Package 'ReducedExperiment'

April 7, 2026

Type Package

Title Containers and tools for dimensionally-reduced -omics representations

Version 1.3.0

Date 2024-07-30

Description Provides SummarizedExperiment-like containers for storing and manipulating dimensionally-reduced assay data. The ReducedExperiment classes allow users to simultaneously manipulate their original dataset and their decomposed data, in addition to other method-specific outputs like feature loadings. Implements utilities and specialised classes for the application of stabilised independent component analysis (sICA) and weighted gene correlation network analysis (WGCNA).

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.4.0), SummarizedExperiment, methods

Imports WGCNA, ica, moments, clusterProfiler, msigdbr, RColorBrewer, car, lme4, lmerTest, pheatmap, biomaRt, stats, grDevices, BiocParallel, ggplot2, patchwork, BiocGenerics, S4Vectors

Suggests knitr, rmarkdown, testthat, biocViews, BiocCheck, BiocStyle, airway

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

URL https://github.com/jackgisby/ReducedExperiment

BugReports https://github.com/jackgisby/ReducedExperiment/issues

biocViews GeneExpression, Infrastructure, DataRepresentation, Software, DimensionReduction, Network

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

git_branch devel

git_last_commit 8a1ee25

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2026-04-06

Author Jack Gisby [aut, cre] (ORCID: <https://orcid.org/0000-0003-0511-8123>), Michael Barnes [aut] (ORCID: <https://orcid.org/0000-0001-9097-7381>)

Maintainer Jack Gisby <jackgisby@gmail.com>

Contents

`.DollarNames.FactorisedExperiment` *Command line completion for* `$`

Description

Command line completion for `$`. This function is not intended to be used directly by users but provides auto-completion capabilities. Autocompletes based on column data names (i.e., the column names of the `colData`).

Usage

```
## S3 method for class 'FactorisedExperiment'
.DollarNames(x, pattern = "")

## S3 method for class 'ModularExperiment'
.DollarNames(x, pattern = "")

## S3 method for class 'ReducedExperiment'
.DollarNames(x, pattern = "")
```

Arguments

`x`	The ReducedExperiment object.
`pattern`	Search pattern.

Value

The names of the matching columns of `colData`.

See Also

`utils:::.DollarNames()`

`assessSoftThreshold` *Assess soft thresholding power for WGCNA*

Description

A wrapper around pickSoftThreshold, allowing assessment and automatic selection of soft-thresholding power. Extends the function to accept a SummarizedExperiment as input and additionally considers mean connectivity when selecting the soft-thresholding power to recommend.

Usage

```
assessSoftThreshold(
  X,
  assay_name = "normal",
  powerVector = 1:30,
  RsquaredCut = 0.85,
  max_mean_connectivity = 100,
  cor_type = "pearson",
  networkType = "signed",
  maxBlockSize = 30000,
  verbose = 0,
  ...
)
```

Arguments

X	Either a SummarizedExperiment object or a matrix containing data to be subject to WGCNA. X should have rows as features and columns as samples.
assay_name	If X is a SummarizedExperiment, then this should be the name of the assay to be subject to WGCNA.
powerVector	a vector of soft thresholding powers for which the scale free topology fit indices are to be calculated.
RsquaredCut	desired minimum scale free topology fitting index R^2.
max_mean_connectivity	The maximal mean connectivity required. Used to select the soft-thresholding power.
cor_type	The type of correlation to be used to generate a correlation matrix during network formation. One of "pearson" (cor) and "bicor" (bicor).
networkType	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
maxBlockSize	The chunk size (in terms of the number of features/genes) to process the data. The default (30000) should process standard transcriptomic datasets in a single chunk. Results may differ if the number of features exceeds the chunk size. Lower values of this parameter will use less memory to calculate networks.
verbose	integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
...	Additional arguments to be passed to pickSoftThreshold.

Details

The pickSoftThreshold function estimates the power by selecting the lowest value with a minimum scale free topology fitting index exceeding RsquaredCut. The assessSoftThreshold function mirrors this behaviour when max_mean_connectivity is NULL. When max_mean_connectivity is specified, however, we additionally require that the selected power does not exceed this connectivity threshold.

Value

Returns a `data.frame`, generated by pickSoftThreshold, with scale free topology fitting indices and connectivity statistics. Additionally contains a column, `estimated_power`, indicating the recommended power to use (see `details`). We suggest manually considering suitability of the soft-thresholding power rather than solely relying on this automated approach.

Author(s)

Jack Gisby

See Also

`WGCNA::pickSoftThreshold()`, `runWGCNA()`

Examples

```
# Get the airway data as a SummarizedExperiment (with a subset of features)
set.seed(2)
airway_se <- ReducedExperiment:::.getAirwayData(n_features = 500)

# Select soft-thresholding power to use (use capture.output to hide WGCNA's prints)
WGCNA::disableWGCNAThreads()
invisible(capture.output(fit_indices <- assessSoftThreshold(airway_se)))

print(fit_indices)
print(paste0("Estimated power: ", fit_indices$Power[fit_indices$estimated_power]))
```

---

assignments *Get and set module feature assignments*

Description

Retrieves a vector of features (usually genes) named by the modules they belong to. Assignment can be used to modify all or part of the vector.

Usage

```
## S4 method for signature 'ModularExperiment'
assignments(object, as_list = FALSE)

## S4 replacement method for signature 'ModularExperiment'
assignments(object) <- value
```

Arguments

object	ModularExperiment object.
as_list	If TRUE, the results are returned as a list, with an entry for each module containing a list of features.
value	New value to replace existing assignments.

Value

A vector with values representing features and names representing feature assignments (i.e., modules).

Author(s)

Jack Gisby

Examples

```
# Create ModularExperiment with random data (100 features, 50 samples,
# 10 modules)
me <- ReducedExperiment:::.createRandomisedModularExperiment(100, 50, 10)
me

# Assignment of features to groups/modules
assignments(me)

# We can reassign a feature to a new module if we like:
names(assignments(me))[6] <- "new_module"
assignments(me)[1:10]

# We shouldn't, however, attempt to change the feature names here:
# assignments(me)[5] <- "modified_gene_name"

# Instead, we should change the object's feature names as so:
featureNames(me)[5] <- "modified_gene_name"
assignments(me)[1:10]
```

---

associateComponents *Runs linear models for components and sample-level data*

Description

Runs either standard linear or linear mixed models, with reduced components (e.g., factors or modules) as the outcomes and sample-level information (e.g., treatment, disease status) as predictors.

Usage

```
associateComponents(
  re,
  formula,
  method = "lm",
  scale_reduced = TRUE,
  center_reduced = TRUE,
  type = "II",
  adj_method = "BH",
  ...
)
```

Arguments

re	An object inheriting from ReducedExperiment.
formula	The model formula to apply. Only the right hand side of the model need be specified (e.g., "~ x + y"). The left hand side (outcome) represents the components themselves. The variables in this formula should be present in the colData of re.
method	If "lm", then the lm function is used to run linear models (in tandem with Anova for running anovas on the model terms). If "lmer", then linear mixed models are run through lmer.
scale_reduced	If TRUE, the reduced data are scaled (to have a standard deviation of 1) before modelling.
center_reduced	If TRUE, the reduced data are centered (to have a mean of 0) before modelling.
type	The type of anova to be applied to the terms of the linear model.
adj_method	The method for adjusting for multiple testing. Passed to the p.adjust method parameter.
...	Additional arguments passed to lmer, given that method is set to "lmer".

Details

Multiple testing adjustment is performed separately for each term in the model across all factors. In other words, p-values are adjusted for the number of factors, but not the number of model terms. If you are testing a large number of terms, you could consider applying a custom adjustment method or using penalised regression.

Value

Returns a list with the entry "models" including a list of the model objects, "anovas" containing the output of anova-based testing, and "summaries" containing the results of running summary on the models.

Author(s)

Jack Gisby

See Also

stats::lm(), car::Anova(), lmerTest::lmer()

Examples

```
# Create FactorisedExperiment with random data (100 features, 50 samples,
# 10 factors)
set.seed(1)
fe <- ReducedExperiment::::.createRandomisedFactorisedExperiment(100, 50, 10)
fe

# Create a sample-level variable describing some sort of treatment
colData(fe)$treated <- c(rep("control", 25), rep("treatment", 25))
colData(fe)$treated <- factor(colData(fe)$treated, c("control", "treatment"))

# Increase the value of factor 1 for the treated samples, simulating some
# kind of treatment-related effect picked up by factor analysis
reduced(fe)[, 1][colData(fe)$treated == "treatment"] <-
    reduced(fe)[, 1][colData(fe)$treated == "treatment"] +
    rnorm(25, mean = 1.5, sd = 0.1)

# Create a sample-level variable describing a covariate we want to adjust for
# We will make the treated patients slightly older on average
colData(fe)$age <- 0
colData(fe)$age[colData(fe)$treated == "control"] <- rnorm(25, mean = 35, sd = 8)
colData(fe)$age[colData(fe)$treated == "treatment"] <- rnorm(25, mean = 40, sd = 8)

# Associate the factors with sample-level variable in the colData
lm_res <- associateComponents(
    fe,
    formula = "~ treated + age", # Our model formula
    method = "lm", # Use a linear model
    adj_method = "BH" # Adjust our p-values with Benjamini-Hochberg
)

# We see that treatment is significantly associated with factor 1 (adjusted
# p-value < 0.05) and is higher in the treated patients. Age is not
# significantly associated with factor 1, but there is a slight positive
# relationship
print(head(lm_res$summaries[
    ,
    c("term", "component", "estimate", "stderr", "pvalue", "adj_pvalue")
]))

# But what if these aren't 50 independent patients, but rather 25 patients
# sampled before and after treatment? We can account for this using a
# linear mixed model, which can account for repeated measures and paired
# designs

# First we add in this information
colData(fe)$patient_id <- c(paste0("patient_", 1:25), paste0("patient_", 1:25))
```

```
# Then we run the linear mixed model with a random intercept for patient
lmm_res <- associateComponents(
    fe,
    formula = "~ treated + age + (1 | patient_id)", # Add a random intercept
    method = "lmer", # Use a linear mixed model
    adj_method = "BH"
)

# We used a different method, but can obtain a similar summary output
print(head(lmm_res$summaries[
    ,
    c("term", "component", "estimate", "stderr", "pvalue", "adj_pvalue")
]))
```

`calcEigengenes` *Calculate eigengenes for new data*

Description

Calculates eigengenes for modules in new data. By default, eigengenes are calculated from scratch using PCA, in a similar manner to the moduleEigengenes function. The function also offers a projection approach, which functions in a similar fashion to the `predict` method of prcomp.

