Package ‘monocle’

March 5, 2024

Type Package
Title Clustering, differential expression, and trajectory analysis for single-cell RNA-Seq
Version 2.30.0
Date 2022-06-06
Author Cole Trapnell
Maintainer Cole Trapnell <coletrap@uw.edu>
Description Monocle performs differential expression and time-series analysis for single-cell expression experiments. It orders individual cells according to progress through a biological process, without knowing ahead of time which genes define progress through that process. Monocle also performs differential expression analysis, clustering, visualization, and other useful tasks on single cell expression data. It is designed to work with RNA-Seq and qPCR data, but could be used with other types as well.
License Artistic-2.0
Depends R (>= 2.10.0), methods, Matrix (>= 1.2-6), Biobase, ggplot2 (>= 1.0.0), VGAM (>= 1.0-6), DDRTree (>= 0.1.4),
Imports parallel, igraph (>= 1.0.1), BiocGenerics, HSMMSingleCell (>= 0.101.5), plyr, cluster, combinat, fastICA, grid, irlba (>= 2.0.0), matrixStats, Rtsne, MASS, reshape2, leidenbase (>= 0.1.9), limma, tibble, dplyr, qlcMatrix, pheatmap, stringr, proxy, slam, viridis, stats, biocViews, RANN(>= 2.5), Rcpp (>= 0.12.0)
LinkingTo Rcpp
Suggests destiny, Hmisc, knitr, Seurat, scater, testthat
VignetteBuilder knitr
Roxygen list(wrap = FALSE)
LazyData true
biocViews ImmunoOncology, Sequencing, RNASeq, GeneExpression, DifferentialExpression, Infrastructure, DataImport, DataRepresentation, Visualization, Clustering, MultipleComparison, QualityControl
RoxygenNote 7.1.2

git url https://git.bioconductor.org/packages/monocle

git_branch RELEASE_3_18

git_last_commit 92ad7db

git_last_commit_date 2023-10-24

Repository Bioconductor 3.18

Date/Publication 2024-03-04

R topics documented:

addCellType .......................................................... 4
BEAM ................................................................. 4
branchTest ........................................................... 6
buildBranchCellDataSet ............................................. 7
calABCs ............................................................... 8
calibrate_per_cell_total_proposal ................................. 9
callILRs ............................................................... 10
CellDataSet .......................................................... 11
CellDataSet-methods .................................................. 12
cellPairwiseDistances ............................................... 13
cellPairwiseDistances<-. ........................................... 14
CellType .............................................................. 15
CellTypeHierarchy .................................................... 15
clusterCells .......................................................... 16
clusterGenes .......................................................... 18
compareModels ....................................................... 19
detectBifurcationPoint .............................................. 19
detectGenes .......................................................... 21
differentialGeneTest .................................................. 22
diff_test_helper ....................................................... 23
dispersionTable ....................................................... 24
estimateDispersionsForCellDataSet ............................... 24
estimateSizeFactorsForMatrix ...................................... 25
estimate_t ............................................................. 26
exportCDS .............................................................. 27
extract_good_branched_ordering .................................... 27
fitModel ............................................................... 28
fit_model_helper ....................................................... 29
genSmoothCurveResiduals ........................................... 30
genSmoothCurves ..................................................... 30
get_classic_muscle_markers ....................................... 31
importCDS ............................................................. 32
load_HSMM ........................................................... 32
load_HSMM_markers .................................................. 33
load_lung .............................................................. 33
markerDiffTable ....................................................... 33
R topics documented:

mcesApply ................................................................. 34
minSpanningTree ......................................................... 35
minSpanningTree<- ....................................................... 36
newCellDataSet ......................................................... 37
newCellTypeHierarchy .................................................. 38
orderCells ................................................................. 40
order_p_node ............................................................. 41
plot_cell_clusters ....................................................... 42
plot_cell_trajectory .................................................... 43
plot_clusters .............................................................. 45
plot_coexpression_matrix .............................................. 46
plot_complex_cell_trajectory ........................................ 47
plot_genes_branched_heatmap ........................................ 48
plot_genes_branched_pseudotime ..................................... 50
plot_genes_in_pseudotime ............................................. 52
plot_genes_jitter ......................................................... 53
plot_genes_positive_cells ............................................. 54
plot_genes_violin ......................................................... 56
plot_multiple_branches_heatmap ...................................... 57
plot_multiple_branches_pseudotime .................................. 59
plot_ordering_genes ..................................................... 60
plot_pc_variance_explained .......................................... 60
plot_pseudotime_heatmap ............................................. 62
plot_rho_delta ............................................................ 63
plot_spanning_tree ...................................................... 64
pq_helper ................................................................. 65
reducedDimA .............................................................. 66
reducedDimA<- ........................................................... 66
reducedDimK .............................................................. 67
reducedDimK<- ........................................................... 68
reducedDimS .............................................................. 68
reducedDimS<- ........................................................... 69
reducedDimW .............................................................. 70
reducedDimW<- ........................................................... 70
reduceDimension ......................................................... 71
relative2abs .............................................................. 73
residualMatrix ............................................................ 75
responseMatrix .......................................................... 75
selectTopMarkers ......................................................... 76
setOrderingFilter ......................................................... 76
spike_df ................................................................. 77
vstExprs ................................................................. 77

Index 78
addCellType  
*Add a new cell type*

**Description**

adds a cell type to a pre-existing CellTypeHierarchy and produces a function that accepts expression data from a CellDataSet. When the function is called on a CellDataSet a boolean vector is returned that indicates whether each cell is or is not the cell type that was added by addCellType.

**Usage**

```
addCellType(cth, cell_type_name, classify_func, parent_cell_type_name = "root")
```

**Arguments**

- `cth`  
The CellTypeHierarchy object
- `cell_type_name`  
The name of the new cell type. Can’t already exist in cth
- `classify_func`  
A function that returns true when a cell is of the new type
- `parent_cell_type_name`  
If this cell type is a subtype of another, provide its name here

**BEAM**  
*Branched expression analysis modeling (BEAM).*

**Description**

Identify genes with branch-dependent expression. Branches in single-cell trajectories are generated by cell fate decisions in development and also arise when analyzing genetic, chemical, or environmental perturbations. Branch expression analysis modeling is a statistical approach for finding genes that are regulated in a manner that depends on the branch. Consider a progenitor cell that generates two distinct cell types. A single-cell trajectory that includes progenitor cells and both differentiated cell types will capture the "decision" as a branch point, with progenitors upstream of the branch and the differentiated cells positioned along distinct branches. These branches will be characterized by distinct gene expression programs. BEAM aims to find all genes that differ between the branches. Such "branch-dependent" genes can help identify the mechanism by which the fate decision is made. BEAM() Takes a CellDataSet and either a specified branch point, or a pair of trajectory outcomes (as States). If a branch point is provided, the function returns a dataframe of test results for dependence on that branch. If a pair of outcomes is provided, it returns test results for the branch that unifies those outcomes into a common path to the trajectory’s root state. BEAM() compares two models with a likelihood ratio test for branch-dependent expression. The full model is the product of smooth Pseudotime and the Branch a cell is assigned to. The reduced model just includes Pseudotime. You can modify these to include arbitrary additional effects in the full or both models.
BEAM

Usage

BEAM(
  cds,
  fullModelFormulaStr = "~sm.ns(Pseudotime, df = 3)*Branch",
  reducedModelFormulaStr = "~sm.ns(Pseudotime, df = 3)",
  branch_states = NULL,
  branch_point = 1,
  relative_expr = TRUE,
  branch_labels = NULL,
  verbose = FALSE,
  cores = 1,
  ...
)

Arguments

cds a CellDataSet object upon which to perform this operation
fullModelFormulaStr a formula string specifying the full model in differential expression tests (i.e.
likelihood ratio tests) for each gene/feature.
reducedModelFormulaStr a formula string specifying the reduced model in differential expression tests
(i.e. likelihood ratio tests) for each gene/feature.
branch_states ids for the immediate branch branch which obtained from branch construction
based on MST
branch_point The ID of the branch point to analyze. Can only be used when reduceDimension
is called with method = "DDRTree".
relative_expr a logic flag to determine whether or not the relative gene expression should be
used
branch_labels the name for each branch, for example, "AT1" or "AT2"
verbose Whether to generate verbose output
cores the number of cores to be used while testing each gene for differential expression
... additional arguments to be passed to differentialGeneTest

Value

da data frame containing the p values and q-values from the BEAM test, with one row per gene.
Test for branch-dependent expression

Description

Testing for branch-dependent expression with `BEAM()` first involves constructing a CellDataSet that assigns each cell to a branch, and then performing a likelihood ratio test to see if the branch assignments significantly improves the fit over a null model that does not split the cells. `branchTest()` implements these two steps.

Usage

```r
branchTest(
  cds, 
  fullModelFormulaStr = "~sm.ns(Pseudotime, df = 3)*Branch", 
  reducedModelFormulaStr = "~sm.ns(Pseudotime, df = 3)", 
  branch_states = NULL, 
  branch_point = 1, 
  relative_expr = TRUE, 
  cores = 1, 
  branch_labels = NULL, 
  verbose = FALSE, 
  ...
)
```

Arguments

cds a CellDataSet object upon which to perform this operation

fullModelFormulaStr a formula string specifying the full model in differential expression tests (i.e. likelihood ratio tests) for each gene/feature.

reducedModelFormulaStr a formula string specifying the reduced model in differential expression tests (i.e. likelihood ratio tests) for each gene/feature.

branch_states states corresponding to two branches

branch_point The ID of the branch point to analyze. Can only be used when reduceDimension is called with method = "DDRTree".

relative_expr a logic flag to determine whether or not the relative gene expression should be used

cores the number of cores to be used while testing each gene for differential expression

branch_labels the name for each branch, for example, AT1 or AT2

verbose Whether to show VGAM errors and warnings. Only valid for cores = 1.

... Additional arguments passed to differentialGeneTest
buildBranchCellDataSet

*Build a CellDataSet that splits cells among two branches*

**Value**

a data frame containing the p values and q-values from the likelihood ratio tests on the parallel arrays of models.

**Description**

Analyzing branches with `BEAM()` requires fitting two models to the expression data for each gene. The full model assigns each cell to one of the two outcomes of the branch, and the reduced model excludes this assignment. `buildBranchBranchCellDataSet()` takes a CellDataSet object and returns a version where the cells are assigned to one of two branches. The branch for each cell is encoded in a new column, “Branch”, in the pData table in the returned CellDataSet.

**Usage**

```r
buildBranchCellDataSet(
  cds,
  progenitor_method = c("sequential_split", "duplicate"),
  branch_states = NULL,
  branch_point = 1,
  branch_labels = NULL,
  stretch = TRUE
)
```

**Arguments**

- `cds`: CellDataSet for the experiment
- `progenitor_method`: The method to use for dealing with the cells prior to the branch
- `branch_states`: The states for two branching branches
- `branch_point`: The ID of the branch point to analyze. Can only be used when `reduceDimension()` is called with `reduction_method = "DDRTree"`. 
- `branch_labels`: The names for each branching branch
- `stretch`: A logical flag to determine whether or not the pseudotime trajectory for each branch should be stretched to the same range or not

**Value**

a CellDataSet with the duplicated cells and stretched branches
**calABCs**

*Compute the area between curves (ABC) for branch-dependent genes*

**Description**

This function is used to calculate the ABC score based on the nature spline curves fitted for each branch. ABC score is used to quantify the total magnitude of divergence between two branches. By default, the ABC score is the area between two fitted spline curves. The ABC score can be used to rank gene divergence. When coupled with p-val calculated from the branchTest, it can be used to identify potential major regulators for branch bifurcation.

