transformation such as sqrt, log, log2, log10, or log1p. If NULL, no transformation will be applied. Occurs after normalization. Default sqrt.

scaleRow  Function. Which function to use to scale each individual row. Set to NULL to disable. Occurs after normalization and log transformation. For example, scale will Z-score transform each row. Default scale.

showFeatureNames  Logical. Whether feature names should be displayed. Default TRUE.

displayName  Character. The column name of rowData(altExp(x, altExpName)) that specifies the display names for the features. Default NULL, which displays the row names. Only works if showFeatureNames is TRUE and x is a SingleCellExperiment object.

trim  Numeric vector. Vector of length two that specifies the lower and upper bounds for plotting the data. This threshold is applied after row scaling. Set to NULL to disable. Default c(-2, 2).

rowFontSize  Numeric. Font size for feature names. If NULL, then the size will automatically be determined. Default NULL.

showHeatmapLegend  Passed to Heatmap. Show legend for expression levels.

showTopAnnotationLegend  Passed to HeatmapAnnotation. Show legend for cell annotation.

showTopAnnotationName  Passed to HeatmapAnnotation. Show heatmap top annotation name.
nonzero

get row and column indices of non zero elements in the matrix

Description
get row and column indices of non zero elements in the matrix

Usage
nonzero(R_counts)
**normalizeCounts**

**Arguments**

- `counts`: A matrix

**Value**

An integer matrix where each row is a row, column indices pair

---

**normalizeCounts** *Normalization of count data*

**Description**

Performs normalization, transformation, and/or scaling of a counts matrix

**Usage**

```r
normalizeCounts(
  counts,
  normalize = c("proportion", "cpm", "median", "mean"),
  scaleFactor = NULL,
  transformationFun = NULL,
  scaleFun = NULL,
  pseudocountNormalize = 0,
  pseudocountTransform = 0
)
```

**Arguments**

- `counts`: Integer, Numeric or Sparse matrix. Rows represent features and columns represent cells.
- `normalize`: Character. Divides counts by the library sizes for each cell. One of 'proportion', 'cpm', 'median', or 'mean'. 'proportion' uses the total counts for each cell as the library size. 'cpm' divides the library size of each cell by one million to produce counts per million. 'median' divides the library size of each cell by the median library size across all cells. 'mean' divides the library size of each cell by the mean library size across all cells.
- `scaleFactor`: Numeric. Sets the scale factor for cell-level normalization. This scale factor is multiplied to each cell after the library size of each cell had been adjusted in `normalize`. Default NULL which means no scale factor is applied.
- `transformationFun`: Function. Applies a transformation such as `sqrt`, `log`, `log2`, `log10`, or `log1p`. If NULL, no transformation will be applied. Occurs after normalization. Default NULL.
- `scaleFun`: Function. Scales the rows of the normalized and transformed count matrix. For example, 'scale' can be used to z-score normalize the rows. Default NULL.
pseudocountNormalize
    Numeric. Add a pseudocount to counts before normalization. Default 0.

pseudocountTransform
    Numeric. Add a pseudocount to normalized counts before applying the transformation function. Adding a pseudocount can be useful before applying a log transformation. Default 0.

Value
    Numeric Matrix. A normalized matrix.

Examples
    data(celdaCGSim)
    normalizedCounts <- normalizeCounts(celdaCGSim$counts, "proportion",
                                        pseudocountNormalize = 1)

params
    Get parameter values provided for celdaModel creation

Description
    Retrieves the K/L, model priors (e.g. alpha, beta), and count matrix checksum parameters provided during the creation of the provided celdaModel.

Usage
    params(celdaMod)

    ## S4 method for signature 'celdaModel'
    params(celdaMod)

Arguments
    celdaMod celdaModel. Options available in celda::availableModels.

Value
    List. Contains the model-specific parameters for the provided celda model object depending on its class.

Examples
    data(celdaCGMod)
    params(celdaCGMod)
perplexity

Calculate the perplexity of a celda model

Description

Perplexity is a statistical measure of how well a probability model can predict new data. Lower perplexity indicates a better model.

Usage

perplexity(
  x,
  celdaMod,
  useAssay = "counts",
  altExpName = "featureSubset",
  newCounts = NULL
)

## S4 method for signature 'SingleCellExperiment,ANY'
perplexity(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  newCounts = NULL
)

## S4 method for signature 'ANY,celda_CG'
perplexity(x, celdaMod, newCounts = NULL)

## S4 method for signature 'ANY,celda_C'
perplexity(x, celdaMod, newCounts = NULL)

## S4 method for signature 'ANY,celda_G'
perplexity(x, celdaMod, newCounts = NULL)

Arguments

x          Can be one of

  • A SingleCellExperiment object returned by celda_C, celda_G or celda_CG, with the matrix located in the useAssay assay slot. Rows represent features and columns represent cells.
  • Integer counts matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate celdaMod.

  celdaMod Celda model object. Only works if x is an integer counts matrix.

  useAssay A string specifying which assay slot to use if x is a SingleCellExperiment object. Default "counts".

plotCeldaViolin

Description

Outputs a violin plot for feature expression data.

Usage

plotCeldaViolin(
  x, 
  celdaMod, 
  features, 
  displayName = NULL, 
  useAssay = "counts", 
  altExpName = "featureSubset", 
  exactMatch = TRUE, 
  plotDots = TRUE, 
  dotSize = 0.1 
)

# S4 method for signature 'SingleCellExperiment'
plotCeldaViolin(
  x, 
  features, 
  displayName = NULL, 
  useAssay = "counts", 
  altExpName = "featureSubset", 
  exactMatch = TRUE, 
  plotDots = TRUE, 
  dotSize = 0.1 
)

altExpName The name for the altExp slot to use. Default "featureSubset".
newCounts A new counts matrix used to calculate perplexity. If NULL, perplexity will be calculated for the matrix in useAssay slot in x. Default NULL.

Value

Numeric. The perplexity for the provided x (and celdaModel).

Examples

data(sceCeldaCG)
perplexity <- perplexity(sceCeldaCG)
data(celdaCGSim, celdaCGMod)
perplexity <- perplexity(celdaCGSim$counts, celdaCGMod)
data(celdaCSim, celdaCMod)
perplexity <- perplexity(celdaCSim$counts, celdaCMod)
data(celdaGSim, celdaGMod)
perplexity <- perplexity(celdaGSim$counts, celdaGMod)
plotCeldaViolin = TRUE,
   dotSize = 0.1
)

## S4 method for signature 'ANY'
plotCeldaViolin(
   x,
   celdaMod,
   features,
   exactMatch = TRUE,
   plotDots = TRUE,
   dotSize = 0.1
)

Arguments

x Numeric matrix or a SingleCellExperiment object with the matrix located in the assay slot under useAssay. Rows represent features and columns represent cells.
celdaMod Celda object of class "celda_G" or "celda_CG". Used only if x is a matrix object.
features Character vector. Uses these genes for plotting.
displayName Character. The column name of rowData(x) that specifies the display names for the features. Default NULL, which displays the row names. Only works if x is a SingleCellExperiment object.
useAssay A string specifying which assay slot to use if x is a SingleCellExperiment object. Default "counts".
altExpName The name for the altExp slot to use. Default "featureSubset".
exactMatch Logical. Whether an exact match or a partial match using grep() is used to look up the feature in the rownames of the counts matrix. Default TRUE.
plotDots Boolean. If TRUE, the expression of features will be plotted as points in addition to the violin curve. Default TRUE.
dotSize Numeric. Size of points if plotDots = TRUE. Default 0.1.

Value

Violin plot for each feature, grouped by celda cluster

Examples

data(sceCeldaCG)
plotCeldaViolin(x = sceCeldaCG, features = "Gene_1")
data(celdaCGSim, celdaCGMod)
plotCeldaViolin(x = celdaCGSim$counts,
celdaMod = celdaCGMod,
features = "Gene_1")
plotDecontXContamination

Plots contamination on UMAP coordinates

Description

A scatter plot of the UMAP dimensions generated by DecontX with cells colored by the estimated percntation of contamination.

Usage

plotDecontXContamination(
  x,
  batch = NULL,
  colorScale = c("blue", "green", "yellow", "orange", "red"),
  size = 1
)

Arguments

x
  Either a SingleCellExperiment with decontX results stored in metadata(x)$decontX or the result from running decontX on a count matrix.

batch
  Character. Batch of cells to plot. If NULL, then the first batch in the list will be selected. Default NULL.

colorScale
  Character vector. Contains the color spectrum to be passed to scale_colour_gradientn from package ‘ggplot2’. Default c("blue","green","yellow","orange","red").

size

Value

Returns a ggplot object.

Author(s)

Shiyi Yang, Joshua Campbell

See Also

See decontX for a full example of how to estimate and plot contamination.
plotDecontXMarkerExpression

Plots expression of marker genes before and after decontamination

Description

Generates a violin plot that shows the counts of marker genes in cells across specific clusters or cell types. Can be used to view the expression of marker genes in different cell types before and after decontamination with decontX.

Usage

plotDecontXMarkerExpression(
  x,
  markers,
  groupClusters = NULL,
  assayName = c("counts", "decontXcounts"),
  z = NULL,
  exactMatch = TRUE,
  by = "rownames",
  log1p = FALSE,
  ncol = NULL,
  plotDots = FALSE,
  dotSize = 0.1
)

Arguments

x
Either a SingleCellExperiment or a matrix-like object of counts.

markers
Character Vector or List. A character vector or list of character vectors with the names of the marker genes of interest.

groupClusters
List. A named list that allows cell clusters labels coded in z to be regrouped and renamed on the fly. For example, list(Tcells=c(1, 2), Bcells=7) would recode clusters 1 and 2 to "Tcells" and cluster 7 to "Bcells". Note that if this is used, clusters in z not found in groupClusters will be excluded. Default NULL.

assayName
Character vector. Name(s) of the assay(s) to plot if x is a SingleCellExperiment. If more than one assay is listed, then side-by-side violin plots will be generated. Default c("counts", "decontXcounts").

z
Character, Integer, or Vector. Indicates the cluster labels for each cell. If x is a SingleCellExperiment and z = NULL, then the cluster labels from decontX will be retrieved from the colData of x (i.e. colData(x)$decontX_clusters). If z is a single character or integer, then that column will be retrieved from colData of x (i.e. colData(x)[,z]). If x is a counts matrix, then z will need to be a vector the same length as the number of columns in x that indicate the cluster to which each cell belongs. Default NULL.
plotDecontXMarkerPercentage

Plots percentage of cells cell types expressing markers

Description

Generates a barplot that shows the percentage of cells within clusters or cell types that have detectable levels of given marker genes. Can be used to view the expression of marker genes in different cell types before and after decontamination with decontX.

Usage

plotDecontXMarkerPercentage(
  x,
  markers,
  groupClusters = NULL,
  assayName = c("counts", "decontXcounts"),
  z = NULL,
  threshold = 1,
  exactMatch = TRUE,
Arguments

x
Either a SingleCellExperiment or a matrix-like object of counts.

markers
List. A named list indicating the marker genes for each cell type of interest. Multiple markers can be supplied for each cell type. For example, `list(Tcell_Markers=c("CD3E", "CD3D"), Bcell_Markers=c("CD79A", "CD79B", "MS4A1"))` would specify markers for human T-cells and B-cells. A cell will be considered "positive" for a cell type if it has a count greater than `threshold` for at least one of the marker genes in the list.

groupClusters
List. A named list that allows cell clusters labels coded in `z` to be regrouped and renamed on the fly. For example, `list(Tcells=c(1, 2), Bcells=7)` would recode clusters 1 and 2 to "Tcells" and cluster 7 to "Bcells". Note that if this is used, clusters in `z` not found in `groupClusters` will be excluded from the barplot. Default `NULL`.

assayName
Character vector. Name(s) of the assay(s) to plot if `x` is a SingleCellExperiment. If more than one assay is listed, then side-by-side barplots will be generated. Default `c("counts", "decontXcounts")`.

z
Character, Integer, or Vector. Indicates the cluster labels for each cell. If `x` is a SingleCellExperiment and `z = NULL`, then the cluster labels from `decontX` will be retrieved from the colData of `x` (i.e. `colData(x)$decontX_clusters`). If `z` is a single character or integer, then that column will be retrieved from `colData` of `x` (i.e. `colData(x)[,z]`). If `x` is a counts matrix, then `z` will need to be a vector the same length as the number of columns in `x` that indicate the cluster to which each cell belongs. Default `NULL`.

threshold
Numeric. Markers greater than or equal to this value will be considered detected in a cell. Default 1.

exactMatch
Boolean. Whether to only identify exact matches for the markers or to identify partial matches using `grep`. See `retrieveFeatureIndex` for more details. Default `TRUE`.

by
Character. Where to search for the markers if `x` is a SingleCellExperiment. See `retrieveFeatureIndex` for more details. If `x` is a matrix, then this must be set to "rownames". Default "rownames".

ncol
Integer. Number of columns to make in the plot. Default `round(sqrt(length(markers)))`.

labelBars
Boolean. Whether to display percentages above each bar. Default `TRUE`.

labelSize
Numeric. Size of the percentage labels in the barplot. Default 3.

