Package ‘BiocSingular’

March 10, 2025

Version 1.22.0

Date 2024-09-21

Title Singular Value Decomposition for Bioconductor Packages

Imports BiocGenerics, S4Vectors, Matrix, methods, utils, DelayedArray, BiocParallel, ScaledMatrix, irlba, rsvd, Rcpp, beachmat (>= 2.21.1)

Suggests testthat, BiocStyle, knitr, rmarkdown, ResidualMatrix

biocViews Software, DimensionReduction, PrincipalComponent

Description Implements exact and approximate methods for singular value decomposition and principal components analysis, in a framework that allows them to be easily switched within Bioconductor packages or workflows. Where possible, parallelization is achieved using the BiocParallel framework.

License GPL-3

LinkingTo Rcpp, beachmat, assorthead

VignetteBuilder knitr

SystemRequirements C++17

RoxygenNote 7.2.3

BugReports `https://github.com/LTLA/BiocSingular/issues`

URL `https://github.com/LTLA/BiocSingular`

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

git_branch RELEASE_3_20

git_last_commit aa0f642

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2025-03-10

Author Aaron Lun [aut, cre, cph]

Maintainer Aaron Lun <`infinite.monkeys.with.keyboards@gmail.com`>

Contents

BiocSingular options 2
BiocSingularParam 3
DeferredMatrix 5
LowRankMatrix 5
ResidualMatrix 6
runExactSVD 7
runIrlbaSVD 8
runPCA 10
runRandomSVD 11
runSVD 12

Index **14**

`BiocSingular options` *Global SVD options*

Description

An overview of the available options when performing SVD with any algorithm.

Computing the cross-product

If the dimensions of the input matrix are very different, it may be faster to compute the cross-product and perform the SVD on the resulting square matrix, rather than performing SVD directly on a very fat or tall input matrix. The cross-product can often be computed very quickly due to good data locality, yielding a small square matrix that is easily handled by any SVD algorithm. This is especially true in cases where the input matrix is not held in memory. Calculation of the cross-product only involves one read across the entire data set, while direct application of approximate methods like `irlba` or `rsvd` would need to access the data multiple times.

The various **BiocSingular** SVD functions allow users to specify the minimum fold difference (via the `fold` argument) at which a cross-product should be computed. Setting `fold=1` will always compute the cross-product for any matrix - this is probably unwise. By contrast, setting `fold=Inf` means that the cross-product is never computed. This is currently the default in all functions, to provide the most expected behaviour unless specifically instructed otherwise.

Centering and scaling

In general, each SVD function performs the SVD on `t((t(x) - C)/S)` where `C` and `S` are numeric vectors of length equal to `ncol(x)`. The values of `C` and `S` are defined according to the `center` and `scale` options.

- If `center=TRUE`, `C` is defined as the column means of `x`. If `center=NULL` or `FALSE`, all elements of `C` are set to zero. If `center` is a numeric vector with length equal to `ncol(x)`, it is used to directly define `C`.
- If `scale=TRUE`, the `i`th element of `S` is defined as the square root of `sum((x[,i] - C[i])^2)/(ncol(x)-1)`, for whatever `C` was defined above. This mimics the behaviour of `scale`. If `scale=NULL` or `FALSE`, all elements of `S` are set to unity. If `scale` is a numeric vector with length equal to `ncol(x)`, it is used to directly define `S`.

Setting `center` or `scale` is more memory-efficient than modifiying the input `x` directly. This is because the function will avoid constructing intermediate centered (possibly non-sparse) matrices.

Deferred centering and scaling

Many of the SVD algorithms (and computation of the cross-product) involve repeated matrix multiplications. We speed this up by using the ScaledMatrix class to defer centering (and to some extent, scaling) during matrix multiplication. The matrix multiplication is performed on the original matrix, and then the centering/scaling operations are applied to the matrix product. This allows direct use of the `%*%` method for each matrix representation, to exploit features of the underlying representation (e.g., sparsity) for greater speed.