**Usage**

```r
calABCs(
  cds,
  trend_formula = "~sm.ns(Pseudotime, df = 3)*Branch",
  branch_point = 1,
  trajectory_states = NULL,
  relative_expr = TRUE,
  stretch = TRUE,
  cores = 1,
  verbose = F,
  min_expr = 0.5,
  integer_expression = FALSE,
  num = 5000,
  branch_labels = NULL,
  ...
)
```

**Arguments**

- **cds**: a CellDataSet object upon which to perform this operation
- **trend_formula**: a formula string specifying the full model in differential expression tests (i.e. likelihood ratio tests) for each gene/feature.
- **branch_point**: the point where two branches diverge
- **trajectory_states**: States corresponding to two branches
- **relative_expr**: a logic flag to determine whether or not the relative gene expression should be used
- **stretch**: a logic flag to determine whether or not each branch should be stretched
- **cores**: the number of cores to be used while testing each gene for differential expression
- **verbose**: a logic flag to determine whether or not we should output detailed running information
- **min_expr**: the lower limit for the expressed gene
calibrate_per_cell_total_proposal

integer_expression

the logic flag to determine whether or not the integer numbers are used for calculating the ABCs. Default is False.

num

number of points on the fitted branch trajectories used for calculating the ABCs. Default is 5000.

branch_labels

the name for each branch, for example, AT1 or AT2

... Additional arguments passed to buildBranchCellDataSet

Value

a data frame containing the ABCs (Area under curves) score as the first column and other meta information from fData

calibrate_per_cell_total_proposal

Calibrate_per_cell_total_proposal

Description

Calibrate_per_cell_total_proposal

Usage

calibrate_per_cell_total_proposal(
    relative_exprs_matrix, t_estimate, expected_capture_rate,
    method = c("num_genes", "tpm_fraction")
)

Arguments

relative_exprs_matrix

The matrix of relative TPM expression values

t_estimate

the TPM value that corresponds to 1 cDNA copy per cell

expected_capture_rate

The fraction of mRNAs captured as cDNAs

method

the formula to estimate the total mRNAs (num_genes corresponds to the second formula while tpm_fraction corresponds to the first formula, see the announcement on Trapnell lab website for the Census paper)
Calculate the Instantaneous Log Ratio between two branches

Description

This function is used to calculate the Instant Log Ratio between two branches which can be used to prepare the heatmap demonstrating the branch gene expression divergence hierarchy. If "stretch" is specified, each branch will be firstly stretched into maturation level from 0-100. Since the results when we use "stretching" are always better and IRLs for non-stretched spline curves are often mismatched, we may only turn down "non-stretch" functionality in future versions. Then, we fit two separate nature spline curves for each individual lineages. The log-ratios of the value on each spline curve corresponding to each branch are calculated, which can be used as a measure for the magnitude of divergence between two branching branches.

Usage

calILRs(
  cds,
  trend_formula = "~sm.ns(Pseudotime, df = 3)*Branch",
  branch_point = 1,
  trajectory_states = NULL,
  relative_expr = TRUE,
  stretch = TRUE,
  cores = 1,
  ILRs_limit = 3,
  label_by_short_name = TRUE,
  useVST = FALSE,
  round_exprs = FALSE,
  output_type = "all",
  branch_labels = NULL,
  file = NULL,
  return_all = F,
  verbose = FALSE,
  ...
)

Arguments

cds          CellDataSet for the experiment
trend_formula trend_formula a formula string specifying the full model in differential expression tests (i.e. likelihood ratio tests) for each gene/feature.
branch_point  the point where two branches diverge
trajectory_states states corresponding to two branches
relative_expr  A logic flag to determine whether or not the relative expressed should be used when we fitting the spline curves
**stretch**

a logic flag to determine whether or not each branch should be stretched

**cores**

Number of cores when fitting the spline curves

**ILRs_limit**

the minimum Instant Log Ratio used to make the heatmap plot

**label_by_short_name**

label the rows of the returned matrix by gene_short_name (TRUE) or feature id (FALSE)

**useVST**

A logic flag to determine whether or not the Variance Stabilization Transformation should be used to stabilize the gene expression. When VST is used, the difference between two branches are used instead of the log-ratio.

**round_exprs**

A logic flag to determine whether or not the expression value should be rounded into integer

**output_type**

A character either of "all" or "after_bifurcation". If "after_bifurcation" is used, only the time points after the bifurcation point will be selected

**branch_labels**

the name for each branch, for example, AT1 or AT2

**file**

the name for storing the data. Since the calculation of the Instant Log Ratio is very time consuming, so by default the result will be stored

**return_all**

A logic flag to determine whether or not all the results from the analysis should be returned, this includes a dataframe for the log fold change, normalized log fold change, raw divergence, normalized divergence, fitting curves for each branch

**verbose**

Whether or not detailed running information should be returned

... Additional arguments passed to `buildBranchCellDataSet`

**Value**

a ggplot2 plot object

---

**CellDataSet**

**The CellDataSet class**

**Description**

The main class used by Monocle to hold single cell expression data. `CellDataSet` extends the basic Bioconductor `ExpressionSet` class.

**Details**

This class is initialized from a matrix of expression values. Methods that operate on `CellDataSet` objects constitute the basic Monocle workflow.
Fields

- **reducedDimS**: Matrix of class numeric, containing the source values computed by Independent Components Analysis.
- **reducedDimW**: Matrix of class numeric, containing the whitened expression values computed during Independent Components Analysis.
- **reducedDimA**: Matrix of class numeric, containing the weight values computed by Independent Components Analysis.
- **reducedDimK**: A Matrix of class numeric, containing the pre-whitening matrix computed by Independent Components Analysis.
- **minSpanningTree**: An Object of class igraph, containing the minimum spanning tree used by Monocle to order cells according to progress through a biological process.
- **cellPairwiseDistances**: A Matrix of class numeric, containing the pairwise distances between cells in the reduced dimension space.
- **expressionFamily**: An Object of class vglmff, specifying the VGAM family function used for expression responses.
- **lowerDetectionLimit**: A numeric value specifying the minimum expression level considered to be true expression.
- **dispFitInfo**: An environment containing lists, one for each set of estimated dispersion values. See `estimateDispersions`.
- **dim_reduce_type**: A string encoding how this CellDataSet has been reduced in dimensionality.
- **auxOrderingData**: An environment of auxiliary data structures used by various steps in Monocle. Not to be accessed by users directly.

---

**CellDataSet-methods**  
Methods for the *CellDataSet* class

**Description**

Methods for the CellDataSet class

**Usage**

```r
## S4 method for signature 'CellDataSet'
sizeFactors(object)

## S4 replacement method for signature 'CellDataSet,numeric'
sizeFactors(object) <- value

## S4 method for signature 'CellDataSet'
estimateSizeFactors(object, locfunc = median, ...)

## S4 method for signature 'CellDataSet'
estimateDispersions(
```
cellPairwiseDistances

function cellPairwiseDistances(cds)

Arguments

- **cds**: expression data matrix for an experiment

Value

A square, symmetric matrix containing the distances between each cell in the reduced-dimensionality space.
cellPairwiseDistances<-  

Examples

## Not run:
D <- cellPairwiseDistances(HSMM)

## End(Not run)

cellPairwiseDistances<-  

Sets the matrix containing distances between each pair of cells used by Monocle during cell ordering. Not intended to be called directly.

Description

Sets the matrix containing distances between each pair of cells used by Monocle during cell ordering. Not intended to be called directly.

Usage

cellPairwiseDistances(cds) <- value

Arguments

cds A CellDataSet object.
value a square, symmetric matrix containing pairwise distances between cells.

Value

An updated CellDataSet object

Examples

## Not run:
cds <- cellPairwiseDistances(D)

## End(Not run)
CellType

The CellType class

Description

Classifies cells using a criterion function.

Details

Classifies cells via a user-defined gating function. The gating function accepts as input the entire matrix of expression data from a CellDataSet, and return TRUE or FALSE for each cell in it, depending on whether each meets the criteria in the gating function.

Slots

classify_func: A function that accepts a matrix of expression values as input, and returns a logical vector (of length equal to the number of columns in the matrix) as output.

CellTypeHierarchy

The CellTypeHierarchy class

Description

Classifies cells according to a hierarchy of types.

Details

Classifies cells according to a hierarchy of types via user-defined gating functions.

Slots

classificationTree: Object of class "igraph"
clusterCells

Cluster cells into a specified number of groups based on.

Description

Unsupervised clustering of cells is a common step in many single-cell expression workflows. In an experiment containing a mixture of cell types, each cluster might correspond to a different cell type. This method takes a CellDataSet as input along with a requested number of clusters, clusters them with an unsupervised algorithm (by default, density peak clustering), and then returns the CellDataSet with the cluster assignments stored in the pData table. When number of clusters is set to NULL (num_clusters = NULL), the decision plot as introduced in the reference will be plotted and the users are required to check the decision plot to select the rho and delta to determine the number of clusters to cluster. When the dataset is big, for example > 50 k, we recommend the user to use the Leiden or Louvain clustering algorithm which is inspired from phenograph paper. Note Louvain doesn’t support the num_cluster argument but the k (number of k-nearest neighbors) is relevant to the final clustering number. The implementation of Louvain clustering is based on the Rphenograph package but updated based on our requirement (for example, changed the jaccard_coef function as well as adding louvain_iter argument, etc.) The density peak clustering method was removed because CRAN removed the densityClust package. Consequently, the parameters skip_rho_sigma, inspect_rho_sigma, rho_threshold, delta_threshold, peaks, and gaussian no longer have an effect.

Usage

clusterCells(
  cds,
  skip_rho_sigma = F,
  num_clusters = NULL,
  inspect_rho_sigma = F,
  rho_threshold = NULL,
  delta_threshold = NULL,
  peaks = NULL,
  gaussian = T,
  cell_type_hierarchy = NULL,
  frequency_thresh = NULL,
  enrichment_thresh = NULL,
  clustering_genes = NULL,
  k = 50,
  louvain_iter = 1,
  weight = FALSE,
  method = c("leiden", "louvain", "DDRTree"),
  verbose = F,
  resolution_parameter = 0.1,
  ...
)

Arguments

cds the CellDataSet upon which to perform this operation
**clusterCells**

- **skip_rho_sigma**: A logic flag to determine whether or not you want to skip the calculation of rho / sigma.
- **num_clusters**: Number of clusters. The algorithm uses 0.5 of the rho as the threshold of rho and the delta corresponding to the number_clusters sample with the highest delta as the density peaks and for assigning clusters.
- **inspect_rho_sigma**: A logical flag to determine whether or not you want to interactively select the rho and sigma for assigning clusters.
- **rho_threshold**: The threshold of local density (rho) used to select the density peaks.
- **delta_threshold**: The threshold of local distance (delta) used to select the density peaks.
- **peaks**: A numeric vector indicating the index of density peaks used for clustering. This vector should be retrieved from the decision plot with caution. No checking involved. Will automatically calculated based on the top num_cluster product of rho and sigma.
- **gaussian**: A logic flag passed to densityClust function in densityClust package to determine whether or not Gaussian kernel will be used for calculating the local density.
- **cell_type_hierarchy**: A data structure used for organizing functions that can be used for organizing cells.
- **frequency_thresh**: When a CellTypeHierarchy is provided, cluster cells will impute cell types in clusters that are composed of at least this much of exactly one cell type.
- **enrichment_thresh**: Fraction to be multiplied by each cell type percentage. Only used if frequency_thresh is NULL, both cannot be NULL.
- **clustering_genes**: A vector of feature ids (from the CellDataSet’s featureData) used for ordering cells.
- **k**: Number of kNN used in creating the k nearest neighbor graph for Leiden and Louvain clustering. The number of kNN is related to the resolution of the clustering result, bigger number of kNN gives low resolution and vice versa. Default to be 50.
- **louvain_iter**: Number of iterations used for Leiden and Louvain clustering. The clustering result gives the largest modularity score will be used as the final clustering result. Default to be 1.
- **weight**: A logic argument to determine whether or not we will use Jaccard coefficient for two nearest neighbors (based on the overlapping of their kNN) as the weight used for Louvain clustering. Default to be FALSE.
- **method**: Method for clustering cells. Three methods are available, including leiden, louvian and DDRTree. By default, we use the leiden algorithm for clustering.
- **verbose**: Verbose. A logic flag to determine whether or not we should print the running details.
- **resolution_parameter**: A real value that controls the resolution of the leiden clustering. Default is .1.
- **...**: Additional arguments passed to densityClust.
**Value**

an updated CellDataSet object, in which phenoData contains values for Cluster for each cell

**References**


---

**clusterGenes**

Clusters genes by pseudotime trend.

**Description**

This function takes a matrix of expression values and performs k-means clustering on the genes.

**Usage**

```r
clusterGenes(
  expr_matrix,
  k,
  method = function(x) { as.dist((1 - cor(Matrix::t(x)))/2),
...    }
)
```

**Arguments**

- `expr_matrix` A matrix of expression values to cluster together. Rows are genes, columns are cells.
- `k` How many clusters to create
- `method` The distance function to use during clustering
- `...` Extra parameters to pass to pam() during clustering

**Value**

a pam cluster object
compareModels

Examples

```r
## Not run:
full_model_fits <- fitModel(HSMM[sample(nrow(fData(HSMM_filtered)), 100),],
  modelFormulaStr="sm.ns(Pseudotime)")
expression_curve_matrix <- responseMatrix(full_model_fits)
clusters <- clusterGenes(expression_curve_matrix, k=4)
plot_clusters(HSMM_filtered[ordering_genes,,], clusters)

## End(Not run)
```

compareModels  

Compare model fits

Description

Performs likelihood ratio tests on nested vector generalized additive models

Usage

```r
compareModels(full_models, reduced_models)
```

Arguments

- `full_models` a list of models, e.g. as returned by fitModels(), forming the numerators of the L.R.Ts.
- `reduced_models` a list of models, e.g. as returned by fitModels(), forming the denominators of the L.R.Ts.

