

# Package ‘dreamlet’

May 20, 2024

**Type** Package

**Title** Scalable differential expression analysis of single cell transcriptomics datasets with complex study designs

**Version** 1.3.0

**Date** 2024-02-14

**Description** Recent advances in single cell/nucleus transcriptomic technology has enabled collection of cohort-scale datasets to study cell type specific gene expression differences associated disease state, stimulus, and genetic regulation. The scale of these data, complex study designs, and low read count per cell mean that characterizing cell type specific molecular mechanisms requires a user-friendly, purpose-build analytical framework. We have developed the dreamlet package that applies a pseudobulk approach and fits a regression model for each gene and cell cluster to test differential expression across individuals associated with a trait of interest. Use of precision-weighted linear mixed models enables accounting for repeated measures study designs, high dimensional batch effects, and varying sequencing depth or observed cells per biosample.

**VignetteBuilder** knitr

**License** Artistic-2.0

**Encoding** UTF-8

**URL** <https://DiseaseNeurogenomics.github.io/dreamlet>

**BugReports** <https://github.com/DiseaseNeurogenomics/dreamlet/issues>

**Suggests** BiocStyle, knitr, pander, rmarkdown, muscat, ExperimentHub, RUnit, scater, scuttle

**biocViews** RNASeq, GeneExpression, DifferentialExpression, BatchEffect, QualityControl, Regression, GeneSetEnrichment, GeneRegulation, Epigenetics, FunctionalGenomics, Transcriptomics, Normalization, SingleCell, Preprocessing, Sequencing, ImmunoOncology, Software

**Depends** R (>= 4.3.0), variancePartition (>= 1.33.11), SingleCellExperiment, ggplot2

**Imports** edgeR, SummarizedExperiment, DelayedMatrixStats, sparseMatrixStats, MatrixGenerics, Matrix, methods, purrr,

GSEABase, data.table, zenith ( $\geq 1.1.2$ ), mashr ( $\geq 0.2.52$ ),  
 ashr, dplyr, BiocParallel, ggbeeswarm, S4Vectors, IRanges,  
 irlba, limma, metafor, remaCor, broom, tidyr, rlang,  
 BiocGenerics, DelayedArray, gtools, reshape2, ggrepel,  
 scattermore, Rcpp, lme4 ( $\geq 1.1-33$ ), MASS, Rdpack, utils, stats

**RoxygenNote** 7.2.3

**RdMacros** Rdpack

**SystemRequirements** C++11

**LinkingTo** Rcpp, beachmat

**git\_url** <https://git.bioconductor.org/packages/dreamlet>

**git\_branch** devel

**git\_last\_commit** cf98b09

**git\_last\_commit\_date** 2024-04-30

**Repository** Bioconductor 3.20

**Date/Publication** 2024-05-20

**Author** Gabriel Hoffman [aut, cre] (<<https://orcid.org/0000-0002-0957-0224>>)

**Maintainer** Gabriel Hoffman <[gabriel.hoffman@mssm.edu](mailto:gabriel.hoffman@mssm.edu)>

## Contents

aggregateNonCountSignal . . . . .	4
aggregateToPseudoBulk . . . . .	5
aggregateVar . . . . .	7
as.dreamletResult . . . . .	9
assay,dreamletResult,ANY-method . . . . .	10
assayNames,dreamletResult-method . . . . .	11
buildClusterTreeFromPB . . . . .	11
cellCounts . . . . .	12
cellSpecificityValues-class . . . . .	13
cellTypeSpecificity . . . . .	14
checkFormula . . . . .	15
coefNames . . . . .	16
colData,dreamletProcessedData-method . . . . .	17
colData<-,dreamletProcessedData,ANY-method . . . . .	17
compositePosteriorTest . . . . .	18
computeCellCounts . . . . .	19
computeLogCPM . . . . .	20
computeNormCounts . . . . .	21
details . . . . .	22
diffVar,dreamletResult-method . . . . .	23
dreamlet . . . . .	25
dreamletCompareClusters . . . . .	27
dreamletProcessedData-class . . . . .	30
dreamletResult-class . . . . .	31

dreamlet_mash_result-class . . . . .	31
dropRedundantTerms . . . . .	32
equalFormulas . . . . .	32
extractData . . . . .	33
fitVarPart . . . . .	34
getExprGeneNames . . . . .	36
getTreat,dreamletResult-method . . . . .	37
metadata,dreamletProcessedData-method . . . . .	38
meta_analysis . . . . .	39
outlier . . . . .	40
outlierByAssay . . . . .	41
pbWeights . . . . .	42
plotBeeswarm . . . . .	44
plotCellComposition . . . . .	45
plotForest . . . . .	46
plotGeneHeatmap . . . . .	47
plotHeatmap . . . . .	49
plotPCA . . . . .	50
plotPercentBars,vpDF-method . . . . .	52
plotProjection . . . . .	53
plotVarPart,DataFrame-method . . . . .	54
plotViolin . . . . .	56
plotVolcano . . . . .	57
plotVoom . . . . .	59
print,dreamletResult-method . . . . .	60
processAssays . . . . .	61
processOneAssay . . . . .	63
removeConstantTerms . . . . .	64
residuals,dreamletResult-method . . . . .	65
run_mash . . . . .	66
seeErrors . . . . .	68
show,dreamletResult-method . . . . .	69
sortCols,vpDF-method . . . . .	70
stackAssays . . . . .	71
tabToMatrix . . . . .	72
topTable,dreamletResult-method . . . . .	73
vpDF-class . . . . .	74
zenith_gsa,dreamletResult,GeneSetCollection-method . . . . .	75
[,dreamletResult,ANY,ANY,ANY-method . . . . .	77

---

```
aggregateNonCountSignal
```

*Aggregation of single-cell signals*

---

### Description

Aggregation of single-cell to pseudobulk data for non-count data.

### Usage

```
aggregateNonCountSignal(
  sce,
  assay = NULL,
  sample_id = NULL,
  cluster_id = NULL,
  min.cells = 10,
  min.signal = 0.01,
  min.samples = 4,
  min.prop = 0.4,
  verbose = TRUE,
  BPPARAM = SerialParam(progressbar = verbose)
)
```

### Arguments

sce	a <a href="#">SingleCellExperiment</a> .
assay	character string specifying the assay slot to use as input data. Defaults to the 1st available ( <code>assayNames(x)[1]</code> ).
sample_id	character string specifying which variable to use as sample id
cluster_id	character string specifying which variable to use as cluster id
min.cells	minimum number of observed cells for a sample to be included in the analysis
min.signal	minimum signal value for a gene to be considered expressed in a sample. Proper value for this cutoff depends on the type of signal value
min.samples	minimum number of samples passing cutoffs for cell cluster to be retained
min.prop	minimum proportion of retained samples with non-zero counts for a gene to be
verbose	logical. Should information on progress be reported?
BPPARAM	a <a href="#">BiocParallelParam</a> object specifying how aggregation should be parallelized.

### Details

The dreamlet workflow can also be applied to non-count data. In this case, a signal is averaged across all cells from a given sample and cell type. Here `aggregateNonCountSignal()` performs the roles of `aggregateToPseudoBulk()` followed by `processAssays()` but using non-count data.

For each cell cluster, samples with at least `min.cells` are retained. Only clusters with at least `min.samples` retained samples are kept. Features are retained if they have at least `min.signal` in at least `min.prop` fraction of the samples.

The precision of a measurement is the inverse of its sampling variance. The precision weights are computed as  $1/\text{sem}^2$ , where  $\text{sem} = \text{sd}(\text{signal}) / \text{sqrt}(n)$ , `signal` stores the values averaged across cells, and `n` is the number of cells.

## Value

a `dreamletProcessedData` object

## Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
# using non-count signal
pb.signal <- aggregateNonCountSignal(example_sce,
  assay = "logcounts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(pb.signal, ~group_id)
```

---

`aggregateToPseudoBulk` *Aggregation of single-cell to pseudobulk data*

---

## Description

Aggregation of single-cell to pseudobulk data. Adapted from `muscat::aggregateData` and has same syntax and results. But can be much faster for `SingleCellExperiment` backed by H5AD files using on-disk storage.

## Usage

```
aggregateToPseudoBulk(
  x,
  assay = NULL,
  sample_id = NULL,
  cluster_id = NULL,
  fun = c("sum", "mean", "median", "prop.detected", "num.detected", "sem", "number"),
```

```

scale = FALSE,
verbose = TRUE,
BPPARAM = SerialParam(progressbar = verbose),
checkValues = TRUE,
h5adBlockSizes = 1e+09
)

```

### Arguments

x	a <a href="#">SingleCellExperiment</a> .
assay	character string specifying the assay slot to use as input data. Defaults to the 1st available ( <code>assayNames(x)[1]</code> ).
sample_id	character string specifying which variable to use as sample id
cluster_id	character string specifying which variable to use as cluster id
fun	a character string. Specifies the function to use as summary statistic. Passed to <code>summarizeAssayByGroup2</code> .
scale	logical. Should pseudo-bulks be scaled with the effective library size & multiplied by 1M?
verbose	logical. Should information on progress be reported?
BPPARAM	a <a href="#">BiocParallelParam</a> object specifying how aggregation should be parallelized.
checkValues	logical. Should we check that signal values are positive integers?
h5adBlockSizes	set the automatic block size block size (in bytes) for <code>DelayedArray</code> to read an H5AD file. Larger values use more memory but are faster.

### Details

Adapted from `muscat::aggregateData` and has similar syntax and same results. This is much faster for `SingleCellExperiment` backed by H5AD files using `DelayedMatrix` because this summarizes counts using `DelayedMatrixStats`. But this function also includes optimizations for `sparseMatrix` used by `Seurat` by using `sparseMatrixStats`.

Keeps variables from `colData()` that are constant within `sample_id`. For example, `sex` will be constant for all cells from the same `sample_id`, so it is retained as a variable in the pseudobulk result. But number of expressed genes varies across cells within each `sample_id`, so it is dropped from `colData()`. Instead the mean value per cell type is stored in `metadata(pb)$aggr_means`, and these can be included in regression formulas downstream. In that case, the value of the covariates used per sample will depend on the cell type analyzed.

### Value

a [SingleCellExperiment](#).

Aggregation parameters (`assay`, `by`, `fun`, `scaled`) are stored in `metadata()``$aggr_pars`, where `by = c(cluster_id, sample_id)`. The number of cells that were aggregated are accessible in `int_colData()``$n_cells`.

### Author(s)

Gabriel Hoffman, Helena L Crowell & Mark D Robinson

## References

Crowell, HL, Soneson, C, Germain, P-L, Calini, D, Collin, L, Raposo, C, Malhotra, D & Robinson, MD: Muscat detects subpopulation-specific state transitions from multi-sample multi-condition single-cell transcriptomics data. *Nature Communications* **11(1):6077** (2020). doi: <https://doi.org/10.1038/s41467-020-19894-4>

## Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# pseudobulk data from each cell type
# is stored as its own assay
pb

# aggregate by cluster only,
# collapsing all samples into the same pseudobulk
pb2 <- aggregateToPseudoBulk(example_sce,
  cluster_id = "cluster_id",
  verbose = FALSE)

pb2
#
```

---

aggregateVar

*Per-sample variance of single-cell counts*

---

## Description

Aggregation function for single-cell log-normalized counts to calculate per-sample variance for dreamlet.