Value

Returns a ggplot object.
Author(s)
Shiyi Yang, Joshua Campbell

See Also
See decontX for a full example of how to estimate and plot contamination.

plotDimReduceCluster  Plotting the cell labels on a dimension reduction plot

Description
Create a scatterplot for each row of a normalized gene expression matrix where x and y axis are from a data dimension reduction tool. The cells are colored by "celda_cell_cluster" column in colData(altExp(x, altExpName)) if x is a SingleCellExperiment object, or x if x is a integer vector of cell cluster labels.

Usage
plotDimReduceCluster(
  x,
  reducedDimName,
  altExpName = "featureSubset",
  dim1 = NULL,
  dim2 = NULL,
  size = 0.5,
  xlab = NULL,
  ylab = NULL,
  specificClusters = NULL,
  labelClusters = FALSE,
  groupBy = NULL,
  labelSize = 3.5
)

# S4 method for signature 'SingleCellExperiment'
plotDimReduceCluster(
  x,
  reducedDimName,
  altExpName = "featureSubset",
  dim1 = 1,
  dim2 = 2,
  size = 0.5,
  xlab = NULL,
  ylab = NULL,
  specificClusters = NULL,
  labelClusters = FALSE,
  groupBy = NULL,
Arguments

x Integer vector of cell cluster labels or a SingleCellExperiment object containing cluster labels for each cell in "celda_cell_cluster" column in colData(x).

reducedDimName The name of the dimension reduction slot in reducedDimNames(x) if x is a SingleCellExperiment object. Ignored if both dim1 and dim2 are set.

altExpName The name for the altExp slot to use. Default "featureSubset".

dim1 Integer or numeric vector. If reducedDimName is supplied, then, this will be used as an index to determine which dimension will be plotted on the x-axis. If reducedDimName is not supplied, then this should be a vector which will be plotted on the x-axis. Default 1.

dim2 Integer or numeric vector. If reducedDimName is supplied, then, this will be used as an index to determine which dimension will be plotted on the y-axis. If reducedDimName is not supplied, then this should be a vector which will be plotted on the y-axis. Default 2.

size Numeric. Sets size of point on plot. Default 0.5.

xlab Character vector. Label for the x-axis. Default NULL.

ylab Character vector. Label for the y-axis. Default NULL.

specificClusters Numeric vector. Only color cells in the specified clusters. All other cells will be grey. If NULL, all clusters will be colored. Default NULL.

labelClusters Logical. Whether the cluster labels are plotted. Default FALSE.

groupBy Character vector. Contains sample labels for each cell. If NULL, all samples will be plotted together. Default NULL.

labelSize Numeric. Sets size of label if labelClusters is TRUE. Default 3.5.

Value

The plot as a ggplot object
Examples

```
data(sceCeldaCG)
sce <- celdaTsne(sceCeldaCG)
plotDimReduceCluster(x = sce,
    reducedDimName = "celda_tSNE",
    specificClusters = c(1, 2, 3))
library(SingleCellExperiment)
data(sceCeldaCG, celdaCGMod)
sce <- celdaTsne(sceCeldaCG)
plotDimReduceCluster(x = celdaClusters(celdaCGMod)$z,
    dim1 = reducedDim(altExp(sce), "celda_tSNE")[, 1],
    dim2 = reducedDim(altExp(sce), "celda_tSNE")[, 2],
    specificClusters = c(1, 2, 3))
```

Description

Create a scatterplot for each row of a normalized gene expression matrix where x and y axis are from a data dimension reduction tool. The cells are colored by expression of the specified feature.

Usage

```
plotDimReduceFeature(
    x, features,
    reducedDimName = NULL,
    displayName = NULL,
    dim1 = NULL,
    dim2 = NULL,
    headers = NULL,
    useAssay = "counts",
    altExpName = "featureSubset",
    normalize = FALSE,
    zscore = TRUE,
    exactMatch = TRUE,
    trim = c(-2, 2),
    limits = c(-2, 2),
    size = 0.5,
    xlab = NULL,
    ylab = NULL,
    colorLow = "blue4",
    colorMid = "grey90",
    colorHigh = "firebrick1",
    midpoint = 0,
    ncol = NULL,
)
## S4 method for signature 'SingleCellExperiment'
plotDimReduceFeature(
  x,
  features,
  reducedDimName = NULL,
  displayName = NULL,
  dim1 = 1,
  dim2 = 2,
  headers = NULL,
  useAssay = "counts",
  altExpName = "featureSubset",
  normalize = FALSE,
  zscore = TRUE,
  exactMatch = TRUE,
  trim = c(-2, 2),
  limits = c(-2, 2),
  size = 0.5,
  xlab = NULL,
  ylab = NULL,
  colorLow = "blue4",
  colorMid = "grey90",
  colorHigh = "firebrick1",
  midpoint = 0,
  ncol = NULL,
  decreasing = FALSE
)

## S4 method for signature 'ANY'
plotDimReduceFeature(
  x,
  features,
  dim1,
  dim2,
  headers = NULL,
  normalize = FALSE,
  zscore = TRUE,
  exactMatch = TRUE,
  trim = c(-2, 2),
  limits = c(-2, 2),
  size = 0.5,
  xlab = "Dimension_1",
  ylab = "Dimension_2",
  colorLow = "blue4",
  colorMid = "grey90",
  colorHigh = "firebrick1",
```
midpoint = 0,
ncol = NULL,
decreasing = FALSE
)
```

### Arguments

- **x**
  - Numeric matrix or a `SingleCellExperiment` object with the matrix located in the assay slot under `useAssay`. Rows represent features and columns represent cells.

- **features**
  - Character vector. Features in the rownames of counts to plot.

- **reducedDimName**
  - The name of the dimension reduction slot in `reducedDimNames(x)` if `x` is a `SingleCellExperiment` object. If `NULL`, then both `dim1` and `dim2` need to be set. Default `NULL`.

- **displayName**
  - Character. The column name of `rowData(x)` that specifies the display names for the features. Default `NULL`, which displays the row names. Only works if `x` is a `SingleCellExperiment` object. Overwrites `headers`.

- **dim1**
  - Integer or numeric vector. If `reducedDimName` is supplied, then, this will be used as an index to determine which dimension will be plotted on the x-axis. If `reducedDimName` is not supplied, then this should be a vector which will be plotted on the x-axis. Default `1`.

- **dim2**
  - Integer or numeric vector. If `reducedDimName` is supplied, then, this will be used as an index to determine which dimension will be plotted on the y-axis. If `reducedDimName` is not supplied, then this should be a vector which will be plotted on the y-axis. Default `2`.

- **headers**
  - Character vector. If `NULL`, the corresponding rownames are used as labels. Otherwise, these headers are used to label the features. Only works if `displayName` is `NULL` and `exactMatch` is `FALSE`.

- **useAssay**
  - A string specifying which `assay` slot to use if `x` is a `SingleCellExperiment` object. Default "counts".

- **altExpName**
  - The name for the `altExp` slot to use. Default "featureSubset".

- **normalize**
  - Logical. Whether to normalize the columns of 'counts'. Default `FALSE`.

- **zscore**
  - Logical. Whether to scale each feature to have a mean 0 and standard deviation of 1. Default `TRUE`.

- **exactMatch**
  - Logical. Whether an exact match or a partial match using `grep()` is used to look up the feature in the rownames of the counts matrix. Default `TRUE`.

- **trim**
  - Numeric vector. Vector of length two that specifies the lower and upper bounds for the data. This threshold is applied after row scaling. Set to `NULL` to disable. Default `c(-1,1)`.

- **limits**
  - Passed to `scale_colour_gradient2`. The range of color scale.

- **size**
  - Numeric. Sets size of point on plot. Default `1`.

- **xlab**
  - Character vector. Label for the x-axis. If `reducedDimName` is used, then this will be set to the column name of the first dimension of that object. Default "Dimension_1".
### plotDimReduceGrid

Mapping the dimension reduction plot

#### Description

Creates a scatterplot given two dimensions from a data dimension reduction tool (e.g tSNE) output.

#### ylab
Character vector. Label for the y-axis. If `reducedDimName` is used, then this will be set to the column name of the second dimension of that object. Default "Dimension_2".

#### colorLow
Character. A color available from `colors()`. The color will be used to signify the lowest values on the scale.

#### colorMid
Character. A color available from `colors()`. The color will be used to signify the midpoint on the scale.

#### colorHigh
Character. A color available from `colors()`. The color will be used to signify the highest values on the scale.

#### midpoint
Numeric. The value indicating the midpoint of the diverging color scheme. If NULL, defaults to the mean with 10 percent of values trimmed. Default 0.

#### ncol
Integer. Passed to `facet_wrap`. Specify the number of columns for facet wrap.

#### decreasing
logical. Specifies the order of plotting the points. If FALSE, the points will be plotted in increasing order where the points with largest values will be on top. TRUE otherwise. If NULL, no sorting is performed. Points will be plotted in their current order in x. Default FALSE.

#### Value

The plot as a ggplot object

#### Examples

```r
data(sceCeldaCG)
sce <- celdaTsne(sceCeldaCG)
plotDimReduceFeature(x = sce, 
                     reducedDimName = "celda_tSNE", 
                     normalize = TRUE, 
                     features = c("Gene_98", "Gene_99"), 
                     exactMatch = TRUE)
```

```r
library(SingleCellExperiment)
data(sceCeldaCG)
sce <- celdaTsne(sceCeldaCG)
plotDimReduceFeature(x = counts(sce), 
                     dim1 = reducedDim(altExp(sce), "celda_tSNE")[, 1], 
                     dim2 = reducedDim(altExp(sce), "celda_tSNE")[, 2], 
                     normalize = TRUE, 
                     features = c("Gene_98", "Gene_99"), 
                     exactMatch = TRUE)
```
Usage

plotDimReduceGrid(
  x,
  reducedDimName,
  dim1 = NULL,
  dim2 = NULL,
  useAssay = "counts",
  altExpName = "featureSubset",
  size = 1,
  xlab = "Dimension_1",
  ylab = "Dimension_2",
  limits = c(-2, 2),
  colorLow = "blue4",
  colorMid = "grey90",
  colorHigh = "firebrick1",
  midpoint = 0,
  varLabel = NULL,
  ncol = NULL,
  headers = NULL,
  decreasing = FALSE
)

## S4 method for signature 'SingleCellExperiment'
plotDimReduceGrid(
  x,
  reducedDimName,
  dim1 = NULL,
  dim2 = NULL,
  useAssay = "counts",
  altExpName = "featureSubset",
  size = 1,
  xlab = "Dimension_1",
  ylab = "Dimension_2",
  limits = c(-2, 2),
  colorLow = "blue4",
  colorMid = "grey90",
  colorHigh = "firebrick1",
  midpoint = 0,
  varLabel = NULL,
  ncol = NULL,
  headers = NULL,
  decreasing = FALSE
)

## S4 method for signature 'ANY'
plotDimReduceGrid(
  x,
  dim1,
plotDimReduceGrid

```r
dim2,
size = 1,
xlab = "Dimension_1",
ylab = "Dimension_2",
limits = c(-2, 2),
colorLow = "blue4",
colorMid = "grey90",
colorHigh = "firebrick1",
midpoint = 0,
varLabel = NULL,
ncol = NULL,
headers = NULL,
decreasing = FALSE
)
```

Arguments

- **x**: Numeric matrix or a `SingleCellExperiment` object with the matrix located in the assay slot under `useAssay`. Each row of the matrix will be plotted as a separate facet.
- **reducedDimName**: The name of the dimension reduction slot in `reducedDimNames(x)` if `x` is a `SingleCellExperiment` object. Ignored if both `dim1` and `dim2` are set.
- **dim1**: Numeric vector. Second dimension from data dimension reduction output.
- **dim2**: Numeric vector. Second dimension from data dimension reduction output.
- **useAssay**: A string specifying which assay slot to use if `x` is a `SingleCellExperiment` object. Default "counts".
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".
- **size**: Numeric. Sets size of point on plot. Default 1.
- **xlab**: Character vector. Label for the x-axis. Default 'Dimension_1'.
- **ylab**: Character vector. Label for the y-axis. Default 'Dimension_2'.
- **limits**: Passed to `scale_colour_gradient2`. The range of color scale.
- **colorLow**: Character. A color available from `colors()`. The color will be used to signify the lowest values on the scale. Default "blue4".
- **colorMid**: Character. A color available from `colors()`. The color will be used to signify the midpoint on the scale. Default "grey90".
- **colorHigh**: Character. A color available from `colors()`. The color will be used to signify the highest values on the scale. Default "firebrick1".
- **midpoint**: Numeric. The value indicating the midpoint of the diverging color scheme. If `NULL`, defaults to the mean with 10 percent of values trimmed. Default 0.
- **varLabel**: Character vector. Title for the color legend.
- **ncol**: Integer. Passed to `facet_wrap`. Specify the number of columns for facet wrap.
- **headers**: Character vector. If 'NULL', the corresponding rownames are used as labels. Otherwise, these headers are used to label the genes.
decreasing logical. Specifies the order of plotting the points. If FALSE, the points will be plotted in increasing order where the points with largest values will be on top. TRUE otherwise. If NULL, no sorting is performed. Points will be plotted in their current order in x. Default FALSE.

Value

The plot as a ggplot object

Examples