Usage

```
## S4 method for signature 'ModularExperiment,matrix'
calcEigengenes(
  object,
  newdata,
  project = FALSE,
  scale_reduced = TRUE,
  return_loadings = FALSE,
  scale_newdata = NULL,
  center_newdata = NULL,
  realign = TRUE,
  min_module_genes = 10
)

## S4 method for signature 'ModularExperiment,data.frame'
calcEigengenes(
  object,
  newdata,
  project = FALSE,
  scale_reduced = TRUE,
  return_loadings = FALSE,
  scale_newdata = NULL,
  center_newdata = NULL,
```

```
  realign = TRUE,
  min_module_genes = 10
)

## S4 method for signature 'ModularExperiment,SummarizedExperiment'
calcEigengenes(
  object,
  newdata,
  project = FALSE,
  scale_reduced = TRUE,
  assay_name = "normal",
  scale_newdata = NULL,
  center_newdata = NULL,
  realign = TRUE,
  min_module_genes = 10
)

## S4 method for signature 'ModularExperiment'
predict(object, newdata, ...)
```

Arguments

object	A ModularExperiment object. By default, the scale and center slots are used to apply the original transformation to the new data. The loadings slot of this class will be used if project is TRUE.
newdata	New data for eigengenes to be calculated in. Must be a data.frame or matrix with features as rows and samples as columns, or a SummarizedExperiment object. Assumes that the rows of newdata match those of the ModularExperiment object.
project	If FALSE (default), calculate eigengenes from scratch in the new dataset using an approach similar to moduleEigengenes (i.e., performing PCA for each module in newdata). If FALSE, perform projection, using PCA rotation matrix from the original data to calculate module eigengenes. Projection approach is experimental.
scale_reduced	Whether or not the reduced data should be scaled after calculation.
return_loadings	If TRUE, additionally returns the feature loadings for the eigengenes.
scale_newdata	Controls whether the newdata are scaled. If NULL, performs scaling based on the ModularExperiment object's scale slot. The value of this argument will be passed to the scale argument of scale.
center_newdata	Controls whether the newdata are centered If NULL, performs centering based on the ModularExperiment object's center slot. The value of this argument will be passed to the center argument of scale.
realign	If project is TRUE, this argument is ignored. Else, controls whether eigengenes are realigned after PCA is performed to ensure the resultant signatures are positively correlated with average expression of the module. Similar to the align argument of moduleEigengenes.

`min_module_genes`
: If `project` is `FALSE`, this argument is ignores. Else, controls the minimum number of genes required in a module for projection. Projected eigengenes are not calculated for modules with sizes below this threshold.

`assay_name`
: If a SummarizedExperiment object is passed as new data, this argument indicates which assay should be used for projection.

`...`
: Additional arguments to be passed to calcEigengenes.

Details

If `scale_newdata` and `center_newdata` are left as `NULL`, then the projection method assumes that the `newdata` are on the same scale as the original data of the `object`. It will therefore use the values of the `center` and `scale` slots of the `object`. For instance, if the `scale` slot is `TRUE`, the `newdata` will be scaled. If the `scale` slot is a vector, the values of this vector will be applied to scale the `newdata`.

Value

If return_loadings is `TRUE`, returns a list with the "reduced" matrix and "loadings" vector (one value per feature). If `FALSE`, returns only the reduced matrix.

The reduced matrix has samples as rows and modules as columns. If `newdata` was a `matrix` or `data.frame`, this will be returned as a matrix. If a SummarizedExperiment object was passed instead, then a If a ModularExperiment object will be created containing this matrix in its `reduced` slot.

Author(s)

Jack Gisby

See Also

`projectData`, `moduleEigengenes`

Examples

```
# Create ModularExperiment with random data (100 features, 50 samples,
# 10 modules)
me_1 <- ReducedExperiment:::.createRandomisedModularExperiment(100, 50, 10)

# Generate a new dataset with the same features (100 rows) but different
# samples/observations (20 columns)
X_2 <- ReducedExperiment:::.makeRandomData(100, 20, "gene", "sample")

# We can use the projection approach to calculate the eigengenes for
# the modules identified in dataset 1 for the samples in dataset 2
# This approach is based on the module loadings
me_2_project <- calcEigengenes(me_1, X_2, project = TRUE)
me_2_project[1:5, 1:5]

# Alternatively, we can calculate eigengenes from scratch in the second
```

```
# dataset. This still uses the modules identified in the first dataset (me_1)
# but does not make use of the loadings. This approach is similar to
# that applied by WGCNA::moduleEigengenes.
me_2_eig <- calcEigengenes(me_1, X_2, project = FALSE)
me_2_eig[1:5, 1:5]
```

`cbind,FactorisedExperiment-method`
Combine ReducedExperiment objects by columns or rows

Description

Combines ReducedExperiment objects by columns (samples) or rows (features).

Usage

```
## S4 method for signature 'FactorisedExperiment'
cbind(..., deparse.level = 1)

## S4 method for signature 'FactorisedExperiment'
rbind(..., deparse.level = 1)

## S4 method for signature 'ModularExperiment'
cbind(..., deparse.level = 1)

## S4 method for signature 'ModularExperiment'
rbind(..., deparse.level = 1)

## S4 method for signature 'ReducedExperiment'
cbind(..., deparse.level = 1)

## S4 method for signature 'ReducedExperiment'
rbind(..., deparse.level = 1)
```

Arguments

`...`	A series of ReducedExperiment objects to be combined. See cbind,SummarizedExperiment-method for further details.
`deparse.level`	Integer, see cbind.

Details

`cbind` assumes that objects have identical features and components (i.e., factors or modules). If they are not, an error is returned.

So, this means that the feature-level slots should be equivalent, for example the assay rownames and values of the `loadings` available in FactorisedExperiment and ModularExperimentobjects. The

component slots should also be equivalent, such as the column names of the `reduced` matrix or the column names of the aformentioned factor `loadings` matrix.

`rbind` assumes that objects have identical samples and components. If they are not, an error is returned. This means that the sample-level slots should be equivalent, including for example the assay column names.

The SummarizedExperiment package includes separate methods for `cbind` (cbind,SummarizedExperiment-method) and (combineRows). The latter is supposed to be more flexible, permitting differences in the number and identity of the rows. For ReducedExperiment objects we only implement a single, less flexible, method that assumes the rows and components (i.e., factors or modules) are identical across objects. Attempting to apply `combineRows` to a ReducedExperiment object will result in the objects being treated as if they were SummarizedExperiments, and a single SummarizedExperiment object will be returned.

Value

Returns a single ReducedExperiment object containing all of the columns in the objects passed to `cbind`.

Author(s)

Jack Gisby

See Also

`base::cbind()`, `base::rbind()`, cbind,SummarizedExperiment-method, rbind,SummarizedExperiment-method

Examples

```
# Create randomised containers with different numbers of samples
i <- 300 # Number of features
k <- 10 # Number of components (i.e., factors/modules)

# Same features and components, different samples (30 vs. 50 columns)
re_1 <- ReducedExperiment:::.createRandomisedReducedExperiment(i, 50, k)
re_2 <- ReducedExperiment:::.createRandomisedReducedExperiment(i, 30, k)

# Make a new object with 80 columns
cbind(re_1, re_2)

# Create randomised containers with different numbers of features
j <- 100 # Number of samples
k <- 10 # Number of components (i.e., factors/modules)

# Same features and components, different samples (30 vs. 50 columns)
re_3 <- ReducedExperiment:::.createRandomisedReducedExperiment(200, j, k)
re_4 <- ReducedExperiment:::.createRandomisedReducedExperiment(150, j, k)
reduced(re_3) <- reduced(re_4) # rbind assumes identical reduced data

# Make a new object with 80 columns
rbind(re_3, re_4)
```

```
# We can apply combineRows and combineCols to `ReducedExperiment` objects
# but the resulting object will be a `SummarizedExperiment`
combineCols(re_1, re_2)
combineRows(re_3, re_4)
```

---

componentNames<-,FactorisedExperiment-method

Get names of dimensionally-reduced components

Description

Retrieves the feature names post-dimensionality reduction In the case of module analysis, these are the names of the gene modules; in the case of factor analysis, these are the names of the factors.

Usage

```
## S4 replacement method for signature 'FactorisedExperiment'
componentNames(object) <- value

## S4 replacement method for signature 'ModularExperiment'
componentNames(object) <- value

## S4 method for signature 'ModularExperiment'
moduleNames(object)

## S4 replacement method for signature 'ModularExperiment'
moduleNames(object) <- value

## S4 method for signature 'ReducedExperiment'
componentNames(object)

## S4 replacement method for signature 'ReducedExperiment'
componentNames(object) <- value
```

Arguments

object	A ReducedExperiment object.
value	New value to replace existing names.