## Usage

```
aggregateVar(
  sce,
  assay = NULL,
  cluster_id = NULL,
  sample_id = NULL,
```

```

min.cells = 10,
min.var = 0.01,
min.samples = 4,
min.prop = 0.4,
verbose = TRUE,
BPPARAM = SerialParam(progressbar = verbose)
)

```

### Arguments

sce	a <a href="#">SingleCellExperiment</a> .
assay	character string specifying the assay slot to use as input data. Defaults to the 1st available ( <code>assayNames(x)[1]</code> ).
cluster_id	character string specifying which variable to use as cluster id
sample_id	character string specifying which variable to use as sample id
min.cells	minimum number of observed cells for a sample to be included in the analysis
min.var	minimum variance for a gene to be considered expressed in a sample
min.samples	minimum number of samples passing cutoffs for cell cluster to be retained
min.prop	minimum proportion of retained samples with non-zero counts for a gene to be
verbose	logical. Should information on progress be reported?
BPPARAM	a <a href="#">BiocParallelParam</a> object specifying how aggregation should be parallelized.

### Details

The dreamlet workflow can also be applied to model gene expression variance. In this case, a per-sample per-gene variance is calculated across all cells from a given sample and cell type. Here `aggregateVar()` performs the roles of `aggregateToPseudoBulk()` followed by `processAssays()` but using log-normalized count data.

For each cell cluster, samples with at least `min.cells` are retained. Only clusters with at least `min.samples` retained samples are kept. Features are retained if they have at least `min.prop` fraction of the samples.

The precision of a measurement is the inverse of its sampling variance. The precision weights are computed as  $1/\text{sem}^2$ , where  $\text{sem} = \text{sd} / \sqrt{n}$  and  $n$  is the number of cells.

### Value

a `dreamletProcessedData` object

### Examples

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

# Compute variance for each sample and cell cluster

```



```
pbVar <- aggregateVar(example_sce,  
  assay = "counts",  
  cluster_id = "cluster_id",  
  sample_id = "sample_id",  
  verbose = FALSE  
)
```

---

as.dreamletResult      *Convert list of regression fits to dreamletResult*

---

### Description

Convert list of regression fits to dreamletResult for downstream analysis

### Usage

```
as.dreamletResult(fitList, df_details = NULL)
```

### Arguments

fitList	list of regression fit with dream()
df_details	data.frame storing assay details

### Details

Useful for combining multiple runs of dreamletCompareClusters() into a single dreamletResult for downstream analysis

### Value

object of class dreamletResult

### Examples

```
library(muscat)  
library(SingleCellExperiment)  
  
data(example_sce)  
  
# create pseudobulk for each sample and cell cluster  
pb <- aggregateToPseudoBulk(example_sce,  
  assay = "counts",  
  cluster_id = "cluster_id",  
  sample_id = "sample_id",  
  verbose = FALSE  
)  
  
# first comparison  
ct.pairs <- c("B cells", "CD14+ Monocytes")
```

```

fit <- dreamletCompareClusters(pb, ct.pairs, method = "fixed")

# second comparison
ct.pairs2 <- c("B cells", "CD8 T cells")
fit2 <- dreamletCompareClusters(pb, ct.pairs2, method = "fixed")

# Make a list storing each result with a meaningful name
fitList <- list()

id <- paste0("[", ct.pairs[1], "]_vs_[", ct.pairs[2], "]")
fitList[[id]] <- fit

id <- paste0("[", ct.pairs2[1], "]_vs_[", ct.pairs2[2], "]")
fitList[[id]] <- fit2

# create a dreamletResult form this list
res.compare <- as.dreamletResult(fitList)
res.compare

```

---

assay,dreamletResult,ANY-method

*Get assay*

---

### Description

Get assay

Get assay

Get assays by name

### Usage

```
## S4 method for signature 'dreamletResult,ANY'
assay(x, i, withDimnames = TRUE, ...)
```

```
## S4 method for signature 'dreamletProcessedData,ANY'
assay(x, i, withDimnames = TRUE, ...)
```

```
## S4 method for signature 'vpDF,ANY'
assay(x, i, withDimnames = TRUE, ...)
```

### Arguments

x	vpDF object
i	number indicating index, or string indicating assay
withDimnames	not used
...	other arguments

**Value**

return ith assay

---

assayNames,dreamletResult-method  
*Get assayNames*

---

**Description**

Get assayNames

Get assayNames

Get assayNames

**Usage**

```
## S4 method for signature 'dreamletResult'
assayNames(x, ...)
```

```
## S4 method for signature 'dreamletProcessedData'
assayNames(x, ...)
```

```
## S4 method for signature 'vpDF'
assayNames(x, ...)
```

**Arguments**

x	vpDF object
...	additional arguments

**Value**

array of assay names

---

buildClusterTreeFromPB  
*Hierarchical clustering on cell types from pseudobulk*

---

**Description**

Perform hierarchical clustering on cell types from pseudobulk by aggregating read counts from each cell type.

**Usage**

```
buildClusterTreeFromPB(
  pb,
  method = c("complete", "ward.D", "single", "average", "mcquitty", "median", "centroid",
            "ward.D2"),
  dist.method = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
  assays = assayNames(pb)
)
```

**Arguments**

pb	SingleCellObject storing pseudobulk for each cell type in in assay() field
method	clustering method for hclust()
dist.method	distance metric
assays	which assays to include

**Value**

hierarchical clustering object of class hclust

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# Hierarchical clustering of cell types
hcl <- buildClusterTreeFromPB(pb)

plot(hcl)
```

---

cellCounts

*Extract cell counts*


---

**Description**

Extract matrix of cell counts from SingleCellExperiment

**Usage**

```
cellCounts(x)
```

**Arguments**

```
x                a SingleCellExperiment
```

**Value**

matrix of cell counts with samples as rows and cell types as columns

**See Also**

```
computeCellCounts()
```

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# get matrix of cell counts for each sample
cellCounts(pb)
```

---

cellSpecificityValues-class

*Class cellSpecificityValues*

---

**Description**

Class cellSpecificityValues cell type specificity values for each gene and cell type

**Value**

none

---

cellTypeSpecificity *Get cell type specificity of gene expression*

---

**Description**

For each gene, compute fraction of overall expression attributable to each cell type

**Usage**

```
cellTypeSpecificity(pb, ...)
```

**Arguments**

pb	SingleCellExperiment of pseudobulk data where easy assay is a cell type.
...	other arguments passed to edgeR::calcNormFactors()

**Details**

Sum counts for each cell type, and compute the fraction of counts-per-million attributable to each cell type for each gene

**Value**

matrix of the fraction of expression attributable to each cell type for each gene.

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# Compute cell type specificity of each gene
df <- cellTypeSpecificity(pb)

# Violin plot of specificity scores for each cell type
# Dashed line indicates genes that are equally expressed
# across all cell types. For K cell types, this is 1/K
plotViolin(df)

# Compute the maximum specificity score for each gene
```

```
scoreMax <- apply(df, 1, max)
head(scoreMax)

# For each cell type, get most specific gene
genes <- rownames(df)[apply(df, 2, which.max)]

# Barplot of 5 genes
plotPercentBars(df, genes = genes)

# heatmap of 5 genes that are most cell type specific
dreamlet::plotHeatmap(df, genes = genes)
```

---

checkFormula	<i>Check variables in a formula</i>
--------------	-------------------------------------

---

## Description

Check that variables in formula are present in the data

## Usage

```
checkFormula(formula, data)
```

## Arguments

formula	formula of variables to check
data	data.frame storing variables in the formula

## Value

If formula is valid, return TRUE. Else throw error

## Examples

```
# Valid formula
dreamlet:::checkFormula(~speed, cars)

# Not valid formula
# dreamlet:::checkFormula( ~ speed + a, cars)
```

---

`coefNames`*Get coefficient names*

---

**Description**

Get coefficient names

**Usage**

```
coefNames(obj)

## S4 method for signature 'dreamletResult'
coefNames(obj)
```

**Arguments**

`obj`            A `dreamletResult` object

**Value**

array storing names of coefficients

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# show coefficients estimated for each cell type
coefNames(res.dl)
```



---

colData,dreamletProcessedData-method  
*Extract colData from dreamletProcessedData*

---

**Description**

Extract colData from dreamletProcessedData

**Usage**

```
## S4 method for signature 'dreamletProcessedData'  
colData(x, ...)
```

**Arguments**

x	A dreamletProcessedData object
...	other arguments

**Value**

object from colData field

---

colData<- ,dreamletProcessedData,ANY-method  
*Set colData*

---

**Description**

Set colData of dreamletProcessedData, and check for same dimensions and rownames

**Usage**

```
## S4 replacement method for signature 'dreamletProcessedData,ANY'  
colData(x, ...) <- value
```

**Arguments**

x	dreamletProcessedData object
...	other arguments
value	data.frame or object that can be coerced to it

**Value**

none

---

 compositePosteriorTest

*Perform composite test on results from mashr*


---

## Description

The posterior probabilities for all genes and conditions is obtained as 1-1FSR. Let prob be an array storing results for one gene. The probability that `_no_` conditions in the exclusion set are non-zero is `prod(1 - prob[exclude])`. The probability that `_all_` conditions in the inclusion set are non-zero is `prod(prob[include])`. The probability that `_at least one_` condition in the inclusion set is non-zero is `1 - prod(1 - prob[include])`. The composite test is the product of the probabilities computed from the inclusion and exclusion sets.

## Usage

```
compositePosteriorTest(
  x,
  include,
  exclude = NULL,
  test = c("at least 1", "all")
)
```

## Arguments

<code>x</code>	"dreamlet_mash_result" from <code>run_mash()</code>
<code>include</code>	array of conditions in the inclusion set
<code>exclude</code>	array of conditions in the exclusion set. Defaults to NULL for no exclusion
<code>test</code>	evaluate the posterior probability of a non-zero effect in "at least 1" or "all" conditions

## Details

Perform composite test evaluating the specificity of an effect. Evaluate the posterior probability that an a non-zero effect present in `_all_` or `_at least one_` condition in the inclusion set, but `_no_` conditions\_ in the exclusion set.

## See Also

`run_mash()`

## Examples

```
library(muscat)
library(mashr)
library(SingleCellExperiment)

data(example_sce)
```

```

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce[1:100, ],
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# run MASH model
# This can take 10s of minutes on real data
# This small datasets should take ~30s
res_mash <- run_mash(res.dl, "group_idstim")

# Composite test based on posterior probabilities
# to identify effect present in *at least 1* monocyte type
# and *NO* T-cell type.
include <- c("CD14+ Monocytes", "FCGR3A+ Monocytes")
exclude <- c("CD4 T cells", "CD8 T cells")