```r
data(sceCeldaCG)
sce <- celdaTsne(sceCeldaCG)
plotDimReduceGrid(x = sce,
    reducedDimName = "celda_tSNE",
    xlab = "Dimension1",
    ylab = "Dimension2",
    varLabel = "tSNE")
library(SingleCellExperiment)
data(sceCeldaCG)
sce <- celdaTsne(sceCeldaCG)
plotDimReduceGrid(x = counts(sce),
    dim1 = reducedDim(altExp(sce), "celda_tSNE")[, 1],
    dim2 = reducedDim(altExp(sce), "celda_tSNE")[, 2],
    xlab = "Dimension1",
    ylab = "Dimension2",
    varLabel = "tSNE")
```

plotDimReduceModule  Plotting Celda module probability on a dimension reduction plot

Description

Create a scatterplot for each row of a normalized gene expression matrix where x and y axis are from a data dimension reduction tool. The cells are colored by the module probability.

Usage

```r
plotDimReduceModule(
    x,
    reducedDimName,
    useAssay = "counts",
    altExpName = "featureSubset",
    celdaMod,
    modules = NULL,
    dim1 = NULL,
    dim2 = NULL,
    size = 0.5,
    xlab = NULL,
)```
plotDimReduceModule

ylab = NULL,
rescale = TRUE,
limits = c(0, 1),
colorLow = "grey90",
colorHigh = "firebrick1",
ncol = NULL,
decreasing = FALSE
)

## S4 method for signature 'SingleCellExperiment'
plotDimReduceModule(
  x,
  reducedDimName,
  useAssay = "counts",
  altExpName = "featureSubset",
  modules = NULL,
  dim1 = 1,
  dim2 = 2,
  size = 0.5,
  xlab = NULL,
  ylab = NULL,
  rescale = TRUE,
  limits = c(0, 1),
  colorLow = "grey90",
  colorHigh = "firebrick1",
  ncol = NULL,
  decreasing = FALSE
)

## S4 method for signature 'ANY'
plotDimReduceModule(
  x,
  celdaMod,
  modules = NULL,
  dim1,
  dim2,
  size = 0.5,
  xlab = "Dimension_1",
  ylab = "Dimension_2",
  rescale = TRUE,
  limits = c(0, 1),
  colorLow = "grey90",
  colorHigh = "firebrick1",
  ncol = NULL,
  decreasing = FALSE
)
Arguments

- **x**: Numeric matrix or a `SingleCellExperiment` object with the matrix located in the assay slot under `useAssay`. Rows represent features and columns represent cells.
- **reducedDimName**: The name of the dimension reduction slot in `reducedDimNames(x)` if `x` is a `SingleCellExperiment` object. Ignored if both `dim1` and `dim2` are set.
- **useAssay**: A string specifying which assay slot to use if `x` is a `SingleCellExperiment` object. Default "counts".
- **altExpName**: The name for the altExp slot to use. Default "featureSubset".
- **celdaMod**: Celda object of class "celda_G" or "celda(CG". Used only if `x` is a matrix object.
- **modules**: Character vector. Module(s) from celda model to be plotted. e.g. c("1", "2").
- **dim1**: Integer or numeric vector. If `reducedDimName` is supplied, then, this will be used as an index to determine which dimension will be plotted on the x-axis. If `reducedDimName` is not supplied, then this should be a vector which will be plotted on the x-axis. Default 1.
- **dim2**: Integer or numeric vector. If `reducedDimName` is supplied, then, this will be used as an index to determine which dimension will be plotted on the y-axis. If `reducedDimName` is not supplied, then this should be a vector which will be plotted on the y-axis. Default 2.
- **size**: Numeric. Sets size of point on plot. Default 0.5.
- **xlab**: Character vector. Label for the x-axis. Default "Dimension_1".
- **ylab**: Character vector. Label for the y-axis. Default "Dimension_2".
- **rescale**: Logical. Whether rows of the matrix should be rescaled to [0, 1]. Default TRUE.
- **limits**: Passed to `scale_colour_gradient`. The range of color scale.
- **colorLow**: Character. A color available from `colors()`. The color will be used to signify the lowest values on the scale.
- **colorHigh**: Character. A color available from `colors()`. The color will be used to signify the highest values on the scale.
- **ncol**: Integer. Passed to `facet_wrap`. Specify the number of columns for facet wrap.
- **decreasing**: Logical. Specifies the order of plotting the points. If FALSE, the points will be plotted in increasing order where the points with largest values will be on top. TRUE otherwise. If NULL, no sorting is performed. Points will be plotted in their current order in x. Default FALSE.

Value

The plot as a ggplot object

Examples

data(sceCeldaCG)
sce <- celdaTsne(sceCeldaCG)
plotDimReduceModule(x = sce, reducedDimName = "celda_tSNE",
```r
modules = c("1", "2")
library(SingleCellExperiment)
data(sceCeldaCG, celdaCGMod)
sce <- celdaTsne(sceCeldaCG)
plotDimReduceModule(x = counts(sce),
  dim1 = reducedDim(altExp(sce), "celda_tSNE")[, 1],
  dim2 = reducedDim(altExp(sce), "celda_tSNE")[, 2],
  celdaMod = celdaCGMod,
  modules = c("1", "2"))
```

**plotGridSearchPerplexity**

*Visualize perplexity of a list of celda models*

**Description**

Visualize perplexity of every model in a celdaList, by unique K/L combinations

**Usage**

```r
plotGridSearchPerplexity(x, altExpName = "featureSubset", sep = 5, alpha = 0.5)
```

```r
## S4 method for signature 'SingleCellExperiment'
plotGridSearchPerplexity(x, altExpName = "featureSubset", sep = 5, alpha = 0.5)
```

```r
## S4 method for signature 'celdaList'
plotGridSearchPerplexity(x, sep = 5, alpha = 0.5)
```

**Arguments**

- **x**
  - Can be one of
    - A `SingleCellExperiment` object returned from `celdaGridSearch`, `recursiveSplitModule`, or `recursiveSplitCell`. Must contain a list named "celda_grid_search" in metadata(x).
    - `celdaList` object.

- **altExpName**
  - The name for the `altExp` slot to use. Default "featureSubset". Only works if `x` is a `SingleCellExperiment` object.

- **sep**
  - Numeric. Breaks in the x axis of the resulting plot.

- **alpha**
  - Numeric. Passed to `geom_jitter`. Opacity of the points. Values of alpha range from 0 to 1, with lower values corresponding to more transparent colors.

**Value**

A `ggplot` plot object showing perplexity as a function of clustering parameters.
Examples

```r
data(sceCeldaCGGridSearch)
sce <- resamplePerplexity(sceCeldaCGGridSearch)
plotGridSearchPerplexity(sce)
data(celdaCGSim, celdaCGGridSearchRes)
## Run various combinations of parameters with `celdaGridSearch`
celdaCGGridSearchRes <- resamplePerplexity(
  celdaCGSim$counts,
  celdaCGGridSearchRes)
plotGridSearchPerplexity(celdaCGGridSearchRes)
```

---

**plotHeatmap**

*Plots heatmap based on Celda model*

---

**Description**

Renders a heatmap based on a matrix of counts where rows are features and columns are cells.

**Usage**

```r
plotHeatmap(
  counts,
  z = NULL,
  y = NULL,
  scaleRow = scale,
  trim = c(-2, 2),
  featureIx = NULL,
  cellIx = NULL,
  clusterFeature = TRUE,
  clusterCell = TRUE,
  colorScheme = c("divergent", "sequential"),
  colorSchemeSymmetric = TRUE,
  colorSchemeCenter = 0,
  col = NULL,
  annotationCell = NULL,
  annotationFeature = NULL,
  annotationColor = NULL,
  breaks = NULL,
  legend = TRUE,
  annotationLegend = TRUE,
  annotationNamesFeature = TRUE,
  annotationNamesCell = TRUE,
  showNamesFeature = FALSE,
  showNamesCell = FALSE,
  rowGroupOrder = NULL,
  colGroupOrder = NULL,
  hclustMethod = "ward.D2",
```
treeheightFeature = ifelse(clusterFeature, 50, 0),
treeheightCell = ifelse(clusterCell, 50, 0),
silent = FALSE,
)

Arguments

counts Numeric or sparse matrix. Normalized counts matrix where rows represent features and columns represent cells.
z Numeric vector. Denotes cell population labels.
y Numeric vector. Denotes feature module labels.
scaleRow Function. A function to scale each individual row. Set to NULL to disable. Occurs after normalization and log transformation. Default is 'scale' and thus will Z-score transform each row.
trim Numeric vector. Vector of length two that specifies the lower and upper bounds for the data. This threshold is applied after row scaling. Set to NULL to disable. Default c(-2,2).
featureIx Integer vector. Select features for display in heatmap. If NULL, no subsetting will be performed. Default NULL.
cellIx Integer vector. Select cells for display in heatmap. If NULL, no subsetting will be performed. Default NULL.
clusterFeature Logical. Determines whether rows should be clustered. Default TRUE.
clusterCell Logical. Determines whether columns should be clustered. Default TRUE.
colorScheme Character. One of "divergent" or "sequential". A "divergent" scheme is best for highlighting relative data (denoted by 'colorSchemeCenter') such as gene expression data that has been normalized and centered. A "sequential" scheme is best for highlighting data that are ordered low to high such as raw counts or probabilities. Default "divergent".
colorSchemeSymmetric Logical. When the colorScheme is "divergent" and the data contains both positive and negative numbers, TRUE indicates that the color scheme should be symmetric from \([-\max(\text{abs}(\text{data})), \max(\text{abs}(\text{data}))]\). For example, if the data ranges goes from -1.5 to 2, then setting this to TRUE will force the color scheme to range from -2 to 2. Default TRUE.
colorSchemeCenter Numeric. Indicates the center of a "divergent" colorScheme. Default 0.
col Color for the heatmap.
annotationCell Data frame. Additional annotations for each cell will be shown in the column color bars. The format of the data frame should be one row for each cell and one column for each annotation. Numeric variables will be displayed as continuous color bars and factors will be displayed as discrete color bars. Default NULL.
annotationFeature A data frame for the feature annotations (rows).
annotationColor
List. Contains color scheme for all annotations. See ‘?pheatmap’ for more details.

breaks
Numeric vector. A sequence of numbers that covers the range of values in the normalized ‘counts’. Values in the normalized ‘matrix’ are assigned to each bin in ‘breaks’. Each break is assigned to a unique color from ‘col’. If NULL, then breaks are calculated automatically. Default NULL.

legend
Logical. Determines whether legend should be drawn. Default TRUE.

annotationLegend
Logical. Whether legend for all annotations should be drawn. Default TRUE.

annotationNamesFeature
Logical. Whether the names for features should be shown. Default TRUE.

annotationNamesCell
Logical. Whether the names for cells should be shown. Default TRUE.

showNamesFeature
Logical. Specifies if feature names should be shown. Default TRUE.

showNamesCell
Logical. Specifies if cell names should be shown. Default FALSE.

rowGroupOrder
Vector. Specifies the order of feature clusters when semisupervised clustering is performed on the y labels.

colGroupOrder
Vector. Specifies the order of cell clusters when semisupervised clustering is performed on the z labels.

hclustMethod
Character. Specifies the method to use for the ‘hclust’ function. See ‘?hclust’ for possible values. Default “ward.D2”.

treeheightFeature
Numeric. Width of the feature dendrogram. Set to 0 to disable plotting of this dendrogram. Default: if clusterFeature == TRUE, then treeheightFeature = 50, else treeheightFeature = 0.

treeheightCell
Numeric. Height of the cell dendrogram. Set to 0 to disable plotting of this dendrogram. Default: if clusterCell == TRUE, then treeheightCell = 50, else treeheightCell = 0.

silent
Logical. Whether to plot the heatmap.

... Other arguments to be passed to underlying pheatmap function.

