Package ‘monocle’

April 14, 2017

Type Package

Title Clustering, differential expression, and trajectory analysis for single- cell RNA-Seq

Version 2.2.0

Date 2016-06-06

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 (>= 1.0-1), 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, MASS, reshape2, limma, dplyr, qlcMatrix, pheatmap, stringr, proxy, slam, stats

VignetteBuilder knitr

Suggests knitr, Hmisc, testthat

Roxygen list(wrap = FALSE)

LazyData true

biocViews Sequencing, RNASeq, GeneExpression, DifferentialExpression, Infrastructure, DataImport, DataRepresentation, Visualization, Clustering, MultipleComparison, QualityControl

RoxygenNote 5.0.1

NeedsCompilation no

R topics documented:

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

Description

Identify genes with branch-dependent expression

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

Details

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.

Value

a data frame containing the p values and q-values from the BEAM test, with one row per gene.

`branchTest` *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

```
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

Value

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

`buildBranchCellDataSet` *Build a CellDataSet that splits cells among two branches*

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

```
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 the nature spline curves fitted for each branch. ABC score is used to quantify the total magnitude of divergence between two branchs. 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

```
calABCs(cds, trend_formula = "~sm.ns(Pseudotime, df = 3)*Branch",
  trajectory_states = c(2, 3), 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.
`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
`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)
```

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

`calILRs` *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 hirearchy. If "stretch" is specifified, 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 linages. 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 branchs.

Usage

```
calILRs(cds, trend_formula = "~sm.ns(Pseudotime, df = 3)*Branch",
  trajectory_states = c(2, 3), 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.
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 Stablization Transformation should be used to stablize the gene expression. When VST is used, the difference between two branchs 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.

Slots

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 auxilliary 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

```
## 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(object, modelFormulaStr = "~ 1",
  relative_expr = TRUE, min_cells_detected = 1, remove_outliers = TRUE,
  cores = 1, ...)
```

Arguments

`object`	The CellDataSet object
`value`	A vector of size factors, with length equal to the cells in object
`locfunc`	A function applied to the geometric-mean-scaled expression values to derive the size factor.
`...`	Additional arguments to be passed to estimateSizeFactorsForMatrix
`modelFormulaStr`	A model formula, passed as a string, specifying how to group the cells prior to estimated dispersion. The default groups all cells together.
`relative_expr`	Whether to transform expression into relative values
`min_cells_detected`	Only include genes detected above lowerDetectionLimit in at least this many cells in the dispersion calculation
`remove_outliers`	Whether to remove outliers (using Cook's distance) when estimating dispersions
`cores`	The number of cores to use for computing dispersions

`cellPairwiseDistances` *Get the matrix of pairwise distances between cells*

Description

Retrieves a matrix capturing distances between each cell used during cell ordering.

Usage

```
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.

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.*

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, and then returns the CellDataSet with the cluster assignments stored in the pData table.

Usage

```
clusterCells(cds, num_clusters, cell_type_hierarchy = NULL,
  frequency_thresh = 0.1, clustering_genes = NULL, max_components = 10,
  residualModelFormulaStr = NULL, param.gamma = 100, verbose = F, ...)
```

Arguments

`cds` the CellDataSet upon which to perform this operation

`num_clusters` number of desired cell clusters

`cell_type_hierarchy` the CellTypeHierarchy that divides the cells from cds into different types of 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.

`clustering_genes` a vector of genes used to differentiate between the cell types in the CellTypeHierarchy

`max_components` number of dimensions to project the data into via `reduceDimension()`

`residualModelFormulaStr` A model formula specifying the effects to subtract from the data before clustering.

`param.gamma` gamma parameter for DDRTree

`verbose` Verbose parameter for DDRTree

`...` Additional arguments passed to `reduceDimension()`

Value

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

---

clusterGenes *Clusters genes by pseudotime trend.*

Description

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

Usage

```
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

Examples

```
## 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

```
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.

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 branchs

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
`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 branchs 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` *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.*

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

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 covariates 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 expression.
`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.

See Also

`vglm`

`diff_test_helper` *Helper function for parallel differential expression testing*

Description

test

Usage

```
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.