# Perform composite test
prob <- compositePosteriorTest(res_mash, include, exclude)

# examine the lFSR for top gene
get_lfsr(res_mash$model)[which.max(prob), , drop = FALSE]

# Test if *all* cell types have non-zero effect
prob <- compositePosteriorTest(res_mash, assayNames(res.dl))

```

---

computeCellCounts      *Get cell counts with metadata*

---

### Description

Get cell counts with metadata for each sample

### Usage

```
computeCellCounts(sce, annotation, sampleIDs)
```

### Arguments

sce                      SingleCellExperiment

annotation      string indicating column in colData(sce) storing cell type annotations  
 sampleIDs      string indicating column in colData(sce) storing sample identifiers

**Value**

matrix storing cell counts

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

counts <- computeCellCounts(example_sce, "cluster_id", "sample_id")

counts[1:4, 1:4]
```

---

computeLogCPM	<i>Compute log normalized counts</i>
---------------	--------------------------------------

---

**Description**

Compute normalized counts as log<sub>2</sub> counts per million

**Usage**

```
computeLogCPM(
  sce,
  lib.size = colSums2(counts(sce)),
  prior.count = 2,
  scaledByLib = FALSE
)
```

**Arguments**

sce              SingleCellExperiment with counts stored as counts(sce)  
 lib.size        library size for each cell  
 prior.count    average count to be added to each observation to avoid taking log of zero  
 scaledByLib    if TRUE, scale pseudocount by lib.size. Else do standard constant pseudocount addition

**Details**

This function gives same result as edgeR: :cpm(counts(sce), log=TRUE)

**Value**

matrix of log CPM values

**See Also**

also edgeR: : cpm()

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

logcounts(example_sce) <- computeLogCPM(example_sce)
```

---

`computeNormCounts`      *Compute normalized counts*

---

**Description**

Compute normalized counts as counts per million

**Usage**

```
computeNormCounts(sce)
```

**Arguments**

sce                      SingleCellExperiment with counts stored as counts(sce)

**Details**

This function gives same result as edgeR: : cpm(counts(sce), log=FALSE)

**Value**

matrix of CPM values

**See Also**

also edgeR: : cpm()

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

normcounts(example_sce) <- computeNormCounts(example_sce)
```

---

details

*Extract details from dreamletProcessedData*

---

**Description**

Extract details from dreamletProcessedData

**Usage**

```
details(object)

## S4 method for signature 'dreamletProcessedData'
details(object)

## S4 method for signature 'dreamletResult'
details(object)

## S4 method for signature 'vpDF'
details(object)
```

**Arguments**

object            A dreamletProcessedData object

**Value**

Extract detailed information from some classes

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
```

```

    verbose = FALSE
  )

  # voom-style normalization
  res.proc <- processAssays(pb, ~group_id)

  # For each cell type, number of samples retained,
  # and variables retained
  details(res.proc)

```

---

diffVar,dreamletResult-method  
*Test differential variance*

---

## Description

Test the association between a covariate of interest and the response's deviation from expectation.

## Usage

```

## S4 method for signature 'dreamletResult'
diffVar(
  fit,
  method = c("AD", "SQ"),
  scale = c("leverage", "none"),
  BPPARAM = SerialParam(),
  ...
)

```

## Arguments

fit	model fit from dream()
method	transform the residuals using absolute deviation ("AD") or squared deviation ("SQ").
scale	scale each observation by "leverage", or no scaling ("none")
BPPARAM	parameters for parallel evaluation
...	other parameters passed to dream()

## Details

This method performs a test of differential variance between two subsets of the data, in a way that generalizes to multiple categories, continuous variables and metrics of spread beyond variance. For the two category test, this method is similar to Levene's test. This model was adapted from Phipson, et al (2014), extended to linear mixed models, and adapted to be compatible with `variancePartition::dream()` and `dreamlet::dreamlet()`.

This method is composed of multiple steps where 1) a typical linear (mixed) model is fit with `dreamlet()`, 2) residuals are computed and transformed based on an absolute value or squaring transform, 3) a second regression is performed with `dreamlet()` to test if a variable is associated with increased deviation from expectation. Both regression take advantage of the `dreamlet()` linear (mixed) modelling framework followed by empirical Bayes shrinkage that extends the `limma::voom()` framework.

Note that `diffVar()` takes the results of the first regression as a parameter to use as a starting point.

## References

Phipson B, Oshlack A (2014). “DiffVar: a new method for detecting differential variability with application to methylation in cancer and aging.” *Genome biology*, **15**(9), 1–16.

## See Also

```
variancePartition::diffVar()
variancePartition::diffVar(), missMethyl::diffVar()
```

## Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# Differential variance analysis
# result is a dreamlet fit
res.dvar <- diffVar(res.dl)

# Examine results
res.dvar

# Examine details for each assay
details(res.dvar)

# show coefficients estimated for each cell type
```



```

coefNames(res.dvar)

# extract results using limma-style syntax
# combines all cell types together
# adj.P.Val gives study-wide FDR
topTable(res.dvar, coef = "group_idstim", number = 3)

# Plot top hit to see differential variance
# Note that this is a toy example with only 4 samples
cellType <- "CD4 T cells"
gene <- "DYNLRB1"

y <- res.proc[[cellType]]$E[gene, ]
x <- colData(res.proc)$group_id

boxplot(y ~ x,
        xlab = "Stimulation status",
        ylab = "Gene expression",
        main = paste(cellType, gene)
)
#

```

---

dreamlet

*Differential expression for each assay*


---

## Description

Perform differential expression for each assay using linear (mixed) models

## Usage

```

dreamlet(
  x,
  formula,
  data = colData(x),
  assays = assayNames(x),
  contrasts = NULL,
  min.cells = 10,
  robust = FALSE,
  quiet = FALSE,
  BPPARAM = SerialParam(),
  use.eBayes = TRUE,
  ...
)

## S4 method for signature 'dreamletProcessedData'
dreamlet(
  x,
  formula,

```

```

data = colData(x),
assays = assayNames(x),
contrasts = NULL,
min.cells = 10,
robust = FALSE,
quiet = FALSE,
BPPARAM = SerialParam(),
use.eBayes = TRUE,
...
)

```

### Arguments

x	SingleCellExperiment or dreamletProcessedData object
formula	regression formula for differential expression analysis
data	metadata used in regression formula
assays	array of assay names to include in analysis. Defaults to assayNames(x)
contrasts	character vector specifying contrasts specifying linear combinations of fixed effects to test. This is fed into makeContrastsDream( formula, data, contrasts=contrasts)
min.cells	minimum number of observed cells for a sample to be included in the analysis
robust	logical, use eBayes method that is robust to outlier genes
quiet	show messages
BPPARAM	parameters for parallel evaluation
use.eBayes	should eBayes be used on result? (default: TRUE)
...	other arguments passed to dream

### Details

Fit linear (mixed) model on each cell type separately. For advanced use of contrasts see `variancePartition::makeContrasts` and vignette <https://gabrielhoffman.github.io/variancePartition/articles/dream.html#advanced-hypothesis-testing-1>.

### Value

Object of class `dreamletResult` storing results for each cell type

### See Also

`variancePartition::dream()`, `variancePartition::makeContrastsDream()`

### Examples

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

```

```
# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# Examine results
res.dl

# Examine details for each assay
details(res.dl)

# show coefficients estimated for each cell type
coefNames(res.dl)

# extract results using limma-style syntax
# combines all cell types together
# adj.P.Val gives study-wide FDR
topTable(res.dl, coef = "group_idstim", number = 3)
```

---

dreamletCompareClusters

*Differential expression between pair of assays*

---

## Description

Perform differential expression between a pair of assays using linear (mixed) models

## Usage

```
dreamletCompareClusters(
  pb,
  assays,
  method = c("fixed", "random", "none"),
  formula = ~0,
  collapse = TRUE,
  min.cells = 10,
  min.count = 10,
  min.samples = 4,
```

```

isCounts = TRUE,
normalize.method = "TMM",
robust = FALSE,
quiet = FALSE,
contrasts = c(compare = paste("cellClustertest - cellClusterbaseline")),
BPPARAM = SerialParam(),
errorsAsWarnings = FALSE,
...
)

```

### Arguments

pb	pseudobulk data as SingleCellExperiment object
assays	array of two entries specifying assays (i.e. cell clusters) to compare, or a list of two sets of assays.
method	account for repeated measures from donors using a "random" effect, a "fixed" effect, or "none"
formula	covariates to include in the analysis.
collapse	if TRUE (default), combine all cell clusters within the test set, and separately the baseline set. If FALSE, estimate coefficient for each cell cluster and then identify differential expression using linear contrasts with <code>variancePartition::makeContrastsDream()</code>
min.cells	minimum number of observed cells for a sample to be included in the analysis
min.count	minimum number of reads for a gene to be consider expressed in a sample. Passed to <code>edgeR::filterByExpr</code>
min.samples	minimum number of samples passing cutoffs for cell cluster to be retained
isCounts	logical, indicating if data is raw counts
normalize.method	normalization method to be used by <code>calcNormFactors</code>
robust	logical, use eBayes method that is robust to outlier genes
quiet	show messages
contrasts	cell type is encoded in variable <code>cellCluster</code> with levels <code>test</code> and <code>baseline</code> . <code>contrasts</code> specifies contrasts passed to <code>variancePartition::makeContrastsDream()</code> . Note, advanced users only.
BPPARAM	parameters for parallel evaluation
errorsAsWarnings	if TRUE, convert error to a warning and return NULL
...	other arguments passed to <code>dream</code>

### Details

Analyze pseudobulk data to identify differential gene expression between two cell clusters or sets of clusters while modeling the cross-donor expression variation and other aspects of the study design. `dreamletCompareClusters()` is useful for finding genes that are differentially expressed between cell clusters and estimating their fold change. However, the p-values and number of differentially expressed genes are problematic for two reasons, so users must be careful not to overinterpret them:

1. Cell clusters are typically identified with the same gene expression data used for this differential expression analysis between clusters. The same data is used both for discovery and testing, and this means that the p-values from the differential expression analysis will not be uniform under the null. This will produce a lot of findings with small p-values even in the absence of true biological differences.
2. The dreamlet package is designed for large datasets with many subjects. The sample sizes from cohort studies are an order of magnitude larger than typical single cell studies. This means that these analyses have huge power to detect even subtle difference in expression between cell clusters. While cluster-specific marker genes are often discovered from an handful of samples, the dreamlet package is applicable to 100s or 1000s of subjects.

method indicates the regression method used to test differential expression between sets of cell clusters. Since the same biosample will usually be represented in both sets of cell clusters, method determines how the paired design is modeled. For method = "mixed", the sample is modeled as a random effect:  $\sim (1|Sample) + \dots$ . For method = "fixed", the sample is modeled as a fixed effect:  $\sim Sample + \dots$ . For method = "none", the pairing is ignored.

When collapse=TRUE (default) combine all cell clusters within the test set, and separately the baseline set, and estimate a coefficient indicating the differential expression between sets for a given gene. If collapse=FALSE, estimate a coefficient for each cell type and then identify differential expression using linear contrasts with `variancePartition::makeContrastsDream()`.

### Value

Object of class `dreamletResult` storing results for each comparison

### Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# Evaluate the specificity of each gene for each cluster
df_cts <- cellTypeSpecificity(pb)

# compare first two assays (i.e. cell types)
ct.pairs <- c("B cells", "CD14+ Monocytes")

# run comparison
# use method = 'fixed' here since it is faster
fit <- dreamletCompareClusters(pb, ct.pairs, method = "fixed")

# Extract top 10 differentially expressed genes
```

```

# The coefficient 'compare' is the value logFC between test and baseline:
# compare = cellClustertest - cellClusterbaseline
res <- topTable(fit, coef = "compare", number = 10)

# genes with highest logFC are most highly expressed in
# B cells compared to CD14+ Monocytes
head(res)

dreamlet::plotHeatmap(df_cts, genes = rownames(res)[1:5])

# compare B cells versus the rest of the cell types
# 'rest' is a keyword indicating all other assays
fit <- dreamletCompareClusters(pb, c("B cells", "rest"), method = "fixed")

res <- topTable(fit, coef = "compare", number = 10)

# genes with highest logFC are most highly expressed in
# B cells compared to all others
head(res)

# Get genes upregulated in B cells
idx <- with(res, which(logFC > 0))[1:5]
dreamlet::plotHeatmap(df_cts, genes = rownames(res)[idx])

lst <- list(
  test = c("CD14+ Monocytes", "FCGR3A+ Monocytes"),
  baseline = c("CD4 T cells", "CD8 T cells")
)

# compare 2 monocyte clusters to two T cell clusters
fit <- dreamletCompareClusters(pb, lst, method = "fixed")

res <- topTable(fit, coef = "compare", number = 10)

# genes with highest logFC are most highly expressed in
# monocytes compared to T cells
head(res)

# Get genes upregulated in monocytes
idx <- with(res, which(logFC > 0))[1:5]
dreamlet::plotHeatmap(df_cts, genes = rownames(res)[idx])