Value

A data frame containing the p values and q-values from the likelihood ratio tests on the parallel arrays of models.

detectBifurcationPoint

Calculate divergence times for branch-dependent genes

Description

Branch-dependent genes may diverge at different points in pseudotime. detectBifurcationPoint() calculates these times. Although the branch times will be shaped by and distributed around the branch point in the trajectory, upstream regulators tend to branch earlier in pseudotime than their targets.
detectBifurcationPoint

Usage

detectBifurcationPoint(
  str_log_df = NULL,
  ILRs_threshold = 0.1,
  detect_all = T,
  cds = cds,
  Branch = "Branch",
  branch_point = NULL,
  branch_states = c(2, 3),
  stretch = T,
  cores = 1,
  trend_formula = "~sm.ns(Pseudotime, df = 3)",
  ILRs_limit = 3,
  relative_expr = TRUE,
  label_by_short_name = TRUE,
  useVST = FALSE,
  round_exprs = FALSE,
  output_type = "all",
  return_cross_point = T,
  file = "bifurcation_heatmap",
  verbose = FALSE,
  ...
)

Arguments

str_log_df the ILRs dataframe calculated from calILRs function. If this data.frame is provided, all the following parameters are ignored. Note that we need to only use the ILRs after the bifurcation point if we duplicated the progenitor cell state.

ILRs_threshold the ILR value used to determine the earliest divergence time point

detect_all a logic flag to determine whether or not genes without ILRs pass the threshold will still report a bifurcation point

cds CellDataSet for the experiment

Branch The column in pData used for calculating the ILRs (If not equal to "Branch", a warning will report)

branch_point The ID of the branch point to analyze. Can only be used when reduceDimension is called with method = "DDRTree".

branch_states The states for two branching branches

stretch a logic flag to determine whether or not each branch should be stretched

cores Number of cores when fitting the spline curves

trend_formula the model formula to be used for fitting the expression trend over pseudotime

ILRs_limit the minimum Instant Log Ratio used to make the heatmap plot

relative_expr A logic flag to determine whether or not the relative expressed should be used when we fitting the spline curves
detectGenes

label_by_short_name
label the rows of the returned matrix by gene_short_name (TRUE) or feature id (FALSE)

useVST
A logic flag to determine whether or not the Variance Stablization Transformation should be used to stablize the gene expression. When VST is used, the difference between two branches are used instead of the log-ratio.

round_exprs
A logic flag to determine whether or not the expression value should be rounded into integer

output_type
A character either of "all" or "after_bifurcation". If "after_bifurcation" is used, only the time points after the bifurcation point will be selected. Note that, if Branch is set to "Branch", we will only use "after_bifurcation" since we duplicated the progenitor cells and the bifurcation should only happen after the largest mature level from the progenitor cells

return_cross_point
A logic flag to determine whether or not only return the cross point

file
the name for storing the data. Since the calculation of the Instant Log Ratio is very time consuming, so by default the result will be stored

verbose
Whether to report verbose output

... Additional arguments passed to calILRs

Value
a vector containing the time for the bifurcation point with gene names for each value

detectGenes
detects genes above minimum threshold.

Description
Sets the global expression detection threshold to be used with this CellDataSet. Counts how many cells each feature in a CellDataSet object that are detectably expressed above a minimum threshold. Also counts the number of genes above this threshold are detectable in each cell.

Usage
detectGenes(cds, min_expr = NULL)

Arguments
cds the CellDataSet upon which to perform this operation
min_expr the expression threshold

Value
an updated CellDataSet object
differentialGeneTest

Examples

## Not run:
HSMM <- detectGenes(HSMM, min_expr=0.1)

## End(Not run)

differentialGeneTest Test genes for differential expression

Description
Tests each gene for differential expression as a function of pseudotime or according to other co-
variates as specified. differentialGeneTest is Monocle’s main differential analysis routine. It
accepts a CellDataSet and two model formulae as input, which specify generalized lineage models
as implemented by the VGAM package.

Usage
differentialGeneTest(
cds,
  fullModelFormulaStr = "~sm.ns(Pseudotime, df=3)",
  reducedModelFormulaStr = "~1",
  relative_expr = TRUE,
  cores = 1,
  verbose = FALSE
)

Arguments
cds a CellDataSet object upon which to perform this operation
fullModelFormulaStr a formula string specifying the full model in differential expression tests (i.e.
  likelihood ratio tests) for each gene/feature.
reducedModelFormulaStr a formula string specifying the reduced model in differential expression tests
  (i.e. likelihood ratio tests) for each gene/feature.
relative_expr Whether to transform expression into relative values.
cores the number of cores to be used while testing each gene for differential expres-
  sion.
verbose Whether to show VGAM errors and warnings. Only valid for cores = 1.

Value
a data frame containing the p values and q-values from the likelihood ratio tests on the parallel
arrays of models.
**diff_test_helper**

*Helper function for parallel differential expression testing*

**Description**

test

**Usage**

```r
diff_test_helper(
  x,
  fullModelFormulaStr,
  reducedModelFormulaStr,
  expressionFamily,
  relative_expr,
  weights,
  disp_func = NULL,
  verbose = FALSE
)
```

**Arguments**

- `x`: test
- `fullModelFormulaStr`: a formula string specifying the full model in differential expression tests (i.e. likelihood ratio tests) for each gene/feature.
- `reducedModelFormulaStr`: a formula string specifying the reduced model in differential expression tests (i.e. likelihood ratio tests) for each gene/feature.
- `expressionFamily`: specifies the VGAM family function used for expression responses
- `relative_expr`: Whether to transform expression into relative values
- `weights`: test
- `disp_func`: test
- `verbose`: Whether to show VGAM errors and warnings. Only valid for cores = 1.

**See Also**

- `vglm`
dispersionTable

Retrieve a table of values specifying the mean-variance relationship

Description

Calling estimateDispersions computes a smooth function describing how variance in each gene's expression across cells varies according to the mean. This function only works for CellDataSet objects containing count-based expression data, either transcripts or reads.

Usage

dispersionTable(cds)

Arguments

cds The CellDataSet from which to extract a dispersion table.

Value

A data frame containing the empirical mean expression, empirical dispersion, and the value estimated by the dispersion model.

estimateDispersionsForCellDataSet

Helper function to estimate dispersions

Description

Helper function to estimate dispersions

Usage

estimateDispersionsForCellDataSet(
  cds,
  modelFormulaStr,
  relative_expr,
  min_cells_detected,
  removeOutliers,
  verbose = FALSE
)
estimateSizeFactorsForMatrix

Function to calculate the size factor for the single-cell RNA-seq data

@importFrom stats median

Description

Function to calculate the size factor for the single-cell RNA-seq data

@importFrom stats median

Usage

estimateSizeFactorsForMatrix(
  counts,
  locfunc = median,
  round_exprs = TRUE,
  method = "mean-geometric-mean-total"
)

Arguments

counts The matrix for the gene expression data, either read counts or FPKM values or transcript counts
locfunc The location function used to find the representative value
round_exprs A logic flag to determine whether or not the expression value should be rounded
method A character to specify the size factor calculation approaches. It can be either "mean-geometric-mean-total" (default), "weighted-median", "median-geometric-mean", "median", "mode", "geometric-mean-total".
**estimate_t**  
*Find the most commonly occurring relative expression value in each cell*

**Description**

Converting relative expression values to mRNA copies per cell requires knowing the most commonly occurring relative expression value in each cell. This value typically corresponds to an RPC value of 1. This function finds the most commonly occurring (log-transformed) relative expression value for each column in the provided expression matrix.

**Usage**

```r
estimate_t(relative_expr_matrix, relative_expr_thresh = 0.1)
```

**Arguments**

- `relative_expr_matrix`  
  a matrix of relative expression values for values with each row and column representing genes/isoforms and cells, respectively. Row and column names should be included. Expression values should not be log-transformed.

- `relative_expr_thresh`  
  Relative expression values below this threshold are considered zero.

**Details**

This function estimates the most abundant relative expression value ($t^*$) using a gaussian kernel density function. It can also optionally output the $t^*$ based on a two gaussian mixture model based on the `smsn.mixture` from `mixsmsn` package.

**Value**

- an vector of most abundant relative_expr value corresponding to the RPC 1.

**Examples**

```r
## Not run:
HSMM_fpkm_matrix <- exprs(HSMM)
t_estimate = estimate_t(HSMM_fpkm_matrix)
## End(Not run)
```
exportCDS

Export a monocle CellDataSet object to the Seurat single cell analysis toolkit.

Description

This function takes a monocle CellDataSet and converts it into a Seurat object.

Usage

exportCDS(monocle_cds, export_to = c("Seurat"), export_all = FALSE)

Arguments

- `monocle_cds`: the Monocle CellDataSet you would like to export into a Seurat object.
- `export_to`: the object type you would like to export to. Seurat is supported.
- `export_all`: Whether or not to export all the slots in Monocle and keep in another object type. Default is FALSE (or only keep minimal dataset). If export_all is setted to be true, the original monocle cds will be kept in the other cds object too. This argument is only applicable when export_to is Seurat.

Value

a new object in the format of Seurat, as described in the export_to argument.

Examples

```r
## Not run:
lung <- load_lung()
seurat_lung <- exportCDS(lung)
seurat_lung_all <- exportCDS(lung, export_all = T)
## End(Not run)
```

extract_good_branched_ordering

Extract a linear ordering of cells from a PQ tree

Description

Extract a linear ordering of cells from a PQ tree
Usage

extract_good_branched_ordering(
    orig_pq_tree,
    curr_node,
    dist_matrix,
    num_branches,
    reverse_main_path = FALSE
)

Arguments

orig_pq_tree    The PQ object to use for ordering
curr_node      The node in the PQ tree to use as the start of ordering
dist_matrix    A symmetric matrix containing pairwise distances between cells
num_branches   The number of outcomes allowed in the trajectory.
reverse_main_path
                  Whether to reverse the direction of the trajectory

fitModel        Fits a model for each gene in a CellDataSet object.

Description

This function fits a vector generalized additive model (VGAM) from the VGAM package for each
gene in a CellDataSet. By default, expression levels are modeled as smooth functions of the Pseudo-
time value of each cell. That is, expression is a function of progress through the biological process.
More complicated formulae can be provided to account for additional covariates (e.g. day collected,
genotype of cells, media conditions, etc).

Usage

fitModel(
    cds,
    modelFormulaStr = "~sm.ns(Pseudotime, df=3)",
    relative_expr = TRUE,
    cores = 1
)

Arguments

cds            the CellDataSet upon which to perform this operation
modelFormulaStr
                  a formula string specifying the model to fit for the genes.
relative_expr  Whether to fit a model to relative or absolute expression. Only meaningful for
counter-based expression data. If TRUE, counts are normalized by Size_Factor
before fitting.
cores          the number of processor cores to be used during fitting.
Details

This function fits a vector generalized additive model (VGAM) from the VGAM package for each gene in a CellDataSet. By default, expression levels are modeled as smooth functions of the Pseudo-time value of each cell. That is, expression is a function of progress through the biological process. More complicated formulae can be provided to account for additional covariates (e.g. day collected, genotype of cells, media conditions, etc).

Value

a list of VGAM model objects

Usage

fit_model_helper(
  x,
  modelFormulaStr,
  expressionFamily,
  relative_expr,
  disp_func = NULL,
  verbose = FALSE,
  ...
)

Arguments

x  test
modelFormulaStr  a formula string specifying the model to fit for the genes.
expressionFamily  specifies the VGAM family function used for expression responses
relative_expr  Whether to transform expression into relative values
disp_func  test
verbose  Whether to show VGAM errors and warnings. Only valid for cores = 1.
...  test
genSmoothCurveResiduals

*Fit smooth spline curves and return the residuals matrix*

**Description**

This function will fit smooth spline curves for the gene expression dynamics along pseudotime in a gene-wise manner and return the corresponding residuals matrix. This function is built on other functions (fit_models and residualsMatrix).

**Usage**

```r
genSmoothCurveResiduals(
  cds,
  trend_formula = "~sm.ns(Pseudotime, df = 3)",
  relative_expr = T,
  residual_type = "response",
  cores = 1
)
```

**Arguments**

- `cds` a CellDataSet object upon which to perform this operation
- `trend_formula` a formula string specifying the model formula used in fitting the spline curve for each gene/feature.
- `relative_expr` a logic flag to determine whether or not the relative gene expression should be used
- `residual_type` the response desired, as accepted by VGAM’s predict function
- `cores` the number of cores to be used while testing each gene for differential expression

**Value**

a data frame containing the data for the fitted spline curves.

