Package 'SingleCellExperiment'

October 16, 2023

Version 1.22.0

Date 2023-03-15

Title S4 Classes for Single Cell Data

Depends SummarizedExperiment

Imports methods, utils, stats, S4Vectors, BiocGenerics, GenomicRanges, DelayedArray

Suggests testthat, BiocStyle, knitr, rmarkdown, Matrix, scRNAseq (>= 2.9.1), Rtsne

biocViews ImmunoOncology, DataRepresentation, DataImport, Infrastructure, SingleCell

Description Defines a S4 class for storing data from single-cell experiments. This includes specialized methods to store and retrieve spike-in information, dimensionality reduction coordinates and size factors for each cell, along with the usual metadata for genes and libraries.

License GPL-3

VignetteBuilder knitr

RoxygenNote 7.2.1

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

git_branch RELEASE_3_17

git_last_commit d477e8e

git_last_commit_date 2023-04-25

Date/Publication 2023-10-15

Author Aaron Lun [aut, cph],
Davide Risso [aut, cre, cph],
Keegan Korthauer [ctb],
Kevin Rue-Albrecht [ctb]

Maintainer Davide Risso <risso.davide@gmail.com>

R topics documented:

`altExps` *Alternative Experiment methods*

Description

In some experiments, different features must be normalized differently or have different row-level metadata. Typical examples would be for spike-in transcripts in plate-based experiments and antibody or CRISPR tags in CITE-seq experiments. These data cannot be stored in the main `assays` of the SingleCellExperiment itself. However, it is still desirable to store these features *somewhere* in the SingleCellExperiment. This simplifies book-keeping in long workflows and ensure that samples remain synchronised.

To facilitate this, the SingleCellExperiment class allows for "alternative Experiments". Nested SummarizedExperiment-class objects are stored inside the SingleCellExperiment object `x`, in a manner that guarantees that the nested objects have the same columns in the same order as those in `x`. Methods are provided to enable convenient access to and manipulation of these alternative Experiments. Each alternative Experiment should contain experimental data and row metadata for a distinct set of features.

Getters

In the following examples, x is a SingleCellExperiment object.

`altExp(x, e, withDimnames=TRUE, withColData=FALSE)`: Retrieves a SummarizedExperiment containing alternative features (rows) for all cells (columns) in x. e should either be a string specifying the name of the alternative Experiment in x to retrieve, or a numeric scalar specifying the index of the desired Experiment, defaulting to the first Experiment is missing.

If `withDimnames=TRUE`, the column names of the output object are set to `colnames(x)`. In addition, if `withColData=TRUE`, `colData(x)` is cbinded to the front of the column data of the output object.

`altExpNames(x)`: Returns a character vector containing the names of all alternative Experiments in x. This is guaranteed to be of the same length as the number of results, though the names may not be unique.

`altExps(x, withDimnames=TRUE, withColData=FALSE)`: Returns a named List of matrices containing one or more SummarizedExperiment objects. Each object is guaranteed to have the same number of columns, in a 1:1 correspondence to those in x.

If `withDimnames=TRUE`, the column names of each output object are set to `colnames(x)`. In addition, if `withColData=TRUE`, `colData(x)` is cbinded to the front of the column data of each output object.

Single-object setter

`altExp(x, e, withDimnames=TRUE, withColData=FALSE) <- value` will add or replace an alternative Experiment in a SingleCellExperiment object x. The value of e determines how the result is added or replaced:

- If e is missing, value is assigned to the first result. If the result already exists, its name is preserved; otherwise it is given a default name `"unnamed1"`.
- If e is a numeric scalar, it must be within the range of existing results, and value will be assigned to the result at that index.
- If e is a string and a result exists with this name, value is assigned to to that result. Otherwise a new result with this name is append to the existing list of results.

value is expected to be a SummarizedExperiment object with number of columns equal to `ncol(x)`. Alternatively, if value is NULL, the alternative Experiment at e is removed from the object.

If `withDimnames=TRUE`, the column names of value are checked against those of x. A warning is raised if these are not identical, with the only exception being when `value=NULL`. This is inspired by the argument of the same name in `assay<-`.

If `withColData=TRUE`, we assume that the left-most columns of `colData(value)` are identical to `colData(x)`. If so, these columns are removed, effectively reversing the `withColData=TRUE` setting for the `altExp` getter. Otherwise, a warning is raised.

Other setters

In the following examples, x is a SingleCellExperiment object.

`altExps(x, withDimnames=TRUE, withColData=FALSE) <- value`**:** Replaces all alterrnative Experiments in `x` with those in `value`. The latter should be a list-like object containing any number of SummarizedExperiment objects with number of columns equal to `ncol(x)`.

If `value` is named, those names will be used to name the alternative Experiments in `x`. Otherwise, unnamed results are assigned default names prefixed with `"unnamed"`.

If `value` is `NULL`, all alternative Experiments in `x` are removed.

If `value` is a Annotated object, any `metadata` will be retained in `altExps(x)`. If `value` is a Vector object, any `mcols` will also be retained.

If `withDimnames=TRUE`, the column names of each entry of `value` are checked against those of `x`. A warning is raised if these are not identical.

If `withColData=TRUE`, we assume that the left-most columns of the `colData` for each entry of `value` are identical to `colData(x)`. If so, these columns are removed, effectively reversing the `withColData=TRUE` setting for the `altExps` getter. Otherwise, a warning is raised.

`altExpNames(x) <- value`**:** Replaces all names for alternative Experiments in `x` with a character vector `value`. This should be of length equal to the number of results currently in `x`.

`removeAltExps(x)` will remove all alternative Experiments from `x`. This has the same effect as `altExps(x) <- NULL` but may be more convenient as it directly returns a SingleCellExperiment.

Main Experiment naming

The alternative Experiments are naturally associated with names (`e` during assignment). However, we can also name the main Experiment in a SingleCellExperiment `x`:

`mainExpName(x) <- value`**:** Set the name of the main Experiment to a non-`NA` string `value`. This can also be used to unset the name if `value=NULL`.

`mainExpName(x)`**:** Returns a string containing the name of the main Experiment. This may also be `NULL` if no name is specified.

The presence of a non-`NULL` main Experiment name is helpful for functions like `swapAltExp`. An appropriate name is automatically added by functions like `splitAltExps`.

Note that, if a SingleCellExperiment is assigned as an alternative Experiment to another SingleCellExperiment via `altExp(x, e) <- value`, no attempt is made to synchronize `mainExpName(value)` with `e`. In such cases, we suggest setting `mainExpName(value)` to `NULL` to avoid any confusion during interpretation.

Author(s)

Aaron Lun

See Also

`splitAltExps`, for a convenient way of adding alternative Experiments from existing features.

`swapAltExp`, to swap the main and alternative Experiments.

Examples

```
example(SingleCellExperiment, echo=FALSE) # Using the class example
dim(counts(sce))

# Mocking up some alternative Experiments.
se1 <- SummarizedExperiment(matrix(rpois(1000, 5), ncol=ncol(se)))
rowData(se1)$stuff <- sample(LETTERS, nrow(se1), replace=TRUE)
se2 <- SummarizedExperiment(matrix(rpois(500, 5), ncol=ncol(se)))
rowData(se2)$blah <- sample(letters, nrow(se2), replace=TRUE)

# Setting the alternative Experiments.
altExp(sce, "spike-in") <- se1
altExp(sce, "CRISPR") <- se2

# Getting alternative Experimental data.
altExpNames(sce)
altExp(sce, "spike-in")
altExp(sce, 2)

# Setting alternative Experimental data.
altExpNames(sce) <- c("ERCC", "Ab")
altExp(sce, "ERCC") <- se1[1:2,]
```

`applySCE` *Applying over parts of a SingleCellExperiment*

Description

Apply a function over the main and alternative Experiments of a SingleCellExperiment.

Usage

```
applySCE(
  X,
  FUN,
  WHICH = altExpNames(X),
  ...,
  MAIN.ARGS = list(),
  ALT.ARGS = list(),
  SIMPLIFY = TRUE
)
```

Arguments

`X`	A SingleCellExperiment object.
`FUN`	A function to apply to each Experiment.

WHICH	A character or integer vector containing the names or positions of alternative Experiments to loop over.
...	Further (named) arguments to pass to all calls to FUN.
MAIN.ARGS	A named list of arguments to pass to FUN for the main Experiment only. Alternatively NULL, in which case the function is *not* applied to the main Experiment.
ALT.ARGS	A named list where each entry is named after an alternative Experiment and contains named arguments to use in FUN for that Experiment.
SIMPLIFY	Logical scalar indicating whether the output should be simplified to a single SingleCellExperiment.