Value
list A list containing dendrogram information and the heatmap grob

Examples

data(celdaCGSim, celdaCGMod)
plotHeatmap(celdaCGSim$counts,
  z = celdaClusters(celdaCGMod)$z, y = celdaClusters(celdaCGMod)$y
)
plotRPC

Visualize perplexity differences of a list of celda models

Description

Visualize perplexity differences of every model in a celdaList, by unique K/L combinations.

Usage

plotRPC(x, altExpName = "featureSubset", sep = 5, alpha = 0.5)

## S4 method for signature 'SingleCellExperiment'
plotRPC(x, altExpName = "featureSubset", sep = 5, alpha = 0.5)

## S4 method for signature 'celdaList'
plotRPC(x, sep = 5, alpha = 0.5)

Arguments

x Can be one of
- A SingleCellExperiment object returned from celdaGridSearch, recursiveSplitModule, or recursiveSplitCell. Must contain a list named "celda_grid_search" in metadata(x).
- celdaList object.

altExpName The name for the altExp slot to use. Default "featureSubset".

sep Numeric. Breaks in the x axis of the resulting plot.

alpha Numeric. Passed to geom_jitter. Opacity of the points. Values of alpha range from 0 to 1, with lower values corresponding to more transparent colors.

Value

A ggplot plot object showing perplexity differences as a function of clustering parameters.

Examples

data(sceCeldaCGGridSearch)
sce <- resamplePerplexity(sceCeldaCGGridSearch)
plotRPC(sce)
data(celdaCGSim, celdaCGGridSearchRes)
## Run various combinations of parameters with 'celdaGridSearch'
celdaCGGridSearchRes <- resamplePerplexity(
celdaCGSim$counts,
celdaCGGridSearchRes)
plotRPC(celdaCGGridSearchRes)
recodeClusterY  
*Recode feature module labels*

**Description**

Recode feature module clusters using a mapping in the `from` and `to` arguments.

**Usage**

```r
recodeClusterY(sce, from, to, altExpName = "featureSubset")
```

**Arguments**

- **sce**: SingleCellExperiment object returned from `celda_G` or `celda_CG`. Must contain column `celda_feature_module` in `rowData(altExp(sce, altExpName))`.
- **from**: Numeric vector. Unique values in the range of `seq(celdaModules(sce))` that correspond to the original module labels in `sce`.
- **to**: Numeric vector. Unique values in the range of `seq(celdaModules(sce))` that correspond to the new module labels.
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".

**Value**

@return SingleCellExperiment object with recoded feature module labels.

**Examples**

```r
data(sceCeldaCG)
sceReorderedY <- recodeClusterY(sceCeldaCG, c(1, 3), c(3, 1))
```

---

recodeClusterZ  
*Recode cell cluster labels*

**Description**

Recode cell subpopulation clusters using a mapping in the `from` and `to` arguments.

**Usage**

```r
recodeClusterZ(sce, from, to, altExpName = "featureSubset")
```
**Arguments**

- **sce** `SingleCellExperiment` object returned from `celda_C` or `celda.CG`. Must contain column `celda_cell_cluster` in `colData(altExp(sce, altExpName))`.
- **from** Numeric vector. Unique values in the range of `seq(max(as.integer(celdaClusters(sce, altExpName = altExpName))))` that correspond to the original cluster labels in `sce`.
- **to** Numeric vector. Unique values in the range of `seq(max(as.integer(celdaClusters(sce, altExpName = altExpName))))` that correspond to the new cluster labels.
- **altExpName** The name for the `altExp` slot to use. Default "featureSubset".

**Value**

`SingleCellExperiment` object with recoded cell cluster labels.

**Examples**

```r
data(sceCeldaCG)
sceReorderedZ <- recodeClusterZ(sceCeldaCG, c(1, 3), c(3, 1))
```

---

**recursiveSplitCell**

**Recursive cell splitting**

**Description**

Uses the `celda.C` model to cluster cells into population for range of possible K’s. The cell population labels of the previous "K-1" model are used as the initial values in the current model with K cell populations. The best split of an existing cell population is found to create the K-th cluster. This procedure is much faster than randomly initializing each model with a different K. If module labels for each feature are given in `yInit`, the `celda.CG` model will be used to split cell populations based on those modules instead of individual features. Module labels will also be updated during sampling and thus may end up slightly different than `yInit`.

**Usage**

```r
recursiveSplitCell(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  sampleLabel = NULL,
  initialK = 5,
  maxK = 25,
  tempL = NULL,
  yInit = NULL,
  alpha = 1,
  beta = 1,
  delta = 1,
)```
gamma = 1,
minCell = 3,
reorder = TRUE,
seed = 12345,
perplexity = TRUE,
doResampling = FALSE,
numResample = 5,
logfile = NULL,
verbose = TRUE
)

## S4 method for signature 'SingleCellExperiment'
recursiveSplitCell(
x,
useAssay = "counts",
altExpName = "featureSubset",
sampleLabel = NULL,
initialK = 5,
maxK = 25,
tempL = NULL,
yInit = NULL,
alpha = 1,
beta = 1,
delta = 1,
gamma = 1,
minCell = 3,
reorder = TRUE,
seed = 12345,
perplexity = TRUE,
doResampling = FALSE,
numResample = 5,
logfile = NULL,
verbose = TRUE
)

## S4 method for signature 'matrix'
recursiveSplitCell(
x,
useAssay = "counts",
altExpName = "featureSubset",
sampleLabel = NULL,
initialK = 5,
maxK = 25,
tempL = NULL,
yInit = NULL,
alpha = 1,
beta = 1,
delta = 1,
gamma = 1,
minCell = 3,
reorder = TRUE,
seed = 12345,
perplexity = TRUE,
doResampling = FALSE,
numResample = 5,
logfile = NULL,
verbose = TRUE
)

Arguments

x A numeric matrix of counts or a SingleCellExperiment with the matrix located in
the assay slot under useAssay. Rows represent features and columns represent
cells.

useAssay A string specifying the name of the assay slot to use. Default "counts".

altExpName The name for the altExp slot to use. Default "featureSubset".
sampleLabel Vector or factor. Denotes the sample label for each cell (column) in the count
matrix.

initialK Integer. Initial number of cell populations to try. Default 5.

maxK Integer. Maximum number of cell populations to try. Default 25.

tempL Integer. Number of temporary modules to identify and use in cell splitting.
Only used if yInit = NULL. Collapsing features to a relatively smaller number
of modules will increase the speed of clustering and tend to produce better cell
populations. This number should be larger than the number of true modules
expected in the dataset. Default NULL.

yInit Integer vector. Module labels for features. Cells will be clustered using the
celda_CG model based on the modules specified in yInit rather than the counts
of individual features. While the features will be initialized to the module labels
in yInit, the labels will be allowed to move within each new model with a
different K.

alpha Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell
population in each sample. Default 1.

beta Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature
in each cell (if yInit is NULL) or to each module in each cell population (if
yInit is set). Default 1.

delta Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature
in each module. Only used if yInit is set. Default 1.

gamma Numeric. Concentration parameter for Eta. Adds a pseudocount to the number
of features in each module. Only used if yInit is set. Default 1.

minCell Integer. Only attempt to split cell populations with at least this many cells.

reorder Logical. Whether to reorder cell populations using hierarchical clustering after
each model has been created. If FALSE, cell populations numbers will corre-
spond to the split which created the cell populations (i.e. 'K15' was created at
split 15, 'K16' was created at split 16, etc.). Default TRUE.
recursiveSplitCell

seed  Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

perplexity  Logical. Whether to calculate perplexity for each model. If FALSE, then perplexity can be calculated later with resamplePerplexity. Default TRUE.

doResampling  Boolean. If TRUE, then each cell in the counts matrix will be resampled according to a multinomial distribution to introduce noise before calculating perplexity. Default FALSE.

numResample  Integer. The number of times to resample the counts matrix for evaluating perplexity if doResampling is set to TRUE. Default 5.

logfile  Character. Messages will be redirected to a file named "logfile". If NULL, messages will be printed to stdout. Default NULL.

verbose  Logical. Whether to print log messages. Default TRUE.

Value

A SingleCellExperiment object. Function parameter settings and celda model results are stored in the metadata "celda_grid_search" slot. The models in the list will be of class celda_C if yInit = NULL or celda_CG if zInit is set.

See Also

recursiveSplitModule for recursive splitting of feature modules.

Examples

data(sceCeldaCG)
## Create models that range from K = 3 to K = 7 by recursively splitting
## cell populations into two to produce \link{celda_C} cell clustering models
sce <- recursiveSplitCell(sceCeldaCG, initialK = 3, maxK = 7)

## Alternatively, first identify features modules using
## \link{recursiveSplitModule}
moduleSplit <- recursiveSplitModule(sceCeldaCG, initialL = 3, maxL = 15)
plotGridSearchPerplexity(moduleSplit)
moduleSplitSelect <- subsetCeldaList(moduleSplit, list(L = 10))

## Then use module labels for initialization in \link{recursiveSplitCell} to
## produce \link{celda_CG} bi-clustering models
cellSplit <- recursiveSplitCell(sceCeldaCG,
    initialK = 3, maxK = 7, yInit = celdaModules(moduleSplitSelect))
plotGridSearchPerplexity(cellSplit)
sce <- subsetCeldaList(cellSplit, list(K = 5, L = 10))
data(celdaCGSim, celdaCSim)

## Create models that range from K = 3 to K = 7 by recursively splitting
## cell populations into two to produce \link{celda_C} cell clustering models
sce <- recursiveSplitCell(celdaCSim$counts, initialK = 3, maxK = 7)

## Alternatively, first identify features modules using
## \link{recursiveSplitModule}
moduleSplit <- recursiveSplitModule(celdaCGSim$counts,
Uses the \code{celda_G} model to cluster features into modules for a range of possible L's. The module labels of the previous "L-1" model are used as the initial values in the current model with L modules. The best split of an existing module is found to create the L-th module. This procedure is much faster than randomly initializing each model with a different L.

**Usage**