Unfortunately, the speed-up with deferred centering comes at the cost of increasing the risk of catastrophic cancellation. The procedure requires subtraction of one large intermediate number from another to obtain the values of the final matrix product. This could result in a loss of numerical precision that compromises the accuracy of the various SVD algorithms.

The default approach is to explicitly create a dense in-memory centred/scaled matrix, possibly via block processing (see `blockGrid` in the **DelayedArray** package). This avoids problems with numerical precision as large intermediate values are not formed. In doing so, we consistently favour accuracy over speed unless the functions are specifically instructed to do otherwise, i.e., with `deferred=TRUE`.

Author(s)

Aaron Lun

`BiocSingularParam` *BiocSingularParam classes*

Description

Classes for specifying the type of singular value decomposition (SVD) algorithm and associated parameters.

Usage

```
ExactParam(deferred=FALSE, fold=Inf)

IrlbaParam(deferred=FALSE, fold=Inf, extra.work=7, ...)

RandomParam(deferred=FALSE, fold=Inf, ...)

FastAutoParam(deferred=FALSE, fold=Inf)

bsparam()
```

Arguments

`deferred`	Logical scalar indicating whether centering/scaling should be deferred, see ?`"BiocSingular-option`
`fold`	Numeric scalar specifying the minimum fold-difference for cross-product calculation, see ?`"BiocSingular-options"`.
`extra.work`	Integer scalar, additional dimensionality of the workspace in `runIrlbaSVD`.
`...`	Additional arguments to pass to `runIrlbaSVD` or `runRandomSVD`. This should not include any named arguments in those functions.

Details

The BiocSingularParam class controls dispatch of the `runSVD` generic to specific algorithms for SVD. The BiocSingularParam class itself is virtual, with several concrete subclasses available in this package:

- **`ExactParam`:** exact SVD with `runExactSVD`.
- **`IrlbaParam`:** approximate SVD with **irlba** via `runIrlbaSVD`.
- **`RandomParam`:** approximate SVD with **rsvd** via `runRandomSVD`.
- **`FastAutoParam`:** fast approximate SVD, chosen based on the matrix representation.

These objects also hold parameters specifying how each algorithm should be run on an arbitrary data set. See the associated documentation pages for each specific SVD method for more details.

Value

Each constructor returns a BiocSingularParam subclass of the same type, containing the specified parameters.

Methods

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

- **`show(object)`:** Display the class of `object` and a summary of the set parameters.
- **`bsfold(object)`:** Return a numeric scalar specifying the fold-difference for cross-product calculation, see “Computing the cross-product” in `?"BiocSingular-options"`.
- **`bsdeferred(object)`:** Return a logical scalar indicating whether centering and scaling should be deferred. see “Deferred centering and scaling” in `?"BiocSingular-options"`.

Automatically choosing a fast SVD algorithm

Running `runSVD` with a FastAutoParam object will dispatch to `runIrlbaSVD` for most matrices. This is the default choice as IRLBA is fast and the approximation is highly similar to the exact SVD result. The exception is that of instances of the DelayedMatrix class, or any subclass that lacks its own specialized `%*%` method. In such cases, `runSVD` with a FastAutoParam will dispatch to `runRandomSVD`, which minimizes the number of matrix multiplication steps and the associated costly block processing. However, if a DelayedMatrix subclass has its own `%*%` method, it is assumed to be efficient enough to afford repeated multiplications in `runIrlbaSVD`.

Setting a session-wide default

`bsparam()` will return a session-wide value for the BiocSingularParam object, analogous to `bpparam()`. This defaults to a FastAutoParam object but can be modified by setting the `"BiocSingularParam.default"` global option to any BiocSingularParam object. Any code that uses `bsparam()` will automatically use this new default without needing to explicitly pass the BiocSingularParam object to those functions.

Author(s)

Aaron Lun

See Also

`runSVD` for generic dispatch.