Details

The behavior of this function is equivalent to creating a list containing X as the first entry and altExps(X) in the subsequent entries, and then lapplying over this list with FUN and the specified arguments. In this manner, users can easily apply the same function to all the Experiments (main and alternative) in a SingleCellExperiment object.

Arguments in ... are passed to all calls to FUN. Arguments in MAIN.ARGS are only used in the call to FUN on the main Experiment. Arguments in ALT.ARGS are passed to the call to FUN on the alternative Experiment of the same name. For the last two, any arguments therein will override arguments of the same name in ....

By default, looping is performed over all alternative Experiments, but the order and identities can be changed by setting WHICH. Values of WHICH should be unique if any simplification of the output is desired. If MAIN.ARGS=NULL, the main Experiment is ignored and the function is only applied to the alternative Experiments.

The default of SIMPLIFY=TRUE is intended as a user-level convenience when all calls to FUN return a SingleCellExperiment with the same number of columns, and WHICH itself contains no more than one reference to each alternative Experiment in x. Under these conditions, the results are collated into a single SingleCellExperiment for easier downstream manipulation.

Value

In most cases or when SIMPLIFY=FALSE, a list is returned containing the output of FUN applied to each Experiment. If MAIN.ARGS is not NULL, the first entry corresponds to the result generated from the main Experiment; all other results are generated according to the entries specified in WHICH and are named accordingly.

If SIMPLIFY=TRUE and certain conditions are fulfilled, a SingleCellExperiment is returned where the results of FUN are mapped to the relevant main or alternative Experiments. This mirrors the organization of Experiments in X.

Developer note

When using this function inside other functions, developers should set SIMPLIFY=FALSE to guarantee consistent output for arbitrary WHICH. If simplification is necessary, the output of this function can be explicitly passed to simplifyToSCE, typically with warn.level=3 to throw an appropriate error if simplification is not possible.

Author(s)

Aaron Lun

See Also

simplifyToSCE, which is used when SIMPLIFY=TRUE.

altExps, to manually extract the alternative Experiments for operations.

Examples

```
ncells <- 10
u <- matrix(rpois(200, 5), ncol=ncells)
sce <- SingleCellExperiment(assays=list(counts=u))
altExp(sce, "BLAH") <- SingleCellExperiment(assays=list(counts=u*10))
altExp(sce, "WHEE") <- SingleCellExperiment(assays=list(counts=u/10))

# Here, using a very simple function that just
# computes the mean of the input for each cell.
FUN <- function(y, multiplier=1) {
    colMeans(assay(y)) * multiplier
}

# Applying over all of the specified parts of 'sce'.
applySCE(sce, FUN=FUN)

# Adding general arguments.
applySCE(sce, FUN=FUN, multiplier=5)

# Adding custom arguments.
applySCE(sce, FUN=FUN, MAIN.ARGS=list(multiplier=5))
applySCE(sce, FUN=FUN, ALT.ARGS=list(BLAH=list(multiplier=5)))

# Skipping Experiments.
applySCE(sce, FUN=FUN, MAIN.ARGS=NULL) # skipping the main
applySCE(sce, FUN=FUN, WHICH=NULL) # skipping the alternatives
```

---

colLabels *Get or set column labels*

Description

Get or set column labels in an instance of a SingleCellExperiment class. Labels are expected to represent information about the the biological state of each cell.

Usage

```
## S4 method for signature 'SingleCellExperiment'
colLabels(x, onAbsence = "none")

## S4 replacement method for signature 'SingleCellExperiment'
colLabels(x, ...) <- value
```

Arguments

x	A SingleCellExperiment object.
onAbsence	String indicating an additional action to take when labels are absent: nothing ("none"), a warning ("warn") or an error ("error").
...	Additional arguments, currently ignored.
value	Any vector-like object of length equal to ncol(object), containing labels for all cells. Alternatively NULL, in which case existing label information is removed.

Details

A frequent task in single-cell data analyses is to label cells with some annotation, e.g., cluster identities, predicted cell type classifications and so on. In a SummarizedExperiment, the colData represents the ideal place for such annotations, which can be easily set and retrieved with standard methods, e.g., x$label <- my.labels.

That said, it is desirable to have some informal standardization of the name of the column used to store these annotations as this makes it easier to programmatically set sensible defaults for retrieval of the labels in downstream functions. To this end, the colLabels function will get or set labels from the "label" field of the colData. This considers the use case where there is a "primary" set of labels that represents the default grouping of cells in downstream analyses.

To illustrate, let's say we have a downstream function that accepts a SingleCellExperiment object and requires labels. When defining our function, we can set colLabels(x) as the default value for our label argument. This pattern is useful as it accommodates on-the-fly changes to a secondary set of labels in x without requiring the user to run colLabels(x) <- second.labels, while facilitating convenient use of the primary labels by default.

For developers, onAbsence is provided to make it easier to mandate that x actually has labels. This avoids silent NULL values that flow to the rest of the function and make debugging difficult.

Value

For colLabels, a vector or equivalent is returned containing label assignments for all cells. If no labels are available, a NULL is returned (and/or a warning or error, depending on onAbsence).

For colLabels<-, a modified x is returned with labels in its colData.

Author(s)

Aaron Lun

See Also

SingleCellExperiment, for the underlying class definition.

Examples

```
example(SingleCellExperiment, echo=FALSE) # Using the class example
colLabels(sce) <- sample(LETTERS, ncol(sce), replace=TRUE)
colLabels(sce)
```

`colPairs` *Column pair methods*

Description

Methods to get or set column pairings in a SingleCellExperiment object. These are typically used to store and retrieve relationships between cells, e.g., in nearest-neighbor graphs or for inferred cell-cell interactions.

Getters

In the following examples, `x` is a SingleCellExperiment object.

`colPair(x, type, asSparse=FALSE)`**:** Retrieves a SelfHits object where each entry represents a pair of columns of `x` and has number of nodes equal to `ncol(x)`. `type` is either a string specifying the name of the column pairing in `x` to retrieve, or a numeric scalar specifying the index of the desired result.

If `asSparse=TRUE`, a sparse matrix is returned instead, see below for details.

`colPairNames(x)`**:** Returns a character vector containing the names of all column pairings in `x`. This is guaranteed to be of the same length as the number of results, though the names may not be unique.

`colPairs(x, asSparse=FALSE)`**:** Returns a named List of matrices containing one or more column pairings as SelfHits objects. If `asSparse=FALSE`, each entry is instead a sparse matrix.

When `asSparse=TRUE`, the return value will be a triplet-form sparse matrix where each row/column corresponds to a column of `x`. The values in the matrix will be taken from the first metadata field of the underlying SelfHits object, with an error being raised if the first metadata field is not of an acceptable type. If there are duplicate pairs, only the value from the last pair is used. If no metadata is available, the matrix values are set to `TRUE` for all pairs.

Single setter

`colPair(x, type) <- value` will add or replace a column pairing in a SingleCellExperiment object `x`. The value of `type` determines how the pairing is added or replaced:

- If `type` is missing, `value` is assigned to the first pairing. If the pairing already exists, its name is preserved; otherwise it is given a default name `"unnamed1"`.

- If `type` is a numeric scalar, it must be within the range of existing pairings, and `value` will be assigned to the pairing at that index.
- If `type` is a string and a pairing exists with this name, `value` is assigned to to that pairing. Otherwise a new pairing with this name is append to the existing list of pairings.

`value` is expected to be a SelfHits with number of nodes equal to `ncol(x)`. Any number of additional fields can be placed in `mcols(value)`. Duplicate column pairs are supported and will not be collapsed into a single entry.

`value` may also be a sparse matrix with number of rows and columns equal to `ncol(x)`. This is converted into a SelfHits object with values stored in the metadata as the `"x"` field.

Alternatively, if `value` is `NULL`, the pairings corresponding to `type` are removed from `x`.

Other setters

In the following examples, `x` is a SingleCellExperiment object.

`colPairs(x) <- value`: Replaces all column pairings in `x` with those in `value`. The latter should be a list-like object containing any number of SelfHits or sparse matrices, each of which is subject to the constraints described for the single setter.

If `value` is named, those names will be used to name the column pairings in `x`. Otherwise, unnamed pairings are assigned default names prefixed with `"unnamed"`.