---

genSmoothCurves

*Fit smooth spline curves and return the response matrix*

**Description**

This function will fit smooth spline curves for the gene expression dynamics along pseudotime in a gene-wise manner and return the corresponding response matrix. This function is built on other functions (fit_models and responseMatrix) and used in calILRs and calABCs functions.
Usage

genSmoothCurves(
  cds,
  new_data,
  trend_formula = "~sm.ns(Pseudotime, df = 3)",
  relative_expr = T,
  response_type = "response",
  cores = 1
)

Arguments

cds a CellDataSet object upon which to perform this operation
new_data a data.frame object including columns (for example, Pseudotime) with names specified in the model formula. The values in the data.frame should be consist with the corresponding values from cds object.
trend_formula a formula string specifying the model formula used in fitting the spline curve for each gene/feature.
relative_expr a logic flag to determine whether or not the relative gene expression should be used
response_type the response desired, as accepted by VGAM’s predict function
cores the number of cores to be used while testing each gene for differential expression

Value

a data frame containing the data for the fitted spline curves.

get_classic_muscle_markers

Return the names of classic muscle genes

Description

Returns a list of classic muscle genes. Used to add convenience for loading HSMM data.

Usage

generateMarkers()
importCDS

Import a Seurat object and convert it to a monocle cds.

Description

This function takes a Seurat object and converts it to a monocle cds. It currently supports only the Seurat package.

Usage

importCDS(otherCDS, import_all = FALSE)

Arguments

otherCDS the object you would like to convert into a monocle cds
import_all Whether or not to import all the slots in seurat. Default is FALSE (or only keep minimal dataset).

Value

a new monocle cell dataset object converted from Seurat object.

Examples

## Not run:
lung <- load_lung()
seurat_lung <- exportCDS(lung)
seurat_lung_all <- exportCDS(lung, export_all = T)

importCDS(seurat_lung)
importCDS(seurat_lung, import_all = T)
importCDS(seurat_lung_all)
importCDS(seurat_lung_all, import_all = T)

## End(Not run)

load_HSMM

Build a CellDataSet from the HSMMSingleCell package

Description

Creates a cellDataSet using the data from the HSMMSingleCell package.

Usage

load_HSMM()
**load_HSM_Markes**

Return a CellDataSet of classic muscle genes.

**Description**

Return a CellDataSet of classic muscle genes.

**Usage**

`load_HSM_Markes()`

**Value**

A CellDataSet object

**load_lung**

Build a CellDataSet from the data stored in inst/extdata directory.

**Description**

Build a CellDataSet from the data stored in inst/extdata directory.

**Usage**

`load_lung()`

**markerDiffTable**

Test genes for cell type-dependent expression

**Description**

takes a CellDataSet and a CellTypeHierarchy and classifies all cells into types passed functions passed into the CellTypeHierarchy. The function will remove all “Unknown” and “Ambiguous” types before identifying genes that are differentially expressed between types.
Usage

```r
markerDiffTable(
  cds,
  cth,
  residualModelFormulaStr = "~1",
  balanced = FALSE,
  reclassify_cells = TRUE,
  remove_ambig = TRUE,
  remove_unknown = TRUE,
  verbose = FALSE,
  cores = 1
)
```

Arguments

- **cds**: A `CellDataSet` object containing cells to classify
- **cth**: The `CellTypeHierarchy` object to use for classification
- **residualModelFormulaStr**: A model formula string specifying effects you want to exclude when testing for cell type-dependent expression
- **balanced**: Whether to downsample the cells so that there's an equal number of each type prior to performing the test
- **reclassify_cells**: A boolean that indicates whether or not the `cds` and `cth` should be run through `classifyCells` again
- **remove_ambig**: A boolean that indicates whether or not ambiguous cells should be removed from the `cds`
- **remove_unknown**: A boolean that indicates whether or not unknown cells should be removed from the `cds`
- **verbose**: Whether to emit verbose output during the search for cell-type dependent genes
- **cores**: The number of cores to use when testing

Value

A table of differential expression test results

---

**mcesApply**

*Multicore apply-like function for CellDataSet*

Description

`mcesApply` computes the row-wise or column-wise results of `FUN`, just like `esApply`. Variables in `pData` from `X` are available in `FUN`. 
Usage

```r
mcesApply(
  X,
  MARGIN,
  FUN,
  required_packages,
  cores = 1,
  convert_to_dense = TRUE,
  ...
)
```

Arguments

- **X**: a CellDataSet object
- **MARGIN**: The margin to apply to, either 1 for rows (samples) or 2 for columns (features)
- **FUN**: Any function
- **required_packages**: A list of packages FUN will need. Failing to provide packages needed by FUN will generate errors in worker threads.
- **cores**: The number of cores to use for evaluation
- **convert_to_dense**: Whether to force conversion a sparse matrix to a dense one before calling FUN
- **...**: Additional parameters for FUN

Value

The result of `with(pData(X) apply(exprs(X), MARGIN, FUN, ...))`

Description

`minSpanningTree` retrieves the minimum spanning tree (MST) that Monocle constructs during `orderCells()`. This MST is mostly used in `plot_spanning_tree` to help assess the accuracy of Monocle's ordering.

Usage

```r
minSpanningTree(cds)
```

Arguments

- **cds**: expression data matrix for an experiment
minSpanningTree<-  

**Value**

An igraph object representing the CellDataSet’s minimum spanning tree.

**Examples**

```r
## Not run:
T <- minSpanningTree(HSMM)

## End(Not run)
```

---

**minSpanningTree<-**  

Set the minimum spanning tree generated by Monocle during cell ordering.

**Description**

Sets the minimum spanning tree used by Monocle during cell ordering. Not intended to be called directly.

**Usage**

```r
minSpanningTree(cds) <- value
```

**Arguments**

- `cds`  
  A CellDataSet object.
- `value`  
  An igraph object describing the minimum spanning tree.

**Value**

An updated CellDataSet object

**Examples**

```r
## Not run:
cds <- minSpanningTree(T)

## End(Not run)
```
newCellDataSet

Creates a new CellDataSet object.

Description

Creates a new CellDataSet object.

Usage

newCellDataSet(
  cellData,  
  phenoData = NULL, 
  featureData = NULL, 
  lowerDetectionLimit = 0.1, 
  expressionFamily = VGAM::negbinomial.size() 
)

Arguments

- **cellData**: expression data matrix for an experiment
- **phenoData**: data frame containing attributes of individual cells
- **featureData**: data frame containing attributes of features (e.g. genes)
- **lowerDetectionLimit**: the minimum expression level that constitutes true expression
- **expressionFamily**: the VGAM family function to be used for expression response variables

Value

a new CellDataSet object

Examples

```r
## Not run:
sample_sheet_small <- read.delim("../data/sample_sheet_small.txt", row.names=1)
sample_sheet_small$Time <- as.factor(sample_sheet_small$Time)
gene_annotations_small <- read.delim("../data/gene_annotations_small.txt", row.names=1)
fpkm_matrix_small <- read.delim("../data/fpkm_matrix_small.txt")
pd <- new("AnnotatedDataFrame", data = sample_sheet_small)
fd <- new("AnnotatedDataFrame", data = gene_annotations_small)
HSMM <- new("CellDataSet", exprs = as.matrix(fpkm_matrix_small), phenoData = pd, featureData = fd)
## End(Not run)
```
newCellTypeHierarchy  

Classify cells according to a set of markers

Description

Creates a CellTypeHierarchy object which can store cell types with the addCellType() function. When classifyCells is used with a CellDataSet and a CellTypeHierarchy cells in the CellDataSet can be classified as cell types found in the CellTypeHierarchy

classifyCells accepts a cellDataSet and and a cellTypeHierarchy. Each cell in the cellDataSet is checked against the functions in the cellTypeHierarchy to determine each cell’s type

Usage

newCellTypeHierarchy()

classifyCells(cds, cth, frequency_thresh = NULL, enrichment_thresh = NULL, ...)

calculateMarkerSpecificity(
    cds,
    cth,
    remove_ambig = TRUE,
    remove_unknown = TRUE
)

Arguments

cds  The CelllDataSet you want to classify
cth  CellTypeHierarchy
frequency_thresh  If at least this fraction of group of cells meet a cell types marker criteria, impute them all to be of that type.
enrichment_thresh  fraction to be multiplied by each cell type percentage. Only used if frequency_thresh is NULL, both cannot be NULL
...
character strings that you wish to pass to dplyr’s group_by_ routine
remove_ambig  a boolean that determines if ambiguous cells should be removed
remove_unknown  a boolean that determines whether unknown cells should be removed

Details

CellTypeHierarchy objects are Monocle’s mechanism for classifying cells into types based on known markers. To classify the cells in a CellDataSet object according to known markers, first construct a CellTypeHierarchy with newCellTypeHierarchy() and addCellType() and then provide both the CellDataSet and the CellTypeHierarchy to classifyCells(). Each call to addCellType()
newCellTypeHierarchy 39

registers a classification function that accepts the expression data from a CellDataSet object as input, and returns a boolean vector indicating whether each cell is of the given type. When you call classifyCells(), each cell will be checked against the classification functions in the CellTypeHierarchy. If you wish to make a cell type a subtype of another that’s already been registered with a CellTypeHierarchy object, make that one the “parent” type with the cell_type_name argument. If you want two types to be mutually exclusive, make them “siblings” by giving them the same parent. The classification functions in a CellTypeHierarchy must take a single argument, a matrix of expression values, as input. Note that this matrix could either be a sparseMatrix or a dense matrix. Explicitly casting the input to a dense matrix inside a classification function is likely to drastically slow down classifyCells and other routines that use CellTypeHierarchy objects. Successive calls to addCellType build up a tree of classification functions inside a CellTypeHierarchy. When two functions are siblings in the tree, classifyCells expects that a cell will meet the classification criteria for at most one of them. For example, you might place classification functions for T cells and B cells as siblings, because a cell cannot be both of these at the same time. When a cell meets the criteria for more than one function, it will be tagged as "Ambiguous". If classifyCells reports a large number of ambiguous cells, consider adjusting your classification functions. For example, some cells are defined by very high expression of a key gene that is expressed at lower levels in other cell types. Raising the threshold for this gene in a classification could resolve the ambiguities.

A classification function can also have child functions. You can use this to specify subtypes of cells. For example, T cells express the gene CD3, and there are many subtypes. You can encode each subtype by first adding a general T cell classification function that recognizes CD3, and then adding an additional function that recognizes CD4 (for CD4+ helper T cells), one for CD8 (to identify CD8+ cytotoxic T cells), and so on. classifyCells will aim to assign each cell to its most specific subtype in the "CellType" column. By default, classifyCells applies the classification functions to individual cells, but you can also apply it to cells in a "grouped" mode to impute the type of cells that are missing expression of your known markers. You can specify additional (quoted) grouping variables to classifyCells. The function will group the cells according to these factors, and then classify the cells. It will compute the frequency of each cell type in each group, and if a cell type is present at the frequency specified in frequency_thresh, all the cells in the group are classified as that type. If group contains more one cell type at this frequency, all the cells are marked "Ambiguous". This allows you to impute cell type based on unsupervised clustering results (e.g. with clusterCells()) or some other grouping criteria.

Value

newCellTypeHierarchy and addCellType both return an updated CellTypeHierarchy object. classifyCells returns an updated CellDataSet with a new column, "CellType", in the pData table.

For a CellDataset with N genes, and a CellTypeHierarchy with k types, returns a dataframe with N x k rows. Each row contains a gene and a specificity score for one of the types.

Functions

• classifyCells: Add a cell type to a CellTypeHierarchy
• calculateMarkerSpecificity: Calculate each gene’s specificity for each cell type

Computes the Jensen-Shannon distance between the distribution of a gene’s expression across cells and a hypothetical gene that is perfectly restricted to each cell type. The Jensen-Shannon distance is an information theoretic metric between two probability distributions. It is a widely
accepted measure of cell-type specificity. For a complete description see Cabili et al, Genes & Development (2011).

Examples

```r
## Not run:
# Initialize a new CellTypeHierarchy
cth <- newCellTypeHierarchy()

# Register a set of classification functions. There are multiple types of T cells
# A cell cannot be both a B cell and a T cell, a T cell and a Monocyte, or
# a B cell and a Monocyte.

cth <- addCellType(cth, "T cell",
  classify_func=function(x) {x["CD3D",] > 0})

cth <- addCellType(cth, "CD4+ T cell",
  classify_func=function(x) {x["CD4",] > 0},
  parent_cell_type_name = "T cell")

cth <- addCellType(cth, "CD8+ T cell",
  classify_func=function(x) {
    x["CD8A",] > 0 | x["CD8B",] > 0
  },
  parent_cell_type_name = "T cell")

cth <- addCellType(cth, "B cell",
  classify_func=function(x) {x["MS4A1",] > 0})

cth <- addCellType(cth, "Monocyte",
  classify_func=function(x) {x["CD14",] > 0})

# Classify each cell in the CellDataSet "mix" according to these types
mix <- classifyCells(mix, cth)

# Group the cells by the pData table column "Cluster". Apply the classification
functions to the cells groupwise. If a group is at least 5% of a type, make
them all that type. If the group is 5% one type, and 5% a different, mutually
exclusive type, mark the whole cluster "Ambiguous"
mix <- classifyCells(mix, Cluster, 0.05)

## End(Not run)
```

**orderCells**  
Orders cells according to pseudotime.

**Description**

Learns a "trajectory" describing the biological process the cells are going through, and calculates where each cell falls within that trajectory. Monocle learns trajectories in two steps. The
The first step is reducing the dimensionality of the data with `reduceDimension()`. The second is this function. This function takes as input a CellDataSet and returns it with two new columns: Pseudotime and State, which together encode where each cell maps to the trajectory. `orderCells()` optionally takes a "root" state, which you can use to specify the start of the trajectory. If you don’t provide a root state, one is selected arbitrarily.