Details

componentNames is valid for all ReducedExperiment objects, whereas moduleNames is only valid for ModularExperiments.

Value

A vector containing the names of the components.

Author(s)

Jack Gisby

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of factors

rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "component")

re <- ReducedExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data
)

stopifnot(all.equal(componentNames(re), colnames(rand_reduced_data)))

print(paste0("Component name at [2]: ", componentNames(re)[2]))
componentNames(re)[2] <- "custom_component_name"
print(paste0("Component name at [2]: ", componentNames(re)[2]))
```

---

dendrogram *Get the dendrogram stored in a ModularExperiment*

Description

Get the dendrogram stored in a ModularExperiment

Usage

```
## S4 method for signature 'ModularExperiment'
dendrogram(object)

## S4 replacement method for signature 'ModularExperiment'
dendrogram(object) <- value
```

Arguments

object	A ModularExperiment object.
value	New value to replace existing dendrogram.

Value

Returns a dendrogram describing relationships between genes. Usually produced through hierarchical clustering using the blockwiseModules function.

Author(s)

Jack Gisby

See Also

WGCNA::blockwiseModules(), stats::hclust()

Examples

```
# Create ModularExperiment with random data (100 features, 50 samples,
# 10 modules)
me <- ReducedExperiment::::.createRandomisedModularExperiment(100, 50, 10)
me

# The dendrogram is usually produced during module discovery, but we can
# assign any dendrogram to the slot. Let's do hierarchical clustering on the
# features in our object and assign it
dendrogram(me) <- hclust(dist(assay(me)))
dendrogram(me)

# Can use default plotting approach
plot(dendrogram(me))

# Or class method that calls WGCNA::plotDendroAndColors
plotDendro(me)
```

---

dim,ReducedExperiment-method

Get the dimensions of a Reducedexperiment object

Description

Get the dimensions of a Reducedexperiment object

Usage

```
## S4 method for signature 'ReducedExperiment'
dim(x)
```

Arguments

x	A ReducedExperiment object.

Value

Returns a named vector containing the dimensions of the samples, features and reduced dimensions.

Author(s)

Jack Gisby

Examples

```
# Create a randomised ReducedExperiment
re <- ReducedExperiment:::.createRandomisedReducedExperiment(100, 50, 10)

# Get the dimensions
dim(re)
```

`estimateFactors` *Perform dimensionality reduction using Independent Component Analysis*

Description

Performs independent component analysis (ICA) and packages both the input data and subsequent results into a FactorisedExperiment container. Calls runICA to perform the analysis; see its documentation page for more information on the ICA method, parameters and outputs.

Usage

```
estimateFactors(
  X,
  nc,
  center_X = TRUE,
  scale_X = FALSE,
  assay_name = "normal",
  ...
)
```

Arguments

`X`	Either a SummarizedExperiment object or a matrix containing data to be subject to ICA. `X` should have rows as features and columns as samples.
`nc`	The number of components to be identified. See estimateStability for a method to estimate the optimal number of components.
`center_X`	If `TRUE`, X is centered (i.e., features / rows are transformed to have a mean of 0) prior to ICA. Generally recommended.
`scale_X`	If `TRUE`, X is scaled (i.e., features / rows are transformed to have a standard deviation of 1) before ICA.

assay_name	If X is a SummarizedExperiment, then this should be the name of the assay to be subject to ICA.
...	Additional arguments to be passed to runICA.

Value

A FactorisedExperiment is returned containing the input data (i.e., the original data matrix in addition to other slots if a SummarizedExperiment was used as input). Additionally contains the results of factor analysis, stored in the reduced and loadings slots. The center_X, scale_X and stability slots may also be filled depending on the arguments given to estimateFactors.

Author(s)

Jack Gisby

See Also

runICA(), ica::ica()

Examples

```
# Get a random matrix with rnorm, with 100 rows (features)
# and 20 columns (observations)
X <- ReducedExperiment:::.makeRandomData(100, 20, "feature", "obs")

# Estimate 5 factors based on the data matrix
set.seed(1)
fe_1 <- estimateFactors(X, nc = 5)
fe_1

# Convert the data matrix to a SummarizedExperiment, then estimate 5 factors
se <- SummarizedExperiment(assays = list("normal" = X))
set.seed(1)
fe_2 <- estimateFactors(se, nc = 5)
fe_2
```

---

estimateStability *Estimate stability of factors as a function of the number of components*

Description

Estimates the stability of factors over a range of component numbers to aid in the identification of the optimal factor number. Based on the Most Stable Transcriptome Dimension (MSTD) approach (see details).

Usage

```
estimateStability(
  X,
  min_components = 10,
  max_components = 60,
  by = 2,
  n_runs = 30,
  resample = FALSE,
  mean_stability_threshold = NULL,
  center_X = TRUE,
  scale_X = FALSE,
  assay_name = "normal",
  BPPARAM = BiocParallel::SerialParam(),
  verbose = TRUE,
  ...
)
```

Arguments

X	Either a SummarizedExperiment object or a matrix containing data to be subject to ICA. X should have rows as features and columns as samples.
min_components	The minimum number of components to estimate the stability for.
max_components	The maximum number of components to estimate the stability for.
by	The number by which to increment the numbers of components tested.
n_runs	The number of times to run ICA to estimate factors and quantify stability. Ignored if use_stability is FALSE.
resample	If TRUE, a boostrap approach is used to estimate factors and quantify stability. Else, random initialisation of ICA is employed. Ignored if use_stability is FALSE.
mean_stability_threshold	A threshold for the mean stability of factors.
center_X	If TRUE, X is centered (i.e., features / rows are transformed to have a mean of 0) prior to ICA. Generally recommended.
scale_X	If TRUE, X is scaled (i.e., features / rows are transformed to have a standard deviation of 1) before ICA.
assay_name	If X is a SummarizedExperiment, then this should be the name of the assay to be subject to ICA.
BPPARAM	A class containing parameters for parallel evaluation. Uses SerialParam by default, running only a single ICA computation at a time. Ignored if use_stability is FALSE.
verbose	If TRUE, shows a progress bar that updates for each number of components tested. Note that the time taken may not be linear, because the time taken to run ICA generally increases with the number of components.
...	Additional arguments to be passed to runICA.

Details

Runs the stability-based ICA algorithm (see runICA) for a range of component numbers. Estimates stability for each, allowing for selection of the optimal number of components to be used for ICA. The results of this function can be plotted by plotStability.

This algorithm is based on the Most Stable Transcriptome Dimension (MSTD) approach (https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-017-4112-9).

The function automatically selects a number of components based on `mean_stability_threshold`. However, this choice should be made after visualisating the stabilities as a function of the number of components, which may be done using plotStability. The aformentioned MSTD paper provides additional context and advice for choosing the number of components based on these results.

Value

Returns a list containing:

stability A data.frame indicating factor stabilities as a function of the number of components.

selected_nc a naive estimate for the optimal number of components based on the `mean_stability_threshold`.

Author(s)

Jack Gisby

See Also

`runICA()`, `plotStability()`

Examples

```
# Get a random matrix with rnorm, with 200 rows (features)
# and 100 columns (observations)
X <- ReducedExperiment:::.makeRandomData(200, 100, "feature", "obs")

# Estimate stability across 10 to 30 components
# Note: We could have provided a SummarizedExperiment object instead of a matrix
stab_res_1 <- estimateStability(
    X,
    min_components = 10,
    max_components = 30,
    n_runs = 5,
    verbose = FALSE
)
```

FactorisedExperiment-class

FactorisedExperiment: A container for the results of factor analysis

Description

A container inheriting from the ReducedExperiment class, that contains one or more data matrices, to which factor analysis has been applied to identify a reduced set of features. A FactorisedExperiment can be created directly in a similar manner to a SummarizedExperiment. Alternatively, the estimateFactors function can be used to both apply factor analysis and generate a FactorisedExperiment from the results.

Usage

```
FactorisedExperiment(
  reduced = new("matrix"),
  scale = TRUE,
  center = TRUE,
  loadings = new("matrix"),
  stability = NULL,
  ...
)
```

Arguments

reduced	A matrix, produced by factor analysis, with rows representing samples and columns representing factors.
scale	Either a boolean, representing whether or not the original data has been scaled to unit variance, or a numeric vector indicating the standard deviations of the original features (as produced by scale.)
center	Either a boolean, representing whether or not the original data has been centered to have a mean of 0, or a numeric vector indicating the means of the original features (as produced by scale.)
loadings	A matrix, produced by factor analysis, with rows representing features and columns representing factors.
stability	A vector containing some measure of stability or variance explained for each factor. If factor analysis was performed using estimateFactors and use_stability = TRUE, this slot will indicate the stability of the factors across multiple runs of ICA.
...	Additional arguments to be passed to ReducedExperiment.

Value

Constructor method returns a FactorisedExperiment object.

Author(s)

Jack Gisby

See Also

ReducedExperiment(), ModularExperiment(), estimateFactors()

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of factors

# In this case we use random assay, reduced and loadings data, but in
# practice these will likely be the result of applying some kind of factor
# analysis to the assay data (e.g., gene expression data) from some study.
rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "factor")
rand_loadings <- ReducedExperiment:::.makeRandomData(i, k, "gene", "factor")

fe <- FactorisedExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data,
    loadings = rand_loadings
)

fe
```

---

getAlignedFeatures *Get feature alignments with factors*

Description

Retrieves features (usually genes) and their alignment (`loadings`) with the factors. Allows for the selection of features whose alignments are high relative to other features. Useful for functional interpretation of factors.