If `value` is `NULL`, all column pairings in `x` are removed.

`colPairNames(x) <- value`: Replaces all names for column pairings in `x` with a character vector `value`. This should be of length equal to the number of pairings currently in `x`.

Interaction with SingleCellExperiment operations

When column-subset replacement is performed on a SingleCellExperiment object (i.e., `x[,i] <- y`), a pair of columns in `colPair(x)` is only replaced if both columns are present in `i`. This replacement not only affects the `value` of the pair but also whether it even exists in `y`. For example, if a pair exists between two columns in `x[,i]` but not in the corresponding columns of `y`, it is removed upon subset replacement.

Importantly, pairs in `x` with only one column in `i` are preserved by replacement. This ensures that `x[,i] <- x[,i]` is a no-op. However, if the replacement is fundamentally altering the identity of the features in `x[,i]`, it is unlikely that the pairings involving the old identities are applicable to the replacement features in `y`. In such cases, additional pruning may be required to remove all pairs involving `i` prior to replacement.

Another interesting note is that, for some `i <- 1:n` where `n` is in `[1, ncol(x))`, `cbind(x[,i], x[,-i])` will not return a SingleCellExperiment equal to `x` with respect to `colPairs`. This operation will remove any pairs involving one column in `i` and another column outside of `i`, simply because each individual subset operation will remove pairs involving columns outside of the subset.

Author(s)

Aaron Lun

See Also

`rowPairs`, for the row equivalent.

Examples

```
example(SingleCellExperiment, echo=FALSE)

# Making up some regulatory pairings:
hits <- SelfHits(
    sample(ncol(sce), 10),
    sample(ncol(sce), 10),
    nnode=ncol(sce)
)
mcols(hits)$value <- runif(10)

colPair(sce, "regulators") <- hits
colPair(sce, "regulators")

as.mat <- colPair(sce, "regulators", asSparse=TRUE)
class(as.mat)

colPair(sce, "coexpression") <- hits
colPairs(sce)

colPair(sce, "regulators") <- NULL
colPairs(sce)

colPairs(sce) <- SimpleList()
colPairs(sce)
```

`Combining LEMs` *LEM combining methods*

Description

Methods to combine LinearEmbeddingMatrix objects.

Usage

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

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

Arguments

`...`	One or more LinearEmbeddingMatrix objects.
`deparse.level`	An integer scalar; see `?base::cbind` for a description of this argument.

Details

For rbind, LinearEmbeddingMatrix objects are combined row-wise, i.e., rows in successive objects are appended to the first object. This corresponds to adding more samples to the first object. Note that featureLoadings and factorData will only be taken from the first element in the list; no checks are performed to determine whether they are consistent or not across objects.

For cbind, LinearEmbeddingMatrix objects are combined columns-wise, i.e., columns in successive objects are appended to the first object. This corresponds to adding more factors to the first object. featureLoadings will also be combined column-wise across objects, provided that the number of features is the same across objects. Similarly, factorData will be combined row-wise across objects.

Combining objects with and without row names will result in the removal of all row names; similarly for column names. Duplicate row names are currently supported by duplicate column names are not, and will be de-duplicated appropriately.

Value

A LinearEmbeddingMatrix object containing all rows/columns of the supplied objects.

Author(s)

Aaron Lun

Examples

```
example(LinearEmbeddingMatrix, echo=FALSE) # using the class example
rbind(lem, lem)
cbind(lem, lem)
```

---

defunct *Defunct methods*

Description

Defunct methods in the **SingleCellExperiment** package.

Named size factors

The class now only supports one set of size factors, accessible via sizeFactors. This represents a simplification of the class and removes a difficult part of the API (that had to deal with both NULL and strings to specify the size factor set of interest).

Spike-ins

It is recommended to handle spike-ins and other "alternative" features via altExps.

Author(s)

Aaron Lun

Getter/setter methods *LinearEmbeddingMatrix getters/setters*

Description

Getter/setter methods for the LinearEmbeddingMatrix class.

Usage

```
## S4 method for signature 'LinearEmbeddingMatrix'
sampleFactors(x, withDimnames=TRUE)

## S4 replacement method for signature 'LinearEmbeddingMatrix'
sampleFactors(x) <- value

## S4 method for signature 'LinearEmbeddingMatrix'
featureLoadings(x, withDimnames=TRUE)

## S4 replacement method for signature 'LinearEmbeddingMatrix'
featureLoadings(x) <- value

## S4 method for signature 'LinearEmbeddingMatrix'
factorData(x)

## S4 replacement method for signature 'LinearEmbeddingMatrix'
factorData(x) <- value

## S4 method for signature 'LinearEmbeddingMatrix'
as.matrix(x, ...)

## S4 method for signature 'LinearEmbeddingMatrix'
dim(x)

## S4 method for signature 'LinearEmbeddingMatrix'
dimnames(x)

## S4 replacement method for signature 'LinearEmbeddingMatrix'
dimnames(x) <- value

## S4 method for signature 'LinearEmbeddingMatrix'
x$name

## S4 replacement method for signature 'LinearEmbeddingMatrix'
x$name <- value
```

Arguments

x	A LinearEmbeddingMatrix object.

value	An appropriate value to assign to the relevant slot.
withDimnames	A logical scalar indicating whether dimension names should be attached to the returned object.
name	A string specifying a field of the factorData slot.
...	Further arguments, ignored.

Details

Any value to assign to sampleFactors and featureLoadings should be matrix-like objects, while factorData should be a DataFrame - ee LinearEmbeddingMatrix for details.

The as.matrix method will return the matrix of sample factors, consistent with the fact that the LinearEmbeddingMatrix mimics a sample-factor matrix. However, unlike the sampleFactors method, this is always guaranteed to return an ordinary R matrix, even if an alternative representation was stored in the slot. This ensures consistency with as.matrix methods for other matrix-like S4 classes.

For assignment to dimnames, a list of length 2 should be used containing vectors of row and column names.

Value

For the getter methods sampleFactors, featureLoadings and factorData, the value of the slot with the same name is returned. For the corresponding setter methods, a LinearEmbeddingMatrix is returned with modifications to the named slot.

For dim, the dimensions of the sampleFactors slot are returned in an integer vector of length 2. For dimnames, a list of length 2 containing the row and column names is returned. For as.matrix, an ordinary matrix derived from sampleFactors is returned.

For $, the value of the named field of the factorData slot is returned. For $<-, a LinearEmbeddingMatrix is returned with the modified field in factorData.

Author(s)

Keegan Korthauer, Davide Risso and Aaron Lun

See Also

LinearEmbeddingMatrix

Examples

```
example(LinearEmbeddingMatrix, echo=FALSE) # Using the class example

sampleFactors(lem)
sampleFactors(lem) <- sampleFactors(lem) * -1

featureLoadings(lem)
featureLoadings(lem) <- featureLoadings(lem) * -1

factorData(lem)
```

```
factorData(lem)$whee <- 1

nrow(lem)
ncol(lem)
colnames(lem) <- LETTERS[seq_len(ncol(lem))]
as.matrix(lem)
```

`LinearEmbeddingMatrix` *LinearEmbeddingMatrix class*

Description

A description of the LinearEmbeddingMatrix class for storing low-dimensional embeddings from linear dimensionality reduction methods.

Usage

```
LinearEmbeddingMatrix(sampleFactors = matrix(nrow = 0, ncol = 0),
    featureLoadings = matrix(nrow = 0, ncol = 0), factorData = NULL,
    metadata = list())
```

Arguments

`sampleFactors`	A matrix-like object of sample embeddings, where rows are samples and columns are factors.
`featureLoadings`	A matrix-like object of feature loadings, where rows are features and columns are factors.
`factorData`	A DataFrame containing factor-level information, with one row per factor.
`metadata`	An optional list of arbitrary content describing the overall experiment.

Details

The LinearEmbeddingMatrix class is a matrix-like object that supports `dim`, `dimnames` and `as.matrix`. It is designed for the storage of results from linear dimensionality reduction methods like principal components analysis (PCA), factor analysis and non-negative matrix factorization.

The `sampleFactors` slot is intended to store The low-dimensional representation of the samples, such as the principal coordinates from PCA. The feature loadings contributing to each factor are stored in `featureLoadings`, and should have the same number of columns as `sampleFactors`. The `factorData` stores additional factor-level information, such as the percentage of variance explained by each factor, and should have the same number of rows as `sampleFactors`.