```

---

dreamletProcessedData-class

*Class dreamletProcessedData*

---

## Description

Class dreamletProcessedData

**Value**

none

none

---

*dreamletResult-class*    *Class dreamletResult*

---

**Description**

Class `dreamletResult` stores results produced by `dreamlet()` to give a standard interface for downstream analysis

Class `dreamletResult` stores results produced by `dreamlet()` to give a standard interface for downstream analysis

**Value**

none

none

---

*dreamlet\_mash\_result-class*  
*Class dreamlet\_mash\_result*

---

**Description**

Class `dreamlet_mash_result`

**Value**

`dreamlet_mash_result` class

---

dropRedundantTerms      *Drop redundant terms from the model*

---

**Description**

Detect co-linear fixed effects and drop the last one

**Usage**

```
dropRedundantTerms(formula, data, tol = 0.001)
```

**Arguments**

formula	original formula
data	data.frame
tol	tolerance to test difference of correlation from 1 or -1

**Value**

a formula, possibly with terms omitted.

**Examples**

```
# Valid formula
dropRedundantTerms(~ group + extra, sleep)
```

---

equalFormulas      *Check if two formulas are equal*

---

**Description**

Check if two formulas are equal by evaluating the formulas and extracting terms

**Usage**

```
equalFormulas(formula1, formula2)
```

**Arguments**

formula1	first formula
formula2	second formula

**Value**

boolean value indicating if formulas are equivalent



**Examples**

```
# These formulas are equivalent
formula1 <- ~ Size + 1
formula2 <- ~ 1 + Size

dreamlet:::equalFormulas(formula1, formula2)
```

---

extractData	<i>Extract normalized expression and colData</i>
-------------	--

---

**Description**

Extract normalized expression and colData

Extract normalized (i.e. log2 CPM) expression and colData from dreamletProcessedData

**Usage**

```
extractData(x, assay, cols = colnames(colData(x)), genes = rownames(x))
```

```
## S4 method for signature 'dreamletProcessedData,character'
```

```
extractData(
  x,
  assay,
  cols = colnames(colData(x)),
  genes = rownames(assay(x, assay))
)
```

**Arguments**

x	dreamletProcessedData object
assay	assay to extract
cols	columns in colData(x) to extract. defaults to all columns as colnames(colData(x))
genes	genes to extract from assay(x, assay)\$E. defaults to all genes as rownames(x)

**Value**

data.frame or DataFrame of merged expression and colData

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
```

```

pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Extract all:
# Extract tibble of colData merged with expression.
# variables and genes are stored as columns, samples as rows
df_merge <- extractData(res.proc, "B cells")

# first few columns
df_merge[, 1:6]

# Extract subset:
df_merge <- extractData(res.proc, "B cells", cols = "group_id", genes = c("SSU72", "U2AF1"))

df_merge

# Boxplot of expression
boxplot(SSU72 ~ group_id, df_merge)
#

```

---

fitVarPart

*Variance Partition analysis for each assay*


---

## Description

Perform Variance Partition analysis for each assay

## Usage

```

fitVarPart(
  x,
  formula,
  data = colData(x),
  assays = assayNames(x),
  quiet = FALSE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'dreamletProcessedData'
fitVarPart(
  x,

```

```

    formula,
    data = colData(x),
    assays = assayNames(x),
    quiet = FALSE,
    BPPARAM = SerialParam(),
    ...
  )

```

### Arguments

x	SingleCellExperiment or dreamletProcessedData object
formula	regression formula for differential expression analysis
data	metadata used in regression formula
assays	array of assay names to include in analysis. Defaults to assayNames(x)
quiet	show messages
BPPARAM	parameters for parallel evaluation
...	other arguments passed to dream

### Value

Object of class vpDF inheriting from DataFrame storing the variance fractions for each gene and cell type.

### Examples

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# variance partitioning analysis
vp <- fitVarPart(res.proc, ~group_id)

# Show variance fractions at the gene-level for each cell type
genes <- vp$gene[2:4]
plotPercentBars(vp[vp$gene %in% genes, ])

# Summarize variance fractions genome-wide for each cell type

```

```
plotVarPart(vp)
```

---

```
getExprGeneNames      Get list of expressed genes for each assay
```

---

### Description

Get list of expressed genes for each assay using same filters as `processAssays()`.

### Usage

```
getExprGeneNames(
  sceObj,
  assays = assayNames(sceObj),
  min.cells = 5,
  min.count = 5,
  min.samples = 4,
  min.prop = 0.4,
  min.total.count = 15,
  normalize.method = "TMM"
)
```

### Arguments

<code>sceObj</code>	SingleCellExperiment object
<code>assays</code>	array of assay names to include in analysis. Defaults to <code>assayNames(sceObj)</code>
<code>min.cells</code>	minimum number of observed cells for a sample to be included in the analysis
<code>min.count</code>	minimum number of reads for a gene to be considered expressed in a sample. Passed to <code>edgeR::filterByExpr</code>
<code>min.samples</code>	minimum number of samples passing cutoffs for cell cluster to be retained
<code>min.prop</code>	minimum proportion of retained samples with non-zero counts for a gene to be retained
<code>min.total.count</code>	minimum total count required per gene for inclusion
<code>normalize.method</code>	normalization method to be used by <code>calcNormFactors</code>

### Examples

```
library(muscat)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
```

```

    assay = "counts",
    sample_id = "sample_id",
    cluster_id = "cluster_id",
    verbose = FALSE
  )

  # Gene expressed genes for each cell type
  geneList = getExprGeneNames(pb)

  # Create precision weights for pseudobulk
  # By default, weights are set to cell count,
  # which is the default in processAssays()
  # even when no weights are specified
  weightsList <- pbWeights(example_sce,
    sample_id = "sample_id",
    cluster_id = "cluster_id",
    geneList = geneList
  )

  # voom-style normalization using initial weights
  res.proc <- processAssays(pb, ~group_id, weightsList = weightsList)

```

---

getTreat,dreamletResult-method

*Test if coefficient is different from a specified value*

---

## Description

Test if coefficient is different from a specified value

## Usage

```

## S4 method for signature 'dreamletResult'
getTreat(fit, lfc = log2(1.2), coef = NULL, number = 10, sort.by = "p")

```

## Arguments

fit	dreamletResult object
lfc	a minimum log2-fold-change below which changes not considered scientifically meaningful
coef	which coefficient to test
number	number of genes to return
sort.by	column to sort by

## Value

DataFrame storing hypothesis test for each gene and cell type

**See Also**

```
limma::topTreat(), variancePartition::getTreat()
```

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# show coefficients estimated for each cell type
coefNames(res.dl)

# extract results using limma-style syntax
# combines all cell types together
# adj.P.Val gives study-wide FDR
getTreat(res.dl, coef = "group_idstim", number = 3)
```

---

metadata,dreamletProcessedData-method

*Extract metadata from dreamletProcessedData*

---

**Description**

Extract metadata from dreamletProcessedData

**Usage**

```
## S4 method for signature 'dreamletProcessedData'
metadata(x)
```

**Arguments**

x                    A dreamletProcessedData object

**Value**

object from metadata field

---

meta_analysis	<i>Meta-analysis across multiple studies</i>
---------------	--

---

**Description**

Meta-analysis across multiple studies

**Usage**

```
meta_analysis(  
  x,  
  method = "FE",  
  group = c("ID", "assay"),  
  control = list(maxiter = 2000)  
)
```

**Arguments**

x	data.frame rbind'ing results across genes, cell types and datasets
method	meta-analysis method. Values are fed into metafor::rma(), except for 'RE2C' which calls remaCor::RE2C().
group	columns in x to group by. For results from dreamlet::topTable(), results are aggregated by gene and cell type (i.e. 'ID' and 'assay'). If x is not from this function, this argument allows the function to group results properly
control	passed to rma(.., control)

**Details**

- 'FE': fixed effects meta-analysis
- 'REML': random effects meta-analysis
- 'RE2C': joint testing of fixed and random effects

**Examples**

```
library(dreamlet)  
library(muscat)  
  
data(example_sce)  
  
# create pseudobulk for each sample and cell cluster  
pb <- aggregateToPseudoBulk(example_sce,  
  assay = "counts",  
  cluster_id = "cluster_id",
```

```

    sample_id = "sample_id",
    verbose = FALSE
  )

  # voom-style normalization
  # just 'CD14+ Monocytes' for speed
  res.proc <- processAssays(pb, ~group_id, assays = "CD14+ Monocytes")

  # dreamlet
  res.dl <- dreamlet(res.proc, ~group_id)

  tab1 <- topTable(res.dl, coef = "group_idstim", number = Inf)
  tab1$Dataset <- "1"

  # Results from a second cohort
  # Here, just a copy of the same results for simplicity
  tab2 <- tab1
  tab2$Dataset <- "2"

  # rbind
  tab_combined <- rbind(tab1, tab2)

  # Perform fixed effects meta-analysis
  res <- meta_analysis(tab_combined, method = "FE")

  res[1:3, ]
  #

```

---

outlier

*Multivariate outlier detection*


---

## Description

Detect multivariate outliers using Mahalanobis distance using mean and covariance estimated either with standard or robust methods.

## Usage