`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, cores)
```

Arguments

`cds`	a CellDataSet that contains all cells user wants evaluated
`modelFormulaStr`	a formula string specifying the model to fit for the genes.
`relative_expr`	Whether to transform expression into relative values
`min_cells_detected`	Only include genes detected above lowerDetectionLimit in at least this many cells in the dispersion calculation
`removeOutliers`	a boolean it determines whether or not outliers from the data should be removed
`cores`	the number of cores to be used while testing each gene for differential expression.

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 representive 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 appraoches. 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 occuring relative expression value in each cell*

Description

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

Usage

```
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

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

## 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

Fits a model for each gene in a CellDataSet object.

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 count-based expression data. If TRUE, counts are normalized by Size_Factor prior to 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 Pseudotime 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

---

fit_model_helper — *Helper function for parallel VGAM fitting*

Description

test

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 build on other functions (fit_models and residualsMatrix)

Usage

```
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 build 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

Return the names of classic muscle genes

Usage

```
get_classic_muscle_markers()
```

`load_HSMM` *Build a CellDataSet from the HSMMSingleCell package*

Description

Build a CellDataSet from the HSMMSingleCell package

Usage

```
load_HSMM()
```

`load_HSMM_markers` *Return a CellDataSet of classic muscle genes*

Description

Return a CellDataSet of classic muscle genes

Usage

```
load_HSMM_markers()
```

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

Test genes for cell type-dependent expression

Usage

```
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 specify 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 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 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

```
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, ...))

---

minSpanningTree *Retrieves the minimum spanning tree generated by Monocle during cell ordering.*

Description

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

```
minSpanningTree(cds)
```

Arguments

cds	expression data matrix for an experiment

Value

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

Examples

```
## 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

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

Arguments

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

Value

An updated CellDataSet object

Examples

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

## End(Not run)
```

`newCellDataSet` *Creates a new CellDateSet object.*

Description

Creates a new CellDateSet object.

Usage

```
newCellDataSet(cellData, phenoData = NULL, featureData = NULL,
  lowerDetectionLimit = 0.1, expressionFamily = VGAM::tobit(Lower =
  log10(lowerDetectionLimit), lmu = "identitylink"))
```

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 consistitutes true expression
`expressionFamily`	the VGAM family function to be used for expression response variables

Value

a new CellDataSet object

Examples

```
## 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

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 CellTypeHierachy with `newCellTypeHierarchy()` and `addCellType()` and then provide both the `CellDataSet` and the `CellTypeHierachy` to `classifyCells()`. Each call to `addCellType()` 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 `CellTypeHierachy`. 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.

Usage

```
newCellTypeHierarchy()

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

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

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

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

`cds` The CelllDataSet you want to classify

`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.

`...` 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

The classifcation 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 CellTypeHierarhcy 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 subset 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 frquency 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 "Ambigious". 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 specifity score for one of the types.

Functions

- `addCellType`: Add a cell type to a CellTypeHierarchy
- `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

```
## Not run:
# Initialize a new CellTypeHierachy

# 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 <- newCellTypeHierarchy()

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 first step is reducing the dimensionality of the data with `reduceDimension()`. The second is this function. 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

---

`order_p_node` *Return an ordering for a P node in the PQ tree*

Description

Return an ordering for a P node in the PQ tree

Usage

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

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_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, 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
`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)

Value

a ggplot2 plot object

Examples

```
## Not run:
data(HSMM)
plot_cell_trajectory(HSMM)
plot_cell_trajectory(HSMM, color_by="Pseudotime", show_backbone=FALSE)
plot_cell_trajectory(HSMM, markers="MYH3")

## End(Not run)
```

---

plot_clusters *Plots kinetic clusters of genes.*

Description

Plots kinetic clusters of genes.

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 pseduotime with separate branch curves

---

Description

Not sure we're ready to release this one quite yet: Plot the branch genes in pseduotime 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

`cds`	CellDataSet for the experiment
`rowgenes`	Gene ids or short names to be arrayed on the vertical axis.
`colgenes`	Gene ids or short names to be arrayed on the horizontal axis
`relative_expr`	Whether to transform expression into relative values
`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
`label_by_short_name`	a boolean that indicates whether cells should be labeled by their short name
`show_density`	a boolean that indicates whether a 2D density estimation should be shown in the plot
`round_expr`	a boolean that indicates whether cds_expr values should be rounded or not

Value

a ggplot2 plot object

`plot_genes_branched_heatmap`

Create a heatmap to demonstrate the bifurcation of gene expression along two branchs