The intended use of this class is to allow PCA and other results to be stored in the `reducedDims` slot of a SingleCellExperiment object. This means that feature loadings remain attached to the embedding, allowing it to be used in downstream analyses.

Value

A LinearEmbeddingMatrix object is returned from the constructor.

Author(s)

Aaron Lun, Davide Risso and Keegan Korthauer

Examples

```
lem <- LinearEmbeddingMatrix(matrix(rnorm(1000), ncol=5),
    matrix(runif(20000), ncol=5))
lem
```

Miscellaneous LEM *Miscellaneous LEM methods*

Description

Various methods for the LinearEmbeddingMatrix class.

Usage

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

Arguments

object	A LinearEmbeddingMatrix object.

Details

The show method will print out information about the data contained in object. This includes the number of samples, the number of factors, the number of genes and the fields available in factorData.

Value

A message is printed to screen describing the data stored in object.

Author(s)

Davide Risso

See Also

LinearEmbeddingMatrix

Examples

```
example(LinearEmbeddingMatrix, echo=FALSE) # Using the class example
show(lem)
```

`reduced.dim.matrix` *The reduced.dim.matrix class*

Description

A matrix class that retains its attributes upon being subsetted or combined. This is useful for storing metadata about a dimensionality reduction result alongside the matrix, and for ensuring that the metadata persists when the matrix is stored inside `reducedDims`.

Constructor

`reduced.dim.matrix(x, ...)` will return a reduced.dim.matrix object, given a matrix input `x`. Arguments in `...` should be named and are stored as custom attributes in the output. Any arguments named `dim` or `dimnames` are ignored.

Subsetting

`x[i, j, ..., drop=FALSE]` will subset a reduced.dim.matrix `x` in the same manner as a base matrix. The only difference is that a reduced.dim.matrix will be returned, retaining any custom attributes in `x`. Note that no custom attributes are retained if the return value is a vector with `drop=TRUE`.

Combining

`rbind(...)` will combine multiple reduced.dim.matrix inputs in `...` by row, while `cbind(...)` will combine those inputs by column.

If the custom attributes are the same across all objects `...`, a reduced.dim.matrix is returned containing all combined rows/columns as well as the custom attributes.

If the custom attributes are different, a warning is issued. A matrix is returned containing all combined rows/columns; no custom attributes are retained.

Author(s)

Aaron Lun

See Also

`reducedDims`, to store these objects in a SingleCellExperiment.

Examples

```
# Typical PC result, with metadata stored in the attributes:
pc <- matrix(runif(500), ncol=5)
attr(pc, "sdev") <- 1:100
attr(pc, "rotation") <- matrix(rnorm(20), ncol=5)

# Disappears upon subsetting and combining!
attributes(pc[1:10,])
attributes(rbind(pc, pc))
```

```
# Transformed into a reduced.dim.matrix:
rd.pc <- reduced.dim.matrix(pc)

attributes(rd.pc[1:10,])
attributes(rbind(rd.pc, rd.pc))
```

`reducedDims` *Reduced dimensions methods*

Description

Methods to get or set dimensionality reduction results in a SingleCellExperiment object. These are typically used to store and retrieve low-dimensional representations of single-cell datasets. Each row of a reduced dimension result is expected to correspond to a column of the SingleCellExperiment object.

Getters

In the following examples, `x` is a SingleCellExperiment object.

`reducedDim(x, type, withDimnames=TRUE)`**:** Retrieves a matrix (or matrix-like object) containing reduced dimension coordinates for cells (rows) and dimensions (columns). `type` is either a string specifying the name of the dimensionality reduction result in `x` to retrieve, or a numeric scalar specifying the index of the desired result, defaulting to the first entry if missing.

If `withDimnames=TRUE`, row names of the output matrix are replaced with the column names of `x`.

`reducedDimNames(x)`**:** Returns a character vector containing the names of all dimensionality reduction results in `x`. This is guaranteed to be of the same length as the number of results, though the names may not be unique.

`reducedDims(x, withDimnames=TRUE)`**:** Returns a named List of matrices containing one or more dimensionality reduction results. Each result is a matrix (or matrix-like object) with the same number of rows as `ncol(x)`.

If `withDimnames=TRUE`, row names of each matrix are replaced with the column names of `x`.

Single-result setter

`reducedDim(x, type, withDimnames=TRUE) <- value` will add or replace a dimensionality reduction result in a SingleCellExperiment object `x`. The value of `type` determines how the result is added or replaced:

- If `type` is missing, `value` is assigned to the first result. If the result already exists, its name is preserved; otherwise it is given a default name `"unnamed1"`.
- If `type` is a numeric scalar, it must be within the range of existing results, and `value` will be assigned to the result at that index.

- If type is a string and a result exists with this name, value is assigned to to that result. Otherwise a new result with this name is append to the existing list of results.

value is expected to be a matrix or matrix-like object with number of rows equal to ncol(x). Alternatively, if value is NULL, the result corresponding to type is removed from the object.

If withDimnames=TRUE, any non-NULL rownames(value) is checked against colnames(x) and a warning is emitted if they are not the same. Otherwise, any differences in the row names are ignored. This is inspired by the argument of the same name in assay<- but is more relaxed for practicality's sake - it raises a warning rather than an error and allows NULL rownames to pass through without complaints.

Other setters

In the following examples, x is a SingleCellExperiment object.

reducedDims(x, withDimnames=TRUE) <- value**:** Replaces all dimensionality reduction results in x with those in value. The latter should be a list-like object containing any number of matrices or matrix-like objects with number of rows equal to ncol(x).

If value is named, those names will be used to name the dimensionality reduction results in x. Otherwise, unnamed results are assigned default names prefixed with "unnamed".

If value is NULL, all dimensionality reduction results in x are removed.

If value is a Annotated object, any metadata will be retained in reducedDims(x). If value is a Vector object, any mcols will also be retained.

If withDimnames=TRUE, any non-NULL row names in each entry of value is checked against colnames(x) and a warning is emitted if they are not the same. Otherwise, any differences in the row names are ignored.

reducedDimNames(x) <- value**:** Replaces all names for dimensionality reduction results in x with a character vector value. This should be of length equal to the number of results currently in x.

Storing dimensionality reduction metadata

When performing dimensionality reduction, we frequently generate metadata associated with a particular method. The typical example is the percentage of variance explained and the rotation matrix from PCA; model-based methods may also report some model information that can be used later to project points onto the embedding. Ideally, we would want to store this information alongside the coordinates themselves.

Our recommended approach is to store this metadata as attributes of the coordinate matrix. This is simple to do, easy to extract, and avoids problems with synchronization (when the coordinates are separated from the metadata). The biggest problem with this approach is that attributes are not retained when the matrix is subsetted or combined. To persist these attributes, we suggest wrapping the coordinates and metadata in a reduced.dim.matrix. More complex matrix-like objects like the LinearEmbeddingMatrix can also be used but may not be immediately compatible with downstream functions that expect an ordinary matrix.

The path less taken is to store the metadata in the mcols of the reducedDims List. This approach avoids the subsetting problem with the attributes but is less ideal as it separates the metadata from the coordinates. Such separation makes the metadata harder to find and remember to keep in sync with the coordinates when the latter changes. The structure of mcols is best suited to situations

where there are some commonalities in the metadata across entries, but this rarely occurs for different dimensionality reduction strategies.

Author(s)

Aaron Lun and Kevin Rue-Albrecht

Examples

```
example(SingleCellExperiment, echo=FALSE)
reducedDim(sce, "PCA")
reducedDim(sce, "tSNE")
reducedDims(sce)

reducedDim(sce, "PCA") <- NULL
reducedDims(sce)

reducedDims(sce) <- SimpleList()
reducedDims(sce)
```

`rowPairs` *Row pair methods*

Description

Methods to get or set row pairings in a SingleCellExperiment object. These are typically used to store and retrieve relationships between features, e.g., in gene regulatory or co-expression networks.

Getters

In the following examples, `x` is a SingleCellExperiment object.

`rowPair(x, type, asSparse=FALSE)`: Retrieves a SelfHits object where each entry represents a pair of rows of `x` and has number of nodes equal to `nrow(x)`. `type` is either a string specifying the name of the row pairing in `x` to retrieve, or a numeric scalar specifying the index of the desired pairing.

If `asSparse=TRUE`, a sparse matrix is returned instead, see below for details.

`rowPairNames(x)`: Returns a character vector containing the names of all row pairings in `x`. This is guaranteed to be of the same length as the number of pairings, though the names may not be unique.