```r
recursiveSplitModule(
  x, 
  useAssay = "counts", 
  altExpName = "featureSubset", 
  initialL = 10, 
  maxL = 100, 
  tempK = 100, 
  zInit = NULL, 
  sampleLabel = NULL, 
  alpha = 1, 
  beta = 1, 
  delta = 1, 
  gamma = 1, 
  minFeature = 3, 
  reorder = TRUE, 
  seed = 12345, 
  perplexity = TRUE, 
  doResampling = FALSE, 
  numResample = 5, 
  verbose = TRUE, 
  logfile = NULL
)
```

**Description**

The \code{recursiveSplitModule} function identifies modules in a set of features for a range of L values. The function first utilizes the \code{celda_G} model to cluster features into modules, taking into account the initial module labels from the previous model. The best module split is determined for each L to create a new module, which is subsequently added to the existing model. This approach is notably more efficient than randomly initializing each model with a different L. The function allows for customization through various parameters, including the number of features to consider, reassessment of modules, and options for additional computational processes.

The function returns an object of class \code{celda_CG}, which contains the identified modules and other relevant information. It is important to note that the function was tested with a specific set of parameters (initialL = 3, maxL = 15) and under certain conditions to ensure accurate and efficient module identification.

The documentation also mentions a grid search for perplexity, which is a measure of how well the model fits the data. The grid search involves evaluating different L values to find the one that maximizes perplexity, thereby optimizing the model for the dataset.

The recursive nature of this function allows for a sequential process of splitting modules, starting from an initial model and progressively creating new modules. This approach is particularly useful in scenarios where the dataset is large and the number of possible modules is substantial. By employing this method, researchers can efficiently analyze complex datasets, identifying meaningful patterns and structures within the data.
recursiveSplitModule

## S4 method for signature 'matrix'
recursiveSplitModule(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  initialL = 10,
  maxL = 100,
  tempK = 100,
  zInit = NULL,
  sampleLabel = NULL,
  alpha = 1,
  beta = 1,
  delta = 1,
  gamma = 1,
  minFeature = 3,
  reorder = TRUE,
  seed = 12345,
  perplexity = TRUE,
  doResampling = FALSE,
  numResample = 5,
  verbose = TRUE,
  logfile = NULL
)
Arguments

x  A numeric matrix of counts or a SingleCellExperiment with the matrix located in the assay slot under useAssay. Rows represent features and columns represent cells.

useAssay  A string specifying which assay slot to use if x is a SingleCellExperiment object. Default "counts".

altExpName  The name for the altExp slot to use. Default "featureSubset".

initialL  Integer. Initial number of modules.

maxL  Integer. Maximum number of modules.

tempK  Integer. Number of temporary cell populations to identify and use in module splitting. Only used if zInit = NULL. Collapsing cells to a relatively smaller number of cell populations will increase the speed of module clustering and tend to produce better modules. This number should be larger than the number of true cell populations expected in the dataset. Default 100.

zInit  Integer vector. Collapse cells to cell populations based on labels in zInit and then perform module splitting. If NULL, no collapsing will be performed unless tempK is specified. Default NULL.

sampleLabel  Vector or factor. Denotes the sample label for each cell (column) in the count matrix. Default NULL.

alpha  Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Only used if zInit is set. Default 1.

beta  Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell. Default 1.

delta  Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1.

gamma  Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 1.

minFeature  Integer. Only attempt to split modules with at least this many features.

reorder  Logical. Whether to reorder modules using hierarchical clustering after each model has been created. If FALSE, module numbers will correspond to the split which created the module (i.e. 'L15' was created at split 15, 'L16' was created at split 16, etc.). Default TRUE.

seed  Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

perplexity  Logical. Whether to calculate perplexity for each model. If FALSE, then perplexity can be calculated later with resamplePerplexity. Default TRUE.

doResampling  Boolean. If TRUE, then each cell in the counts matrix will be resampled according to a multinomial distribution to introduce noise before calculating perplexity. Default FALSE.

numResample  Integer. The number of times to resample the counts matrix for evaluating perplexity if doResampling is set to TRUE. Default 5.

verbose  Logical. Whether to print log messages. Default TRUE.

logfile  Character. Messages will be redirected to a file named "logfile". If NULL, messages will be printed to stdout. Default NULL.
Value

A SingleCellExperiment object. Function parameter settings and celda model results are stored in the metadata "celda_grid_search" slot. The models in the list will be of class celda_G if zInit = NULL or celda_CG if zInit is set.

See Also

recursiveSplitCell for recursive splitting of cell populations.

Examples

data(sceCeldaCG)
## Create models that range from L=3 to L=20 by recursively splitting modules
## into two
moduleSplit <- recursiveSplitModule(sceCeldaCG, initialL = 3, maxL = 20)

## Example results with perplexity
plotGridSearchPerplexity(moduleSplit)

## Select model for downstream analysis
celdaMod <- subsetCeldaList(moduleSplit, list(L = 10))

# Repeat for celdaCGSim

data(celdaCGSim)
## Create models that range from L=3 to L=20 by recursively splitting modules
## into two
moduleSplit <- recursiveSplitModule(celdaCGSim$data, initialL = 3, maxL = 20)

## Example results with perplexity
plotGridSearchPerplexity(moduleSplit)

## Select model for downstream analysis
celdaMod <- subsetCeldaList(moduleSplit, list(L = 10))

reorderCelda
Reorder cells populations and/or features modules using hierarchical clustering

Description

Apply hierarchical clustering to reorder the cell populations and/or feature modules and group similar ones together based on the cosine distance of the factorized matrix from factorizeMatrix.

Usage

reorderCelda(
  x,
  celdaMod,
  useAssay = "counts",
  altExpName = "featureSubset",
)
reorderCelda

method = "complete"
)

## S4 method for signature 'SingleCellExperiment,ANY'
reorderCelda(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  method = "complete"
)

## S4 method for signature 'matrix,celda_CG'
reorderCelda(x, celdaMod, method = "complete")

## S4 method for signature 'matrix,celda_C'
reorderCelda(x, celdaMod, method = "complete")

## S4 method for signature 'matrix,celda_G'
reorderCelda(x, celdaMod, method = "complete")

Arguments

  x  Can be one of
      • A SingleCellExperiment object returned by celda_C, celda_G or celda_CG, with the matrix located in the useAssay assay slot in altExp(x, altExpName). Rows represent features and columns represent cells.
      • Integer count matrix. Rows represent features and columns represent cells. This matrix should be the same as the one used to generate celdaMod.
  celdaMod  Celda model object. Only works if x is an integer counts matrix. Ignored if x is a SingleCellExperiment object.
  useAssay  A string specifying which assay slot to use if x is a SingleCellExperiment object. Default "counts".
  altExpName  The name for the altExp slot. Default "featureSubset".
  method  Passed to hclust. The agglomeration method to be used to be used. Default "complete".

Value

  A SingleCellExperiment object (or Celda model object) with updated cell cluster and/or feature module labels.

Examples

data(sceCeldaCG)
reordersce <- reorderCelda(sceCeldaCG)
data(celdaCGSim, celdaCGMod)
reorderCeldaCG <- reorderCelda(celdaCGSim$counts, celdaCGMod)
data(celdaCSim, celdaCMod)
reorderCeldaC <- reorderCelda(celdaCSim$counts, celdaCMod)
data(celdaGSim, celdaGMod)
reorderCeldaG <- reorderCelda(celdaGSim$counts, celdaGMod)

---

**reportCeldaCG**

*Generate an HTML report for celda_CG*

**Description**

*reportCeldaCGRun* will run *recursiveSplitModule* and *recursiveSplitCell* to find the number of modules (L) and the number of cell populations (K). A final *celda_CG* model will be selected from *recursiveSplitCell*. After a *celda_CG* model has been fit, *reportCeldaCGPlotResults* can be used to create an HTML report for visualization and exploration of the *celda_CG* model results. Some of the plotting and feature selection functions require the installation of the Bioconductor package *singleCellTK*.

**Usage**

```r
reportCeldaCGRun(
  sce,
  L,
  K,
  sampleLabel = NULL,
  altExpName = "featureSubset",
  useAssay = "counts",
  initialL = 10,
  maxL = 150,
  initialK = 5,
  maxK = 50,
  minCell = 3,
  minCount = 3,
  maxFeatures = 5000,
  output_file = "CeldaCG_RunReport",
  output_sce_prefix = "celda_cg",
  output_dir = ".",
  pdf = FALSE,
  showSession = TRUE
)
```

```r
reportCeldaCGPlotResults(
  sce,
  reducedDimName,
  features = NULL,
  displayName = NULL,
  altExpName = "featureSubset",
  useAssay = "counts",
  cellAnnot = NULL,
)```
`reportCellaCG`

cellAnnotLabel = NULL,
exactMatch = TRUE,
moduleFilePrefix = "module_features",
output_file = "CellaCG_ResultReport",
output_dir = ".",
pdf = FALSE,
showSetup = TRUE,
showSession = TRUE
)

Arguments

c `sce` A SingleCellExperiment with the matrix located in the assay slot under useAssay. Rows represent features and columns represent cells.

L Integer. Final number of feature modules. See cella_CG for more information.

K Integer. Final number of cell populations. See cella_CG for more information.

sampleLabel Vector or factor. Denotes the sample label for each cell (column) in the count matrix.

altExpName The name for the altExp slot to use. Default "featureSubset".

useAssay A string specifying which assay slot to use. Default "counts".

initialL Integer. Minimum number of modules to try. See recursiveSplitModule for more information. Default 10.

maxL Integer. Maximum number of modules to try. See recursiveSplitModule for more information. Default 150.

initialK Integer. Initial number of cell populations to try.

maxK Integer. Maximum number of cell populations to try.

minCell Integer. Minimum number of cells required for feature selection. See selectFeatures for more information. Default 3.

minCount Integer. Minimum number of counts required for feature selection. See selectFeatures for more information. Default 3.

maxFeatures Integer. Maximum number of features to include. If the number of features after filtering for minCell and minCount are greater than maxFeature, then Seurat’s VST function is used to select the top variable features. Default 5000.


output_sce_prefix Character. The sce object with cella_CG results will be saved to an .rds file starting with this prefix. Default cella_cg.

output_dir Character. Path to save the html file. Default ..

pdf Boolean. Whether to create PDF versions of each plot in addition to PNGs. Default FALSE.

showSession Boolean. Whether to show the session information at the end. Default TRUE.

reducedDimName Character. Name of the reduced dimensional object to be used in 2-D scatter plots throughout the report. Default cella_UMAP.
features  Character vector. Expression of these features will be displayed on a reduced dimensional plot defined by reducedDimName. If NULL, then no plotting of features on a reduced dimensional plot will be performed. Default NULL.

displayName  Character. The name to use for display in scatter plots and heatmaps. If NULL, then the rownames of the sce object will be used. This can also be set to the name of a column in the row data of sce or altExp(sce, altExpName). Default NULL.

cellAnnot  Character vector. The cell-level annotations to display on the reduced dimensional plot. These variables should be present in the column data of the sce object. Default NULL.

cellAnnotLabel  Character vector. Additional cell-level annotations to display on the reduced dimensional plot. Variables will be treated as categorial and labels for each group will be placed on the plot. These variables should be present in the column data of the sce object. Default NULL.

exactMatch  Boolean. Whether to only identify exact matches or to identify partial matches using grep. Default FALSE.

moduleFilePrefix  Character. The features in each module will be written to a csv file starting with this name. If NULL, then no file will be written. Default "module_features".

showSetup  Boolean. Whether to show the setup code at the beginning. Default TRUE.

Value

.html file

Examples

data(sceCeldaCG)

## Not run:
library(SingleCellExperiment)
sceCeldaCG$sum <- colSums(counts(sceCeldaCG))
rowData(sceCeldaCG)$rownames <- rownames(sceCeldaCG)
sceCeldaCG <- reportCeldaCGRun(sceCeldaCG,
   initialL = 5, maxL = 20, initialK = 5,
   maxK = 20, L = 10, K = 5)
reportCeldaCGPlotResults(sce = sceCeldaCG,
   reducedDimName = "celda_UMAP",
   features = c("Gene_1", "Gene_100"),
   displayName = "rownames",
   cellAnnot="sum")

## End(Not run)
resamplePerplexity

**Calculate and visualize perplexity of all models in a celdaList**

### Description

Calculates the perplexity of each model's cluster assignments given the provided `countMatrix`, as well as resamplings of that count matrix, providing a distribution of perplexities and a better sense of the quality of a given K/L choice.

### Usage

```r
resamplePerplexity(
  x,
  celdaList,
  useAssay = "counts",
  altExpName = "featureSubset",
  doResampling = FALSE,
  numResample = 5,
  seed = 12345
)
```

#### S4 method for signature 'SingleCellExperiment'

```r
resamplePerplexity(
  x,
  useAssay = "counts",
  altExpName = "featureSubset",
  doResampling = FALSE,
  numResample = 5,
  seed = 12345
)
```

#### S4 method for signature 'ANY'

```r
resamplePerplexity(
  x,
  celdaList,
  doResampling = FALSE,
  numResample = 5,
  seed = 12345
)
```

### Arguments

- **x**: A numeric `matrix` of counts or a `SingleCellExperiment` returned from `celdaGridSearch` with the matrix located in the assay slot under `useAssay`. Rows represent features and columns represent cells. Must contain "celda_grid_search" slot in `metadata(x)` if `x` is a `SingleCellExperiment` object.
- **celdaList**: Object of class 'celdaList'. Used only if `x` is a matrix object.
**useAssay**
A string specifying which assay slot to use if x is a SingleCellExperiment object. Default "counts".

**altExpName**
The name for the altExp slot to use. Default "featureSubset".

**doResampling**
Boolean. If TRUE, then each cell in the counts matrix will be resampled according to a multinomial distribution to introduce noise before calculating perplexity. Default FALSE.

**numResample**
Integer. The number of times to resample the counts matrix for evaluating perplexity if doResampling is set to TRUE. Default 5.

**seed**
Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

**Value**
A SingleCellExperiment object or celdaList object with a perplexity property, detailing the perplexity of all K/L combinations that appeared in the celdaList’s models.

**Examples**

data(sceCeldaCGGridSearch)
sce <- resamplePerplexity(sceCeldaCGGridSearch)
plotGridSearchPerplexity(sce)
data(celdaCGSim, celdaCGGridSearchRes)
celdaCGGridSearchRes <- resamplePerplexity(
  celdaCGSim$counts,
  celdaCGGridSearchRes
)
plotGridSearchPerplexity(celdaCGGridSearchRes)

---
resList

Get final celdaModels from a celda model SCE or celdaList object

**Description**
Returns all celda models generated during a celdaGridSearch run.

**Usage**
resList(x, altExpName = "featureSubset")

## S4 method for signature 'SingleCellExperiment'
resList(x, altExpName = "featureSubset")

## S4 method for signature 'celdaList'
resList(x)
retrieveFeatureIndex

Arguments

features Character vector of feature names to find in the rows of x.
x A data.frame, matrix, or SingleCellExperiment object to search.
by Character. Where to search for features in x. If set to "rownames" then the features will be searched for among rownames(x). If x inherits from class SummarizedExperiment, then by can be one of the fields in the row annotation data.frame (i.e. one of colnames(rowData(x))).
extactMatch Boolean. Whether to only identify exact matches or to identify partial matches using grep.
removeNA Boolean. If set to FALSE, features not found in x will be given NA and the returned vector will be the same length as features. If set to TRUE, then the NA values will be removed from the returned vector. Default FALSE.

Value

List. Contains one celdaModel object for each of the parameters specified in runParams(x).

Examples

data(sceCeldaCGGridSearch)
celdaCGGridModels <- resList(sceCeldaCGGridSearch)
data(celdaCGGridSearchRes)
celdaCGGridModels <- resList(celdaCGGridSearchRes)

retrieveFeatureIndex Retrieve row index for a set of features

Description

This will return indices of features among the rownames or rowData of a data.frame, matrix, or a SummarizedExperiment object including a SingleCellExperiment. Partial matching (i.e. grepping) can be used by setting exactMatch = FALSE.