Usage

```
## S4 method for signature 'FactorisedExperiment'
getAlignedFeatures(
  object,
  loading_threshold = 0.5,
  proportional_threshold = 0.01,
  feature_id_col = "rownames",
  format = "list",
  center_loadings = FALSE
)
```

Arguments

object	A FactorisedExperiment object.
loading_threshold	A value between 0 and 1 indicating the proportion of the maximal loading to be used as a threshold. A value of 0.5 (default) means that genes will be selected if their factor alignment (derived from the loadings slot) exceeds or equals 50% of the maximally aligned feature.
proportional_threshold	A value between 0 and 1 indicating the maximal proportion of features to be returned. A value of 0.01 (default) means that a maximum of 1% of the input features (usually genes) will be returned for each factor. These will be the genes in the top percentile with respect to the loadings
feature_id_col	The column in rowData(object) that will be used as a feature ID. Setting this to "rownames" (default) instead uses rownames(object).
format	A string specifying the format in which to return the results. See the value section below.
center_loadings	If TRUE, loadings will be centered column-wise to have a mean of 0.

Value

If the format argument is "list", then a list will be returned with an entry for each factor, each containing a vector of input features. Otherwise, if format is "data.frame", a data.frame is returned with a row for each gene-factor combination. The format argument can also be a function to be applied to the output data.frame before returning the results.

Author(s)

Jack Gisby

See Also

getCommonFeatures()

Examples

```
# Get a random matrix with rnorm, with 100 rows (features)
# and 20 columns (observations)
X <- ReducedExperiment:::.makeRandomData(100, 20, "feature", "obs")

# Estimate 5 factors based on the data matrix
fe <- estimateFactors(X, nc = 5)

# Get the genes highly aligned with each factor as a list
aligned_features <- getAlignedFeatures(fe, proportional_threshold = 0.03)
aligned_features

# Can also view as a data.frame
head(getAlignedFeatures(fe, format = "data.frame", proportional_threshold = 0.03))
```

`getCentrality` *Get correlation of features with module eigengenes*

Description

Provides a wrapper around signedKME. Provides a measure of module centrality/connectivity of each feature. Calculates correlation (Pearson's r) of each feature with the module eigengene (i.e., the column of `reduced` to which the feature belongs).

Usage

```
## S4 method for signature 'ModularExperiment'
getCentrality(object, assay_name = "normal", feature_id_col = "rownames")
```

Arguments

`object`	A ModularExperiment object.
`assay_name`	The name of the assay to be used for calculation of module centrality.
`feature_id_col`	The column in `rowData(object)` that will be used as a feature ID. Setting this to "rownames" (default) instead uses `rownames(object)`.

Value

Returns a `data.frame` with columns for `feature`, `r` (signed correlation with the eigengene), `rsq` (squared correlation with the eigengene), `rank_r` (feature rank based on `r`) and `rank_rsq` (feature rank based on rsq).

Author(s)

Jack Gisby

See Also

`WGCNA::signedKME()`

Examples

```
# Create ModularExperiment with random data (100 features, 50 samples,
# 10 modules)
me <- ReducedExperiment:::.createRandomisedModularExperiment(100, 50, 10)
me

# Calculate centrality of each feature for the corresponding module
head(getCentrality(me))
```

getCommonFeatures *Get common factor features*

Description

Function to count how many genes are aligned with multiple factors.

Usage

```
getCommonFeatures(factor_features)
```

Arguments

factor_features	A data.frame as returned by getAlignedFeatures.

Value

A data.frame for each factor pair with the numbers and proportions of the genes in the input that overlap.

Author(s)

Jack Gisby

See Also

plotCommonFeatures(), getAlignedFeatures()

Examples

```
# Get a random matrix with rnorm, with 100 rows (features)
# and 20 columns (observations)
X <- ReducedExperiment:::.makeRandomData(100, 20, "feature", "obs")

# Estimate 5 factors based on the data matrix
fe <- estimateFactors(X, nc = 5)

# Get the genes highly aligned with each factor
aligned_features <- getAlignedFeatures(
    fe,
    format = "data.frame",
    proportional_threshold = 0.3
)

# Identify overlap between common features for each factor
common_features <- getCommonFeatures(aligned_features)
head(common_features)
```

getGeneIDs *Gets alternative gene annotations from biomaRt*

Description

Uses getBM to get alternative gene IDs for ReducedExperiment objects. The new annotations are added as columns to the input object's rowData

Usage

```
## S4 method for signature 'ReducedExperiment'
getGeneIDs(
  object,
  gene_id_col = "rownames",
  gene_id_type = "ensembl_gene_id",
  ids_to_get = c("hgnc_symbol", "entrezgene_id"),
  dataset = "hsapiens_gene_ensembl",
  mart = NULL,
  biomart_out = NULL
)
```

Arguments

object	ReducedExperiment object.
gene_id_col	The column in rowData(object) that will be used to query biomaRt. Setting this to "rownames" instead uses rownames(object) for matching.
gene_id_type	The type of attribute to be used to query with biomaRt. See the filters argument of getBM.
ids_to_get	The type of attribute to get from biomaRt. See the attributes argument of getBM.
dataset	The Ensembl dataset to retrieve. See the dataset argument of useEnsembl. If mart is not NULL, this argument is ignored.
mart	An optional mart object to use. See the mart argument of getBM. If provided, this object is used to query biomart for the conversion of gene IDs. If biomart_out is not NULL, this argument is ignored.
biomart_out	An optional data.frame containing the output of a call to getBM. If provided, this object is used for the conversion of gene IDs.

Value

Returns the original object, with additional variables added to the rowData slot.

Author(s)

Jack Gisby

See Also

biomaRt::useEnsembl(), biomaRt::getBM()

Examples

```
set.seed(2)
airway <- ReducedExperiment:::.getAirwayData(n_features = 500)

set.seed(1)
airway_fe <- estimateFactors(airway, nc = 2, use_stability = FALSE, method = "imax")

# rowData before getting additional gene IDs
rowData(airway_fe)

# For this example we run `getGeneIDs` using a preloaded biomart query
# (`biomart_out`) to avoid actually querying ensembl during testing
# Note: do not use this file for your actual data
biomart_out <- readRDS(system.file(
    "extdata",
    "biomart_out.rds",
    package = "ReducedExperiment"
))
airway_fe <- getGeneIDs(airway_fe, biomart_out = biomart_out)

# rowData after getting additional gene IDs
rowData(airway_fe)
```

---

getMsigdbT2G *Get TERM2GENE dataframe from MSigDB*

Description

Gets pathways from the MSigDB database in the format required by `clusterProfiler` enrichment functions, such as enricher and GSEA. May be used as input to runEnrich. By default, retrieves the C2 canonical pathways.

Usage

```
getMsigdbT2G(
  species = "Homo sapiens",
  category = "C2",
  subcategory = NULL,
  subcategory_to_remove = "CGP",
  gene_id = "ensembl_gene"
)
```

Arguments

species	The species for which to obtain MSigDB pathways. See msigdbr for more details.
category	The MSigDB category to retrieve pathways for. See msigdbr for more details.
subcategory	The MSigDB subcategory to retrieve pathways for. See msigdbr for more details.
subcategory_to_remove	If not NULL, this is a character string indicating a subcategory to be removed from the results of msigdbr.
gene_id	The name to be given to the gene_id column of the resulting data.frame.

Value

Returns a data.frame, where the `gs_name` column indicates the name of a pathway, and the `gene_id` column indicates genes that belong to said pathway.

Author(s)

Jack Gisby

Examples

```
pathways <- getMsigdbT2G(
    species = "Homo sapiens",
    category = "C2",
    subcategory_to_remove = "CGP",
    gene_id = "ensembl_gene"
)

# A data.frame indicating gene-pathway mappings for use in pathway analysis
head(pathways)
```

---

identifyModules *Apply dimensionality reduction using Weighted Gene Correlation Network Analysis*

Description

Performs Weighted gene correlation network analysis (WGCNA) and packages both the input data and subsequent results into a ModularExperiment. Calls runWGCNA to perform the analysis; see its documentation page for more information on the ICA method, parameters and outputs.

Usage

```
identifyModules(
  X,
  power,
  center_X = TRUE,
  scale_X = TRUE,
  assay_name = "normal",
  ...
)
```

Arguments

X	Either a SummarizedExperiment object or a matrix containing data to be subject to WGCNA. X should have rows as features and columns as samples.
power	An integer representing the soft-thresholding power to be used to define modules. See the assessSoftThreshold function for aid in determining this parameter.
center_X	If TRUE, X is centered (i.e., features / rows are transformed to have a mean of 0) prior to WGCNA.
scale_X	If TRUE, X is scaled (i.e., features / rows are transformed to have a standard deviation of 1) before WGCNA.
assay_name	If X is a SummarizedExperiment, then this should be the name of the assay to be subject to WGCNA.
...	Additional arguments to be passed to runWGCNA.

Value

A ModularExperiment is returned containing the input data (i.e., the original data matrix in addition to other slots if a SummarizedExperiment was used as input). Additionally contains the results of module analysis, stored in the reduced and assignments slots. The center_X, scale_X, loadings, threshold and dendrogram slots may also be filled depending on the arguments given to identifyModules.