`rowPairs(x, asSparse=FALSE)`: Returns a named List of matrices containing one or more row pairings as SelfHits objects. If `asSparse=FALSE`, each entry is instead a sparse matrix.

When `asSparse=TRUE`, the return value will be a triplet-form sparse matrix where each row/column corresponds to a row of `x`. The values in the matrix will be taken from the first metadata field of the underlying SelfHits object, with an error being raised if the first metadata field is not of an acceptable type. If there are duplicate pairs, only the value from the last pair is used. If no metadata is available, the matrix values are set to `TRUE` for all pairs.

Single setter

`rowPair(x, type) <- value` will add or replace a row pairing in a SingleCellExperiment object `x`. The value of `type` determines how the pairing is added or replaced:

- If `type` is missing, `value` is assigned to the first pairing. If the pairing already exists, its name is preserved; otherwise it is given a default name `"unnamed1"`.
- If `type` is a numeric scalar, it must be within the range of existing pairings, and `value` will be assigned to the pairing at that index.
- If `type` is a string and a pairing exists with this name, `value` is assigned to to that pairing. Otherwise a new pairing with this name is append to the existing list of pairings.

`value` is expected to be a SelfHits with number of nodes equal to `nrow(x)`. Any number of additional fields can be placed in `mcols(value)`. Duplicate row pairs are supported and will not be collapsed into a single entry.

`value` may also be a sparse matrix with number of rows and columns equal to `nrow(x)`. This is converted into a SelfHits object with values stored in the metadata as the `"x"` field.

Alternatively, if `value` is `NULL`, the pairings corresponding to `type` are removed from `x`.

Other setters

In the following examples, `x` is a SingleCellExperiment object.

`rowPairs(x) <- value`**:** Replaces all row pairings in `x` with those in `value`. The latter should be a list-like object containing any number of SelfHits or sparse matrices, each of which is subject to the constraints described for the single setter.

If `value` is named, those names will be used to name the row pairings in `x`. Otherwise, unnamed pairings are assigned default names prefixed with `"unnamed"`.

If `value` is `NULL`, all row pairings in `x` are removed.

`rowPairNames(x) <- value`**:** Replaces all names for row pairings in `x` with a character vector `value`. This should be of length equal to the number of pairings currently in `x`.

Interaction with SingleCellExperiment operations

When row-subset replacement is performed on a SingleCellExperiment object (i.e., `x[i,] <- y`), a pair of rows in `rowPair(x)` is only replaced if both rows are present in `i`. This replacement not only affects the `value` of the pair but also whether it even exists in `y`. For example, if a pair exists between two rows in `x[i,]` but not in the corresponding rows of `y`, it is removed upon subset replacement.

Importantly, pairs in `x` with only one row in `i` are preserved by replacement. This ensures that `x[i,] <- x[i,]` is a no-op. However, if the replacement is fundamentally altering the identity of the features in `x[i,]`, it is unlikely that the pairings involving the old identities are applicable to the replacement features in `y`. In such cases, additional pruning may be required to remove all pairs involving `i` prior to replacement.

Another interesting note is that, for some `i <- 1:n` where `n` is in `[1, nrow(x))`, `rbind(x[i,], x[-i,])` will not return a SingleCellExperiment equal to `x` with respect to `rowPairs`. This operation will remove any pairs involving one row in `i` and another row outside of `i`, simply because each individual subset operation will remove pairs involving rows outside of the subset.

Author(s)

Aaron Lun

See Also

colPairs, for the column equivalent.

Examples

```
example(SingleCellExperiment, echo=FALSE)

# Making up some regulatory pairings:
hits <- SelfHits(
    sample(nrow(sce), 10),
    sample(nrow(sce), 10),
    nnode=nrow(sce)
)
mcols(hits)$value <- runif(10)

rowPair(sce, "regulators") <- hits
rowPair(sce, "regulators")

as.mat <- rowPair(sce, "regulators", asSparse=TRUE)
class(as.mat)

rowPair(sce, "coexpression") <- hits
rowPairs(sce)

rowPair(sce, "regulators") <- NULL
rowPairs(sce)

rowPairs(sce) <- SimpleList()
rowPairs(sce)
```

---

rowSubset *Get or set the row subset*

Description

Get or set the row subset in an instance of a SingleCellExperiment class. This is assumed to specify some interesting subset of genes to be favored in downstream analyses.

Usage

```
## S4 method for signature 'SingleCellExperiment'
rowSubset(x, field = "subset", onAbsence = "none")

## S4 replacement method for signature 'SingleCellExperiment'
rowSubset(x, field = "subset", ...) <- value
```

Arguments

x	A SingleCellExperiment object.
field	String containing the name of the field in the rowData to get or set subsetting data.
onAbsence	String indicating an additional action to take when labels are absent: nothing ("none"), a warning ("warn") or an error ("error").
...	Additional arguments, currently ignored.
value	Any character, logical or numeric vector specifying rows of x to include in the subset. Alternatively NULL, in which case existing subsetting information is removed.

Details

A frequent task in single-cell data analyses is to focus on a subset of genes of interest, e.g., highly variable genes, derived marker genes for clusters, known markers for cell types. A related task is to filter out uninteresting genes such as ribosomal protein genes or mitochondrial transcripts, in which case we want to subset to exclude those genes.

These functions store a set of genes of interest inside a SingleCellExperiment for later retrieval and use in downstream functions. Character and numeric value are converted to logical vectors that are parallel to the rows of x, allowing them to be added to the rowData for synchronized row-level operations.

For developers, onAbsence is provided to make it easier to mandate that x actually has labels. This avoids silent NULL values that flow to the rest of the function and make debugging difficult.

Value

For rowSubset, a logical vector is returned specifying the rows to retain in the subset of interest. If no subset is available, a NULL is returned (and/or a warning or error, depending on onAbsence).

For rowSubset<-, a modified x is returned a subsetting vector in its rowData.

Author(s)

Aaron Lun

See Also

SingleCellExperiment, for the underlying class definition.

Examples

```
example(SingleCellExperiment, echo=FALSE) # Using the class example

rowSubset(sce, "hvgs") <- 1:10
rowSubset(sce, "hvgs")

rowSubset(sce) <- rbinom(nrow(sce), 1, 0.5)==1
rowSubset(sce)
```

SCE-assays *Named assay getters and setters*

Description

These are methods for getting or setting `assay(sce, i=X, ...)` where `sce` is a SingleCellExperiment object and `X` is the name of the method. For example, `counts` will get or set `X="counts"`. This provides some convenience for users as well as encouraging standardization of assay names across packages.

Available methods

In the following code snippets, `x` is a SingleCellExperiment object, `value` is a matrix-like object with the same dimensions as `x`, and `...` are further arguments passed to `assay` (for the getter) or `assay<-` (for the setter).

`counts(x, ...)`, `counts(x, ...) <- value`: Get or set a matrix of raw count data, e.g., number of reads or transcripts.

`normcounts(x, ...)`, `normcounts(x, ...) <- value`: Get or set a matrix of normalized values on the same scale as the original counts. For example, counts divided by cell-specific size factors that are centred at unity.

`logcounts(x, ...)`, `logcounts(x, ...) <- value`: Get or set a matrix of log-transformed counts or count-like values. In most cases, this will be defined as log-transformed `normcounts`, e.g., using log base 2 and a pseudo-count of 1.

`cpm(x, ...)`, `cpm(x, ...) <- value`: Get or set a matrix of counts-per-million values. This is the read count for each gene in each cell, divided by the library size of each cell in millions.

`tpm(x, ...)`, `tpm(x, ...) <- value`: Get or set a matrix of transcripts-per-million values. This is the number of transcripts for each gene in each cell, divided by the total number of transcripts in that cell (in millions).

`weights(x, ...)`, `weights(x, ...) <- value`: Get or set a matrix of weights, e.g., observational weights to be used in differential expression analysis.

Author(s)

Aaron Lun

See Also

`assay` and `assay<-`, for the wrapped methods.

Examples

```
example(SingleCellExperiment, echo=FALSE) # Using the class example
counts(sce) <- matrix(rnorm(nrow(sce)*ncol(sce)), ncol=ncol(sce))
dim(counts(sce))
```

```
# One possible way of computing normalized "counts"
sf <- 2^rnorm(ncol(sce))
sf <- sf/mean(sf)
normcounts(sce) <- t(t(counts(sce))/sf)
dim(normcounts(sce))

# One possible way of computing log-counts
logcounts(sce) <- log2(normcounts(sce)+1)
dim(normcounts(sce))
```

SCE-combine *Combining or subsetting SingleCellExperiment objects*

Description

An overview of methods to combine multiple SingleCellExperiment objects by row or column, or to subset a SingleCellExperiment by row or column. These methods are useful for ensuring that all data fields remain synchronized when cells or genes are added or removed.