`runExactSVD`, `runIrlbaSVD` and `runRandomSVD` for specific methods.

Examples

```
ExactParam()

IrlbaParam(tol=1e-8)

RandomParam(q=20)

# Modifying the default.
bsparam()
options(BiocSingularParam.default=IrlbaParam())
bsparam()
```

`DeferredMatrix` *The DeferredMatrix class*

Description

This has been deprecated in favor of the ScaledMatrix class from the **ScaledMatrix** package - use those constructors instead.

`LowRankMatrix` *The LowRankMatrix class*

Description

Definitions of the LowRankMatrixSeed and LowRankMatrix classes and their associated methods. These classes are designed to provide a memory-efficient representation of a low-rank reconstruction, e.g., after a principal components analysis.

Usage

```
LowRankMatrixSeed(rotation, components)

LowRankMatrix(rotation, components)
```

Arguments

`rotation`	A matrix-like object where each row corresponds to a row of the LowRankMatrix object. This can alternatively be a LowRankMatrixSeed, in which case any value of `components` is ignored.
`components`	A matrix-like object where each row corresponds to a column of the LowRankMatrix object.

Value

The `LowRankMatrixSeed` constructor will return a LowRankMatrixSeed object.

The `LowRankMatrix` constructor will return a LowRankMatrix object equivalent to `tcrossprod(rotation, components)`.

Methods for LowRankMatrixSeed objects

LowRankMatrixSeed objects are implemented as DelayedMatrix backends. They support standard operations like `dim`, `dimnames` and `extract_array`.

Passing a LowRankMatrixSeed object to the `DelayedArray` constructor will create a LowRankMatrix object.

Methods for LowRankMatrix objects

LowRankMatrix objects are derived from DelayedMatrix objects and support all of valid operations on the latter. Subsetting, transposition and replacement of row/column names are specialized for greater efficiency when operating on LowRankMatrix instances, and will return a new LowRankMatrix rather than a DelayedMatrix.

All other operations applied to a LowRankMatrix will use the underlying **DelayedArray** machinery. Unary or binary operations will generally create a new DelayedMatrix instance containing a LowRankMatrixSeed.

Author(s)

Aaron Lun

See Also

`runPCA` to generate the rotation and component matrices.

Examples

```
a <- matrix(rnorm(100000), ncol=20)
out <- runPCA(a, rank=10)

lr <- LowRankMatrix(out$rotation, out$x)
```

`ResidualMatrix` *The ResidualMatrix class*

Description

This class is deprecated, see the exact same class in the **ResidualMatrix** package.

Usage

```
ResidualMatrixSeed(x, design=NULL)

ResidualMatrix(x, design=NULL)
```

Arguments

`x`	A matrix-like object. This can alternatively be a ResidualMatrixSeed, in which case `design` is ignored.
`design`	A numeric matrix containing the experimental design, to be used for linear model fitting on each *column* of `x`.

Value

The `ResidualMatrixSeed` constructor will return a ResidualMatrixSeed object.

The `ResidualMatrix` constructor will return a ResidualMatrix object, containing values equivalent to `lm.fit(x=design, y=x)$residuals`.

Author(s)

Aaron Lun

`runExactSVD` *Exact SVD*

Description

Perform an exact singular value decomposition.

Usage

```
runExactSVD(x, k=min(dim(x)), nu=k, nv=k, center=FALSE, scale=FALSE,
    deferred=FALSE, fold=Inf, BPPARAM=SerialParam())
```

Arguments

`x`	A numeric matrix-like object to use in the SVD.
`k`	Integer scalar specifying the number of singular values to return.
`nu`	Integer scalar specifying the number of left singular vectors to return.
`nv`	Integer scalar specifying the number of right singular vectors to return.
`center`	A logical scalar indicating whether columns should be centered. Alternatively, a numeric vector or `NULL` - see `?"BiocSingular-options"`.
`scale`	A logical scalar indicating whether columns should be scaled. Alternatively, a numeric vector or `NULL` - see `?"BiocSingular-options"`.
`deferred`	Logical scalar indicating whether centering/scaling should be deferred, see `?"BiocSingular-option`
`fold`	Numeric scalar specifying the minimum fold difference between dimensions of `x` to compute the cross-product, see `?"BiocSingular-options"`.
`BPPARAM`	A BiocParallelParam object specifying how parallelization should be performed.