**Usage**

```
orderCells(cds, root_state = NULL, num_paths = NULL, reverse = NULL)
```

**Arguments**

- `cds`: the CellDataSet upon which to perform this operation
- `root_state`: The state to use as the root of the trajectory. You must already have called `orderCells()` once to use this argument.
- `num_paths`: the number of end-point cell states to allow in the biological process.
- `reverse`: whether to reverse the beginning and end points of the learned biological process.

**Details**

The `reduction_method` argument to `reduceDimension()` determines which algorithm is used by `orderCells()` to learn the trajectory. If `reduction_method == "ICA"`, this function uses polygonal reconstruction to learn the underlying trajectory. If `reduction_method == "DDRTree"`, the trajectory is specified by the principal graph learned by the `DDRTree()` function.

Whichever algorithm you use, the trajectory will be composed of segments. The cells from a segment will share the same value of State. One of these segments will be selected as the root of the trajectory arbitrarily. The most distal cell on that segment will be chosen as the "first" cell in the trajectory, and will have a Pseudotime value of zero. `orderCells()` will then "walk" along the trajectory, and as it encounters additional cells, it will assign them increasingly large values of Pseudotime.

**Value**

an updated CellDataSet object, in which phenoData contains values for State and Pseudotime for each cell

---

Return an ordering for a P node in the PQ tree

**Order_p_node**

**Description**

Return an ordering for a P node in the PQ tree

**Usage**

```
order_p_node(q_level_list, dist_matrix)
```
plot_cell_clusters

Arguments

q_level_list  A list of Q nodes in the PQ tree
dist_matrix   A symmetric matrix of pairwise distances between cells

plot_cell_clusters  Plots clusters of cells.

Description

Plots clusters of cells.

Usage

plot_cell_clusters(
  cds,
  x = 1,
  y = 2,
  color_by = "Cluster",
  markers = NULL,
  show_cell_names = FALSE,
  cell_size = 1.5,
  cell_name_size = 2,
  ...
)

Arguments

cds  CellDataSet for the experiment
x    the column of reducedDimS(cds) to plot on the horizontal axis
y    the column of reducedDimS(cds) to plot on the vertical axis
color_by the cell attribute (e.g. the column of pData(cds)) to map to each cell’s color
markers a gene name or gene id to use for setting the size of each cell in the plot
show_cell_names draw the name of each cell in the plot
cell_size The size of the point for each cell
cell_name_size the size of cell name labels
...

Value

a ggplot2 plot object
plot_cell_trajectory

Examples

## Not run:
library(HSMMSingleCell)
HSMM <- load_HSMM()
HSMM <- reduceD
plot_cell_clusters(HSMM)
plot_cell_clusters(HSMM, color_by="Pseudotime")
plot_cell_clusters(HSMM, markers="MYH3")

## End(Not run)

plot_cell_trajectory  Plots the minimum spanning tree on cells.

Description

Plots the minimum spanning tree on cells.

Usage

plot_cell_trajectory(
  cds,
  x = 1,
  y = 2,
  color_by = "State",
  show_tree = TRUE,
  show_backbone = TRUE,
  backbone_color = "black",
  markers = NULL,
  use_color_gradient = FALSE,
  markers_linear = FALSE,
  show_cell_names = FALSE,
  show_state_number = FALSE,
  cell_size = 1.5,
  cell_link_size = 0.75,
  cell_name_size = 2,
  state_number_size = 2.9,
  show_branch_points = TRUE,
  theta = 0,
  ...
)

Arguments

cds                  CellDataSet for the experiment
x                     the column of reducedDimS(cds) to plot on the horizontal axis
y                     the column of reducedDimS(cds) to plot on the vertical axis
color_by: the cell attribute (e.g. the column of pData(cds)) to map to each cell's color

show_tree: whether to show the links between cells connected in the minimum spanning tree

show_backbone: whether to show the diameter path of the MST used to order the cells

backbone_color: the color used to render the backbone.

markers: a gene name or gene id to use for setting the size of each cell in the plot

use_color_gradient: Whether or not to use color gradient instead of cell size to show marker expression level

markers_linear: a boolean used to indicate whether you want to scale the markers logarithimically or linearly

show_cell_names: draw the name of each cell in the plot

show_state_number: show state number

cell_size: The size of the point for each cell

cell_link_size: The size of the line segments connecting cells (when used with ICA) or the principal graph (when used with DDRTree)

cell_name_size: the size of cell name labels

state_number_size: the size of the state number

show_branch_points: Whether to show icons for each branch point (only available when reduceDimension was called with DDRTree)

theta: How many degrees you want to rotate the trajectory

... Additional arguments passed into scale_color_viridis function

Value

a ggplot2 plot object

Examples

```r
## Not run:
lung <- load_lung()
plot_cell_trajectory(lung)
plot_cell_trajectory(lung, color_by="Pseudotime", show_backbone=FALSE)
plot_cell_trajectory(lung, markers="MYH3")

## End(Not run)
```
plot_clusters

Plots kinetic clusters of genes.

Description

returns a ggplot2 object showing the shapes of the expression patterns followed by a set of pre-selected genes. The topographic lines highlight the distributions of the kinetic patterns relative to overall trend lines.

Usage

plot_clusters(
  cds, 
  clustering, 
  drawSummary = TRUE, 
  sumFun = mean_cl_boot, 
  ncol = NULL, 
  nrow = NULL, 
  row_samples = NULL, 
  callout_ids = NULL
)

Arguments

cds     CellDataSet for the experiment
clustering a clustering object produced by clusterCells
drawSummary whether to draw the summary line for each cluster
sumFun   whether the function used to generate the summary for each cluster
ncol     number of columns used to layout the faceted cluster panels
nrow     number of columns used to layout the faceted cluster panels
row_samples how many genes to randomly select from the data
callout_ids a vector of gene names or gene ids to manually render as part of the plot

Value

a ggplot2 plot object

Examples

## Not run:
full_model_fits <- fitModel(HSMM_filtered[sample(nrow(fData(HSMM_filtered)), 100),],
                             modelFormulaStr="VGAM::bs(Pseudotime)"
) 
expression_curve_matrix <- responseMatrix(full_model_fits)
clusters <- clusterGenes(expression_curve_matrix, k=4)
plot_clusters(HSMM_filtered[ordering_genes,], clusters)

## End(Not run)
plot_coexpression_matrix

Not sure we’re ready to release this one quite yet: Plot the branch genes in pseudotime with separate branch curves

Description

Not sure we’re ready to release this one quite yet: Plot the branch genes in pseudotime with separate branch curves

Usage

plot_coexpression_matrix(
  cds,
  rowgenes,
  colgenes,
  relative_expr = TRUE,
  min_expr = NULL,
  cell_size = 0.85,
  label_by_short_name = TRUE,
  show_density = TRUE,
  round_expr = FALSE
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cds</td>
<td>CellDataSet for the experiment</td>
</tr>
<tr>
<td>rowgenes</td>
<td>Gene ids or short names to be arrayed on the vertical axis.</td>
</tr>
<tr>
<td>colgenes</td>
<td>Gene ids or short names to be arrayed on the horizontal axis.</td>
</tr>
<tr>
<td>relative_expr</td>
<td>Whether to transform expression into relative values</td>
</tr>
<tr>
<td>min_expr</td>
<td>The minimum level of expression to show in the plot</td>
</tr>
<tr>
<td>cell_size</td>
<td>A number how large the cells should be in the plot</td>
</tr>
<tr>
<td>label_by_short_name</td>
<td>a boolean that indicates whether cells should be labeled by their short name</td>
</tr>
<tr>
<td>show_density</td>
<td>a boolean that indicates whether a 2D density estimation should be shown in the plot</td>
</tr>
<tr>
<td>round_expr</td>
<td>a boolean that indicates whether cds_expr values should be rounded or not</td>
</tr>
</tbody>
</table>

Value

a ggplot2 plot object
plot_complex_cell_trajectory

Plots the minimum spanning tree on cells.

Description

Plots the minimum spanning tree on cells.

Usage

```r
plot_complex_cell_trajectory(
  cds,
  x = 1,
  y = 2,
  root_states = NULL,
  color_by = "State",
  show_tree = TRUE,
  show_backbone = TRUE,
  backbone_color = "black",
  markers = NULL,
  show_cell_names = FALSE,
  cell_size = 1.5,
  cell_link_size = 0.75,
  cell_name_size = 2,
  show_branch_points = TRUE,
  ...
)
```

Arguments

- **cds**: CellDataSet for the experiment
- **x**: the column of reducedDimS(cds) to plot on the horizontal axis
- **y**: the column of reducedDimS(cds) to plot on the vertical axis
- **root_states**: the state used to set as the root of the graph
- **color_by**: the cell attribute (e.g. the column of pData(cds)) to map to each cell’s color
- **show_tree**: whether to show the links between cells connected in the minimum spanning tree
- **show_backbone**: whether to show the diameter path of the MST used to order the cells
- **backbone_color**: the color used to render the backbone
- **markers**: a gene name or gene id to use for setting the size of each cell in the plot
- **show_cell_names**: draw the name of each cell in the plot
- **cell_size**: The size of the point for each cell
cell_link_size  The size of the line segments connecting cells (when used with ICA) or the principal graph (when used with DDRTree)
cell_name_size  the size of cell name labels
show_branch_points  Whether to show icons for each branch point (only available when reduceDimension was called with DDRTree)

...  Additional arguments passed to the scale_color_viridis function

Value

a ggplot2 plot object

Examples

```r
## Not run:
library(HSMMData)
HSM <- load_HSMM()
plot_complex_cell_trajectory(HSM)
plot_complex_cell_trajectory(HSM, color_by="Pseudotime", show_backbone=FALSE)
plot_complex_cell_trajectory(HSM, markers="MYH3")
## End(Not run)
```

Description

Create a heatmap to demonstrate the bifurcation of gene expression along two branches @description returns a heatmap that shows changes in both lineages at the same time. It also requires that you choose a branch point to inspect. Columns are points in pseudotime, rows are genes, and the beginning of pseudotime is in the middle of the heatmap. As you read from the middle of the heatmap to the right, you are following one lineage through pseudotime. As you read left, the other. The genes are clustered hierarchically, so you can visualize modules of genes that have similar lineage-dependent expression patterns.
Usage

```r
plot_genes_branched_heatmap(
  cds_subset,
  branch_point = 1,
  branch_states = NULL,
  branch_labels = c("Cell fate 1", "Cell fate 2"),
  cluster_rows = TRUE,
  hclust_method = "ward.D2",
  num_clusters = 6,
  hmcols = NULL,
  branch_colors = c("#979797", "#F05662", "#7990C8"),
  add_annotation_row = NULL,
  add_annotation_col = NULL,
  show_rownames = FALSE,
  use_gene_short_name = TRUE,
  scale_max = 3,
  scale_min = -3,
  norm_method = c("log", "vstExprs"),
  trend_formula = "~sm.ns(Pseudotime, df=3) * Branch",
  return_heatmap = FALSE,
  cores = 1,
  ...
)
```

Arguments

cds_subset  CellDataSet for the experiment (normally only the branching genes detected with branchTest)
branch_point The ID of the branch point to visualize. Can only be used when reduceDimension is called with method = "DDRTree".
branch_states The two states to compare in the heatmap. Mutually exclusive with branch_point.
branch_labels The labels for the branches.
cluster_rows Whether to cluster the rows of the heatmap.
hclust_method The method used by pheatmap to perform hiearchical clustering of the rows.
num_clusters Number of clusters for the heatmap of branch genes
hmcols The color scheme for drawing the heatmap.
branch_colors The colors used in the annotation strip indicating the pre- and post-branch cells.
add_annotation_row Additional annotations to show for each row in the heatmap. Must be a dataframe with one row for each row in the fData table of cds_subset, with matching IDs.
add_annotation_col Additional annotations to show for each column in the heatmap. Must be a dataframe with one row for each cell in the pData table of cds_subset, with matching IDs.
show_rownames Whether to show the names for each row in the table.
use_gene_short_name

Whether to use the short names for each row. If FALSE, uses row IDs from the fData table.

scale_max

The maximum value (in standard deviations) to show in the heatmap. Values larger than this are set to the max.

scale_min

The minimum value (in standard deviations) to show in the heatmap. Values smaller than this are set to the min.

norm_method

Determines how to transform expression values prior to rendering

trend_formula

A formula string specifying the model used in fitting the spline curve for each gene/feature.

return_heatmap

Whether to return the pheatmap object to the user.

cores

Number of cores to use when smoothing the expression curves shown in the heatmap.