Usage

r retrofitFeatureIndex(
  features,
  x,
  by = "rownames",
  exactMatch = TRUE,
  removeNA = FALSE
)

Arguments

features Character vector of feature names to find in the rows of x.
x An object of class SingleCellExperiment or celdaList.
altExpName The name for the altExp slot to use. Default "featureSubset".
runParams

Get run parameters from a celda model SingleCellExperiment or celdaList object

Description

Returns details on the clustering parameters and model priors from the celdaList object when it was created.

Usage

runParams(x, altExpName = "featureSubset")

## S4 method for signature 'SingleCellExperiment'
runParams(x, altExpName = "featureSubset")

## S4 method for signature 'celdaList'
runParams(x)

Arguments

x An object of class SingleCellExperiment or class celdaList.
altExpName The name for the altExp slot to use. Default "featureSubset".

Value

Data Frame. Contains details on the various K/L parameters, chain parameters, seed, and final log-likelihoods derived for each model in the provided celdaList.
**sampleCells**

**Examples**

```r
data(sceCeldaCGGridSearch)
runParams(sceCeldaCGGridSearch)
data(celdaCGGridSearchRes)
runParams(celdaCGGridSearchRes)
```

---

**sampleCells**

**Description**

A matrix of simulated gene counts.

**Usage**

```r
sampleCells
```

**Format**

A matrix of simulated gene counts with 10 rows (genes) and 10 columns (cells).

**Details**

A toy count matrix for use with celda.

Generated by Josh Campbell.

**Source**

http://github.com/campbio/celda

---

**sampleLabel**

*Get or set sample labels from a celda SingleCellExperiment object*

**Description**

Return or set the sample labels for the cells in sce.
Usage

```r
sampleLabel(x, altExpName = "featureSubset")

## S4 method for signature 'SingleCellExperiment'
sampleLabel(x, altExpName = "featureSubset")

sampleLabel(x, altExpName = "featureSubset") <- value

## S4 replacement method for signature 'SingleCellExperiment'
sampleLabel(x, altExpName = "featureSubset") <- value

## S4 method for signature 'celdaModel'
sampleLabel(x)
```

Arguments

- `x`: Can be one of
  - A `SingleCellExperiment` object returned by `celda_C`, `celda_G`, or `celda(CG)`, with the matrix located in the `useAssay` assay slot. Rows represent features and columns represent cells.
  - A `celda` model object.
- `altExpName`: The name for the `altExp` slot to use. Default "featureSubset".
- `value`: Character vector of sample labels for replacements. Works only if `x` is a `SingleCellExperiment` object.

Value

Character vector. Contains the sample labels provided at model creation, or those automatically generated by `celda`.

Examples

```r
data(sceCeldaCG)
sampleLabel(sceCeldaCG)
data(celdaCGMod)
sampleLabel(celdaCGMod)
```

Description

A `SingleCellExperiment` object containing the results of running `selectFeatures` and `celda_C` on `celdaCSim`.
Usage

sceCeldaC

Format

A SingleCellExperiment object

Examples

data(celdaCSim)
sceCeldaC <- selectFeatures(celdaCSim$counts)
sceCeldaC <- celda_C(sceCeldaC,
    K = celdaCSim$K,
    sampleLabel = celdaCSim$sampleLabel,
    nchains = 1)

sceCeldaCG

Description

A SingleCellExperiment object containing the results of running selectFeatures and celda_C on celdaCGSim.

Usage

sceCeldaCG

Format

A SingleCellExperiment object

Examples

data(celdaCGSim)
sceCeldaCG <- selectFeatures(celdaCGSim$counts)
sceCeldaCG <- celda_CG(sceCeldaCG,
    K = celdaCGSim$K,
    L = celdaCGSim$L,
    sampleLabel = celdaCGSim$sampleLabel,
    nchains = 1)
Description

A SingleCellExperiment object containing the results of running selectFeatures and celdaGridSearch on celdaCGSim.

Usage

sceCeldaCGGridSearch

Format

A SingleCellExperiment object

Examples

data(celdaCGSim)
sce <- selectFeatures(celdaCGSim$counts)
sceCeldaCGGridSearch <- celdaGridSearch(sce,
    model = "celda_CG",
    paramsTest = list(K = seq(4, 6), L = seq(9, 11)),
    paramsFixed = list(sampleLabel = celdaCGSim$sampleLabel),
    bestOnly = TRUE,
    nchains = 1,
    cores = 1,
    verbose = FALSE)
Examples

```r
data(celdaGSim)
sceCeldaG <- selectFeatures(celdaGSim$counts)
sceCeldaG <- celda_G(sceCeldaG, L = celdaGSim$L, nchains = 1)
```

**Description**

Select the chain with the best log likelihood for each combination of tested parameters from a SCE object generated by `celdaGridSearch` or from a `celdaList` object.

**Usage**

```r
selectBestModel(x, asList = FALSE, altExpName = "featureSubset")
```

## S4 method for signature 'SingleCellExperiment'
```
selectBestModel(x, asList = FALSE, altExpName = "featureSubset")
```

## S4 method for signature 'celdaList'
```
selectBestModel(x, asList = FALSE)
```

**Arguments**

- `x` Can be one of
  - A `SingleCellExperiment` object returned from `celdaGridSearch`, `recursiveSplitModule`, or `recursiveSplitCell`. Must contain a list named "celda_grid_search" in `metadata(x)`.
  - `celdaList` object.

- `asList` `TRUE` or `FALSE`. Whether to return the best model as a `celdaList` object or not. If `FALSE`, return the best model as a corresponding `celda` model object.

- `altExpName` The name for the `altExp` slot to use. Default "featureSubset".

**Value**

One of

- A new `SingleCellExperiment` object containing one model with the best log-likelihood for each set of parameters in `metadata(x)`. If there is only one set of parameters, a new `SingleCellExperiment` object with the matching model stored in the `metadata "celda_parameters"` slot will be returned. Otherwise, a new `SingleCellExperiment` object with the subset models stored in the `metadata "celda_grid_search"` slot will be returned.
- A new `celdaList` object containing one model with the best log-likelihood for each set of parameters. If only one set of parameters is in the `celdaList`, the best model will be returned directly instead of a `celdaList` object.
selectFeatures

Simple feature selection by feature counts

Description

A simple heuristic feature selection procedure. Select features with at least minCount counts in at least minCell cells. A SingleCellExperiment object with subset features will be stored in the altExp slot with name altExpName. The name of the assay slot in altExp will be the same as useAssay.

Usage

```r
selectFeatures(
  x,
  minCount = 3,
  minCell = 3,
  useAssay = "counts",
  altExpName = "featureSubset"
)
```

## S4 method for signature 'SingleCellExperiment'
```r
selectFeatures(
  x,
  minCount = 3,
  minCell = 3,
  useAssay = "counts",
  altExpName = "featureSubset"
)
```

## S4 method for signature 'matrix'
```r
selectFeatures(
  x,
  minCount = 3,
  minCell = 3,
  useAssay = "counts",
  altExpName = "featureSubset"
)
```

See Also
celdaGridSearch subsetCeldaList

Examples

```r
data(sceCeldaCGGridSearch)
## Returns same result as running celdaGridSearch with "bestOnly = TRUE"
sce <- selectBestModel(sceCeldaCGGridSearch)
data(celdaCGGridSearchRes)
## Returns same result as running celdaGridSearch with "bestOnly = TRUE"
cgsBest <- selectBestModel(celdaCGGridSearchRes)
```
Arguments

- **x**: A numeric matrix of counts or a `SingleCellExperiment` with the matrix located in the assay slot under `useAssay`. Rows represent features and columns represent cells.
- **minCount**: Minimum number of counts required for feature selection.
- **minCell**: Minimum number of cells required for feature selection.
- **useAssay**: A string specifying the name of the assay slot to use. Default "counts".
- **altExpName**: The name for the `altExp` slot to use. Default "featureSubset".

Value

A `SingleCellExperiment` object with a `altExpName` `altExp` slot. Function parameter settings are stored in the metadata "select_features" slot.

Examples

```r
data(sceCeldaCG)
sce <- selectFeatures(sceCeldaCG)
data(celdaCGSim)
sce <- selectFeatures(celdaCGSim$counts)
```

Description

A function to draw clustered heatmaps where one has better control over some graphical parameters such as cell size, etc.

The function also allows to aggregate the rows using kmeans clustering. This is advisable if number of rows is so big that R cannot handle their hierarchical clustering anymore, roughly more than 1000. Instead of showing all the rows separately one can cluster the rows in advance and show only the cluster centers. The number of clusters can be tuned with parameter `kmeansK`.

Usage

```r
semiPheatmap(mat, 
  color = colorRampPalette(rev(brewer.pal(n = 7, name = "RdYlBu")))(100), 
  kmeansK = NA, 
  breaks = NA, 
  borderColor = "grey60", 
  cellWidth = NA, 
  cellHeight = NA, 
  scale = "none", 
  clusterRows = TRUE, 
  clusterCols = TRUE,
```
semiPheatmap

clusteringDistanceRows = "euclidean",
clusteringDistanceCols = "euclidean",
clusteringMethod = "complete",
clusteringCallback = .identity2,
cutreeRows = NA,
cutreeCols = NA,
treeHeightRow = ifelse(clusterRows, 50, 0),
treeHeightCol = ifelse(clusterCols, 50, 0),
legend = TRUE,
legendBreaks = NA,
legendLabels = NA,
annotationRow = NA,
annotationCol = NA,
annotation = NA,
annotationColors = NA,
annotationLegend = TRUE,
annotationNamesRow = TRUE,
annotationNamesCol = TRUE,
dropLevels = TRUE,
showRownames = TRUE,
showColnames = TRUE,
main = NA,
fontSize = 10,
fontSizeRow = fontSize,
fontSizeCol = fontSize,
displayNumbers = FALSE,
numberFormat = "% .2f",
numberColor = "grey30",
fontSizeNumber = 0.8 * fontSize,
gapsRow = NULL,
gapsCol = NULL,
labelsRow = NULL,
labelsCol = NULL,
fileName = NA,
width = NA,
height = NA,
silent = FALSE,
rowLabel,
colLabel,
rowGroupOrder = NULL,
colGroupOrder = NULL,
...
)

Arguments

mat numeric matrix of the values to be plotted.

color vector of colors used in heatmap.
kmeansK  the number of kmeans clusters to make, if we want to aggregate the rows before drawing heatmap. If NA then the rows are not aggregated.
breaks   Numeric vector. A sequence of numbers that covers the range of values in the normalized 'counts'. Values in the normalized 'matrix' are assigned to each bin in 'breaks'. Each break is assigned to a unique color from 'col'. If NULL, then breaks are calculated automatically. Default NULL.
borderColor color of cell borders on heatmap, use NA if no border should be drawn.
cellWidth individual cell width in points. If left as NA, then the values depend on the size of plotting window.
cellHeight individual cell height in points. If left as NA, then the values depend on the size of plotting window.
scale character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. Corresponding values are "row", "column" and "none".
clusterRows boolean values determining if rows should be clustered or hclust object,
clusterCols boolean values determining if columns should be clustered or hclust object.
clusteringDistanceRows distance measure used in clustering rows. Possible values are "correlation" for Pearson correlation and all the distances supported by dist, such as "euclidean", etc. If the value is none of the above it is assumed that a distance matrix is provided.
clusteringDistanceCols distance measure used in clustering columns. Possible values the same as for clusteringDistanceRows.
clusteringMethod clustering method used. Accepts the same values as hclust.
clusteringCallback callback function to modify the clustering. Is called with two parameters: original hclust object and the matrix used for clustering. Must return a hclust object.
cutreeRows number of clusters the rows are divided into, based on the hierarchical clustering (using cutree), if rows are not clustered, the argument is ignored

cutreeCols similar to cutreeRows, but for columns
treeHeightRow the height of a tree for rows, if these are clustered. Default value 50 points.
treeHeightCol the height of a tree for columns, if these are clustered. Default value 50 points.
legend logical to determine if legend should be drawn or not.
legendBreaks vector of breakpoints for the legend.
legendLabels vector of labels for the legendBreaks.
annotationRow data frame that specifies the annotations shown on left side of the heatmap. Each row defines the features for a specific row. The rows in the data and in the annotation are matched using corresponding row names. Note that color schemes takes into account if variable is continuous or discrete.
annotationCol similar to annotationRow, but for columns.
annotation parameter that currently sets the annotationCol if it is missing.

annotationColors
-list for specifying annotationRow and annotationCol track colors manually. It is possible to define the colors for only some of the features. Check examples for details.