```
outlier(data, robust = FALSE, ...)
```

## Arguments

data	matrix of data
robust	use robust covariance method, defaults to FALSE
...	arguments passed to MASS::cov.rob()

## Details

The distance follow a chisq distribution under the null with standard method for mean and covariance. It is approximate if the robust method is used. So use `qchisq(p = 0.999, df = k)` to get cutoff to keep 99.9% of samples under the null for data with  $k=2$  columns.



**Value**

data.frame storing chisq and z-score for each entry indicating deviation from the mean. The z-score is computed by evaluating the p-value of chisq statistic and converting it into a z-score

**Examples**

```
data <- matrix(rnorm(200), 100, 2)

res <- outlier(data)

res[1:4,]
```

---

outlierByAssay	<i>Outlier analysis for each assay</i>
----------------	--

---

**Description**

Compute outlier score for each sample in each assay using outlier() run on the top principal components. Mahalanobis distance is used for outlier detect and multivariate normal assumption is used to compute p-values

**Usage**

```
outlierByAssay(object, assays = names(object), nPC = 2, robust = FALSE, ...)
```

**Arguments**

object	dreamletProcessedData from processAssays()
assays	assays / cell types to analyze
nPC	number of PCs to uses for outlier score with outlier()
robust	use robust covariance method, defaults to FALSE
...	arguments passed to MASS::cov.rob()

**Value**

- ID:sample identifier
- assay:specify assay
- PCs:principal components
- chisq:mahalanobis distance that is distributed as chisq(k) k = nPC if the data is multivariate gaussian
- z:z-score corresponding to the chisq distance

**See Also**

outlier()

**Examples**

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Compute PCs and outlier scores
outlierByAssay( res.proc, c("B cells", "CD14+ Monocytes"))

```

---

pbWeights

*Compute precision weights for pseudobulk*


---

**Description**

Compute precision weights for pseudobulk using the delta method to approximate the variance of the log2 counts per million considering variation in the number of cells and gene expression variance across cells within each sample. By default, used number of cells; if specified use delta method. Note that processAssays() uses number of cells as weights when no weights are specified

**Usage**

```

pbWeights(
  sce,
  sample_id,
  cluster_id,
  geneList = NULL,
  method = c("delta", "ncells"),
  shrink = TRUE,
  prior.count = 0.5,
  maxRatio = 20,
  h5adBlockSizes = 1e+09,
  details = FALSE,
  verbose = TRUE
)

```

**Arguments**

sce	SingleCellExperiment of where counts(sce) stores the raw count data at the single cell level
sample_id	character string specifying which variable to use as sample id
cluster_id	character string specifying which variable to use as cluster id
geneList	list of genes to be included for each cell type
method	select method to compute precision weights. 'delta' use the delta method based on normal approximation to a negative binomial model, slower but can increase power. 'ncells' use the number of cells, this is faster; Subsequent arguments are ignored. Included for testing
shrink	Defaults to TRUE. Use empirical Bayes variance shrinkage from limma to shrink estimates of expression variance across cells within each sample
prior.count	Defaults to 0.5. Count added to each observation at the pseudobulk level. This is scaled but the number of cells before added to the cell level
maxRatio	When computing precision as the reciprocal of variance $1/(x+\tau)$ select tau to have a maximum ratio between the largest and smallest precision
h5adBlockSizes	set the automatic block size block size (in bytes) for DelayedArray to read an H5AD file. Larger values use more memory but are faster.
details	include data.frame of cell-level statistics as attr(., "details")
verbose	Show messages, defaults to TRUE

**Examples**

```
library(muscat)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  sample_id = "sample_id",
  cluster_id = "cluster_id",
  verbose = FALSE
)

# Gene expressed genes for each cell type
geneList = getExprGeneNames(pb)

# Create precision weights for pseudobulk
# By default, weights are set to cell count,
# which is the default in processAssays()
# even when no weights are specified
weightsList <- pbWeights(example_sce,
  sample_id = "sample_id",
  cluster_id = "cluster_id",
  geneList = geneList
)
```

```
# voom-style normalization using initial weights
res.proc <- processAssays(pb, ~group_id, weightsList = weightsList)
```

---

plotBeeswarm	<i>Beeswarm plot of effect sizes for each assay</i>
--------------	---

---

### Description

Beeswarm plot of effect sizes for each assay, colored by sign and FDR

### Usage

```
plotBeeswarm(res.dl, coef, fdr.range = 4, assays = assayNames(res.dl))
```

### Arguments

res.dl	dreamletResult object from dreamlet()
coef	coefficient name fed to topTable()
fdr.range	range for coloring FDR
assays	which assays to plot

### Value

ggplot2 of logFC by assay

### Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# Beeswarm plot of effect sizes for each assay,
```

```
# colored by sign and FDR
plotBeeswarm(res.dl, "group_idstim")
```

---

plotCellComposition *Bar plot of cell compositions*

---

## Description

Bar plot of cell compositions

## Usage

```
plotCellComposition(obj, col, width = NULL)

## S4 method for signature 'SingleCellExperiment'
plotCellComposition(obj, col, width = NULL)

## S4 method for signature 'matrix'
plotCellComposition(obj, col, width = NULL)

## S4 method for signature 'data.frame'
plotCellComposition(obj, col, width = NULL)
```

## Arguments

obj	matrix of [cells] x [samples] or SingleCellExperiment from aggregateToPseudoBulk
col	array of colors. If missing, use default colors. If names(col) is the same as arrayNames(obj), then colors will be assigned by assay name#'
width	specify width of bars

## Value

Barplot showing cell fractions

## Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE)
```

```

)

# show cell composition bar plots
plotCellComposition(pb)

# extract cell counts
df_cellCounts <- cellCounts(pb)

# show cell composition bar plots
plotCellComposition(df_cellCounts)

```

---

plotForest

*Forest plot*


---

### Description

Forest plot

### Usage

```

plotForest(x, gene, coef, ...)

## S4 method for signature 'dreamletResult'
plotForest(x, gene, coef, assays = names(x), ylim = NULL)

## S4 method for signature 'dreamlet_mash_result'
plotForest(x, gene, coef, assays = colnames(x$logFC.original), ylim = NULL)

```

### Arguments

x	result from dreamlet
gene	gene to show results for
coef	coefficient to test with topTable
...	other arguments
assays	array of assays to plot
ylim	limits for the y axis

### Value

Plot showing effect sizes

**Examples**

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# show coefficients estimated for each cell type
coefNames(res.dl)

# Show estimated log fold change with in each cell type
plotForest(res.dl, gene = "ISG20", coef = "group_idstim")

```

---

plotGeneHeatmap      *Heatmap of genes and assays*

---

**Description**

Heatmap of genes and assays

**Usage**

```

plotGeneHeatmap(
  x,
  coef,
  genes,
  assays = assayNames(x),
  zmax = NULL,
  transpose = FALSE
)

## S4 method for signature 'dreamletResult'
plotGeneHeatmap(
  x,

```

```

  coef,
  genes,
  assays = assayNames(x),
  zmax = NULL,
  transpose = FALSE
)

```

### Arguments

<code>x</code>	A <code>dreamletResult</code> object
<code>coef</code>	column number or column name specifying which coefficient or contrast of the linear model is of interest.
<code>genes</code>	array of genes to include in plot
<code>assays</code>	array of assay names to include in analysis. Defaults to <code>assayNames(x)</code>
<code>zmax</code>	maximum z.std value
<code>transpose</code>	(default: FALSE) Use <code>'coord_flip()'</code> to flip axes

### Value

Heatmap plot for specified genes and assays

Heatmap plot for specified genes and assays

### Examples

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# Heatmap for specified subset of genes
plotGeneHeatmap(res.dl, coef = "group_idstim", genes = rownames(pb)[1:15])

```



---

plotHeatmap	<i>Plot heatmap</i>
-------------	---------------------

---

## Description

Plot heatmap

## Usage

```
plotHeatmap(  
  x,  
  genes = rownames(x),  
  color = "darkblue",  
  assays = colnames(x),  
  useFillScale = TRUE  
)
```

```
## S4 method for signature 'cellSpecificityValues'
```

```
plotHeatmap(  
  x,  
  genes = rownames(x),  
  color = "darkblue",  
  assays = colnames(x),  
  useFillScale = TRUE  
)
```

```
## S4 method for signature 'data.frame'
```

```
plotHeatmap(  
  x,  
  genes = rownames(x),  
  color = "darkblue",  
  assays = colnames(x),  
  useFillScale = TRUE  
)
```

```
## S4 method for signature 'matrix'
```

```
plotHeatmap(  
  x,  
  genes = rownames(x),  
  color = "darkblue",  
  assays = colnames(x),  
  useFillScale = TRUE  
)
```

## Arguments

x                   fractions for each gene

genes	name of genes to plot
color	color of heatmap
assays	array of assays to plot
useFillScale	default TRUE. add scale_fill_gradient() to plot

**Value**

heatmap

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# Compute cell type specificity of each gene
df <- cellTypeSpecificity(pb)

# For each cell type, get most specific gene
genes <- rownames(df)[apply(df, 2, which.max)]

# heatmap of 5 genes that are most cell type specific
dreamlet::plotHeatmap(df, genes = genes)
```

---

plotPCA

*Plot PCA of gene expression for an assay*


---

**Description**

Compute PCA of gene expression for an assay, and plot samples coloring by outlier score

**Usage**

```
## S4 method for signature 'list'
plotPCA(
  object,
  assays = names(object),
  nPC = 2,
  robust = FALSE,
```

```

    ...,
    maxOutlierZ = 20,
    nrow = 2,
    size = 2,
    fdr.cutoff = 0.05
  )

```

### Arguments

object	dreamletProcessedData from processAssays() or a list from residuals()
assays	assays / cell types to analyze
nPC	number of PCs to uses for outlier score with outlier()
robust	use robust covariance method, defaults to FALSE
...	arguments passed to MASS::cov.rob()
maxOutlierZ	cap outlier z-scores at this value for plotting to maintain consistent color scale
nrow	number of rows in plot
size	size passed to geom_point()
fdr.cutoff	FDR cutoff to determine outlier

### See Also

outlierByAssay()

### Examples

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# PCA to identify outliers
# from normalized expression
plotPCA( res.proc, c("B cells", "CD14+ Monocytes"))

# Run on regression residuals
#-----

# Regression analysis

```

```

fit = dreamlet(res.proc, ~ group_id)

# Extract regression residuals
residsObj = residuals(fit)

# PCA on residuals
plotPCA( residsObj, c("B cells", "CD14+ Monocytes"))

```

---

```

plotPercentBars, vpDF-method
Bar plot of variance fractions

```

---

### Description

Bar plot of variance fractions for a subset of genes

### Usage

```

## S4 method for signature 'vpDF'
plotPercentBars(
  x,
  col = c(ggColorHue(ncol(x) - 3), "grey85"),
  genes = unique(x$gene),
  width = NULL,
  ncol = 3,
  ...
)

## S4 method for signature 'cellSpecificityValues'
plotPercentBars(
  x,
  col = ggColorHue(ncol(x)),
  genes = rownames(x),
  width = NULL,
  ...
)