...

Additional arguments passed to buildBranchCellDataSet

Value

A list of heatmap_matrix (expression matrix for the branch committment), ph (pheatmap heatmap object), annotation_row (annotation data.frame for the row), annotation_col (annotation data.frame for the column).

plot_genes_branched_pseudotime

Plot the branch genes in pseudo-time with separate branch curves.

Description

Works similarly to plot_genes_in_pseudotime except it shows one kinetic trend for each lineage.

Usage

plot_genes_branched_pseudotime(
  cds,
  branch_states = NULL,
  branch_point = 1,
  branch_labels = NULL,
  method = "fitting",
  min_expr = NULL,
  cell_size = 0.75,
  nrow = NULL,
  ncol = 1,
  panel_order = NULL,
  color_by = "State",
  expression_curve_linetype_by = "Branch",
  trend_formula = "(~ sm.ns(Pseudotime, df=3) * Branch)"
reducedModelFormulaStr = NULL,
label_by_short_name = TRUE,
relative_expr = TRUE,
...
)

Arguments

**cds**
CellDataSet for the experiment

**branch_states**
The states for two branching branches

**branch_point**
The ID of the branch point to analyze. Can only be used when reduceDimension is called with method = "DDRTree".

**branch_labels**
The names for each branching branch

**method**
The method to draw the curve for the gene expression branching pattern, either loess ("loess") or VGLM fitting ("fitting")

**min_expr**
The minimum (untransformed) expression level to use in plotted the genes.

**cell_size**
The size (in points) of each cell used in the plot

**nrow**
Number of columns used to layout the faceted cluster panels

**ncol**
Number of columns used to layout the faceted cluster panels

**panel_order**
The a character vector of gene short names (or IDs, if that’s what you’re using), specifying order in which genes should be layed out (left-to-right, top-to-bottom)

**color_by**
The cell attribute (e.g. the column of pData(cds)) to be used to color each cell

**expression_curve_linetype_by**
The cell attribute (e.g. the column of pData(cds)) to be used for the linetype of each branch curve

**trend_formula**
The model formula to be used for fitting the expression trend over pseudotime

**reducedModelFormulaStr**
A formula specifying a null model. If used, the plot shows a p value from the likelihood ratio test that uses trend_formula as the full model

**label_by_short_name**
Whether to label figure panels by gene_short_name (TRUE) or feature id (FALSE)

**relative_expr**
Whether or not the plot should use relative expression values (only relevant for CellDataSets using transcript counts)

... Additional arguments passed on to branchTest. Only used when reducedModelFormulaStr is not NULL.

Details

This plotting function is used to make the branching plots for a branch dependent gene goes through the progenitor state and bifurcating into two distinct branches (Similar to the pitch-fork bifurcation in dynamic systems). In order to make the bifurcation plot, we first duplicated the progenitor states and by default stretch each branch into maturation level 0-100. Then we fit two nature spline curves for each branches using VGAM package.
plot_genes_in_pseudotime

Plots expression for one or more genes as a function of pseudotime

Description

Plots expression for one or more genes as a function of pseudotime. Plotting allows you determine if the ordering produced by orderCells() is correct and it does not need to be flipped using the "reverse" flag in orderCells.

Usage

plot_genes_in_pseudotime(
  cds_subset,
  min_expr = NULL,
  cell_size = 0.75,
  nrow = NULL,
  ncol = 1,
  panel_order = NULL,
  color_by = "State",
  trend_formula = "~ sm.ns(Pseudotime, df=3)",
  label_by_short_name = TRUE,
  relative_expr = TRUE,
  vertical_jitter = NULL,
  horizontal_jitter = NULL
)

Arguments

- **cds_subset**: CellDataSet for the experiment
- **min_expr**: the minimum (untransformed) expression level to use in plotted the genes.
- **cell_size**: the size (in points) of each cell used in the plot
- **nrow**: the number of rows used when laying out the panels for each gene’s expression
- **ncol**: the number of columns used when laying out the panels for each gene’s expression
- **panel_order**: the order in which genes should be layed out (left-to-right, top-to-bottom)
- **color_by**: the cell attribute (e.g. the column of pData(cds)) to be used to color each cell
- **trend_formula**: the model formula to be used for fitting the expression trend over pseudotime
- **label_by_short_name**: label figure panels by gene_short_name (TRUE) or feature id (FALSE)
- **relative_expr**: Whether to transform expression into relative values
vertical_jitter
A value passed to ggplot to jitter the points in the vertical dimension. Prevents overplotting, and is particularly helpful for rounded transcript count data.

horizontal_jitter
A value passed to ggplot to jitter the points in the horizontal dimension. Prevents overplotting, and is particularly helpful for rounded transcript count data.

Value
a ggplot2 plot object

Examples
```r
## Not run:
library(HSMMsingleCell)
HSMM <- load_HSMM()
my_genes <- row.names(subset(fData(HSMM), gene_short_name %in% c("CDK1", "MEF2C", "MYH3")))
cds_subset <- HSMM[my_genes,]
plot_genes_in_pseudotime(cds_subset, color_by="Time")
## End(Not run)
```
**plot_genes_positive_cells**

Plots the number of cells expressing one or more genes as a barplot

**Description**

Accepts a `CellDataSet` and a parameter, "grouping", used for dividing cells into groups. Returns one or more bar graphs (one graph for each gene in the `CellDataSet`). Each graph shows the percentage of cells that express a gene in the `CellDataSet` for each sub-group of cells created by "grouping".

Let’s say the `CellDataSet` passed in included genes A, B, and C and the "grouping" parameter divided all of the cells into three groups called X, Y, and Z. Then three graphs would be produced called A, B, and C. In the A graph there would be three bars one for X, one for Y, and one for Z. So X bar in the A graph would show the percentage of cells in the X group that express gene A.
Usage

plot_genes_positive_cells(
  cds_subset,
  grouping = "State",
  min_expr = 0.1,
  nrow = NULL,
  ncol = 1,
  panel_order = NULL,
  plot_as_fraction = TRUE,
  label_by_short_name = TRUE,
  relative_expr = TRUE,
  plot_limits = c(0, 100)
)

Arguments

cds_subset CellDataSet for the experiment
grouping the cell attribute (e.g. the column of pData(cds)) to group cells by on the horizontal axis
min_expr the minimum (untransformed) expression level to use in plotted the genes.
nrow the number of rows used when laying out the panels for each gene’s expression
ncol the number of columns used when laying out the panels for each gene’s expression
panel_order the order in which genes should be layed out (left-to-right, top-to-bottom)
plot_as_fraction whether to show the percent instead of the number of cells expressing each gene
label_by_short_name label figure panels by gene_short_name (TRUE) or feature id (FALSE)
relative_expr Whether to transform expression into relative values
plot_limits A pair of number specifying the limits of the y axis. If NULL, scale to the range of the data.

Value

da ggplot2 plot object

Examples

## Not run:
library(HSMMSingleCell)
HSMM <- load_HSMM()
MYOG_ID1 <- HSMM[row.names(subset(fData(HSMM), gene_short_name %in% c("MYOG", "ID1")))]
plot_genes_positive_cells(MYOG_ID1, grouping="Media", ncol=2)

## End(Not run)
plot_genes_violin  

Plots expression for one or more genes as a violin plot

Description

Accepts a subset of a CellDataSet and an attribute to group cells by, and produces one or more ggplot2 objects that plots the level of expression for each group of cells.

Usage

```r
plot_genes_violin(
  cds_subset,
  grouping = "State",
  min_expr = NULL,
  cell_size = 0.75,
  nrow = NULL,
  ncol = 1,
  panel_order = NULL,
  color_by = NULL,
  plot_trend = FALSE,
  label_by_short_name = TRUE,
  relative_expr = TRUE,
  log_scale = TRUE
)
```

Arguments

cds_subset  
CellDataSet for the experiment

grouping  
the cell attribute (e.g. the column of pData(cds)) to group cells by on the horizontal axis

min_expr  
the minimum (untransformed) expression level to use in plotted the genes.

cell_size  
the size (in points) of each cell used in the plot

nrow  
the number of rows used when laying out the panels for each gene’s expression

ncol  
the number of columns used when laying out the panels for each gene’s expression

panel_order  
the order in which genes should be layed out (left-to-right, top-to-bottom)

color_by  
the cell attribute (e.g. the column of pData(cds)) to be used to color each cell

plot_trend  
whether to plot a trendline tracking the average expression across the horizontal axis.

label_by_short_name  
label figure panels by gene_short_name (TRUE) or feature id (FALSE)

relative_expr  
Whether to transform expression into relative values

log_scale  
a boolean that determines whether or not to scale data logarithmically
Value

a ggplot2 plot object

Examples

```r
## Not run:
library(HSMMSingleCell)
HSMM <- load_HSMM()
my_genes <- HSMM[row.names(subset(fData(HSMM), gene_short_name %in% c("ACTA1", "ID1", "CCNB2"))),]
plot_genes_violin(my_genes, grouping="Hours", ncol=2, min_expr=0.1)
## End(Not run)
```

 DESCRIPTION

Create a heatmap to demonstrate the bifurcation of gene expression along multiple branches

Usage

```r
plot_multiple_branches_heatmap(
  cds,
  branches,
  branches_name = NULL,
  cluster_rows = TRUE,
  hclust_method = "ward.D2",
  num_clusters = 6,
  hmcols = NULL,
  add_annotation_row = NULL,
  add_annotation_col = NULL,
  show_rownames = FALSE,
  use_gene_short_name = TRUE,
  norm_method = c("vstExprs", "log"),
  scale_max = 3,
  scale_min = -3,
  trend_formula = "~sm.ns(Pseudotime, df=3)",
  return_heatmap = FALSE,
  cores = 1
)
```
Arguments

- **cds**: CellDataSet for the experiment (normally only the branching genes detected with BEAM).
- **branches**: The terminal branches (states) on the developmental tree you want to investigate.
- **branches_name**: Name (for example, cell type) of branches you believe the cells on the branches are associated with.
- **cluster_rows**: Whether to cluster the rows of the heatmap.
- **hclust_method**: The method used by pheatmap to perform hierarchical clustering of the rows.
- **num_clusters**: Number of clusters for the heatmap of branch genes.
- **hmcols**: The color scheme for drawing the heatmap.
- **add_annotation_row**: Additional annotations to show for each row in the heatmap. Must be a dataframe with one row for each row in the fData table of cds_subset, with matching IDs.
- **add_annotation_col**: Additional annotations to show for each column in the heatmap. Must be a dataframe with one row for each cell in the pData table of cds_subset, with matching IDs.
- **show_rownames**: Whether to show the names for each row in the table.
- **use_gene_short_name**: Whether to use the short names for each row. If FALSE, uses row IDs from the fData table.
- **norm_method**: Determines how to transform expression values prior to rendering.
- **scale_max**: The maximum value (in standard deviations) to show in the heatmap. Values larger than this are set to the max.
- **scale_min**: The minimum value (in standard deviations) to show in the heatmap. Values smaller than this are set to the min.
- **trend_formula**: A formula string specifying the model used in fitting the spline curve for each gene/feature.
- **return_heatmap**: Whether to return the heatmap object to the user.
- **cores**: Number of cores to use when smoothing the expression curves shown in the heatmap.

Value

A list of heatmap_matrix (expression matrix for the branch commitment), ph (pheatmap heatmap object), annotation_row (annotation data.frame for the row), annotation_col (annotation data.frame for the column).
plot_multiple_branches_pseudotime

Create a kinetic curves to demonstrate the bifurcation of gene expression along multiple branches

Description

Create a kinetic curves to demonstrate the bifurcation of gene expression along multiple branches

Usage

plot_multiple_branches_pseudotime(
  cds,
  branches,
  branches_name = NULL,
  min_expr = NULL,
  cell_size = 0.75,
  norm_method = c("vstExprs", "log"),
  nrow = NULL,
  ncol = 1,
  panel_order = NULL,
  color_by = "Branch",
  trend_formula = "~sm.ns(Pseudotime, df=3)",
  label_by_short_name = TRUE,
  TPM = FALSE,
  cores = 1
)

Arguments

cds CellDataSet for the experiment (normally only the branching genes detected with BEAM)
branches The terminal branches (states) on the developmental tree you want to investigate.
branches_name Name (for example, cell type) of branches you believe the cells on the branches are associated with.
min_expr The minimum level of expression to show in the plot
cell_size A number how large the cells should be in the plot
norm_method Determines how to transform expression values prior to rendering
nrow the number of rows used when laying out the panels for each gene’s expression
ncol the number of columns used when laying out the panels for each gene’s expression
panel_order the order in which genes should be layed out (left-to-right, top-to-bottom)
color_by the cell attribute (e.g. the column of pData(cds)) to be used to color each cell
trend_formula the model formula to be used for fitting the expression trend over pseudotime
**plot_pc_variance_explained**

- **label_by_short_name**
  - Label figure panels by gene_short_name (TRUE) or feature id (FALSE)

- **TPM**
  - Whether to convert the expression value into TPM values.