Author(s)

Jack Gisby

See Also

runWGCNA(), WGCNA::blockwiseModules(), WGCNA::pickSoftThreshold()

Examples

```
# Get the airway data as a SummarizedExperiment (with a subset of features)
set.seed(2)
airway_se <- ReducedExperiment:::.getAirwayData(n_features = 500)

# Select soft-thresholding power to use (use capture.output to hide WGCNA's prints)
WGCNA::disableWGCNAThreads()
```

```
invisible(capture.output(fit_indices <- assessSoftThreshold(airway_se)))
estimated_power <- fit_indices$Power[fit_indices$estimated_power]

# Identify modules using WGCNA
airway_me <- identifyModules(airway_se, verbose = 0, power = estimated_power)
airway_me
```

`loadings,FactorisedExperiment-method`
Get and set loadings

Description

Method for getting and setting loadings for a ReducedExperiment object.

Usage

```
## S4 method for signature 'FactorisedExperiment'
loadings(
  object,
  scale_loadings = FALSE,
  center_loadings = FALSE,
  abs_loadings = FALSE
)

## S4 replacement method for signature 'FactorisedExperiment'
loadings(object) <- value

## S4 method for signature 'ModularExperiment'
loadings(
  object,
  scale_loadings = FALSE,
  center_loadings = FALSE,
  abs_loadings = FALSE
)

## S4 replacement method for signature 'ModularExperiment'
loadings(object) <- value
```

Arguments

`object`	ReducedExperiment object or an object that inherits from this class.
`scale_loadings`	If `TRUE`, loadings will be scaled to have a standard deviation of 0. If the loadings are a matrix, this operation is performed column-wise.
`center_loadings`	If `TRUE`, loadings will be centered to have a mean of 0. If the loadings are a matrix, this operation is performed column-wise.

abs_loadings	If TRUE, the absolute values of the loadings will be returned.
value	New value to replace existing loadings.

Details

When available, the module loadings provide the values of the rotation matrix (usually generated by prcomp) used to calculate the sample-level module vectors available in the reduced slot. Normally, these loadings are calculated for each module separately, so their values are not comparable across modules.

Value

If object is a FactorisedExperiment, the loadings matrix will be returned, with features as rows and reduced components as columns. If object is a ModularExperiment, the loadings will be returned as a vector, with a value for each feature (usually genes).

Author(s)

Jack Gisby

Examples

```
# Create ModularExperiment with random data (100 features, 50 samples,
# 10 modules)
me <- ReducedExperiment:::.createRandomisedModularExperiment(100, 50, 10)
me

# Retrieve the loadings
loadings(me)[1:10]

# Change a loading
loadings(me)[9] <- 8
loadings(me)[1:10]
```

ModularExperiment-class

ModularExperiment: A container for the results of module analysis

---

Description

A container inheriting from the ReducedExperiment class, that contains one or more data matrices, to which module analysis has been applied to identify a reduced set of features. A ModularExperiment can be created directly in a similar manner to a SummarizedExperiment. Alternatively, the identifyModules function can be used to both define modules and generate a ModularExperiment from the results.

Usage

```
ModularExperiment(
  reduced = new("matrix"),
  scale = TRUE,
  center = TRUE,
  loadings = NULL,
  assignments = character(),
  dendrogram = NULL,
  threshold = NULL,
  ...
)
```

Arguments

reduced	A matrix, produced by module analysis, with rows representing samples and columns representing module expression profiles. Typically, this matrix contains "eigengenes" produced by the Weighted Gene Correlation Network Analysis (WGCNA) approach, as is applied by identifyModules.
scale	Either a boolean, representing whether or not the original data has been scaled to unit variance, or a numeric vector indicating the standard deviations of the original features (as produced by scale.)
center	Either a boolean, representing whether or not the original data has been centered to have a mean of 0, or a numeric vector indicating the means of the original features (as produced by scale.)
loadings	A numeric vector representing the loadings used to generate module expression profiles. Typically, these values are obtained from the rotation matrix produced by prcomp, which is used to identify the first principal component of each module. The vector names represent features.
assignments	A vector of features, named according to the module to which the feature belongs.
dendrogram	Either NULL, or the dendrogram used to identify modules from the original data.
threshold	Either NULL, or a matrix produced by pickSoftThreshold indicating the parameters used for network construction.
...	Additional arguments to be passed to ReducedExperiment.

Value

Constructor method returns a ModularExperiment object.

Author(s)

Jack Gisby

See Also

ReducedExperiment(), FactorisedExperiment(), identifyModules()

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of modules

# In this case we use random assay data and reduced data (i.e., module
# eigengenes). We also randomly assign each feature to a module. In practice,
# we would identify modules and eigengenes using a method like WGCNA applied
# to the analysis of assay data (e.g., gene expression data) from some study.
rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "module")
rand_assignments <- paste0("gene_", seq_len(i))
names(rand_assignments) <- paste0("module_", round(stats::runif(i, 1, k), 0))

me <- ModularExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data,
    assignments = rand_assignments
)

me
```

`modulePreservation` *Get module preservation statistics*

Description

Tests whether a set of modules defined in the reference dataset are preserved in the test dataset. Provides a convenient wrapper around modulePreservation for ModularExperiment and SummarizedExperiment objects.

Usage

```
modulePreservation(
  reference_dataset,
  test_dataset,
  reference_assay_name = "normal",
  test_assay_name = "normal",
  module_assignments = NULL,
  greyName = "module_0",
  goldName = "random",
  networkType = "signed",
  corFnc = "cor",
  savePermutedStatistics = FALSE,
  ...
)
```

Arguments

`reference_dataset`
: The dataset that was used to define the modules. Must be a `data.frame` or `matrix` with features as rows and samples as columns, or a ModularExperiment or SummarizedExperiment object.

`test_dataset`
: The dataset that will be used to test for module preservation. Must be a `data.frame` or `matrix` with features as rows and samples as columns, or a SummarizedExperiment object. The features of `test_dataset` should be the same as `reference_dataset` and in the same order.

`reference_assay_name`
: If the reference dataset is a ModularExperiment or SummarizedExperiment object, this argument specifies which assay slot was used to define the modules.

`test_assay_name`
: If the reference dataset is a ModularExperiment or SummarizedExperiment object, this argument specifies which assay slot is to be used in preservation tests.

`module_assignments`
: If the reference dataset is not a ModularExperiment object, this argument is necessary to specify the module assignments.

`greyName`
: The name of the "module" of unassigned genes. Usually "module_0" (ReducedExperiment default) or "grey" (WGCNA default). See modulePreservation.

`goldName`
: The name to be used for the "gold" module (which is made up of a random sample of all network genes). See modulePreservation.

`networkType`
: A string referring to the type of WGCNA network used for the reference and test datasets. One of"unsigned", "signed" or "signed hybrid". See adjacency. See modulePreservation.

`corFnc`
: A string referring to the function to be used to calculate correlation. One of "cor" or "bicor". See modulePreservation.

`savePermutedStatistics`
: If `TRUE`, saves the permutation statistics as a .RData file. See modulePreservation.

`...`
: Additional arguments to be passed to modulePreservation.

Value

A `data.frame` containing preservation statistics, as described by modulePreservation.

Author(s)

Jack Gisby

Examples

```
# Get random ModularExperiments with rnorm, with 100 rows (features),
# 20 columns (observations) and 5/10 modules
me_1 <- ReducedExperiment:::.createRandomisedModularExperiment(100, 20, 5)
me_2 <- ReducedExperiment:::.createRandomisedModularExperiment(100, 20, 10)
```

```
# Test module preservation (test modules from dataset 1 in dataset 2)
mp <- modulePreservation(me_1, me_2, verbose = 0, nPermutations = 3)
```

---

names<-,FactorisedExperiment-method
Get feature names

Description

Gets and sets feature names (i.e., rownames, usually genes).

Usage

```
## S4 replacement method for signature 'FactorisedExperiment'
names(x) <- value

## S4 replacement method for signature 'FactorisedExperiment'
featureNames(x) <- value

## S4 replacement method for signature 'FactorisedExperiment'
rownames(x) <- value

## S4 replacement method for signature 'ModularExperiment'
names(x) <- value

## S4 replacement method for signature 'ModularExperiment'
featureNames(x) <- value

## S4 replacement method for signature 'ModularExperiment'
rownames(x) <- value

## S4 method for signature 'ReducedExperiment'
featureNames(x)

## S4 replacement method for signature 'ReducedExperiment'
names(x) <- value

## S4 replacement method for signature 'ReducedExperiment'
rownames(x) <- value

## S4 replacement method for signature 'ReducedExperiment'
ROWNAMES(x) <- value

## S4 replacement method for signature 'ReducedExperiment'
featureNames(x) <- value
```

Arguments

x	ReducedExperiment object.
value	New value to replace existing names.

Value

A vector containing the names of the features.

Author(s)

Jack Gisby

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of factors

rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "component")

re <- ReducedExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data
)

# Methods return equivalent results
stopifnot(all.equal(featureNames(re), rownames(rand_assay_data)))
stopifnot(all.equal(rownames(re), rownames(rand_assay_data)))
stopifnot(all.equal(names(re), rownames(rand_assay_data)))

# We can change the feature name at a particular position
print(paste0("Feature name at position 55: ", featureNames(re)[55]))
featureNames(re)[55] <- "custom_feature_name"
print(paste0("Reduced data at position 55: ", featureNames(re)[55]))
```

---

nModules,ModularExperiment-method

Prints individual lengths of samples, components and features

---

Description

Prints individual lengths of samples, components and features

Usage

```
## S4 method for signature 'ModularExperiment'
nModules(object)

## S4 method for signature 'ReducedExperiment'
nComponents(object)

## S4 method for signature 'ReducedExperiment'
nSamples(object)

## S4 method for signature 'ReducedExperiment'
nFeatures(object)
```

Arguments

object	ReducedExperiment object.

Value

The number of samples (nSamples), features (nFeatures) or dimensionally-reduced components (nComponents) are returned.