```

### Arguments

x	vpDF object returned by fitVarPart()
col	color of bars for each variable
genes	name of genes to plot
width	specify width of bars
ncol	number of columns in the plot
...	other arguments

**Value**

Bar plot showing variance fractions for each gene

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# variance partitioning analysis
vp <- fitVarPart(res.proc, ~group_id)

# Show variance fractions at the gene-level for each cell type
plotPercentBars(vp, genes = vp$gene[2:4], ncol = 2)
```

---

plotProjection

*Plot 2D projection*

---

**Description**

Plot 2D projection (i.e. UMAP, tSNE) for millions of cells efficiently

**Usage**

```
plotProjection(
  sce,
  type,
  annotation,
  pointsize = 0,
  pixels = c(512, 512),
  legend.position = "none",
  text = TRUE,
  order
)
```

**Arguments**

sce	SingleCellExperiment
type	field in reducedDims(sce) to plot
annotation	column in colData(sce) to annotate each cell
pointsize	Radius of rasterized point. Use 0 for single pixels(fastest).
pixels	Vector with X and Y resolution of the raster, default c(512, 512)
legend.position	legend.position: the position of legends ("none", "left", "right", "bottom", "top", or two-element numeric vector)
text	show annotation as text. Default TRUE
order	specify order of levels for annotation

**Details**

Uses scattermore::geom\_scattermore() to plot millions of points efficiently

**Value**

ggplot2 plot of the projection

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

plotProjection(example_sce, "TSNE", "cluster_id", 1)
```

---

plotVarPart,DataFrame-method

*Violin plot of variance fractions*

---

**Description**

Violin plot of variance fraction for each gene and each variable

**Usage**

```
## S4 method for signature 'DataFrame'
plotVarPart(
  obj,
  col = c(ggColorHue(base::ncol(obj) - 3), "grey85"),
  label.angle = 20,
  main = "",
```

```

  ylab = "",
  convertToPercent = TRUE,
  ncol = 3,
  ...
)

```

### Arguments

obj	varParFrac object returned by fitExtractVarPart or extractVarPart
col	vector of colors
label.angle	angle of labels on x-axis
main	title of plot
ylab	text on y-axis
convertToPercent	multiply fractions by 100 to convert to percent values
ncol	number of columns in the plot
...	additional arguments

### Value

Violin plot showing variance fractions

### Examples

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# variance partitioning analysis
vp <- fitVarPart(res.proc, ~group_id)

# Summarize variance fractions genome-wide for each cell type
plotVarPart(vp)

```

---

`plotViolin`*Plot Violins*

---

**Description**

Plot Violins

**Usage**`plotViolin(x, ...)`

```
## S4 method for signature 'cellSpecificityValues'  
plotViolin(x, assays = colnames(x))
```

**Arguments**

<code>x</code>	fractions for each gene
<code>...</code>	other arguments
<code>assays</code>	array of assays to plot

**Value**

Violin plot

**Examples**

```
library(muscat)  
library(SingleCellExperiment)  
  
data(example_sce)  
  
# create pseudobulk for each sample and cell cluster  
pb <- aggregateToPseudoBulk(example_sce,  
  assay = "counts",  
  cluster_id = "cluster_id",  
  sample_id = "sample_id",  
  verbose = FALSE  
)  
  
# Compute cell type specificity of each gene  
df <- cellTypeSpecificity(pb)  
  
# Violin plot of specificity scores for each cell type  
# Dashed line indicates genes that are equally expressed  
# across all cell types. For K cell types, this is 1/K  
plotViolin(df)
```



---

plotVolcano	<i>Volcano plot for each cell type</i>
-------------	--

---

**Description**

Volcano plot for each cell type

**Usage**

```
plotVolcano(  
  x,  
  coef,  
  nGenes = 5,  
  size = 12,  
  minp = 9.9999999999997e-311,  
  cutoff = 0.05,  
  ncol = 3,  
  ...  
)  
  
## S4 method for signature 'list'  
plotVolcano(  
  x,  
  coef,  
  nGenes = 5,  
  size = 12,  
  minp = 9.9999999999997e-311,  
  cutoff = 0.05,  
  ncol = 3,  
  assays = names(x),  
  ...  
)  
  
## S4 method for signature 'MArrayLM'  
plotVolcano(  
  x,  
  coef,  
  nGenes = 5,  
  size = 12,  
  minp = 9.9999999999997e-311,  
  cutoff = 0.05,  
  ncol = 3,  
  ...  
)  
  
## S4 method for signature 'dreamlet_mash_result'  
plotVolcano(  
  x,  
  coef,  
  nGenes = 5,  
  size = 12,  
  minp = 9.9999999999997e-311,  
  cutoff = 0.05,  
  ncol = 3,  
  ...  
)
```

```

x,
coef,
nGenes = 5,
size = 12,
minp = 1e-16,
cutoff = 0.05,
ncol = 3,
assays = colnames(x$logFC.original),
...
)

```

### Arguments

x	result from dreamlet
coef	coefficient to test with topTable
nGenes	number of genes to highlight in each volcano plot
size	text size
minp	minimum p-value to show on the y-axis
cutoff	adj.P.Val cutoff to distinguish significant from non-significant genes
ncol	number of columns in the plot
...	arguments passed to facet_wrap(). Useful for specifying scales = "free_y"
assays	which assays to plot

### Value

Volcano plot for each cell type

### Examples

```

library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

```

```
# show coefficients estimated for each cell type
coefNames(res.dl)

# volcano plot for each cell type
plotVolcano(res.dl, coef = "group_idstim")

# volcano plot for first two cell types
plotVolcano(res.dl[1:2], coef = "group_idstim")
```

---

plotVoom	<i>Plot voom curves from each cell type</i>
----------	---

---

### Description

Plot voom curves from each cell type

### Usage

```
plotVoom(x, ncol = 3, alpha = 0.5, ...)

## S4 method for signature 'dreamletProcessedData'
plotVoom(x, ncol = 3, alpha = 0.5, assays = names(x))

## S4 method for signature 'EList'
plotVoom(x, ncol = 3, alpha = 0.5)
```

### Arguments

x	dreamletProcessedData
ncol	number of columns in the plot
alpha	transparency of points
...	other arguments
assays	which assays to plot

### Value

Plot of mean-variance trend

### Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
```

```
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Show mean-variance trend from voom
plotVoom(res.proc)

# plot for first two cell types
plotVoom(res.proc[1:2])
```

---

*print,dreamletResult-method*

*Print object*

---

### **Description**

Print object

Print object

### **Usage**

```
## S4 method for signature 'dreamletResult'
print(x, ...)
```

```
## S4 method for signature 'dreamletProcessedData'
print(x, ...)
```

### **Arguments**

x	dreamletProcessedData object
...	other arguments

### **Value**

print data stored in object

processAssays

*Processing SingleCellExperiment to dreamletProcessedData***Description**

For raw counts, estimate precision weights using linear mixed model weighting by number of cells observed for each sample. For normalized data, only weight by number of cells.

**Usage**

```
processAssays(
  sceObj,
  formula,
  assays = assayNames(sceObj),
  min.cells = 5,
  min.count = 5,
  min.samples = 4,
  min.prop = 0.4,
  isCounts = TRUE,
  normalize.method = "TMM",
  span = "auto",
  quiet = FALSE,
  weightsList = NULL,
  BPPARAM = SerialParam(),
  ...
)
```

**Arguments**

sceObj	SingleCellExperiment object
formula	regression formula for differential expression analysis
assays	array of assay names to include in analysis. Defaults to assayNames(sceObj)
min.cells	minimum number of observed cells for a sample to be included in the analysis
min.count	minimum number of reads for a gene to be considered expressed in a sample. Passed to edgeR::filterByExpr
min.samples	minimum number of samples passing cutoffs for cell cluster to be retained
min.prop	minimum proportion of retained samples with non-zero counts for a gene to be retained
isCounts	logical, indicating if data is raw counts
normalize.method	normalization method to be used by calcNormFactors
span	Lowess smoothing parameter using by variancePartition::voomWithDreamWeights()
quiet	show messages

weightsList	list storing matrix of precision weights for each cell type. If NULL precision weights are set to 1
BPPARAM	parameters for parallel evaluation
...	other arguments passed to dream

### Details

For each cell cluster, samples with at least `min.cells` are retained. Only clusters with at least `min.samples` retained samples are kept. Genes are retained if they have at least `min.count` reads in at least `min.prop` fraction of the samples. Current values are reasonable defaults, since genes that don't pass these cutoffs are very underpowered for differential expression analysis and only increase the multiple testing burden. But values of `min.cells = 2` and `min.count = 2` are also reasonable to include more genes in the analysis.

The precision weights are estimated using the residuals fit from the specified formula. These weights are robust to changes in the formula as long as the major variables explaining the highest fraction of the variance are included.

If `weightsList` is NULL, precision weights are set to 1 internally.

### Value

Object of class `dreamletProcessedData` storing voom-style normalized expression data

### Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)
#
```

---

processOneAssay	<i>Processing expression data from assay</i>
-----------------	--

---

### Description

For raw counts, filter genes and samples, then estimate precision weights using linear mixed model weighting by number of cells observed for each sample. For normalized data, only weight by number of cells

### Usage

```
processOneAssay(
  y,
  formula,
  data,
  n.cells,
  min.cells = 5,
  min.count = 5,
  min.samples = 4,
  min.prop = 0.4,
  min.total.count = 15,
  isCounts = TRUE,
  normalize.method = "TMM",
  span = "auto",
  quiet = TRUE,
  weights = NULL,
  BPPARAM = SerialParam(),
  ...
)
```

### Arguments

y	matrix of counts or log2 CPM
formula	regression formula for differential expression analysis
data	metadata used in regression formula
n.cells	array of cell count for each sample
min.cells	minimum number of observed cells for a sample to be included in the analysis
min.count	minimum number of reads for a gene to be considered expressed in a sample. Passed to edgeR::filterByExpr
min.samples	minimum number of samples passing cutoffs for cell cluster to be retained
min.prop	minimum proportion of retained samples with non-zero counts
min.total.count	minimum total count required per gene for inclusion
isCounts	logical, indicating if data is raw counts

normalize.method	normalization method to be used by calcNormFactors
span	Lowess smoothing parameter using by variancePartition::voomWithDreamWeights()
quiet	show messages
weights	matrix of precision weights
BPPARAM	parameters for parallel evaluation
...	other arguments passed to dream

**Value**

EList object storing log2 CPM and precision weights

**See Also**

processAssays()

---

removeConstantTerms     *Remove constant terms from formula*

---

**Description**

Remove constant terms from formula. Also remove categorical variables with a max of one example per category

**Usage**

```
removeConstantTerms(formula, data)
```

**Arguments**

formula	original formula
data	data.frame

**Details**

Adapted from MoEClust::drop\_constants

**Value**

a formula, possibly with terms omitted.

**Examples**

```
# Valid formula
removeConstantTerms(~ group + extra, sleep)

# there is no variation in 'group' in this dataset
removeConstantTerms(~ group + extra, sleep[1:3, ])
```



---

residuals,dreamletResult-method  
*Extract residuals from dreamletResult*

---

## Description

Extract residuals from dreamletResult

## Usage