annotationLegend
-boolean value showing if the legend for annotation tracks should be drawn.

annotationNamesRow
-boolean value showing if the names for row annotation tracks should be drawn.

annotationNamesCol
-boolean value showing if the names for column annotation tracks should be drawn.

dropLevels
-logical to determine if unused levels are also shown in the legend.

showRownames
-boolean specifying if column names are be shown.

showColnames
-boolean specifying if column names are be shown.

main
-the title of the plot

fontSize
-base fontsize for the plot

fontSizeRow
-fontsize for rownames (Default: fontsize)

fontSizeCol
-fontsize for colnames (Default: fontsize)

displayNumbers
-logical determining if the numeric values are also printed to the cells. If this is a matrix (with same dimensions as original matrix), the contents of the matrix are shown instead of original values.

numberFormat
-format strings (C printf style) of the numbers shown in cells. For example ".2f" shows 2 decimal places and ".1e" shows exponential notation (see more in sprintf).

numberColor
-color of the text

fontSizeNumber
-fontsize of the numbers displayed in cells

gapsRow
-vector of row indices that show where to put gaps into heatmap. Used only if the rows are not clustered. See cutreeRow to see how to introduce gaps to clustered rows.

gapsCol
-similar to gapsRow, but for columns.

labelsRow
-custom labels for rows that are used instead of rownames.

labelsCol
-similar to labelsRow, but for columns.

fileName
-file path where to save the picture. Filetype is decided by the extension in the path. Currently following formats are supported: png, pdf, tiff, bmp, jpeg. Even if the plot does not fit into the plotting window, the file size is calculated so that the plot would fit there, unless specified otherwise.

width
-manual option for determining the output file width in inches.

height
-manual option for determining the output file height in inches.

silent
-do not draw the plot (useful when using the gtable output)

rowLabel
-row cluster labels for semi-clustering
colLabel  column cluster labels for semi-clustering
rowGroupOrder Vector. Specifies the order of feature clusters when semisupervised clustering is performed on the y labels.
colGroupOrder Vector. Specifies the order of cell clusters when semisupervised clustering is performed on the z labels.

... graphical parameters for the text used in plot. Parameters passed to grid.text, see gpar.

Value

Invisibly a list of components

- treeRow the clustering of rows as hclust object
- treeCol the clustering of columns as hclust object
- kmeans the kmeans clustering of rows if parameter kmeansK was specified

Author(s)

Raivo Kolde <rkolde@gmail.com>  

```r
# Create test matrix
test = matrix(rnorm(200), 20, 10)
test[seq(10), seq(1, 10, 2)] = test[seq(10), seq(1, 10, 2)] + 3
test[seq(11, 20), seq(2, 10, 2)] = test[seq(11, 20), seq(2, 10, 2)] + 2
test[seq(15, 20), seq(2, 10, 2)] = test[seq(15, 20), seq(2, 10, 2)] + 4
colnames(test) = paste("Test", seq(10), sep = "")
rownames(test) = paste("Gene", seq(20), sep = "")

# Draw heatmaps
pheatmap(test)
pheatmap(test, kmeansK = 2)
pheatmap(test, clusteringDistanceRows = "correlation")
pheatmap(test, color = colorRampPalette(c("navy", "white", "firebrick3"))(50))
pheatmap(test, cluster_row = FALSE)
pheatmap(test, legend = FALSE)

# Show text within cells
pheatmap(test, displayNumbers = TRUE)
pheatmap(test, displayNumbers = TRUE, numberFormat = "%.1e")
pheatmap(test, displayNumbers = TRUE, numberFormat = "%.1e")
pheatmap(test, displayNumbers = TRUE, numberFormat = "%.1e")

# Fix cell sizes and save to file with correct size
pheatmap(test, cellWidth = 15, cellHeight = 12,
main = "Example heatmap")
pheatmap(test, cellWidth = 15, cellHeight = 12, fontSize = 8, fileName = "test.pdf")

# Generate annotations for rows and columns
annotationCol = data.frame(CellType = factor(rep(c("CT1", "CT2"), 5)),
Time = seq(5))
rownames(annotationCol) = paste("Test", seq(10), sep = "")

annotationRow = data.frame(GeneClass = factor(rep(c("Path1", "Path2", "Path3"), c(10, 4, 6))))
rownames(annotationRow) = paste("Gene", seq(20), sep = "")

# Display row and color annotations
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
pheatmap(test, annotationCol = annotationCol)
p heatmap(test, annotationCol = annotationCol)

# Specify colors
ann_colors = list(Time = c("white", "firebrick"),
CellType = c("CT1" = "#1B9E77", "CT2" = "#D95F02"),
GeneClass = c(Path1 = "#7570B3", Path2 = "#E7298A", Path3 = "#66A61E"))
pheatmap(test, annotationCol = annotationCol, annotationColors = ann_colors)
```

### simulateCells

Simulate count data from the celda generative models.

#### Description

This function generates a SingleCellExperiment containing a simulated counts matrix in the "counts" assay slot, as well as various parameters used in the simulation which can be useful for running celda and are stored in metadata slot. The user must provide the desired model (one of celda_C, celda_G, celda_CG) as well as any desired tuning parameters for those model’s simulation functions as detailed below.

#### Usage

```r
simulateCells(
  model = c("celda_CG", "celda_C", "celda_G"),
  S = 5,
  CRange = c(50, 100),
  NRange = c(500, 1000),
  C = 100,
  G = 100,
  K = 5,
  L = 10,
  alpha = 1,
  beta = 1,
  gamma = 5,
  delta = 1,
  seed = 12345
)
```
simulateCells

Arguments

model Character. Options available in celda::availableModels. Can be one of "celda_CG", "celda_C", or "celda_G". Default "celda_CG".

S Integer. Number of samples to simulate. Default 5. Only used if model is one of "celda_CG" or "celda_C".

CRange Integer vector. A vector of length 2 that specifies the lower and upper bounds of the number of cells to be generated in each sample. Default c(50, 100). Only used if model is one of "celda_CG" or "celda_C".

NRange Integer vector. A vector of length 2 that specifies the lower and upper bounds of the number of counts generated for each cell. Default c(500, 1000).

C Integer. Number of cells to simulate. Default 100. Only used if model is "celda_G".

G Integer. The total number of features to be simulated. Default 100.

K Integer. Number of cell populations. Default 5. Only used if model is one of "celda_CG" or "celda_C".

L Integer. Number of feature modules. Default 10. Only used if model is one of "celda_CG" or "celda_G".

alpha Numeric. Concentration parameter for Theta. Adds a pseudocount to each cell population in each sample. Default 1. Only used if model is one of "celda_CG" or "celda_C".

beta Numeric. Concentration parameter for Phi. Adds a pseudocount to each feature module in each cell population. Default 1.

gamma Numeric. Concentration parameter for Eta. Adds a pseudocount to the number of features in each module. Default 5. Only used if model is one of "celda_CG" or "celda_G".

delta Numeric. Concentration parameter for Psi. Adds a pseudocount to each feature in each module. Default 1. Only used if model is one of "celda_CG" or "celda_G".

seed Integer. Passed to with_seed. For reproducibility, a default value of 12345 is used. If NULL, no calls to with_seed are made.

Value

A SingleCellExperiment object with simulated count matrix stored in the "counts" assay slot. Function parameter settings are stored in the metadata slot. For "celda_CG" and "celda_C" models, columns celda_sample_label and celda_cell_cluster in colData contain simulated sample labels and cell population clusters. For "celda_CG" and "celda_G" models, column celda_feature_module in rowData contains simulated gene modules.

Examples

sce <- simulateCells()
Simulate contaminated count matrix

Description

This function generates a list containing two count matrices – one for real expression, the other one for contamination, as well as other parameters used in the simulation which can be useful for running decontamination.

Usage

```r
simulateContamination(
  C = 300,
  G = 100,
  K = 3,
  NRange = c(500, 1000),
  beta = 0.1,
  delta = c(1, 10),
  numMarkers = 3,
  seed = 12345
)
```

Arguments

- `C` Integer. Number of cells to be simulated. Default 300.
- `G` Integer. Number of genes to be simulated. Default 100.
- `K` Integer. Number of cell populations to be simulated. Default 3.
- `NRange` Integer vector. A vector of length 2 that specifies the lower and upper bounds of the number of counts generated for each cell. Default `c(500, 1000)`.
- `beta` Numeric. Concentration parameter for Phi. Default 0.1.
- `delta` Numeric or Numeric vector. Concentration parameter for Theta. If input as a single numeric value, symmetric values for beta distribution are specified; if input as a vector of length 2, the two values will be the shape1 and shape2 parameters of the beta distribution respectively. Default `c(1, 5)`.
- `numMarkers` Integer. Number of markers for each cell population. Default 3.
- `seed` Integer. Passed to `with_seed`. For reproducibility, a default value of 12345 is used. If NULL, no calls to `with_seed` are made.

Value

A list containing the `nativeMatrix` (real expression), `observedMatrix` (real expression + contamination), as well as other parameters used in the simulation.

Author(s)

Shiyi Yang, Yuan Yin, Joshua Campbell
Examples

contaminationSim <- simulateContamination(K = 3, delta = c(1, 10))

splitModule

Split celda feature module

Description

Manually select a celda feature module to split into 2 or more modules. Useful for splitting up
modules that show divergent expression of features in multiple cell clusters.

Usage

splitModule(
  x,
  module,
  useAssay = "counts",
  altExpName = "featureSubset",
  n = 2,
  seed = 12345
)

## S4 method for signature 'SingleCellExperiment'
splitModule(
  x,
  module,
  useAssay = "counts",
  altExpName = "featureSubset",
  n = 2,
  seed = 12345
)

Arguments

x
A SingleCellExperiment object with the matrix located in the assay slot under
useAssay. Rows represent features and columns represent cells.

module
Integer. The module to be split.

useAssay
A string specifying which assay slot to use for x. Default "counts".

altExpName
The name for the altExp slot to use. Default "featureSubset".

n
Integer. How many modules should module be split into. Default 2.

seed
Integer. Passed to with_seed. For reproducibility, a default value of 12345 is
used. If NULL, no calls to with_seed are made.

Value

A updated SingleCellExperiment object with new feature modules stored in column celda_feature_module
in rowData(x).
Examples

data(sceCeldaCG)
# Split module 5 into 2 new modules.
sce <- splitModule(sceCeldaCG, module = 5)

subsetCeldaList subset celda model from SCE object returned from celdaGridSearch

Description

Select a subset of models from a SingleCellExperiment object generated by celdaGridSearch that match the criteria in the argument params.

Usage

subsetCeldaList(x, params, altExpName = "featureSubset")

## S4 method for signature 'SingleCellExperiment'
subsetCeldaList(x, params, altExpName = "featureSubset")

## S4 method for signature 'celdaList'
subsetCeldaList(x, params)

Arguments

x Can be one of
- A SingleCellExperiment object returned from celdaGridSearch, recursiveSplitModule, or recursiveSplitCell. Must contain a list named "celda_grid_search" in metadata(x).
- celdaList object.

params List. List of parameters used to subset the matching celda models in list "celda_grid_search" in metadata(x).

altExpName The name for the altExp slot to use. Default "featureSubset".

Value

One of

- A new SingleCellExperiment object containing all models matching the provided criteria in params. If only one celda model result in the "celda_grid_search" slot in metadata(x) matches the given criteria, a new SingleCellExperiment object with the matching model stored in the metadata "celda_parameters" slot will be returned. Otherwise, a new SingleCellExperiment object with the subset models stored in the metadata "celda_grid_search" slot will be returned.
- A new celdaList object containing all models matching the provided criteria in params. If only one item in the celdaList matches the given criteria, the matching model will be returned directly instead of a celdaList object.
See Also
celdaGridSearch can run Celda with multiple parameters and chains in parallel. selectBestModel
can get the best model for each combination of parameters.

Examples
data(sceCeldaCGGridSearch)
sceK5L10 <- subsetCeldaList(sceCeldaCGGridSearch,  
params = list(K = 5, L = 10))
data(celdaCGGridSearchRes)
resK5L10 <- subsetCeldaList(celdaCGGridSearchRes,  
params = list(K = 5, L = 10))

topRank

Identify features with the highest influence on clustering.

Description
topRank() can quickly identify the top 'n' rows for each column of a matrix. For example, this can be useful for identifying the top 'n' features per cell.

Usage
topRank(matrix, n = 25, margin = 2, threshold = 0, decreasing = TRUE)

Arguments
matrix Numeric matrix.
n Integer. Maximum number of items above 'threshold' returned for each ranked row or column.
margin Integer. Dimension of 'matrix' to rank, with 1 for rows, 2 for columns. Default 2.
threshold Numeric. Only return ranked rows or columns in the matrix that are above this threshold. If NULL, then no threshold will be applied. Default 0.
decreasing Logical. Specifies if the rank should be decreasing. Default TRUE.