Details

If any of `k`, `nu` or `nv` exceeds `min(dim(x))`, they will be capped and a warning will be raised. The exception is when they are explicitly set to `Inf`, in which case all singular values/vectors of `x` are returned without any warning.

Note that parallelization via `BPPARAM` is only applied to the calculation of the cross-product. It has no effect for near-square matrices where the SVD is computed directly.

Value

A list containing:

- `d`, a numeric vector of the first `k` singular values.
- `u`, a numeric matrix with `nrow(x)` rows and `nu` columns. Each column contains a left singular vector.
- `u`, a numeric matrix with `ncol(x)` rows and `nv` columns. Each column contains a right singular vector.

Author(s)

Aaron Lun

See Also

`svd` for the underlying algorithm.

Examples

```
a <- matrix(rnorm(100000), ncol=20)
out <- runExactSVD(a)
str(out)
```

`runIrlbaSVD` *Approximate SVD with* **irlba**

Description

Perform an approximate singular value decomposition with the augmented implicitly restarted Lanczos bidiagonalization algorithm.

Usage

```
runIrlbaSVD(x, k=5, nu=k, nv=k, center=FALSE, scale=FALSE, deferred=FALSE,
    extra.work=7, ..., fold=Inf, BPPARAM=SerialParam())
```

Arguments

`x`	A numeric matrix-like object to use in the SVD.
`k`	Integer scalar specifying the number of singular values to return.
`nu`	Integer scalar specifying the number of left singular vectors to return.
`nv`	Integer scalar specifying the number of right singular vectors to return.
`center`	A logical scalar indicating whether columns should be centered. Alternatively, a numeric vector or `NULL` - see `?"BiocSingular-options"`.
`scale`	A logical scalar indicating whether columns should be scaled. Alternatively, a numeric vector or `NULL` - see `?"BiocSingular-options"`.
`deferred`	Logical scalar indicating whether centering/scaling should be deferred, see `?"BiocSingular-option`
`extra.work`	Integer scalar specifying the additional number of dimensions to use for the working subspace.
`...`	Further arguments to pass to `irlba`.
`fold`	Numeric scalar specifying the minimum fold difference between dimensions of `x` to compute the cross-product, see `?"BiocSingular-options"`.
`BPPARAM`	A BiocParallelParam object specifying how parallelization should be performed.

Details

If `BPPARAM` has only 1 worker and a cross-product is not being computed, this function will use `irlba`'s own `center` and `scale` arguments. This is effectively equivalent to deferred centering and scaling, despite the setting of `deferred=FALSE`.

For multiple workers, this function will parallelize all multiplication operations involving `x` according to the supplied `BPPARAM`.

The total dimensionality of the working subspace is defined as the maximum of `k`, `nu` and `nv`, plus the `extra.work`.

Value

A list containing:

- `d`, a numeric vector of the first `k` singular values.
- `u`, a numeric matrix with `nrow(x)` rows and `nu` columns. Each column contains a left singular vector.
- `u`, a numeric matrix with `ncol(x)` rows and `nv` columns. Each column contains a right singular vector.

Author(s)

Aaron Lun

See Also

`irlba` for the underlying algorithm.

Examples

```
a <- matrix(rnorm(100000), ncol=20)
out <- runIrlbaSVD(a)
str(out)
```

`runPCA` *Principal components analysis*

Description

Perform a principal components analysis (PCA) on a target matrix with a specified SVD algorithm.