Author(s)

Jack Gisby

See Also

dim,ReducedExperiment-method

Examples

```
# Create a randomised ReducedExperiment
re <- ReducedExperiment:::.createRandomisedReducedExperiment(100, 50, 10)

# Get the dimensions
nComponents(re) # 10
nSamples(re) # 50
nFeatures(re) # 10

# For a ModularExperiment we can alternatively use nModules
me <- ReducedExperiment:::.createRandomisedModularExperiment(100, 50, 10)
nComponents(me) # 10
nModules(me) # 10
```

`plotCommonFeatures` *Heatmap comparing commonality across factors*

Description

Heatmap comparing commonality across factors

Usage

```
plotCommonFeatures(
  common_features,
  filename = NA,
  color = (grDevices::colorRampPalette(RColorBrewer::brewer.pal(n = 7, name =
    "YlOrRd")))(100)
)
```

Arguments

`common_features`	The output of getCommonFeatures.
`filename`	The path at which to save the plot.
`color`	The colour palette to be used in the heatmap.

Value

An object generated by pheatmap.

Author(s)

Jack Gisby

See Also

`getCommonFeatures()`, `getAlignedFeatures()`

Examples

```
# Get a random matrix with rnorm, with 100 rows (features)
# and 20 columns (observations)
X <- ReducedExperiment:::.makeRandomData(100, 20, "feature", "obs")

# Estimate 5 factors based on the data matrix
fe <- estimateFactors(X, nc = 5)

# Get the genes highly aligned with each factor
aligned_features <- getAlignedFeatures(
    fe,
    format = "data.frame",
```

```
    proportional_threshold = 0.3
)

# Identify overlap between common features for each factor
common_features <- getCommonFeatures(aligned_features)

# Plot the common features as a heatmap
plotCommonFeatures(common_features)
```

`plotDendro` *Plot a dendrogram stored in a ModularExperiment*

Description

Plots the dendrogram in the `dendrogam` slot of a ModularExperiment object using the plotDendroAndColors function.

Usage

```
## S4 method for signature 'ModularExperiment'
plotDendro(
  object,
  groupLabels = "Module colors",
  dendroLabels = FALSE,
  hang = 0.03,
  addGuide = TRUE,
  guideHang = 0.05,
  color_func = WGCNA::labels2colors,
  modules_are_colors = FALSE,
  ...
)
```

Arguments

`object`	ModularExperiment object.
`groupLabels`	Module label axis label. See plotDendroAndColors.
`dendroLabels`	If `TRUE`, shows feature names in the dendrogram. See plotDendroAndColors.
`hang`	The fraction of the plot height by which labels should hang below the rest of the plot. See plot.hclust.
`addGuide`	If `TRUE`, adds vertical guide lines to the dendrogram. See plotDendroAndColors.
`guideHang`	The fraction of the dendrogram's height to leave between the top end of the guide line and the dendrogram merge height. See plotDendroAndColors.
`color_func`	Function for converting module names to colors. Only used if `modules_are_colors` is `FALSE`.

modules_are_colors
: If TRUE, expects the module names to be colors. Else, assumes that module names are are numbers that can be converted into colours by color_func.

...
: Additional arguments to be passed to plotDendroAndColors.

Value

A plot produced by plotDendroAndColors.

Author(s)

Jack Gisby

See Also

WGCNA::plotDendroAndColors(), plot.hclust

Examples

```
# Create ModularExperiment with random data (100 features, 50 samples,
# 10 modules)
me <- ReducedExperiment:::.createRandomisedModularExperiment(100, 50, 10)
me

# The dendrogram is usually produced during module discovery, but we can
# assign any dendrogram to the slot. Let's do hierarchical clustering on the
# features in our object and assign it
dendrogram(me) <- hclust(dist(assay(me)))
dendrogram(me)

# Plot the dendrogram - modules are random in this instance, but in general
# features within a module should cluster together
plotDendro(me)
```

---

plotModulePreservation

Plot module preservation statistics

Description

Plot module preservation statistics

Usage

```
plotModulePreservation(
  modulePreservation_results,
  show_random = TRUE,
  remove_module = NULL
)
```

Arguments

modulePreservation_results	The output of modulePreservation
show_random	If TRUE, shows the random module in the plots.
remove_module	The name of a module to be hidden from the plots.

Value

Two ggplot2 plot objects combined by patchwork. Plots the module preservation statistics generated by modulePreservation.

Author(s)

Jack Gisby

Examples

```
# Get random ModularExperiments with rnorm, with 100 rows (features),
# 20 columns (observations) and 5/10 modules
me_1 <- ReducedExperiment:::.createRandomisedModularExperiment(100, 20, 5)
me_2 <- ReducedExperiment:::.createRandomisedModularExperiment(100, 20, 10)

# Test module preservation (test modules from dataset 1 in dataset 2)
mp <- modulePreservation(me_1, me_2, verbose = 0, nPermutations = 3)

# No significant preservation, since these were random modules
plotModulePreservation(mp)
```

---

plotStability *Plot component stability as a function of the number of components*

Description

Plots the results of estimateStability. See this function's documentation for more information.

Usage

```
plotStability(
  stability,
  plot_path = NULL,
  stability_threshold = NULL,
  mean_stability_threshold = NULL,
  height = 4,
  width = 10,
  ...
)
```

Arguments

`stability`	The results of estimateStability.
`plot_path`	The path at which the plot will be saved
`stability_threshold`	Plots a stability threshold, below which components can be pruned by runICA.
`mean_stability_threshold`	Plots a stability threshold, which is used by estimateStability to provide a naive estimate for the optimal number of components.
`height`	The height of the plot, to be passed to ggsave.
`width`	The width of the plot, to be passed to ggsave.
`...`	Additional arguments to be passed to ggsave.

Value

Returns a list of three plots as `ggplot2` objects:

combined_plot The two other plots combined with patchwork.

stability_plot A plot in which each line indicates stability as a function of the number of components. A line is shown for each number of components tested.

mean_plot The average component stability as a function of the number of components.

Author(s)

Jack Gisby

Examples

```
# Get a random matrix with rnorm, with 200 rows (features)
# and 100 columns (observations)
X <- ReducedExperiment:::.makeRandomData(200, 100, "feature", "obs")

# Estimate stability across 10 to 30 components
stab_res <- estimateStability(
    X,
    min_components = 10,
    max_components = 30,
    n_runs = 5,
    verbose = FALSE
)

# Intracluster stability similar to extracluster since this is random data
plotStability(stab_res)$combined_plot
```

projectData *Project new data using pre-defined factors*

Description

Uses a projection approach to calculate factors in new data. Functions in a similar fashion to the `predict` method of prcomp. The transposed `newdata` are multiplied by the original loadings matrix.

Usage

```
## S4 method for signature 'FactorisedExperiment,matrix'
projectData(
  object,
  newdata,
  standardise_reduced = TRUE,
  scale_newdata = NULL,
  center_newdata = NULL
)

## S4 method for signature 'FactorisedExperiment,data.frame'
projectData(
  object,
  newdata,
  standardise_reduced = TRUE,
  scale_newdata = NULL,
  center_newdata = NULL
)

## S4 method for signature 'FactorisedExperiment,SummarizedExperiment'
projectData(
  object,
  newdata,
  standardise_reduced = TRUE,
  scale_newdata = NULL,
  center_newdata = NULL,
  assay_name = "normal"
)

## S4 method for signature 'FactorisedExperiment'
predict(object, newdata, ...)
```

Arguments

`object`	A FactorisedExperiment object. The `loadings` slot of this class will be used for projection. Additionally, by default, the `scale` and `center` slots are used to apply the original transformation to the new data.

newdata	New data for projection. Must be a data.frame or matrix with features as rows and samples as columns, or a SummarizedExperiment object. Assumes that the rows of newdata match those of the FactorisedExperiment object.
standardise_reduced	Whether or not the reduced data should be standardised (i.e., transformed to have a mean of 0 and standard deviation of 1) after calculation.
scale_newdata	Controls whether the newdata are scaled. If NULL, performs scaling based on the FactorisedExperiment object's scale slot. The value of this argument will be passed to the scale argument of scale.
center_newdata	Controls whether the newdata are centered If NULL, performs centering based on the FactorisedExperiment object's center slot. The value of this argument will be passed to the center argument of scale.
assay_name	If a SummarizedExperiment object is passed as new data, this argument indicates which assay should be used for projection.
...	Additional arguments to be passed to projectData.

Details

If scale_newdata and center_newdata are left as NULL, then the projection method assumes that the newdata are on the same scale as the original data of the object. It will therefore use the values of the center and scale slots of the object. For instance, if the scale slot is TRUE, the newdata will be scaled. If the scale slot is a vector, the values of this vector will be applied to scale the newdata.

Value

Calculates a matrix with samples as rows and factors as columns. If newdata was a matrix or data.frame, this will be returned as a matrix. If a SummarizedExperiment object was passed instead, then a FactorisedExperiment object will be created containing this matrix in its reduced slot.

Author(s)

Jack Gisby

See Also

calcEigengenes, stats::prcomp

Examples

```
# Get two random matrices with rnorm
# 1: 100 rows (features) and 20 columns (observations)
X_1 <- ReducedExperiment:::.makeRandomData(100, 20, "feature", "obs")

# Both matrices must have the same features, but they may have different obs
# 2: 100 rows (features) and 30 columns (observations)
X_2 <- ReducedExperiment:::.makeRandomData(100, 30, "feature", "obs")
```

```
# Estimate 5 factors based on the data matrix
fe_1 <- estimateFactors(X_1, nc = 5)
fe_1

# Project the fe_1 factors for the samples in X_2
projected_data <- projectData(fe_1, X_2)
projected_data
```

`reduced` *Get and set reduced data*

Description

Retrieves the reduced data matrix, with samples as rows and reduced components as columns. Setter method can be used to replace or modify all or part of the matrix.