- **cores**
  - Number of cores to use when smoothing the expression curves shown in the heatmap.

**Value**

- a ggplot2 plot object

---

**plot_ordering_genes**

Plots genes by mean vs. dispersion, highlighting those selected for ordering.

**Description**

Each gray point in the plot is a gene. The black dots are those that were included in the last call to setOrderingFilter. The red curve shows the mean-variance model learning by estimateDispersions().

**Usage**

`plot_ordering_genes(cds)`

**Arguments**

- **cds**
  - The CellDataSet to be used for the plot.

---

**plot_pc_variance_explained**

Plots the percentage of variance explained by the each component based on PCA from the normalized expression data using the same procedure used in reduceDimension function.

**Description**

Plots the percentage of variance explained by the each component based on PCA from the normalized expression data using the same procedure used in reduceDimension function.
function Description

Usage

plot_pc_variance_explained(
  cds,
  max_components = 100,
  norm_method = c("log", "vstExprs", "none"),
  residualModelFormulaStr = NULL,
  pseudo_expr = NULL,
  return_all = F,
  use_existing_pc_variance = FALSE,
  verbose = FALSE,
  ...
)

Arguments

cds CellDataSet for the experiment after running reduceDimension with reduction_method as tSNE
max_components Maximum number of components shown in the scree plot (variance explained by each component)
norm_method Determines how to transform expression values prior to reducing dimensionality
residualModelFormulaStr A model formula specifying the effects to subtract from the data before clustering.
pseudo_expr amount to increase expression values before dimensionality reduction
return_all A logical argument to determine whether or not the variance of each component is returned
use_existing_pc_variance Whether to plot existing results for variance explained by each PC
verbose Whether to emit verbose output during dimensionality reduction
... additional arguments to pass to the dimensionality reduction function

Examples

## Not run:
library(HSMMSingleCell)
HSMM <- load_HSMM()
plot_pc_variance_explained(HSMM)

## End(Not run)
plot_pseudotime_heatmap

Plots a pseudotime-ordered, row-centered heatmap

Description

The function plot_pseudotime_heatmap takes a CellDataSet object (usually containing a only subset of significant genes) and generates smooth expression curves much like plot_genes_in_pseudotime. Then, it clusters these genes and plots them using the pheatmap package. This allows you to visualize modules of genes that co-vary across pseudotime.

Usage

plot_pseudotime_heatmap(
    cds_subset,
    cluster_rows = TRUE,
    hclust_method = "ward.D2",
    num_clusters = 6,
    hmcols = NULL,
    add_annotation_row = NULL,
    add_annotation_col = NULL,
    show_rownames = FALSE,
    use_gene_short_name = TRUE,
    norm_method = c("log", "vstExprs"),
    scale_max = 3,
    scale_min = -3,
    trend_formula = "~sm.ns(Pseudotime, df=3)",
    return_heatmap = FALSE,
    cores = 1
)

Arguments

cds_subset: CellDataSet for the experiment (normally only the branching genes detected with branchTest)

cluster_rows: Whether to cluster the rows of the heatmap.

hclust_method: The method used by pheatmap to perform hierarchcal clustering of the rows.

num_clusters: Number of clusters for the heatmap of branch genes

hmcols: The color scheme for drawing the heatmap.

add_annotation_row: Additional annotations to show for each row in the heatmap. Must be a dataframe with one row for each row in the fData table of cds_subset, with matching IDs.

add_annotation_col: Additional annotations to show for each column in the heatmap. Must be a dataframe with one row for each cell in the pData table of cds_subset, with matching IDs.
**plot_rho_delta**

Plots the decision map of density clusters.

### Description

Plots the decision map of density clusters.

### Usage

```r
plot_rho_delta(cds, rho_threshold = NULL, delta_threshold = NULL)
```

### Arguments

- **cds**: CellDataSet for the experiment after running `clusterCells_Density_Peak`
- **rho_threshold**: The threshold of local density (rho) used to select the density peaks for plotting
- **delta_threshold**: The threshold of local distance (delta) used to select the density peaks for plotting

### Examples

```r
## Not run:
library(HSMMSingleCell)
HSMM <- load_HSMM()
plot_rho_delta(HSMM)
## End(Not run)
```
plot_spanning_tree  

Plots the minimum spanning tree on cells. This function is deprecated.

Description

This function arranges all of the cells in the cds in a tree and predicts their location based on their pseudotime value

Usage

plot_spanning_tree(
  cds,
  x = 1,
  y = 2,
  color_by = "State",
  show_tree = TRUE,
  show_backbone = TRUE,
  backbone_color = "black",
  markers = NULL,
  show_cell_names = FALSE,
  cell_size = 1.5,
  cell_link_size = 0.75,
  cell_name_size = 2,
  show_branch_points = TRUE
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cds</td>
<td>CellDataSet for the experiment</td>
</tr>
<tr>
<td>x</td>
<td>the column of reducedDimS(cds) to plot on the horizontal axis</td>
</tr>
<tr>
<td>y</td>
<td>the column of reducedDimS(cds) to plot on the vertical axis</td>
</tr>
<tr>
<td>color_by</td>
<td>the cell attribute (e.g. the column of pData(cds)) to map to each cell’s color</td>
</tr>
<tr>
<td>show_tree</td>
<td>whether to show the links between cells connected in the minimum spanning tree</td>
</tr>
<tr>
<td>show_backbone</td>
<td>whether to show the diameter path of the MST used to order the cells</td>
</tr>
<tr>
<td>backbone_color</td>
<td>the color used to render the backbone.</td>
</tr>
<tr>
<td>markers</td>
<td>a gene name or gene id to use for setting the size of each cell in the plot</td>
</tr>
<tr>
<td>show_cell_names</td>
<td>draw the name of each cell in the plot</td>
</tr>
<tr>
<td>cell_size</td>
<td>The size of the point for each cell</td>
</tr>
<tr>
<td>cell_link_size</td>
<td>The size of the line segments connecting cells (when used with ICA) or the principal graph (when used with DDRTree)</td>
</tr>
<tr>
<td>cell_name_size</td>
<td>the size of cell name labels</td>
</tr>
<tr>
<td>show_branch_points</td>
<td>Whether to show icons for each branch point (only available when reduceDimension was called with DDRTree)</td>
</tr>
</tbody>
</table>
pq_helper

Recursively builds and returns a PQ tree for the MST

Description

Recursively builds and returns a PQ tree for the MST

Usage

pq_helper(mst, use_weights = TRUE, root_node = NULL)

Arguments

- **mst**: The minimum spanning tree, as an igraph object.
- **use_weights**: Whether to use edge weights when finding the diameter path of the tree.
- **root_node**: The name of the root node to use for starting the path finding.
reducedDimA <-

| reducedDimA | Get the weights needed to lift cells back to high dimensional expression space. |

**Description**

Retrieves the weights that transform the cells' coordinates in the reduced dimension space back to the full (whitened) space.

**Usage**

reducedDimA(cds)

**Arguments**

- **cds** A CellDataSet object.

**Value**

A matrix that when multiplied by a reduced-dimension set of coordinates for the CellDataSet, recovers a matrix in the full (whitened) space

**Examples**

```r
## Not run:
A <- reducedDimA(HSMM)
## End(Not run)
```

---

reducedDimA<-

| reducedDimA<- | Get the weights needed to lift cells back to high dimensional expression space. |

**Description**

Sets the weights transform the cells’ coordinates in the reduced dimension space back to the full (whitened) space.

**Usage**

reducedDimA(cds) <- value

**Arguments**

- **cds** A CellDataSet object.
- **value** A whitened expression data matrix
reducedDimK

Value

An updated CellDataSet object

Examples

```r
## Not run:
cds <- reducedDimA(A)
## End(Not run)
```

---

reducedDimK

Retrieves the whitening matrix during independent component analysis.

Description

Retrieves the whitening matrix during independent component analysis.

Usage

`reducedDimK(cds)`

Arguments

- `cds`: A CellDataSet object.

Value

A matrix, where each row is a set of whitened expression values for a feature and columns are cells.

Examples

```r
## Not run:
K <- reducedDimW(HSMM)
## End(Not run)
```
reducedDimK <- Sets the whitening matrix during independent component analysis.

Description
Sets the whitening matrix during independent component analysis.

Usage
reducedDimK(cds) <- value

Arguments
- cds: A CellDataSet object.
- value: a numeric matrix

Value
A matrix, where each row is a set of whitened expression values for a feature and columns are cells.

Examples
```r
## Not run:
cds <- reducedDimK(K)
## End(Not run)
```

reducedDimS
Retrieves the coordinates of each cell in the reduced-dimensionality space generated by calls to reduceDimension.

Description
Reducing the dimensionality of the expression data is a core step in the Monocle workflow. After you call reduceDimension(), this function will return the new coordinates of your cells in the reduced space.

Usage
reducedDimS(cds)

Arguments
- cds: A CellDataSet object.
reducedDimS<- 69

Value

A matrix, where rows are cell coordinates and columns correspond to dimensions of the reduced space.

Examples

## Not run:
S <- reducedDimS(HSMM)
## End(Not run)

reducedDimS<-  Set embedding coordinates of each cell in a CellDataSet.

Description

This function sets the coordinates of each cell in a new (reduced-dimensionality) space. Not intended to be called directly.

Usage

reducedDimS(cds) <- value

Arguments

cds        A CellDataSet object.

value      A matrix of coordinates specifying each cell’s position in the reduced-dimensionality space.

Value

An update CellDataSet object

Examples

## Not run:
cds <- reducedDimS(S)
## End(Not run)
reducedDimW <- Get the whitened expression values for a CellDataSet.

Description

Retrieves the expression values for each cell (as a matrix) after whitening during dimensionality reduction.

Usage

reducedDimW(cds)

Arguments

cds A CellDataSet object.

Value

A matrix, where each row is a set of whitened expression values for a feature and columns are cells.

Examples

## Not run:
W <- reducedDimW(HSMM)
## End(Not run)

reducedDimW<- Sets the whitened expression values for each cell prior to independent component analysis. Not intended to be called directly.

Description

Sets the whitened expression values for each cell prior to independent component analysis. Not intended to be called directly.

Usage

reducedDimW(cds) <- value

Arguments

cds A CellDataSet object.
value A whitened expression data matrix
reduceDimension

Value

An updated CellDataSet object

Examples

```r
## Not run:
# ' cds <- reducedDimA(A)
## End(Not run)
```

reduceDimension Compute a projection of a CellDataSet object into a lower dimensional space

Description

Monocle aims to learn how cells transition through a biological program of gene expression changes in an experiment. Each cell can be viewed as a point in a high-dimensional space, where each dimension describes the expression of a different gene in the genome. Identifying the program of gene expression changes is equivalent to learning a trajectory that the cells follow through this space. However, the more dimensions there are in the analysis, the harder the trajectory is to learn. Fortunately, many genes typically co-vary with one another, and so the dimensionality of the data can be reduced with a wide variety of different algorithms. Monocle provides two different algorithms for dimensionality reduction via reduceDimension. Both take a CellDataSet object and a number of dimensions allowed for the reduced space. You can also provide a model formula indicating some variables (e.g. batch ID or other technical factors) to "subtract" from the data so it doesn’t contribute to the trajectory.