Value
List. The 'index' variable provides the top 'n' row (feature) indices contributing the most to each column (cell). The 'names' variable provides the rownames corresponding to these indexes.

Examples
data(sampleCells)
topRanksPerCell <- topRank(sampleCells, n = 5)
topFeatureNamesForCell <- topRanksPerCell$names[1]
Index

* datasets
  availableModels, 4
  celdaCGGridSearchRes, 6
  celdaCGMod, 6
  celdaCGSim, 7
  celdaMod, 8
  celdaCSim, 9
  celdaGMod, 9
  celdaGSim, 12
  contaminationSim, 36
  sampleCells, 99
  sceCeldaC, 100
  sceCeldaCG, 101
  sceCeldaCGGridSearch, 102
  sceCeldaG, 102
  celda, 5, 7, 8, 11, 13, 14, 17, 19, 21, 23, 24, 26, 27, 29–32, 34, 44, 47–49, 51, 54, 60, 61, 67, 70, 73, 76, 77, 81–83, 85, 89, 91, 93, 96–98, 100, 103–105, 113, 114
  appendCeldaList, 4
  assay, 11, 13, 17, 19, 21, 23, 24, 26, 27, 29, 31, 32, 34, 41, 44, 48, 49, 51, 54, 59, 61, 70, 73, 76, 85, 89, 91, 93, 96, 105, 113
  availableModels, 4, 9, 11

  bestLogLikelihood, 5
  bestLogLikelihood, celdaModel-method (bestLogLikelihood), 5
  bestLogLikelihood, SingleCellExperiment-method (bestLogLikelihood), 5

  celda, 5
  celda_C, 5, 7, 11, 13, 14, 17, 18, 21, 23, 24, 30, 33, 34, 44, 51, 59, 83, 91, 100
  celda_C, ANY-method (celda_C), 24
  celda_C, SingleCellExperiment-method (celda_C), 24

  celda_CG, 5, 7, 11, 13, 14, 17, 18, 21, 23, 27, 29, 33, 34, 44, 47, 48, 51, 59, 82, 83, 85, 90–92, 100, 101
  celda_CG, ANY-method (celda_CG), 27
  celda_CG, SingleCellExperiment-method (celda_CG), 27
  celda_G, 5, 11, 13, 14, 17, 21, 23, 27, 30, 31, 34, 44, 47, 48, 51, 59, 82, 87, 90, 91, 100, 102
  celda_G, ANY-method (celda_G), 31
  celda_G, SingleCellExperiment-method (celda_G), 31
  sceCeldaCGGridSearchRes, 6
  celdaCGMod, 6
  celdaCGSim, 7, 101, 102
  celdaClusters, 7
  celdaClusters, celdaModel-method (celdaClusters), 7
  celdaClusters, SingleCellExperiment-method (celdaClusters), 7
  celdaClusters<-, (celdaClusters), 7
  celdaClusters<-, SingleCellExperiment-method (celdaClusters), 7
  celdaMod, 8
  celdaCSim, 9, 100
  celdaGMod, 9
  celdaGridSearch, 9, 27, 30, 33, 95, 96, 102–104, 114, 115
  celdaGridSearch, matrix-method (celdaGridSearch), 9
  celdaGridSearch, SingleCellExperiment-method (celdaGridSearch), 9
  celdaGSim, 12, 102
  celdaHeatmap, 12
  celdaHeatmap, SingleCellExperiment-method (celdaHeatmap), 12
  celdaModel, 13
  celdaModel, SingleCellExperiment-method (celdaModel), 13

116
celdaModules, 14
celdaModules, SingleCellExperiment-method (celdaModules), 14
celdaModules<- (celdaModules), 14
celdaModules<-, SingleCellExperiment-method (celdaModules), 14
celdaPerplexity, 15
celdaPerplexity, celdaList-method, 15
celdaProbabilityMap, 16
celdaProbabilityMap, SingleCellExperiment-method (celdaProbabilityMap), 16
celdatosce, 18
celdatosce, celda_C-method (celdatosce), 18
celdatosce, celda_CG-method (celdatosce), 18
celdatosce, celda_G-method (celdatosce), 18
celdatosce, celdaList-method (celdatosce), 18
celdaTsne, 20
celdaTsne, SingleCellExperiment-method (celdaTsne), 20
celdaUmap, 22
celdaUmap, SingleCellExperiment-method (celdaUmap), 22
clusterProbability, 34
clusterProbability, SingleCellExperiment-method (clusterProbability), 34
colData, 19, 27, 30, 83, 111
compareCountMatrix, 19, 35
compareCountMatrix, ANY, celdaList-method (compareCountMatrix), 35
compareCountMatrix, ANY, celdaModel-method (compareCountMatrix), 35
contaminationSim, 36
countChecksum, 36
countChecksum, celdaList-method, 37
dbscan, 38, 39
decontX, 37, 62–66
decontX, ANY-method (decontX), 37
decontX, SingleCellExperiment-method (decontX), 37
decontXcounts, 41
decontXcounts, SingleCellExperiment-method (decontXcounts), 41
decontXcounts<-, SingleCellExperiment-method (decontXcounts), 41
dist, 107
distinctColors, 41
eigenMatMultiInt, 42
eigenMatMultiNumeric, 43
erichr, 49
facet_wrap, 71, 73, 76
factorizeMatrix, 43, 90
factorizeMatrix, ANY, celda_C-method (factorizeMatrix), 43
factorizeMatrix, ANY, celda_CG-method (factorizeMatrix), 43
factorizeMatrix, ANY, celda_G-method (factorizeMatrix), 43
factorizeMatrix, SingleCellExperiment, ANY-method (factorizeMatrix), 43
fastNormProp, 45
fastNormPropLog, 45
fastNormPropSqrt, 45
featureModuleLookup, 46
featureModuleLookup, SingleCellExperiment-method (featureModuleLookup), 46
featureModuleTable, 48
fit_dirichlet, 39
geneSetEnrich, 49
geneSetEnrich, matrix-method (geneSetEnrich), 49
 geneSetEnrich, SingleCellExperiment-method (geneSetEnrich), 49
gem_jitter, 77, 81
gpar, 56, 109
grep, 64, 65, 94, 97
grid.draw, 56
grid.text, 109
hclust, 91, 107, 109
Heatmap, 17, 18, 54–56
Heatmap-class, 18
HeatmapAnnotation, 55, 56
HeatmapList, 18, 56
listEnrichrDbs, 49
log, 55, 57
log10, 55, 57
log1p, 55, 57
log2, 55, 57
logLikelihood, 50
logLikelihood, matrix, celda_C-method
  (logLikelihood), 50
logLikelihood, matrix, celda_CG-method
  (logLikelihood), 50
logLikelihood, matrix, celda_G-method
  (logLikelihood), 50
logLikelihood, SingleCellExperiment, ANY-method
  (logLikelihood), 50
logLikelihoodHistory, 51
logLikelihoodHistory, celdaModel-method
  (logLikelihoodHistory), 51
logLikelihoodHistory, SingleCellExperiment-method
  (logLikelihoodHistory), 51

marrangeGrob, 56
Matrix, 38
matrix, 11, 19, 49, 54, 56, 85, 89, 95, 105
matrixNames, 52
matrixNames, celdaModel-method
  (matrixNames), 52
metadata, 11, 19, 27, 30, 33, 86, 90, 103, 105, 111, 114
modelGeneVar, 39
moduleHeatmap, 53
moduleHeatmap, SingleCellExperiment-method
  (moduleHeatmap), 53

nonzero, 56
normalizeCounts, 21, 23, 55, 57

params, 58
params, celdaModel-method (params), 58
perplexity, 59
perplexity, ANY, celda_C-method
  (perplexity), 59
perplexity, ANY, celda_CG-method
  (perplexity), 59
perplexity, ANY, celda_G-method
  (perplexity), 59
perplexity, SingleCellExperiment, ANY-method
  (perplexity), 59
plotCeldaViolin, 60
plotCeldaViolin, ANY-method
  (plotCeldaViolin), 60
plotCeldaViolin, SingleCellExperiment-method
  (plotCeldaViolin), 60
plotDecontXContamination, 62
plotDecontXMarkerExpression, 63
plotDecontXMarkerPercentage, 64
plotDimReduceCluster, 66
plotDimReduceCluster, SingleCellExperiment-method
  (plotDimReduceCluster), 66
plotDimReduceCluster, vector-method
  (plotDimReduceCluster), 66
plotDimReduceFeature, 68
plotDimReduceFeature, ANY-method
  (plotDimReduceFeature), 68
plotDimReduceFeature, SingleCellExperiment-method
  (plotDimReduceFeature), 68
plotDimReduceGrid, 71
plotDimReduceGrid, ANY-method
  (plotDimReduceGrid), 71
plotDimReduceGrid, SingleCellExperiment-method
  (plotDimReduceGrid), 71
plotDimReduceModule, 74
plotDimReduceModule, ANY-method
  (plotDimReduceModule), 74
plotDimReduceModule, SingleCellExperiment-method
  (plotDimReduceModule), 74
plotGridSearchPerplexity, 77
plotGridSearchPerplexity, celdaList-method
  (plotGridSearchPerplexity), 77
plotGridSearchPerplexity, SingleCellExperiment-method
  (plotGridSearchPerplexity), 77
plotHeatmap, 13, 78
plotRPC, 81
plotRPC, celdaList-method (plotRPC), 81
plotRPC, SingleCellExperiment-method
  (plotRPC), 81
recodeClusterY, 82
recodeClusterZ, 82
recursiveSplitCell, 83, 92
recursiveSplitCell, matrix-method
  (recursiveSplitCell), 83
recursiveSplitCell, SingleCellExperiment-method
  (recursiveSplitCell), 83
recursiveSplitModule, 86, 87, 92, 93
recursiveSplitModule, matrix-method
  (recursiveSplitModule), 87
recursiveSplitModule, SingleCellExperiment-method
  (recursiveSplitModule), 87
reducedDim, 21, 24
reorderCelda, 90
reorderCelda, matrix, celda_C-method
  (reorderCelda), 90
INDEX

reorderCelda, matrix, celda_CG-method
(reorderCelda), 90
reorderCelda, matrix, celda_G-method
(reorderCelda), 90
reorderCelda, SingleCellExperiment, ANY-method
(reorderCelda), 90
reportCeldaCG, 92
reportCeldaCGPlotResults
(reportCeldaCG), 92
reportCeldaCGRun (reportCeldaCG), 92
resamplePerplexity, 11, 86, 89, 95
resamplePerplexity, ANY-method
(resamplePerplexity), 95
resamplePerplexity, SingleCellExperiment-method
(resamplePerplexity), 95
resList, 96
resList, celdaList-method (resList), 96
resList, SingleCellExperiment-method
(resList), 96
retrieveFeatureIndex, 64, 65, 97
retrieveFeatureInfo, 98
rowAnnotation, 56
rowData, 19, 30, 33, 82, 111, 113
Rtsne, 20
runParams, 98
runParams, celdaList-method (runParams), 98
runParams, SingleCellExperiment-method
(runParams), 98
sampleCells, 99
sampleLabel, 99
sampleLabel, celdaModel-method
(sampleLabel), 99
sampleLabel, SingleCellExperiment-method
(sampleLabel), 99
sampleLabel<- (sampleLabel), 99
sampleLabel<-, SingleCellExperiment-method
(sampleLabel), 99
scale, 55
scale_colour_gradient, 76
scale_colour_gradient2, 70, 73
sceCeldaC, 100
sceCeldaCG, 101
sceCeldaCGGridSearch, 102
sceCeldaG, 102
selectBestModel, 11, 103, 115
selectBestModel, celdaList-method
(selectBestModel), 103
selectBestModel, SingleCellExperiment-method
(selectBestModel), 103
selectFeatures, 93, 100–102, 104
selectFeatures, matrix-method
(selectFeatures), 104
selectFeatures, SingleCellExperiment-method
(selectFeatures), 104
semiPheatmap, 105
simulateCells, 110
simulateContamination, 36, 112
SingleCellExperiment, 5, 7, 8, 11, 13, 14,
17–19, 21, 23, 24, 26, 27, 29–34,
38–41, 44, 47–49, 51, 54, 55, 59,
61–67, 70, 73, 76, 77, 81–83, 85, 86,
89–91, 93, 95–105, 110, 111, 113,
114
splitModule, 113
splitModule, SingleCellExperiment-method
(splitModule), 113
sprintf, 108
sqrt, 55, 57
subsetCeldaList, 11, 104, 114
subsetCeldaList, celdaList-method
(subsetCeldaList), 114
subsetCeldaList, SingleCellExperiment-method
(subsetCeldaList), 114
SummarizedExperiment, 97
topRank, 115
umap, 22–24, 38
unit, 56
with_seed, 11, 21, 23, 26, 30, 33, 39, 86, 89,
96, 111–113