```
## S4 method for signature 'dreamletResult'  
residuals(object, y, ..., type = c("response", "pearson"))
```

## Arguments

object	dreamletResult object
y	dreamletProcessedData object
...	other arguments
type	compute either "response" residuals or "pearson" residuals.

## Details

"response" residuals are the typical residuals returned from `lm()`. "pearson" residuals divides each residual value by its estimated standard error. This requires specifying `y`

## Value

residuals from model fit

## Examples

```
library(muscat)  
library(SingleCellExperiment)  
  
data(example_sce)  
  
# create pseudobulk for each sample and cell cluster  
pb <- aggregateToPseudoBulk(example_sce,  
  assay = "counts",  
  cluster_id = "cluster_id",  
  sample_id = "sample_id",  
  verbose = FALSE  
)  
  
# voom-style normalization  
res.proc <- processAssays(pb, ~group_id)  
  
# Differential expression analysis within each assay,
```

```
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# extract typical residuals for each assay (i.e. cell type)
# Return list with entry for each assay with for retained samples and genes
resid.lst <- residuals(res.dl)

# Get Pearson residuals:
# typical residuals scaled by the standard deviation
residPearson.lst <- residuals(res.dl, res.proc, type = "pearson")
```

---

run\_mash

*Run mash analysis on dreamlet results*

---

## Description

Run mash analysis on dreamlet results

## Usage

```
run_mash(fit, coefList)
```

## Arguments

fit	result from dreamlet()
coefList	coefficient to be analyzed

## Details

Apply **mashr** analysis (Urbut et al. 2019) on the joint set of coefficients for each gene and cell type. mashr is a Bayesian statistical method that borrows strength across tests (i.e. genes and cell types) by learning the distribution of non-zero effects based the observed logFC and standard errors. The method then estimates the posterior distributions of each coefficient based on the observed value and the genome-wide empirical distribution.

mashr has been previously applied to differential expression in **GTEX** data using multiple tissues from the same set of donors (Oliva et al. 2020).

In single cell data, a given gene is often not sufficiently expressed in all cell types. So it is not evaluated in a subsets of cell types, and its coefficient value is NA. Since mashr assumes coefficients and standard errors for every gene and cell type pair, entries with these missing values are set to have coef = 0, and se = 1e6. The output of mashr is then modified to set the corresponding values to NA, to avoid nonsensical results downstream.

## Value

a list storing the mashr model as model and the original coefficients as logFC.original

## References

Oliva M, Munoz-Aguirre M, Kim-Hellmuth S, Wucher V, Gewirtz AD, Cotter DJ, Parsana P, Kasela S, Balliu B, Vinuela A, others (2020). “The impact of sex on gene expression across human tissues.” *Science*, **369**(6509), eaba3066. <https://doi.org/10.1126/science.aba3066>.

Urbut SM, Wang G, Carbonetto P, Stephens M (2019). “Flexible statistical methods for estimating and testing effects in genomic studies with multiple conditions.” *Nature genetics*, **51**(1), 187–195. <https://doi.org/10.1038/s41588-018-0268-8>.

## See Also

`mashr::mash_estimate_corr_em()`, `mashr::cov_canonical`, `mashr::mash_set_data`

## Examples

```
library(muscat)
library(mashr)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce[1:100, ],
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# run MASH model
# This can take 10s of minutes on real data
# This small datasets should take ~30s
res_mash <- run_mash(res.dl, "group_idstim")

# extract statistics from mashr model
# NA values indicate genes not sufficiently expressed
# in a given cell type

# original logFC
head(res_mash$logFC.original)

# posterior mean for logFC
head(get_pm(res_mash$model))
```

```

# how many gene-by-celltype tests are significant
# i.e. if a gene is significant in 2 celltypes, it is counted twice
table(get_lfsr(res_mash$model) < 0.05, useNA = "ifany")

# how many genes are significant in at least one cell type
table(apply(get_lfsr(res_mash$model), 1, min, na.rm = TRUE) < 0.05)

# how many genes are significant in each cell type
apply(get_lfsr(res_mash$model), 2, function(x) sum(x < 0.05, na.rm = TRUE))

# examine top set of genes
# which genes are significant in at least 1 cell type
sort(names(get_significant_results(res_mash$model)))[1:10]

# Lets examine EN01
# There is a lot of variation in the raw logFC
res_mash$logFC.original["EN01", ]

# posterior mean after borrowing across cell type and genes
get_pm(res_mash$model)["EN01", ]

# forest plot based on mashr results
plotForest(res_mash, "EN01")

# volcano plot based on mashr results
# yaxis uses local false sign rate (lfsr)
plotVolcano(res_mash)

# Comment out to reduce package runtime
# gene set analysis using mashr results
# library(zenith)
# go.gs = get_GeneOntology("CC", to="SYMBOL")
# df_gs = zenith_gsa(res_mash, go.gs)

# Heatmap of results
# plotZenithResults(df_gs, 2, 1)

```

---

seeErrors

*Get error text*

---

## Description

Get error text

## Usage

```
seeErrors(obj)
```

```
## S4 method for signature 'dreamletResult'
```

```
seeErrors(obj)

## S4 method for signature 'dreamletProcessedData'
seeErrors(obj)

## S4 method for signature 'vpDF'
seeErrors(obj)
```

### Arguments

obj                    A dreamletResult object

### Value

tibble storing error text

### Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# show errors
# but none are reported
res.err = seeErrors(res.dl)
```

---

show,dreamletResult-method

*Show object*

---

**Description**

Show object

Show object

**Usage**

```
## S4 method for signature 'dreamletResult'
show(object)
```

```
## S4 method for signature 'dreamletProcessedData'
show(object)
```

**Arguments**

object                  dreamletProcessedData object

**Value**

show data stored in object

---

sortCols, vpDF-method    *Sort variance partition statistics*

---

**Description**

Sort variance partition statistics

**Usage**

```
## S4 method for signature 'vpDF'
sortCols(
  x,
  FUN = sum,
  decreasing = TRUE,
  last = c("Residuals", "Measurement.error"),
  ...
)
```

**Arguments**

x                      object returned by fitVarPart()  
 FUN                    function giving summary statistic to sort by. Defaults to sum  
 decreasing            logical. Should the sorting be increasing or decreasing?  
 last                    columns to be placed on the right, regardless of values in these columns  
 ...                    other arguments to sort

**Value**

data.frame with columns sorted by mean value, with Residuals in last column

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# variance partitioning analysis
vp <- fitVarPart(res.proc, ~group_id)

# Summarize variance fractions genome-wide for each cell type
plotVarPart(sortCols(vp))
```

---

stackAssays

*Stack assays from pseudobulk*


---

**Description**

Stack assays from pseudobulk to perform analysis across cell types

**Usage**

```
stackAssays(pb, assays = assayNames(pb))
```

**Arguments**

**pb** pseudobulk SingleCellExperiment from aggregateToPseudoBulk()  
**assays** array of assay names to include in analysis. Defaults to assayNames(pb)

**Value**

pseudobulk SingleCellExperiment cbind'ing expression values and rbind'ing colData. The column stackedAssay in colData() stores the assay information of the stacked data.

## Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# Stack assays for joint analysis
pb.stack <- stackAssays(pb)

# voom-style normalization
# assay (i.e. cell type) can now be included as a covariate
res.proc <- processAssays(pb.stack, ~ group_id + stackedAssay)

# variance partitioning analysis
vp <- fitVarPart(res.proc, ~ group_id + stackedAssay)

# Summarize variance fractions across cell types
plotVarPart(sortCols(vp))

# Interaction analysis allows group_id
# to have a different effect within each stackedAssay
vp2 <- fitVarPart(res.proc, ~ group_id * stackedAssay)

plotVarPart(sortCols(vp2))

# Interaction model using random effects
form <- ~ (1 | group_id) + (1 | stackedAssay) + (1 | group_id:stackedAssay)
#
```

---

tabToMatrix

*Convert results table to matrix*

---

## Description

Convert results table to matrix

## Usage

```
tabToMatrix(tab, col, rn = "ID", cn = "assay")
```



**Arguments**

tab	results table from topTable()
col	which column to extract
rn	column id storing rownames
cn	column id storing colnames

**Value**

matrix storing values of column col in rows defined by rn and columns defined by cn

---

topTable,dreamletResult-method

*Table of Top Genes from dreamlet fit*

---

**Description**

Extract a table of the top-ranked genes from a dreamlet fit.

**Usage**

```
## S4 method for signature 'dreamletResult'
topTable(
  fit,
  coef = NULL,
  number = 10,
  genelist = NULL,
  adjust.method = "BH",
  sort.by = "P",
  resort.by = NULL,
  p.value = 1,
  lfc = 0,
  confint = FALSE
)
```

**Arguments**

fit	dreamletResult object
coef	coef
number	number
genelist	genelist
adjust.method	adjust.method
sort.by	sort.by
resort.by	resort.by
p.value	p.value
lfc	lfc
confint	confint

**Value**

data.frame storing hypothesis test for each gene and cell type

**See Also**

limma::topTable(), variancePartition::topTable()

**Examples**

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# show coefficients estimated for each cell type
coefNames(res.dl)

# extract results using limma-style syntax
# combines all cell types together
# adj.P.Val gives study-wide FDR
topTable(res.dl, coef = "group_idstim", number = 3)
```

---

vpDF-class

*Class vpDF*


---

**Description**

Class vpDF stores results for each gene for each assay

**Value**

none

none

---

```
zenith_gsa,dreamletResult,GeneSetCollection-method
  Perform gene set analysis using zenith
```

---

## Description

Perform a competitive gene set analysis accounting for correlation between genes.

## Usage

```
## S4 method for signature 'dreamletResult,GeneSetCollection'
zenith_gsa(
  fit,
  geneSets,
  coefs,
  use.ranks = FALSE,
  n_genes_min = 10,
  inter.gene.cor = 0.01,
  progressbar = TRUE,
  ...
)

## S4 method for signature 'dreamlet_mash_result,GeneSetCollection'
zenith_gsa(
  fit,
  geneSets,
  coefs,
  use.ranks = FALSE,
  n_genes_min = 10,
  inter.gene.cor = 0.01,
  progressbar = TRUE,
  ...
)
```

## Arguments

fit	results from dreamlet()
geneSets	GeneSetCollection
coefs	coefficients to test using topTable(fit, coef=coefs[i])
use.ranks	do a rank-based test TRUE or a parametric test FALSE? default: FALSE
n_genes_min	minimum number of genes in a geneset
inter.gene.cor	if NA, estimate correlation from data. Otherwise, use specified value
progressbar	if TRUE, show progress bar
...	other arguments

## Details

This code adapts the widely used camera() analysis (Wu and Smyth 2012) in the limma package (Ritchie et al. 2015) to the case of linear (mixed) models used by variancePartition::dream().

## Value

data.frame of results for each gene set and cell type

data.frame of results for each gene set and cell type

## Examples

```
library(muscat)
library(SingleCellExperiment)

data(example_sce)