Usage

```
runPCA(x, ...)

## S4 method for signature 'ANY'
runPCA(x, rank, center=TRUE, scale=FALSE, get.rotation=TRUE,
    get.pcs=TRUE, ...)
```

Arguments

`x`	A numeric matrix-like object with samples as rows and variables as columns.
`rank`	Integer scalar specifying the number of principal components to retain.
`center`	A logical scalar indicating whether columns of `x` should be centered before the PCA is performed. Alternatively, a numeric vector of length `ncol(x)` containing the value to subtract from each column of `x`.
`scale`	A logical scalar indicating whether columns of `x` should be scaled to unit variance before the PCA is performed. Alternatively, a numeric vector of length `ncol(x)` containing the scaling factor for each column of `x`.
`get.rotation`	A logical scalar indicating whether rotation vectors should be returned.
`get.pcs`	A logical scalar indicating whether the principal component scores should be returned.
`...`	For the generic, this contains arguments to pass to methods upon dispatch. For the `ANY` method, this contains further arguments to pass to `runSVD`. This includes `BSPARAM` to specify the algorithm that should be used, and `BPPARAM` to control parallelization.

Details

This function simply calls `runSVD` and converts the results into a format similar to that returned by `prcomp`.

The generic is exported to allow other packages to implement their own `runPCA` methods for other `x` objects, e.g., **scater** for SingleCellExperiment inputs.

Value

A list is returned containing:

- `sdev`, a numeric vector of length `rank` containing the standard deviations of the first `rank` principal components.
- `rotation`, a numeric matrix with `rank` columns and `nrow(x)` rows, containing the first `rank` rotation vectors. This is only returned if `get.rotation=TRUE`.
- `x`, a numeric matrix with `rank` columns and `ncol(x)` rows, containing the scores for the first `rank` principal components. This is only returned if `get.pcs=TRUE`.

Author(s)

Aaron Lun

See Also

`runSVD` for the underlying SVD function.

?`BiocSingularParam` for details on the algorithm choices.

Examples

```
a <- matrix(rnorm(100000), ncol=20)
str(out <- runPCA(a, rank=10))
```

`runRandomSVD` *Approximate SVD with* **rsvd**

Description

Perform a randomized singular value decomposition.

Usage

```
runRandomSVD(x, k=5, nu=k, nv=k, center=FALSE, scale=FALSE, deferred=FALSE,
    ..., fold=Inf, BPPARAM=SerialParam())
```

Arguments

`x`	A numeric matrix-like object to use in the SVD.
`k`	Integer scalar specifying the number of singular values to return.
`nu`	Integer scalar specifying the number of left singular vectors to return.
`nv`	Integer scalar specifying the number of right singular vectors to return.
`center`	A logical scalar indicating whether columns should be centered. Alternatively, a numeric vector or `NULL` - see ?`"BiocSingular-options"`.
`scale`	A logical scalar indicating whether columns should be scaled. Alternatively, a numeric vector or `NULL` - see ?`"BiocSingular-options"`.
`deferred`	Logical scalar indicating whether centering/scaling should be deferred, see ?`"BiocSingular-option`
`...`	Further arguments to pass to `rsvd`.
`fold`	Numeric scalar specifying the minimum fold difference between dimensions of `x` to compute the cross-product, see ?`"BiocSingular-options"`.
`BPPARAM`	A BiocParallelParam object specifying how parallelization should be performed.

Details

All multiplication operations in `rsvd` involving `x` will be parallelized according to the supplied `BPPARAM`.

The dimensionality of the working subspace is defined as the maximum of `k`, `nu` and `nv`, plus the `q` specified in `...`.

Value

A list containing:

- d, a numeric vector of the first `k` singular values.
- u, a numeric matrix with `nrow(x)` rows and `nu` columns. Each column contains a left singular vector.
- u, a numeric matrix with `ncol(x)` rows and `nv` columns. Each column contains a right singular vector.

Author(s)

Aaron Lun

See Also

`rsvd` for the underlying algorithm.