Usage

```r
reduceDimension(
  cds,
  max_components = 2,
  reduction_method = c("DDRTree", "ICA", "tSNE", "SimplePPT", "L1-graph", "SGL-tree"),
  norm_method = c("log", "vstExprs", "none"),
  residualModelFormulaStr = NULL,
  pseudo_expr = 1,
  relative_expr = TRUE,
  auto_param_selection = TRUE,
  verbose = FALSE,
  scaling = FALSE,
  ...
)
```
**Arguments**

- `cds`: the CellDataSet upon which to perform this operation
- `max_components`: the dimensionality of the reduced space
- `reduction_method`: A character string specifying the algorithm to use for dimensionality reduction.
- `norm_method`: Determines how to transform expression values prior to reducing dimensionality
- `residualModelFormulaStr`: A model formula specifying the effects to subtract from the data before clustering.
- `pseudo_expr`: amount to increase expression values before dimensionality reduction
- `relative_expr`: When this argument is set to TRUE (default), we intend to convert the expression into a relative expression.
- `auto_param_selection`: when this argument is set to TRUE (default), it will automatically calculate the proper value for the ncenter (number of centroids) parameters which will be passed into DDRTree call.
- `verbose`: Whether to emit verbose output during dimensionality reduction
- `scaling`: When this argument is set to TRUE (default), it will scale each gene before running trajectory reconstruction.
- ... additional arguments to pass to the dimensionality reduction function

**Details**

You can choose two different reduction algorithms: Independent Component Analysis (ICA) and Discriminative Dimensionality Reduction with Trees (DDRTree). The choice impacts numerous downstream analysis steps, including `orderCells`. Choosing ICA will execute the ordering procedure described in Trapnell and Cacchiarelli et al., which was implemented in Monocle version 1. `DDRTree` is a more recent manifold learning algorithm developed by Qi Mao and colleagues. It is substantially more powerful, accurate, and robust for single-cell trajectory analysis than ICA, and is now the default method.

Often, experiments include cells from different batches or treatments. You can reduce the effects of these treatments by transforming the data with a linear model prior to dimensionality reduction. To do so, provide a model formula through `residualModelFormulaStr`.

Prior to reducing the dimensionality of the data, it usually helps to normalize it so that highly expressed or highly variable genes don't dominate the computation. `reduceDimension()` automatically transforms the data in one of several ways depending on the `expressionFamily` of the CellDataSet object. If the `expressionFamily` is `negbinomial` or `negbinomial.size`, the data are variance-stabilized. If the `expressionFamily` is `Tobit`, the data are adjusted by adding a pseudo-count (of 1 by default) and then log-transformed. If you don’t want any transformation at all, set `norm_method` to "none" and `pseudo_expr` to 0. This may be useful for single-cell qPCR data, or data you’ve already transformed yourself in some way.

**Value**

an updated CellDataSet object
**relative2abs**  
*Transform relative expression values into absolute transcript counts.*

**Description**

Converts FPKM/TPM data to transcript counts. This allows for the use for negative binomial as an expressionFamily. These results are often far more accurate than using tobit().

**Usage**

```r
relative2abs(
  relative_cds,
  t_estimate = estimate_t(exprs(relative_cds)),
  modelFormulaStr = "~1",
  ERCC_controls = NULL,
  ERCC_annotation = NULL,
  volume = 10,
  dilution = 40000,
  mixture_type = 1,
  detection_threshold = 800,
  expected_capture_rate = 0.25,
  verbose = FALSE,
  return_all = FALSE,
  method = c("num_genes", "tpm_fraction"),
  cores = 1
)
```

**Arguments**

- **relative_cds**: the cds object of relative expression values for single cell RNA-seq with each row and column representing genes/isoforms and cells. Row and column names should be included.
- **t_estimate**: an vector for the estimated most abundant FPKM value of isoform for a single cell. Estimators based on gene-level relative expression can also give good approximation but estimators based on isoform FPKM will give better results in general.
- **modelFormulaStr**: modelformula used to grouping cells for transcript counts recovery. Default is "~ 1", which means to recover the transcript counts from all cells.
- **ERCC_controls**: the FPKM matrix for each ERCC spike-in transcript in the cells if user wants to perform the transformation based on their spike-in data. Note that the row and column names should match up with the ERCC_annotation and relative_exprs_matrix respectively.
- **ERCC_annotation**: the ERCC_annotation matrix from illumina USE GUIDE which will be used for calculating the ERCC transcript copy number for performing the transformation.
### volume
the approximate volume of the lysis chamber (nanoliters). Default is 10

### dilution
the dilution of the spikein transcript in the lysis reaction mix. Default is 40,000. The number of spike-in transcripts per single-cell lysis reaction was calculated from

### mixture_type
the type of spikein transcripts from the spikein mixture added in the experiments. By default, it is mixture 1. Note that m/c we inferred are also based on mixture 1.

### detection_threshold
the lowest concentration of spikein transcript considered for the regression. Default is 800 which will ensure (almost) all included spike transcripts expressed in all the cells. Also note that the value of c is based on this concentration.

### expected_capture_rate
the expected fraction of RNA molecules in the lysate that will be captured as cDNAs during reverse transcription

### verbose
a logical flag to determine whether or not we should print all the optimization details

### return_all
parameter for the intended return results. If setting TRUE, matrix of m, c, k^*, b^* as well as the transformed absolute cds will be returned in a list format

### method
the formula to estimate the total mRNAs (num_genes corresponds to the second formula while tpm_fraction corresponds to the first formula, see the announcement on Trapnell lab website for the Census paper)

### cores
number of cores to perform the recovery. The recovery algorithm is very efficient so multiple cores only needed when we have very huge number of cells or genes.

#### Details
Transform a relative expression matrix to absolute transcript matrix based on the inferred linear regression parameters from most abundant isoform relative expression value. This function takes a relative expression matrix and a vector of estimated most abundant expression value from the isoform-level matrix and transform it into absolute transcript number. It is based on the observation that the recovery efficient of the single-cell RNA-seq is relative low and that most expressed isoforms of gene in a single cell therefore only sequenced one copy so that the most abundant isoform log10-FPKM (t^*) will corresponding to 1 copy transcript. It is also based on the fact that the spikein regression parameters k/b for each cell will fall on a line because of the intrinsic properties of spikein experiments. We also assume that if we perform the same spikein experiments as Treutlein et al. did, the regression parameters should also fall on a line in the same way. The function takes the the vector t^* and the detection limit as input, then it uses the t^* and the m/c value corresponding to the detection limit to calculate two parameters vectors k^* and b^* (corresponding to each cell) which correspond to the slope and intercept for the linear conversion function between log10 FPKM and log10 transcript counts. The function will then apply a linear transformation to convert the FPKM to estimated absolute transcript counts based on the the k^* and b^*. The default m/c values used in the algorithm are 3.652201, 2.263576, respectively.

#### Value
an matrix of absolute count for isoforms or genes after the transformation.
Examples

## Not run:
HSMM_relative_expr_matrix <- exprs(HSMM)
HSMM_abs_matrix <- relative2abs(HSMM_relative_expr_matrix, 
t_estimate = estimate_t(HSMM_relative_expr_matrix))
## End(Not run)

residualMatrix  
Response values

Description
Generates a matrix of response values for a set of fitted models

Usage
residualMatrix(models, residual_type = "response", cores = 1)

Arguments
models  
a list of models, e.g. as returned by fitModels()
residual_type  
the response desired, as accepted by VGAM’s predict function
cores  
number of cores used for calculation

Value
a matrix where each row is a vector of response values for a particular feature’s model, and columns are cells.

responseMatrix  
Calculates response values.

Description
Generates a matrix of response values for a set of fitted models

Usage
responseMatrix(models, newdata = NULL, response_type = "response", cores = 1)

Arguments
models  
a list of models, e.g. as returned by fitModels()
newdata  
a dataframe used to generate new data for interpolation of time points
response_type  
the response desired, as accepted by VGAM’s predict function
cores  
number of cores used for calculation


**setOrderingFilter**

Value

a matrix where each row is a vector of response values for a particular feature’s model, and columns are cells.

---

**selectTopMarkers**  
*Select the most cell type specific markers*

Description

This is a handy wrapper function around dplyr’s `top_n` function to extract the most specific genes for each cell type. Convenient, for example, for selecting a balanced set of genes to be used in semi-supervised clustering or ordering.

Usage

`selectTopMarkers(marker_specificities, num_markers = 10)`

Arguments

- **marker_specificities**
  The dataframe of specificity results produced by `calculateMarkerSpecificity()`
- **num_markers**
  The number of markers that will be shown for each cell type

Value

A data frame of specificity results

---

**setOrderingFilter**  
*Marks genes for clustering*

Description

The function marks genes that will be used for clustering in subsequent calls to `clusterCells`. The list of selected genes can be altered at any time.

Usage

`setOrderingFilter(cds, ordering_genes)`

Arguments

- **cds**
  the CellDataSet upon which to perform this operation
- **ordering_genes**
  a vector of feature ids (from the CellDataSet’s featureData) used for ordering cells

Value

an updated CellDataSet object
**spike_df**  

*Spike-in transcripts data.*

**Description**

A dataset containing the information for the 92 ERCC spikein transcripts (This dataset is based on the data from the Nature paper from Stephen Quake group)

**Usage**

```r
spike_df
```

**Format**

A data frame with 92 rows and 9 variables:

- **ERCC_ID**  ID for ERCC transcripts
- **subgroup**  Subgroup for ERCC transcript
- **conc_attomoles_ul_Mix1**  Contraction of Mix 1 (attomoles / ul)
- **conc_attomoles_ul_Mix2**  Contraction of Mix 2 (attomoles / ul)
- **exp_fch_ratio**  expected fold change between mix 1 over mix 2
- **numMolecules**  number of molecules calculated from concentration and volume
- **rounded_numMolecules**  number in rounded digit of molecules calculated from concentration and volume

**vstExprs**  

*Return a variance-stabilized matrix of expression values*

**Description**

This function was taken from the DESeq package (Anders and Huber) and modified to suit Monocle's needs. It accepts a either a CellDataSet or the expression values of one and returns a variance-stabilized matrix based off of them.

**Usage**

```r
vstExprs(cds, dispModelName = "blind", expr_matrix = NULL, round_vals = TRUE)
```

**Arguments**

- **cds**  A CellDataSet to use for variance stabilization.
- **dispModelName**  The name of the dispersion function to use for VST.
- **expr_matrix**  An matrix of values to transform. Must be normalized (e.g. by size factors) already. This function doesn’t do this for you.
- **round_vals**  Whether to round expression values to the nearest integer before applying the transformation.
Index

* datasets
  - spike_df, 77
addCellType, 4
BEAM, 4, 6, 7
branchTest, 6
buildBranchCellDataSet, 7

calABCs, 8
calculateMarkerSpecificity, 76
calculateMarkerSpecificity
(newCellTypeHierarchy), 38
calibrate_per_cell_total_proposal, 9
calILRs, 10
CellDataSet, 11
CellDataSet, ANY, ANY-method
(CellDataSet-methods), 12
CellDataSet-class (CellDataSet), 11
CellDataSet-methods, 12
cellPairwiseDistances, 13
cellPairwiseDistances<-, 14
CellType, 15
CellType-class (CellType), 15
CellTypeHierarchy, 15
CellTypeHierarchy-class
(CellTypeHierarchy), 15
classifyCells (newCellTypeHierarchy), 38
clusterCells, 16, 39
clusterGenes, 18
compareModels, 19

DDRTree, 41, 72
detectBifurcationPoint, 19
detectGenes, 21
diff_test_helper, 23
differentialGeneTest, 22
dispersionTable, 24

estimate_t, 26

estimateDispersions, CellDataSet-method
(CellDataSet-methods), 12
estimateDispersionsForCellDataSet, 24
estimateSizeFactors, CellDataSet-method
(CellDataSet-methods), 12
estimateSizeFactorsForMatrix, 25
exportCDS, 27
extract_good_branched_ordering, 27

fit_model_helper, 29
fitModel, 28
genSmoothCurveResiduals, 30
genSmoothCurves, 30
get_classic_muscle_markers, 31

importCDS, 32
load_HSMM, 32
load_HSMM_markers, 33
load_lung, 33

markerDiffTable, 33
mcesApply, 34
minSpanningTree, 35
minSpanningTree<-, 36

newCellDataSet, 37
newCellTypeHierarchy, 38

order_p_node, 41
orderCells, 40, 72

plot_cell_clusters, 42
plot_cell_trajectory, 43
plot_clusters, 45
plot_coexpression_matrix, 46
plot_complex_cell_trajectory, 47
plot_genes_branched_heatmap, 48
plot_genes_branched_pseudotime, 50
plot_genes_in_pseudotime, 52
plot_genes_jitter, 53
plot_genes_positive_cells, 54
plot_genes_violin, 56
plot_multiple_branches_heatmap, 57
plot_multiple_branches_pseudotime, 59
plot_ordering_genes, 60
plot_pc_variance_explained, 60
plot_pseudotime_heatmap, 62
plot_rho_delta, 63
plot_spanning_tree, 64
pq_helper, 65
reducedDimA, 66
reducedDimA<-, 66
reducedDimK, 67
reducedDimK<-, 68
reducedDimS, 68
reducedDimS<-, 69
reducedDimW, 70
reducedDimW<-, 70
reduceDimension, 7, 41, 71
relative2abs, 73
residualMatrix, 75
responseMatrix, 75
selectTopMarkers, 76
setOrderingFilter, 76
sizeFactors, CellDataSet-method
   (CellDataSet-methods), 12
sizeFactors<-, CellDataSet, numeric-method
   (CellDataSet-methods), 12
sparseMatrix, 39
spike_df, 77
vglm, 23
vstExprs, 77