# create pseudobulk for each sample and cell cluster
pb <- aggregateToPseudoBulk(example_sce,
  assay = "counts",
  cluster_id = "cluster_id",
  sample_id = "sample_id",
  verbose = FALSE
)

# voom-style normalization
res.proc <- processAssays(pb, ~group_id)

# Differential expression analysis within each assay,
# evaluated on the voom normalized data
res.dl <- dreamlet(res.proc, ~group_id)

# Load Gene Ontology database
# use gene 'SYMBOL', or 'ENSEMBL' id
# use get_MSigDB() to load MSigDB
library(zenith)
go.gs <- get_GeneOntology("CC", to = "SYMBOL")

# Run zenith gene set analysis on result of dreamlet
res_zenith <- zenith_gsa(res.dl, go.gs, "group_idstim", progressBar = FALSE)

# for each cell type select 3 genesets with largest t-statistic
# and 1 geneset with the lowest
# Grey boxes indicate the gene set could not be evaluated because
#   to few genes were represented
plotZenithResults(res_zenith, 3, 1)
```

---

[,dreamletResult,ANY,ANY,ANY-method  
*Subset with brackets*

---

### **Description**

Subset with brackets

Subset with brackets

### **Usage**

```
## S4 method for signature 'dreamletResult,ANY,ANY,ANY'  
x[i]
```

```
## S4 method for signature 'dreamletProcessedData,ANY,ANY,ANY'  
x[i]
```

### **Arguments**

x                    dreamletProcessedData object

i                    indeces to extract

### **Value**

entries stored at specified index

# Index

[\[, dreamletProcessedData, ANY, ANY, ANY-method\]](#) [11](#)  
[\(\[, dreamletResult, ANY, ANY, ANY-method\), assayNames, dreamletResult-method\]](#), [11](#)  
[77](#) [assayNames, vpDF, vpDF-method](#)  
[\[, dreamletProcessedData, dreamletProcessedData-method \(assayNames, dreamletResult-method\),](#)  
[\(\[, dreamletResult, ANY, ANY, ANY-method\),\]](#) [11](#)  
[77](#) [assayNames, vpDF-method](#)  
[\[, dreamletResult, ANY, ANY, ANY-method,](#) [\(assayNames, dreamletResult-method\),](#)  
[77](#) [11](#)  
[\[, dreamletResult, dreamletResult-method](#)  
[\(\[, dreamletResult, ANY, ANY, ANY-method\),](#) [BiocParallelParam](#), [4](#), [6](#), [8](#)  
[77](#) [buildClusterTreeFromPB](#), [11](#)  
  
[aggregateNonCountSignal](#), [4](#) [cellCounts](#), [12](#)  
[aggregateToPseudoBulk](#), [5](#) [cellSpecificityValues-class](#), [13](#)  
[aggregateVar](#), [7](#) [cellTypeSpecificity](#), [14](#)  
[as.dreamletResult](#), [9](#) [checkFormula](#), [15](#)  
[assay, dreamletProcessedData, ANY-method](#) [coefNames](#), [16](#)  
[\(assay, dreamletResult, ANY-method\),](#) [coefNames, dreamletResult-method](#)  
[10](#) [\(coefNames\),](#) [16](#)  
[assay, dreamletProcessedData, dreamletProcessedData-method,](#) [Data-method,](#) [dreamletProcessedData-method,](#)  
[\(assay, dreamletResult, ANY-method\),](#) [17](#)  
[10](#) [colData<- , dreamletProcessedData, ANY-method,](#)  
[17](#)  
[assay, dreamletResult, ANY-method,](#) [10](#) [compositePosteriorTest](#), [18](#)  
[assay, dreamletResult, dreamletResult-method](#) [computeCellCounts](#), [19](#)  
[\(assay, dreamletResult, ANY-method\),](#) [computeLogCPM](#), [20](#)  
[10](#) [computeNormCounts](#), [21](#)  
[assay, vpDF, ANY-method](#) [DelayedMatrixStats](#), [6](#)  
[\(assay, dreamletResult, ANY-method\),](#) [10](#) [details](#), [22](#)  
[assay, vpDF, vpDF-method](#) [details, dreamletProcessedData-method](#)  
[\(assay, dreamletResult, ANY-method\),](#) [10](#) [\(details\),](#) [22](#)  
  
[assayNames, dreamletProcessedData, dreamletProcessedData-method,](#) [dreamletResult-method](#)  
[\(assayNames, dreamletResult-method\),](#) [\(details\),](#) [22](#)  
[11](#) [details, vpDF-method \(details\),](#) [22](#)  
[assayNames, dreamletProcessedData-method](#) [diffVar, dreamletResult, dreamletResult-method](#)  
[\(assayNames, dreamletResult-method\),](#) [\(diffVar, dreamletResult-method\),](#)  
[11](#) [23](#)  
  
[assayNames, dreamletResult, dreamletResult-method,](#) [diffVar, dreamletResult-method,](#) [23](#)  
[\(assayNames, dreamletResult-method\),](#) [dreamlet,](#) [25](#)

- dreamlet, dreamletProcessedData-method  
(dreamlet), 25
- dreamlet\_mash\_result-class, 31
- dreamletCompareClusters, 27
- dreamletProcessedData-class, 30
- dreamletResult-class, 31
- dropRedundantTerms, 32
  
- equalFormulas, 32
- extractData, 33
- extractData, dreamletProcessedData, character-method  
(extractData), 33
- extractData, dreamletProcessedData-method  
(extractData), 33
  
- fitVarPart, 34
- fitVarPart, dreamletProcessedData-method  
(fitVarPart), 34
  
- getExprGeneNames, 36
- getTreat, dreamletResult, dreamletResult-method  
(getTreat, dreamletResult-method),  
37
- getTreat, dreamletResult-method, 37
  
- meta\_analysis, 39
- metadata, dreamletProcessedData, dreamletProcessedData-method  
(metadata, dreamletProcessedData-method),  
38
- metadata, dreamletProcessedData-method,  
38
  
- outlier, 40
- outlierByAssay, 41
  
- pbWeights, 42
- plotBeeswarm, 44
- plotCellComposition, 45
- plotCellComposition, data.frame-method  
(plotCellComposition), 45
- plotCellComposition, matrix-method  
(plotCellComposition), 45
- plotCellComposition, SingleCellExperiment-method  
(plotCellComposition), 45
- plotForest, 46
- plotForest, dreamlet\_mash\_result-method  
(plotForest), 46
- plotForest, dreamletResult-method  
(plotForest), 46
- plotGeneHeatmap, 47
- plotGeneHeatmap, dreamletResult, dreamletResult-method  
(plotGeneHeatmap), 47
- plotGeneHeatmap, dreamletResult-method  
(plotGeneHeatmap), 47
- plotHeatmap, 49
- plotHeatmap, cellSpecificityValues, cellSpecificityValues-method  
(plotHeatmap), 49
- plotHeatmap, cellSpecificityValues-method  
(plotHeatmap), 49
- plotHeatmap, data.frame, data.frame-method  
(plotHeatmap), 49
- plotHeatmap, data.frame-method  
(plotHeatmap), 49
- plotHeatmap, matrix, matrix-method  
(plotHeatmap), 49
- plotHeatmap, matrix-method  
(plotHeatmap), 49
- plotPCA, 50
- plotPCA, list-method (plotPCA), 50
- plotPercentBars, cellSpecificityValues, cellSpecificityValue-method  
(plotPercentBars, vpDF-method),  
52
- plotPercentBars, cellSpecificityValues-method  
(plotPercentBars, vpDF-method),  
52
- plotPercentBars, vpDF, vpDF-method  
(plotPercentBars, vpDF-method),  
52
- plotPercentBars, vpDF-method, 52
- plotProjection, 53
- plotVarPart, DataFrame, DataFrame-method  
(plotVarPart, DataFrame-method),  
54
- plotVarPart, DataFrame-method, 54
- plotViolin, 56
- plotViolin, cellSpecificityValues, cellSpecificityValues-method  
(plotViolin), 56
- plotViolin, cellSpecificityValues-method  
(plotViolin), 56
- plotVolcano, 57
- plotVolcano, dreamlet\_mash\_result, dreamlet\_mash\_result-method  
(plotVolcano), 57
- plotVolcano, dreamlet\_mash\_result-method  
(plotVolcano), 57
- plotVolcano, list, list-method  
(plotVolcano), 57
- plotVolcano, list-method (plotVolcano),  
57

- plotVolcano, MArrayLM, MArrayLM-method  
(plotVolcano), [57](#)
- plotVolcano, MArrayLM-method  
(plotVolcano), [57](#)
- plotVoom, [59](#)
- plotVoom, dreamletProcessedData, dreamletProcessedData-method  
(plotVoom), [59](#)
- plotVoom, dreamletProcessedData-method  
(plotVoom), [59](#)
- plotVoom, EList-method (plotVoom), [59](#)
- plotVoom, list, list-method (plotVoom), [59](#)
- print, dreamletProcessedData, dreamletProcessedData-method  
(print, dreamletResult-method),  
[60](#)
- print, dreamletProcessedData-method  
(print, dreamletResult-method),  
[60](#)
- print, dreamletResult, dreamletResult-method  
(print, dreamletResult-method),  
[60](#)
- print, dreamletResult-method, [60](#)
- processAssays, [61](#)
- processOneAssay, [63](#)
  
- removeConstantTerms, [64](#)
- residuals, dreamletResult, dreamletResult-method  
(residuals, dreamletResult-method),  
[65](#)
- residuals, dreamletResult-method, [65](#)
- run\_mash, [66](#)
  
- seeErrors, [68](#)
- seeErrors, dreamletProcessedData-method  
(seeErrors), [68](#)
- seeErrors, dreamletResult-method  
(seeErrors), [68](#)
- seeErrors, vpDF-method (seeErrors), [68](#)
- Seurat, [6](#)
- show, dreamletProcessedData, dreamletProcessedData-method  
(show, dreamletResult-method),  
[69](#)
- show, dreamletProcessedData-method  
(show, dreamletResult-method),  
[69](#)
- show, dreamletResult, dreamletResult-method  
(show, dreamletResult-method),  
[69](#)
- show, dreamletResult-method, [69](#)
- SingleCellExperiment, [4](#), [6](#), [8](#)
- sortCols, vpDF-method, [70](#)
- stackAssays, [71](#)
- tabToMatrix, [72](#)
- topTable, dreamletResult, dreamletResult-method  
(topTable, dreamletResult-method),  
[73](#)
- topTable, dreamletResult-method, [73](#)
- vpDF-class, [74](#)
- zenith\_gsa, dreamlet\_mash\_result, GeneSetCollection, ANY-method  
(zenith\_gsa, dreamletResult, GeneSetCollection-method),  
[75](#)
- zenith\_gsa, dreamlet\_mash\_result, GeneSetCollection-method  
(zenith\_gsa, dreamletResult, GeneSetCollection-method),  
[75](#)
- zenith\_gsa, dreamletResult, GeneSetCollection, ANY-method  
(zenith\_gsa, dreamletResult, GeneSetCollection-method),  
[75](#)
- zenith\_gsa, dreamletResult, GeneSetCollection-method,  
[75](#)