Usage

```
## S4 method for signature 'ReducedExperiment'
reduced(object, scale_reduced = FALSE, center_reduced = FALSE)

## S4 replacement method for signature 'ReducedExperiment'
reduced(object) <- value
```

Arguments

`object`	An object that inherits from ReducedExperiment.
`scale_reduced`	If `TRUE`, data will be scaled column-wise to have a standard deviation of 0.
`center_reduced`	If `TRUE`, data will be centered column-wise to have a mean of 0.
`value`	New value to replace existing reduced data matrix.

Value

A matrix with samples/observations as rows and columns referring to the dimensionally-reduced components.

Author(s)

Jack Gisby

See Also

`ReducedExperiment()`

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of factors

rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "component")

re <- ReducedExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data
)

stopifnot(all.equal(reduced(re), rand_reduced_data))

print(paste0("Reduced data at position (2, 2): ", reduced(re)[2, 2]))
reduced(re)[2, 2] <- 5
print(paste0("Reduced data at position (2, 2): ", reduced(re)[2, 2]))
```

`ReducedExperiment-class` *ReducedExperiment: A container for dimensionally-reduced representations*

Description

Inherits from SummarizedExperiment, a container for one or more matrices with features as rows (e.g., genes) and columns as samples. Additional information on features and samples are contained in DataFrame tables. The `ReducedExperiment` extends SummarizedExperiment by additionally providing access to a "reduced" data matrix, in which rows represent samples and columns represent a second set of dimensionally-reduced features.

The methods available for SummarizedExperiment objects are also available for `ReducedExperiment` and its children, which include FactorisedExperiment and ModularExperiment.

Typically, `ReducedExperiment` objects contain two main assays. The first is, by default, named "normal" and contains some type of normalised assay data, such as gene expression. The second is "transformed", which is typically the result of applying scaling and/or centering to the normalised data matrix.

Usage

```
ReducedExperiment(reduced = new("matrix"), scale = TRUE, center = TRUE, ...)
```

Arguments

reduced	A matrix, usually the result of some type of dimensionality-reduction, with rows representing samples and columns representing a new set of features.
scale	Either a boolean, representing whether or not the original data has been scaled to unit variance, or a numeric vector indicating the standard deviations of the original features (as produced by scale.)
center	Either a boolean, representing whether or not the original data has been centered to have a mean of 0, or a numeric vector indicating the means of the original features (as produced by scale.)
...	Additional arguments to be passed to SummarizedExperiment.

Value

Constructor method returns a ReducedExperiment object.

Author(s)

Jack Gisby

See Also

FactorisedExperiment(), ModularExperiment()

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of factors

# In this case we use random assay and reduced data, but in
# practice these will likely be the result of applying some kind of
# dimensionality-reduction method to the assay data (e.g., gene
# expression data) from some study.
rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "component")

re <- ReducedExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data
)

re
```

runEnrich *Functional enrichment analyses for dimensionally-reduced data*

Description

Method for applying pathway enrichment analysis to components identified through dimensionality reduction (e.g., factors or modules). Enrichment analyses are applied to each component separately.

Usage

```
## S4 method for signature 'FactorisedExperiment'
runEnrich(
  object,
  method = "gsea",
  feature_id_col = "rownames",
  center_loadings = FALSE,
  abs_loadings = FALSE,
  loading_threshold = 0.5,
  proportional_threshold = 0.01,
  as_dataframe = FALSE,
  ...
)

## S4 method for signature 'ModularExperiment'
runEnrich(
  object,
  method = "overrepresentation",
  feature_id_col = "rownames",
  as_dataframe = FALSE,
  ...
)
```

Arguments

object	FactorisedExperiment or ModularExperiment object.
method	The method to use for identifying enriched pathways. One of "overrepresentation" or "gsea". The "overrepresentation" method calls enricher whereas the "gsea" method calls GSEA. Note that "gsea" is not available for modules.
feature_id_col	The column in `rowData(object)` that will be used as a feature ID. Setting this to `"rownames"` (default) instead uses `rownames(object)`.
center_loadings	If `TRUE`, loadings will be centered column-wise to have a mean of 0.
abs_loadings	If `TRUE`, the absolute values of the loadings will be used for enrichment analysis. If `FALSE`, the signed loadings will be used for GSEA enrichment. Note that, regardless of the value of this term, the process used to select genes for overrepresentation analysis will be based on absolute loadings.

`loading_threshold`
: See getAlignedFeatures. Only relevant for overrepresentation analysis.

`proportional_threshold`
: See getAlignedFeatures. Only relevant for overrepresentation analysis.

`as_dataframe`
: If `TRUE`, the results will be returned as a data.frame. Otherwise, the results will be returned as a list of objects created by either enricher, in the case of overrepresentation analysis, or GSEA, in the case of GSEA.

`...`
: Additional arguments to be passed to GSEA (if `method == "gsea"`) or enricher (if `method == "overrepresentation"`).

Details

When running module analysis, the overrepresentation method identifies pathways that are overrepresented in each module.

For factor analysis, the overrepresentation method first identifies the genes most highly aligned with each factor (using getAlignedFeatures), then uses the resulting gene lists to perform overrepresentation analysis. The GSEA method instead uses the entire set of factor loadings, and identifies pathways that are overrepresented in the tails of this distribution.

Value

If `as_dataframe` is `TRUE`, the results will be returned as a data.frame. Otherwise, the results will be returned as a list of objects created by either enricher, in the case of overrepresentation analysis, or GSEA, in the case of GSEA.

Author(s)

Jack Gisby

See Also

`getMsigdbT2G()`

Examples

```
set.seed(2)
airway <- ReducedExperiment:::.getAirwayData(n_features = 2000)
airway_fe <- estimateFactors(
    airway,
    nc = 2,
    use_stability = FALSE,
    method = "imax"
)

# Get pathways (e.g., by using ReducedExperiment::getMsigdbT2G())
t2g <- read.csv(system.file(
    "extdata",
    "msigdb_t2g_filtered.csv",
    package = "ReducedExperiment"
))
```

```
# Run overrepresentation analysis
overrep_res <- runEnrich(
    airway_fe,
    method = "overrepresentation",
    feature_id_col = "rownames",
    as_dataframe = TRUE,
    p_cutoff = 0.1,
    TERM2GENE = t2g,
    universe = rownames(airway_fe)
)

head(overrep_res)
```

`runICA` *Run standard or stabilised Independent Component Analysis*

Description

Runs ICA through ica. If `use_stability` is FALSE, then `X` is passed directly to ica and a standard ICA analysis is performed. If `use_stability` is `TRUE`, then the stabilised ICA procedure is carried out (see `details`).

Usage

```
runICA(
  X,
  nc,
  use_stability = FALSE,
  resample = FALSE,
  method = "fast",
  stability_threshold = NULL,
  center_X = TRUE,
  scale_X = FALSE,
  reorient_skewed = TRUE,
  scale_components = TRUE,
  scale_reduced = TRUE,
  n_runs = 30,
  BPPARAM = BiocParallel::SerialParam(),
  ...
)
```

Arguments

`X`	Either a SummarizedExperiment object or a matrix containing data to be subject to ICA. `X` should have rows as features and columns as samples.

nc — The number of components to be identified. See estimateStability for a method to estimate the optimal number of components.

use_stability — Whether to use a stability-based approach to estimate factors. See details for further information.

resample — If TRUE, a boostrap approach is used to estimate factors and quantify stability. Else, random initialisation of ICA is employed. Ignored if use_stability is FALSE.

method — The ICA method to use. Passed to ica, the options are "fast", "imax" or "jade".

stability_threshold — A stability threshold for pruning factors. Factors with a stability below this threshold will be removed. If used, the threshold can lead to fewer factors being returned than that specified by nc.

center_X — If TRUE, X is centered (i.e., features / rows are transformed to have a mean of 0) prior to ICA. Generally recommended.

scale_X — If TRUE, X is scaled (i.e., features / rows are transformed to have a standard deviation of 1) before ICA.

reorient_skewed — If TRUE, factors are reorientated to ensure that the loadings of each factor (i.e., the source signal matrix) have positive skew. Helps ensure that the most influential features for each factor are positively associated with it.

scale_components — If TRUE, the loadings are standardised (to have a mean of 0 and standard deviation of 1).

scale_reduced — If TRUE, the reduced data (mixture matrix) are standardised (to have a mean of 0 and standard deviation of 1).

n_runs — The number of times to run ICA to estimate factors and quantify stability. Ignored if use_stability is FALSE.

BPPARAM — A class containing parameters for parallel evaluation. Uses SerialParam by default, running only a single ICA computation at a time. Ignored if use_stability is FALSE.

... — Additional arguments to be passed to ica.

Details

Function performs ICA for a data matrix. If use_stability is TRUE, then ICA is performed multiple times with either: i) random initialisation (default); or ii) bootstrap resampling of the data (if resample is TRUE).

Note that the seed must be set if reproducibility is needed. Specifically, one can use set.seed prior to running standard ICA (use_stability = FALSE) or set the RNGseed argument of BPPARAM when running stabilised ICA (use_stability = TRUE).

The stability-based ICA algorithm is similar to the the ICASSO approach (https://www.cs.helsinki.fi/u/ahyvarin/papers/Himberg03.pd) that is implemented in the stabilized-ica Python package (https://github.com/ncaptier/stabilized-ica/tree/master).