Combining

In the following code snippets, ... contains one or more SingleCellExperiment objects.

`rbind(..., deparse.level=1)`**:** Returns a SingleCellExperiment where all objects in ... are combined row-wise, i.e., rows in successive objects are appended to the first object.

Refer to `?"rbind,SummarizedExperiment-method"` for details on how metadata is combined in the output object. Refer to `?rbind` for the interpretation of `deparse.level`.

Note that all objects in ... must have the exact same values for `reducedDims` and `altExps`. Any `sizeFactors` should either be `NULL` or contain the same values across objects.

`cbind(..., deparse.level=1)`**:** Returns a SingleCellExperiment where all objects in ... are combined column-wise, i.e., columns in successive objects are appended to the first object.

Each object `x` in ... must have the same values of `reducedDimNames(x)` (though they can be unordered). Dimensionality reduction results with the same name across objects will be combined row-wise to create the corresponding entry in the output object.

Each object `x` in ... must have the same values of `altExpNames(x)` (though they can be unordered). Alternative Experiments with the same name across objects will be combined column-wise to create the corresponding entry in the output object.

`sizeFactors` should be either set to `NULL` in all objects, or set to a numeric vector in all objects.

Refer to `?"cbind,SummarizedExperiment-method"` for details on how metadata is combined in the output object. Refer to `?cbind` for the interpretation of `deparse.level`.

In the following code snippets, `x` is a SingleCellExperiment and ... contains multiple SingleCellExperiment objects.

`combineCols(x, ..., delayed=TRUE, fill=NA, use.names=TRUE)`: Returns a SingleCellExperiment where all objects are flexibly combined by column. The assays and `colData` are combined as described in `?"combineCols,SummarizedExperiment-method"`, where assays or DataFrame columns missing in any given object are filled in with missing values before combining.

Entries of the `reducedDims` with the same name across objects are combined by row. If a dimensionality reduction result is not present for a particular SingleCellExperiment, it is represented by a matrix of `NA` values instead. If corresponding `reducedDim` entries cannot be combined, e.g., due to inconsistent dimensions, they are omitted from the `reducedDims` of the output object with a warning.

Entries of the `altExps` with the same name across objects are combined by column using the relevant `combineCols` method. If a named entry is not present for a particular SingleCellExperiment, it is represented by a SummarizedExperiment with a single assay full of `fill` values. If entries cannot be combined, e.g., due to inconsistent dimensions, they are omitted from the `altExps` of the output object with a warning.

Entries of the `colPairs` with the same name across objects are concatenated together after adjusting the indices for each column's new position in the combined object. If a named entry is not present for a particular SingleCellExperiments, it is assumed to contribute no column pairings and is ignored.

Entries of the `rowPairs` with the same name should be identical across objects if `use.names=FALSE`. If `use.names=TRUE`, we attempt to merge together entries with the same name by taking the union of all column pairings. However, if the same cell has a different set of pairings across objects, a warning is raised and we fall back to the `rowPair` entry from the first object.

Subsetting

In the following code snippets, `x` is a SingleCellExperiment object.

`x[i, j, ..., drop=TRUE]`: Returns a SingleCellExperiment containing the specified rows `i` and columns `j`.

`i` and `j` can be a logical, integer or character vector of subscripts, indicating the rows and columns respectively to retain. Either can be missing, in which case subsetting is only performed in the specified dimension. If both are missing, no subsetting is performed.

Arguments in `...` and `drop` are passed to to `[,SummarizedExperiment-method`.

`x[i, j, ...] <- value`: Replaces all data for rows `i` and columns j with the corresponding fields in a SingleCellExperiment `value`.

`i` and `j` can be a logical, integer or character vector of subscripts, indicating the rows and columns respectively to replace. Either can be missing, in which case replacement is only performed in the specified dimension. If both are missing, `x` is replaced entirely with `value`.

If `j` is specified, `value` is expected to have the same name and order of `reducedDimNames` and `altExpNames` as `x`. If `sizeFactors` is set for `x`, it should also be set for `value`.

Arguments in `...` are passed to the corresponding SummarizedExperiment method.

Author(s)

Aaron Lun

Examples

```
example(SingleCellExperiment, echo=FALSE) # using the class example

# Combining:
rbind(sce, sce)
cbind(sce, sce)

# Subsetting:
sce[1:10,]
sce[,1:5]

sce2 <- sce
sce2[1:10,] <- sce[11:20,]

# Can also use subset()
sce$WHEE <- sample(LETTERS, ncol(sce), replace=TRUE)
subset(sce, , WHEE=="A")

# Can also use split()
split(sce, sample(LETTERS, nrow(sce), replace=TRUE))
```

SCE-internals *Internal SingleCellExperiment functions*

Description

Methods to get or set internal fields from the SingleCellExperiment class. Thse functions are intended for package developers who want to add protected fields to a SingleCellExperiment. They should *not* be used by ordinary users of the **SingleCellExperiment** package.

Getters

In the following code snippets, `x` is a SingleCellExperiment.

`int_elementMetadata(x)`: Returns a DataFrame of internal row metadata, with number of rows equal to `nrow(x)`. This is analogous to the user-visible `rowData`.

`int_colData(x)`: Returns a DataFrame of internal column metadata, with number of rows equal to `ncol(x)`. This is analogous to the user-visible `colData`.

`int_metadata(x)`: Returns a list of internal metadata, analogous to the user-visible `metadata`.

It may occasionally be useful to return both the visible and the internal `colData` in a single DataFrame. This is facilitated by the following methods:

`rowData(x, ..., internal=FALSE)`: Returns a DataFrame of the user-visible row metadata. If `internal=TRUE`, the internal row metadata is added column-wise to the user-visible metadata. A warning is emitted if the user-visible metadata column names overlap with the internal fields. Any arguments in `...` are passed to `rowData,SummarizedExperiment-method`.

`colData(x, ..., internal=FALSE)`: Returns a DataFrame of the user-visible column metadata. If `internal=TRUE`, the internal column metadata is added column-wise to the user-visible metadata. A warning is emitted if the user-visible metadata column names overlap with the internal fields. Any arguments in `...` are passed to `colData,SummarizedExperiment-method`.

Setters

In the following code snippets, `x` is a SingleCellExperiment.

`int_elementMetadata(x) <- value`: Replaces the internal row metadata with `value`, a DataFrame with number of rows equal to `nrow(x)`. This is analogous to the user-visible `rowData<-`.

`int_colData(x) <- value`: Replaces the internal column metadata with `value`, a DataFrame with number of rows equal to `ncol(x)`. This is analogous to the user-visible `colData<-`.

`int_metadata(x) <- value`: Replaces the internal metadata with `value`, analogous to the user-visible `metadata<-`.

Comments

The internal metadata fields allow easy and extensible storage of additional elements that are parallel to the rows or columns of a SingleCellExperiment class. This avoids the need to specify new slots and adjust the subsetting/combining code for a new data element. For example, `altExps` and `reducedDims` are implemented as fields in the internal column metadata.

That these elements are internal is important as this ensures that the implementation details are abstracted away. Any user interaction with these internal fields should be done via the designated getter and setter methods, e.g., `reducedDim` and friends for retrieving or modifying reduced dimensions. This provides developers with more freedom to change the internal representation without breaking user code.

Package developers intending to use these methods to store their own content should read the development vignette for guidance.

Author(s)

Aaron Lun

See Also

`colData`, `rowData` and `metadata` for the user-visible equivalents.

Examples

```
example(SingleCellExperiment, echo=FALSE) # Using the class example
int_metadata(sce)$whee <- 1
```

SCE-miscellaneous *Miscellaneous SingleCellExperiment methods*

Description

Miscellaneous methods for the SingleCellExperiment class that do not fit in any other documentation category.

Available methods

In the following code snippets, x and object are SingleCellExperiment objects.

show(object): Print a message to screen describing the contents of object.

objectVersion(x): Return the version of the package with which x was constructed.

sizeFactors(object): Return a numeric vector of size factors of length equal to ncol(object). If no size factors are available in object, return NULL instead.

sizeFactors(object) <- value: Replace the size factors with value, usually expected to be a numeric vector or vector-like object. Alternatively, value can be NULL in which case any size factors in object are removed.