---

Description

Create a heatmap to demonstrate the bifurcation of gene expression along two branchs

Usage

```
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("vstExprs", "log"),
  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 branchs.
`cluster_rows`	Whether to cluster the rows of the heatmap.
`hclust_method`	The method used by pheatmap to perform hirearchical 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 pseduotime with separate branch curves.

Description

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 branchs (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 branchs using VGAM package.

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 branchs
`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.

Value

a ggplot2 plot object

---

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

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

```
## Not run:
data(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_jitter *Plots expression for one or more genes as a jittered, grouped points*

Description

Plots expression for one or more genes as a jittered, grouped points

Usage

```
plot_genes_jitter(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)
```

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

Value

a ggplot2 plot object

Examples

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

## End(Not run)
```

---

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

Description

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

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

a ggplot2 plot object

Examples

```
## Not run:
data(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_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_pseudotime_heatmap` *Plots a pseudotime-ordered, row-centered heatmap*

Description

Plots a pseudotime-ordered, row-centered heatmap

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("vstExprs", "log"), 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 hirearchical 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 pheatmap 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 committment), ph (pheatmap heatmap object), annotation_row (annotation data.frame for the row), annotation_col (annotation data.frame for the column).

`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

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
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)

Value

a ggplot2 plot object

See Also

plot_cell_trajectory

Examples

```
## Not run:
data(HSMM)
plot_cell_trajectory(HSMM)
plot_cell_trajectory(HSMM, color_by="Pseudotime", show_backbone=FALSE)
plot_cell_trajectory(HSMM, markers="MYH3")

## End(Not run)
```

---

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` *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

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

## End(Not run)
```

`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

Value

An updated CellDataSet object

Examples

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

## End(Not run)
```

`reducedDimK` *Retrieves the the whitening matrix during independent component analysis.*

Description

Retrieves the 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

```
## Not run:
K <- reducedDimW(HSMM)

## End(Not run)
```

`reducedDimK<-` *Sets the the whitening matrix during independent component analysis.*

Description

Sets the 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

```
## 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.

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

Value

An updated CellDataSet object

Examples

```
## 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

```
reduceDimension(cds, max_components = 2, reduction_method = c("DDRTree",
  "ICA"), norm_method = c("vstExprs", "log", "none"),
  residualModelFormulaStr = NULL, pseudo_expr = NULL, verbose = 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

`verbose` Whether to emit verbose output during dimensionality reduction

`...` 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 colleages. 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 pseudocount (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 maybe 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

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 algoritm are 3.652201, 2.263576, respectively.

Usage

```
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, 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 ued 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

`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.

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 *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

Value

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

scale_pseudotime *Scale pseudotime to be in the range from 0 to 100*

Description

This function transforms the pseudotime scale so that it ranges from 0 to 100. If there are multiple branches, each leaf is set to be 100, with branches stretched accordingly.

Usage

```
scale_pseudotime(cds, verbose = F)
```

Arguments

cds	the CellDataSet upon which to perform this operation
verbose	Whether to emit verbose output

Value

an updated CellDataSet object which an

selectNegentropyGenes *Filter genes with extremely high or low negentropy*

Description

Filter genes with extremely high or low negentropy

Usage

```
selectNegentropyGenes(cds, lower_negentropy_bound = "0%",
  upper_negentropy_bound = "99%", expression_lower_thresh = 0.1,
  expression_upper_thresh = Inf)
```

Arguments

cds	a CellDataSet object upon which to perform this operation
lower_negentropy_bound	the centile below which to exclude to genes
upper_negentropy_bound	the centile above which to exclude to genes
expression_lower_thresh	the expression level below which to exclude genes used to determine negentropy
expression_upper_thresh	the expression level above which to exclude genes used to determine negentropy

Value

a vector of gene names

Examples

```
## Not run:
reasonableNegentropy <- selectNegentropyGenes(HSMM, "07%", "95%", 1, 100)

## End(Not run)
```

---

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 *Sets the features (e.g. genes) to be used for ordering cells in pseudo-time.*

Description

Sets the features (e.g. genes) to be used for ordering cells in pseudotime.

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

```
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 Contration of Mix 1 (attomoles / ul)

conc_attomoles_ul_Mix2 Contration 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

Usage

```
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