In short, the stability-based algorithm consists of:

- Running ICA multiple times with either random initialisation or bootstrap resampling of the input data.
- Clustering the resulting factors across all runs based on the signature matrix.
- Calculating intra- (aics) and extra- (aecs) cluster stability, and defining the final cluster stability as `aics - aecs`.
- Calculating the cluster centrotype as the factor with the highest intra-cluster stability.
- Optionally removing factors below a specified stability threshold (`stability_threshold`).

Results from this function should be broadly similar to those generated by other implementations of stabilised ICA, although they will not be identical. Notable differences include:

ICA algorithm Differences in the underlying implementation of ICA.

Stability threshold The `stability_threshold` argument, if specified, removes unstable components. Such a threshold is not used by stabilized-ica.

Mixture matrix recovery ICA is generally formulated as `X = MS`, where `X` is the input data, `M` is the mixture matrix (reduced data) and `S` is the source signal matrix (feature loadings). The stabilised ICA approach first calculates a source signal matrix before recovering the mixture matrix. To do this, other implementations, including that of the stabilized-ica package, multiply `X` by the pseudo-inverse of `S`. Such an operation is implemented in the `ginv` function of the `MASS` R package. In the development of ReducedExperiment, we noticed that taking the inverse of `S` often failed, particularly when there were correlated factors. For this reason, we instead formulate the mixture matrix as `M = XS`. After standardisation of `M`, both approaches return near-identical results, given that the matrix inverse was successfully calculated.

Value

A list containing the following:

M The mixture matrix (reduced data) with samples as rows and columns as factors.

S The source signal matrix (loadings) with rows as features and columns as factors.

stab If `use_stability` is TRUE, "stab" will be a component of the list. It is a vector indicating the relative stability, as described above.

Author(s)

Jack Gisby

See Also

`ica::ica()`, `estimateStability()`

Examples

```
# Get a random matrix with rnorm, with 100 rows (features)
# and 20 columns (observations)
X <- ReducedExperiment:::.makeRandomData(100, 20, "feature", "obs")

# Run standard ICA on the data with 5 components
```

```
set.seed(1)
ica_res <- runICA(X, nc = 5, use_stability = FALSE)

# Run stabilised ICA on the data with 5 components (low runs for example)
ica_res_stab <- runICA(X, nc = 5, use_stability = TRUE, n_runs = 5,
                       BPPARAM = BiocParallel::SerialParam(RNGseed = 1))
```

---

runWGCNA *Run WGCNA for a data matrix*

Description

Runs WGCNA. Largely a wrapper for the blockwiseModules function that reformats data into a format convenient for creating a ModularExperiment object and changes module names from colours to numbers by default.

Usage

```
runWGCNA(
  X,
  power,
  cor_type = "pearson",
  networkType = "signed",
  module_labels = "numbers",
  maxBlockSize = 30000,
  verbose = 0,
  standardise_reduced = TRUE,
  ...
)
```

Arguments

X	Either a SummarizedExperiment object or a matrix containing data to be subject to WGCNA. X should have rows as features and columns as samples.
power	soft-thresholding power for network construction.
cor_type	The type of correlation to be used to generate a correlation matrix during network formation. One of "pearson" (cor) and "bicor" (bicor).
networkType	network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
module_labels	Specifies whether the modules should be named based on "numbers" or "colours. If module_labels is set to "numbers", then "module_0" represents unclustered genes, whereas if it is set to "colours" then "grey" represents unclustered genes.
maxBlockSize	The chunk size (in terms of the number of features/genes) to process the data. See blockwiseModules for more details. The default (30000) should process standard transcriptomic datasets in a single chunk. Results may differ if the number of features exceeds the chunk size. Lower values of this parameter may use less memory to calculate networks.

`verbose` integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

`standardise_reduced`
If `TRUE`, the reduced data (eigengenes) are standardised to have a mean of 0 and a standard deviation of 1.

`...` Additional arguments to be passed to blockwiseModules.

Details

Note that if `module_labels` is set to "numbers", then "module_0" represents unclustered genes, whereas if it is set to "colours" then "grey" represents unclustered genes.

The function also stores the loadings matrices generated when PCA is performed for each module to calculate eigengenes. These loadings can be used to recalculate the reduced data matrix (eigengenes).

Value

Returns a list containing:

"E" The reduced data (eigengenes).

"L" The module loadings. This represents the values of the PCA rotation matrix for the first principal component generated for each module.

"assignments" A named vector representing the assignments of genes to modules.

Author(s)

Jack Gisby

See Also

`WGCNA::blockwiseModules()`, `assessSoftThreshold()`, `WGCNA::pickSoftThreshold()`,

Examples

```
# Get the airway data as a SummarizedExperiment (with a subset of features)
set.seed(2)
airway_se <- ReducedExperiment:::.getAirwayData(n_features = 500)

# Choose an appropriate soft-thresholding power
WGCNA::disableWGCNAThreads()
fit_indices <- assessSoftThreshold(airway_se)
estimated_power <- fit_indices$Power[fit_indices$estimated_power]

# Identify modules using the airway expression matrix
wgcna_res <- runWGCNA(
    assay(airway_se, "normal"),
    verbose = 0,
    power = estimated_power
)
```

```
# We find just one module for this small dataset (module_0 indicates unclustered genes)
table(names(wgcna_res$assignments))
```

`sampleNames` *Get sample names*

Description

Retrieves sample names (colnames).

Usage

```
## S4 method for signature 'ReducedExperiment'
sampleNames(x)

## S4 replacement method for signature 'ReducedExperiment'
sampleNames(x) <- value

## S4 replacement method for signature 'ReducedExperiment'
colnames(x) <- value
```

Arguments

`x`	ReducedExperiment object.
`value`	New value to replace existing names.

Value

A vector containing the names of the features.

Author(s)

Jack Gisby

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of factors

rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "component")

re <- ReducedExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data
)
```

```
stopifnot(all.equal(sampleNames(re), colnames(rand_assay_data)))
stopifnot(all.equal(colnames(re), colnames(rand_assay_data)))

print(paste0("Sample name at [80]: ", sampleNames(re)[80]))
sampleNames(re)[80] <- "custom_feature_name"
print(paste0("Sample data at [80]: ", sampleNames(re)[80]))
```

---

show *Prints a summary of a ReducedExperiment object*

Description

Prints a summary of a ReducedExperiment object

Usage

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

Arguments

object	ReducedExperiment object.

Value

A character summary describing the object.

Author(s)

Jack Gisby

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of factors

rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "component")

re <- ReducedExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data
)

# Equivalent to `show(re)`
re
```

stability *Get and setting the stability values for factors*

Description

Get and setting the stability values for factors

Usage

```
## S4 method for signature 'FactorisedExperiment'
stability(object)

## S4 replacement method for signature 'FactorisedExperiment'
stability(object) <- value
```

Arguments

object	FactorisedExperiment object.
value	New value to replace existing stability vector.

Value

A vector with a value for each factor indicating the factor stability. More details are available from the estimateStability help page.

Author(s)

Jack Gisby

See Also

estimateStability()

Examples

```
# Get a random matrix with rnorm, with 100 rows (features)
# and 20 columns (observations)
X <- ReducedExperiment:::.makeRandomData(100, 20, "feature", "obs")

# Run stabilised ICA on the data with 5 components
fe <- estimateFactors(X, nc = 5, use_stability = TRUE)

stability(fe)

stability(fe)[2] <- 10
stability(fe)
```

`[,FactorisedExperiment,ANY,ANY,ANY-method`
Extract and replace parts of ReducedExperiment objects

Description

Method permits slicing of ReducedExperiment objects.

Usage

```
## S4 method for signature 'FactorisedExperiment,ANY,ANY,ANY'
x[i, j, k, ..., drop = FALSE]

## S4 replacement method for signature 'FactorisedExperiment,ANY,ANY,FactorisedExperiment'
x[i, j, k, ...] <- value

## S4 method for signature 'ModularExperiment,ANY,ANY,ANY'
x[i, j, k, ..., drop = FALSE]

## S4 replacement method for signature 'ModularExperiment,ANY,ANY,ModularExperiment'
x[i, j, k, ...] <- value

## S4 method for signature 'ReducedExperiment,ANY,ANY,ANY'
x[i, j, k, ..., drop = FALSE]

## S4 replacement method for signature 'ReducedExperiment,ANY,ANY,ReducedExperiment'
x[i, j, k, ...] <- value
```

Arguments

`x`	ReducedExperiment object.
`i`	Slicing by rows (features, usually genes).
`j`	Slicing by columns (samples/observations).
`k`	Slicing by reduced dimensions.
`...`	Additional arguments to be passed to the parent method.
`drop`	Included for consistency with other slicing methods.
`value`	Value to be used to replace part of the object.

Value

A ReducedExperiment object, potentially sliced by rows (`i`), columns (`j`) and components (`k`).

Author(s)

Jack Gisby

Examples

```
# Create randomised data with the following dimensions
i <- 300 # Number of features
j <- 100 # Number of samples
k <- 10 # Number of components (i.e., factors/modules)

rand_assay_data <- ReducedExperiment:::.makeRandomData(i, j, "gene", "sample")
rand_reduced_data <- ReducedExperiment:::.makeRandomData(j, k, "sample", "component")

# Create a randomised ReducedExperiment container
re <- ReducedExperiment(
    assays = list("normal" = rand_assay_data),
    reduced = rand_reduced_data
)

# Slice our object by rows (1:50), columns (1:20) and components (1:5)
# re[i, j, k, ...]
sliced_re <- re[1:50, 1:20, 1:5]
sliced_re

# We can also assign our subsetted object back to the original
re[1:50, 1:20, 1:5] <- sliced_re
re
```

Index