Author(s)

Aaron Lun

See Also

updateObject, where objectVersion is used.

Examples

```
example(SingleCellExperiment, echo=FALSE) # Using the class example

show(sce)

objectVersion(sce)

# Setting/getting size factors.
sizeFactors(sce) <- runif(ncol(sce))
sizeFactors(sce)

sizeFactors(sce) <- NULL
sizeFactors(sce)
```

simplifyToSCE *Simplify a list to a single SingleCellExperiment*

Description

Simplify a list of SingleCellExperiment, usually generated by applySCE on main and alternative Experiments, into a single SingleCellExperiment containing some of the results in its altExps.

Usage

```
simplifyToSCE(results, which.main, warn.level = 2)
```

Arguments

results	A named list of SummarizedExperiment or SingleCellExperiment objects.
which.main	Integer scalar specifying which entry of results contains the output generated from the main Experiment. If NULL or a vector of length zero, this indicates that no entry was generated from the main Experiment. Defaults to the unnamed entry of results.
warn.level	Integer scalar specifying the type of warnings that can be emitted.

Details

Each entry of results should be a SummarizedExperiment with the same number and names of the columns. There should not be any duplicate entries in names(results), as the names are used to represent the names of the alternative Experiments in the output. If which.main is a scalar, the corresponding entry of results should be a SingleCellExperiment. Failure to meet these conditions may result in a warning or error depending on warn.level.

The type of warnings that are emitted can be controlled with warn.level. If warn.level=0, no warnings are emitted. If warn.level=1, all warnings are emitted except for those related to results not being of the appropriate class. If warn.level=2, all warnings are emitted, and if warn.level=3, warnings are promoted to errors.

Value

A SingleCellExperiment corresponding to the entry of results generated from the main Experiment. All results generated from the alternative Experiments of x are stored in the altExps of the output.

If no main Experiment was used to generate results, an empty SingleCellExperiment is used as a container for the various altExps.

If simplification could not be performed, NULL is returned with a warning (depending on warn.level.

Author(s)

Aaron Lun

See Also

applySCE, where this function is used when SIMPLIFY=TRUE.

Examples

```
ncells <- 100
u <- matrix(rpois(20000, 5), ncol=ncells)
sce <- SingleCellExperiment(assays=list(counts=u))
altExp(sce, "BLAH") <- SingleCellExperiment(assays=list(counts=u*10))
altExp(sce, "WHEE") <- SingleCellExperiment(assays=list(counts=u*2))

# Setting FUN=identity just extracts each piece:
results <- applySCE(sce, FUN=identity, SIMPLIFY=FALSE)
results

# Simplifying to an output that mirrors the structure of 'sce'.
simplifyToSCE(results)
```

SingleCellExperiment-class

The SingleCellExperiment class

Description

The SingleCellExperiment class is designed to represent single-cell sequencing data. It inherits from the RangedSummarizedExperiment class and is used in the same manner. In addition, the class supports storage of dimensionality reduction results (e.g., PCA, t-SNE) via reducedDims, and storage of alternative feature types (e.g., spike-ins) via altExps.

Usage

```
SingleCellExperiment(
  ...,
  reducedDims = list(),
  altExps = list(),
  rowPairs = list(),
  colPairs = list(),
  mainExpName = NULL
)
```

Arguments

...	Arguments passed to the SummarizedExperiment constructor to fill the slots of the base class.
reducedDims	A list of any number of matrix-like objects containing dimensionality reduction results, each of which should have the same number of rows as the output SingleCellExperiment object.

altExps	A list of any number of SummarizedExperiment objects containing alternative Experiments, each of which should have the same number of columns as the output SingleCellExperiment object.
rowPairs	A list of any number of SelfHits objects describing relationships between pairs of rows. Each entry should have number of nodes equal to the number of rows of the output SingleCellExperiment object. Alternatively, entries may be square sparse matrices of order equal to the number of rows of the output object.
colPairs	A list of any number of SelfHits objects describing relationships between pairs of columns. Each entry should have number of nodes equal to the number of columns of the output SingleCellExperiment object. Alternatively, entries may be square sparse matrices of order equal to the number of columns of the output object.
mainExpName	String containing the name of the main Experiment. This is comparable to the names assigned to each of the altExps.

Details

In this class, rows should represent genomic features (e.g., genes) while columns represent samples generated from single cells. As with any SummarizedExperiment derivative, different quantifications (e.g., counts, CPMs, log-expression) can be stored simultaneously in the assays slot, and row and column metadata can be attached using rowData and colData, respectively.

The extra arguments in the constructor (e.g., reducedDims altExps) represent the main extensions implemented in the SingleCellExperiment class. This enables a consistent, formalized representation of data structures that are commonly encountered during single-cell data analysis. Readers are referred to the specific documentation pages for more details.

A SingleCellExperiment can also be created by coercing from a SummarizedExperiment or RangedSummarizedExperiment instance.

Value

A SingleCellExperiment object.

Author(s)

Aaron Lun and Davide Risso

See Also

reducedDims, for representation of dimensionality reduction results.

altExps, for representation of data for alternative feature sets.

colPairs and rowPairs, to hold pairing information for rows and columns.

sizeFactors, to store size factors for normalization.

colLabels, to store cell-level labels.

rowSubset, to store a subset of rows.

?"SCE-combine", to combine or subset a SingleCellExperiment object.

?"SCE-internals", for developer use.

Examples

```
ncells <- 100
u <- matrix(rpois(20000, 5), ncol=ncells)
v <- log2(u + 1)

pca <- matrix(runif(ncells*5), ncells)
tsne <- matrix(rnorm(ncells*2), ncells)

sce <- SingleCellExperiment(assays=list(counts=u, logcounts=v),
    reducedDims=SimpleList(PCA=pca, tSNE=tsne))
sce

## coercion from SummarizedExperiment
se <- SummarizedExperiment(assays=list(counts=u, logcounts=v))
as(se, "SingleCellExperiment")
```

---

sizeFactors *Size factor methods*

Description

Gets or sets the size factors for all cells in a SingleCellExperiment object.

Usage

```
## S4 method for signature 'SingleCellExperiment'
sizeFactors(object, onAbsence = "none")

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

Arguments

object	A SingleCellExperiment object.
onAbsence	String indicating an additional action to take when size factors are absent: nothing ("none"), a warning ("warn") or an error ("error").
...	Additional arguments, currently ignored.
value	A numeric vector of length equal to ncol(object), containing size factors for all cells.

Details

A size factor is a scaling factor used to divide the raw counts of a particular cell to obtain normalized expression values, thus allowing downstream comparisons between cells that are not affected by differences in library size or total RNA content. The sizeFactors methods can be used to get or set size factors for all cells in a SingleCellExperiment object.

When setting size factors, the values are stored in the colData as the sizeFactors field. This name is chosen for general consistency with other packages (e.g., **DESeq2**) and to allow the size factors to be easily extracted from the colData for use as covariates.

For developers, onAbsence is provided to make it easier to mandate that object has size factors. This avoids silent NULL values that flow to the rest of the function and make debugging difficult.

Value

For sizeFactors, a numeric vector is returned containing size factors for all cells. If no size factors are available, a NULL is returned (and/or a warning or error, depending on onAbsence).

For sizeFactors<-, a modified object is returned with size factors in its colData.

Author(s)

Aaron Lun

See Also

SingleCellExperiment, for the underlying class definition.

librarySizeFactors from the **scater** package or computeSumFactors from the **scran** package, as examples of functions that compute the size factors.

Examples

```
example(SingleCellExperiment, echo=FALSE) # Using the class example
sizeFactors(sce) <- runif(ncol(sce))
sizeFactors(sce)
```

---

splitAltExps *Split off alternative features*

Description

Split a SingleCellExperiment based on the feature type, creating alternative Experiments to hold features that are not in the majority set.

Usage

```
splitAltExps(x, f, ref = NULL)
```

Arguments

x	A SingleCellExperiment object.
f	A character vector or factor of length equal to nrow(x), specifying the feature type of each row.
ref	String indicating which level of f should be treated as the main set.

Details

This function provides a convenient way to create a SingleCellExperiment with alternative Experiments. For example, a SingleCellExperiment with rows corresponding to all features can be quickly split into endogenous genes (main) and other alternative features like spike-in transcripts and antibody tags.

By default, the most frequent level of `f` is treated as the `ref` if the latter is not specified.

Value

A SingleCellExperiment where each row corresponds to a feature in the main set. Each other feature type is stored as an alternative Experiment, accessible by `altExp`. `ref` is used as the `mainExpName`.

Author(s)

Aaron Lun

See Also

`altExp`, to access and manipulate the alternative Experiment fields.

`unsplitAltExps`, to reverse the splitting.

Examples

```
example(SingleCellExperiment, echo=FALSE)
feat.type <- sample(c("endog", "ERCC", "CITE"), nrow(sce),
    replace=TRUE, p=c(0.8, 0.1, 0.1))

sce2 <- splitAltExps(sce, feat.type)
sce2
```

Subsetting LEMs *LEM subsetting methods*

Description

Methods to subset LinearEmbeddingMatrix objects.

Usage

```
## S4 method for signature 'LinearEmbeddingMatrix,ANY,ANY'
x[i, j, ..., drop=TRUE]

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

Arguments

x	A LinearEmbeddingMatrix object.
i, j	A vector of logical or integer subscripts, indicating the rows and columns to be subsetted for i and j, respectively.
...	Extra arguments that are ignored.
drop	A logical scalar indicating whether the result should be coerced to the lowest possible dimension.
value	A LinearEmbeddingMatrix object with number of rows equal to length of i (or that of x, if i is not specified). The number of columns must be equal to the length of j (or number of columns in x, if j is not specified).

Details

Subsetting yields a LinearEmbeddingMatrix object containing the specified rows (samples) and columns (factors). If column subsetting is performed, values of featureLoadings and factorData will be modified to retain only the selected factors.

If drop=TRUE and the subsetting would produce dimensions of length 1, those dimensions are dropped and a vector is returned directly from sampleFactors. This mimics the expected behaviour from a matrix-like object. Users should set drop=FALSE to ensure that a LinearEmbeddingMatrix is returned.

For subset replacement, if neither i or j are set, x will be effectively replaced by value. However, row and column names will *not* change, consistent with replacement in ordinary matrices.

Value

For [, a subsetted LinearEmbeddingMatrix object is returned.

For [<-, a modified LinearEmbeddingMatrix object is returned.

Author(s)

Aaron Lun

See Also

LinearEmbeddingMatrix-class

Examples

```
example(LinearEmbeddingMatrix, echo=FALSE) # using the class example

lem[1:10,]
lem[,1:5]

lem2 <- lem
lem2[1:10,] <- lem[11:20,]
```

swapAltExp *Swap main and alternative Experiments*

Description

Swap the main Experiment for an alternative Experiment in a SingleCellExperiment object.

Usage

```
swapAltExp(x, name, saved = mainExpName(x), withColData = TRUE)
```

Arguments

x	A SingleCellExperiment object.
name	String or integer scalar specifying the alternative Experiment to use to replace the main Experiment.
saved	String specifying the name to use to save the original x as an alternative experiment in the output. If NULL, the original is not saved.
withColData	Logical scalar specifying whether the column metadata of x should be preserved in the output.

Details

During the course of an analysis, we may need to perform operations on each of the alternative Experiments in turn. This would require us to repeatedly call altExp(x, name) prior to running downstream functions on those Experiments. In such cases, it may be more convenient to switch the main Experiment with the desired alternative Experiments, allowing a particular section of the analysis to be performed on the latter by default.

For example, the initial phases of the analysis might use the entire set of features. At some point, we might want to focus only on a subset of features of interest, but we do not want to discard the rest of the features. This can be achieved by storing the subset as an alternative Experiment and swapping it with the main Experiment, as shown in the Examples below.

If withColData=TRUE, the column metadata of the output object is set to colData(x). As a side-effect, any column data previously altExp(x, name) is stored in the saved alternative Experiment of the output. This is necessary to preserve the column metadata while achieving reversibility (see below). Setting withColData=FALSE will omit the colData exchange.

swapAltExp is almost perfectly reversible, i.e., swapAltExp(swapAltExp(x, name, saved), saved, name) should return something very similar to x. The only exceptions are that the order of altExpNames is changed, and that any non-NULL mainExpName in altExp(x, name) will be lost.

Value

A SingleCellExperiment derived from altExp(x, name). This contains all alternative Experiments in altExps(x), excluding the one that was promoted to the main Experiment. An additional alternative Experiment containing x may be included if saved is specified.

Author(s)

Aaron Lun

See Also

altExps, for a description of the alternative Experiment concept.

Examples

```
example(SingleCellExperiment, echo=FALSE) # using the class example

# Let's say we defined a subset of genes of interest.
# We can save the feature set as its own altExp.
hvgs <- 1:10
altExp(sce, "subset") <- sce[hvgs,]

# At some point, we want to do our analysis on the HVGs only,
# but we want to hold onto the other features for later reference.
sce <- swapAltExp(sce, name="subset", saved="all")
sce

# Once we're done, it is straightforward to switch back.
swapAltExp(sce, "all")
```

---

unsplitAltExps *Unsplit the alternative experiments*

Description

Combine the main and alternative experiments back into one SingleCellExperiment object. This is effectively the reverse operation to splitAltExps.

Usage

```
unsplitAltExps(sce, prefix.rows = TRUE, prefix.cols = TRUE, delayed = TRUE)
```

Arguments

sce	A SingleCellExperiment containing alternative experiments in the altExps slot.
prefix.rows	Logical scalar indicating whether the (non-NULL) row names should be prefixed with the name of the alternative experiment.
prefix.cols	Logical scalar indicating whether the names of column-related fields should be prefixed with the name of the alternative experiment. If NA, any colData of the altExps are ignored.
delayed	Logical scalar indicating whether the combining of the assays should be delayed.

Details

This function is intended for downstream applications that accept a SingleCellExperiment but are not aware of the altExps concept. By consolidating all data together, applications like **iSEE** can use the same machinery to visualize any feature of interest across all modalities. However, for quantitative analyses, it is usually preferable to keep different modalities separate.

Assays with the same name are rbinded together in the output object. If a particular name is not present for any experiment, its values are filled in with the appropriately typed NA instead. By default, this is done efficiently via ConstantMatrix abstractions to avoid actually creating a dense matrix of NAs. If delayed=FALSE, the combining of matrices is done without any DelayedArray wrappers, yielding a simpler matrix representation at the cost of increasing memory usage.

Any colData or reducedDims in the alternative experiments are added to those of the main experiment. The names of these migrated fields are prefixed by the name of the alternative experiment if prefix.cols=TRUE.

Setting prefix.rows=FALSE, prefix.cols=NA and delayed=FALSE will reverse the effects of splitAltExps.

Value

A SingleCellExperiment where all features in the alternative experiments of sce are now features in the main experiment. The output object has no alternative experiments of its own.

Author(s)

Aaron Lun

See Also

splitAltExps, which does the reverse operation of this function.

Examples

```
counts <- matrix(rpois(10000, 5), ncol=100)
sce <- SingleCellExperiment(assays=list(counts=counts))
feat.type <- sample(c("endog", "ERCC", "adt"), nrow(sce),
    replace=TRUE, p=c(0.8, 0.1, 0.1))
sce <- splitAltExps(sce, feat.type)

# Making life a little more complicated.
logcounts(sce) <- log2(counts(sce) + 1)
sce$cluster <- sample(5, ncol(sce), replace=TRUE)
reducedDim(sce, "PCA") <- matrix(rnorm(ncol(sce)*2), ncol=2)

# Now, putting Humpty Dumpty back together again.
restored <- unsplitAltExps(sce)
restored
```

updateObject *Update a SingleCellExperiment object*

Description

Update SingleCellExperiment objects to the latest version of the class structure. This is usually called by methods in the **SingleCellExperiment** package rather than by users or downstream packages.

Usage

```
## S4 method for signature 'SingleCellExperiment'
updateObject(object, ..., verbose = FALSE)
```

Arguments

object	A old SingleCellExperiment object.
...	Additional arguments that are ignored.
verbose	Logical scalar indicating whether a message should be emitted as the object is updated.

Details

This function updates the SingleCellExperiment to match changes in the internal class representation. Changes are as follows:

- Objects created before 1.7.1 are modified to include altExps and reducedDims fields in their internal column metadata. Reduced dimension results previously in the reducedDims slot are transferred to the reducedDims field.
- Objects created before 1.9.1 are modified so that the size factors are stored by sizeFactors<- in colData rather than int_colData.

Value

An updated version of object.

Author(s)

Aaron Lun

See Also

objectVersion, which is used to determine if the object is up-to-date.

Index