Examples

```
a <- matrix(rnorm(100000), ncol=20)
out <- runRandomSVD(a)
str(out)
```

`runSVD` *Run SVD*

Description

Perform a singular value decomposition on an input matrix with a specified algorithm.

Usage

```
runSVD(x, k, nu=k, nv=k, center=FALSE, scale=FALSE,
    BPPARAM=SerialParam(), ..., BSPARAM=ExactParam())
```

Arguments

`x`	A numeric matrix-like object to use in the SVD.
`k`	Integer scalar specifying the number of singular values to return.
`nu`	Integer scalar specifying the number of left singular vectors to return.
`nv`	Integer scalar specifying the number of right singular vectors to return.
`center`	Numeric vector, logical scalar or `NULL`, specifying values to subtract from each column of `x` - see `?"BiocSingular-options"`.
`scale`	Numeric vector, logical scalar or `NULL`, specifying values to divide each column of `x` - see `?"BiocSingular-options"`.
`BPPARAM`	A BiocParallelParam object specifying how parallelization should be performed.
`...`	Further arguments to pass to specific methods.
`BSPARAM`	A BiocSingularParam object specifying the type of algorithm to run.

Details

The class of `BSPARAM` will determine the algorithm that is used, see ?`BiocSingularParam` for more details. The default is to use an exact SVD via `runExactSVD`.

Value

A list containing:

- `d`, a numeric vector of the first `k` singular values.
- `u`, a numeric matrix with `nrow(x)` rows and `nu` columns. Each column contains a left singular vector.
- `v`, a numeric matrix with `ncol(x)` rows and `nv` columns. Each column contains a right singular vector.

Author(s)

Aaron Lun

See Also

`runExactSVD`, `runIrlbaSVD` and `runRandomSVD` for the specific functions.

Examples

```
a <- matrix(rnorm(100000), ncol=20)

out.exact0 <- runSVD(a, k=4)
str(out.exact0)

out.exact <- runSVD(a, k=4, BSPARAM=ExactParam())
str(out.exact)

out.irlba <- runSVD(a, k=4, BSPARAM=IrlbaParam())
str(out.exact)

out.random <- runSVD(a, k=4, BSPARAM=RandomParam())
str(out.random)
```

Index

[,LowRankMatrix,ANY,ANY,ANY-method (LowRankMatrix), 5

BiocParallelParam, *7*, *9*, *11*, *12*
BiocSingular options, 2
BiocSingular-options (BiocSingular options), 2
BiocSingularParam, 3, *11–13*
BiocSingularParam-class (BiocSingularParam), 3
blockGrid, *3*
bpparam, *4*
bsdeferred (BiocSingularParam), 3
bsfold (BiocSingularParam), 3
bsparam (BiocSingularParam), 3

DeferredMatrix, 5
DeferredMatrixSeed (DeferredMatrix), 5
DelayedArray, *6*
DelayedArray,LowRankMatrixSeed-method (LowRankMatrix), 5
DelayedMatrix, *4*, *6*
dim,LowRankMatrixSeed-method (LowRankMatrix), 5
dimnames,LowRankMatrixSeed-method (LowRankMatrix), 5
dimnames<-,LowRankMatrix,ANY-method (LowRankMatrix), 5

ExactParam (BiocSingularParam), 3
ExactParam-class (BiocSingularParam), 3
extract_array,LowRankMatrixSeed-method (LowRankMatrix), 5

FastAutoParam (BiocSingularParam), 3
FastAutoParam-class (BiocSingularParam), 3

irlba, *2*, *9*
IrlbaParam (BiocSingularParam), 3
IrlbaParam-class (BiocSingularParam), 3

LowRankMatrix, 5
LowRankMatrix-class (LowRankMatrix), 5
LowRankMatrixSeed (LowRankMatrix), 5
LowRankMatrixSeed-class (LowRankMatrix), 5

prcomp, *10*

RandomParam (BiocSingularParam), 3
RandomParam-class (BiocSingularParam), 3
ResidualMatrix, 6
ResidualMatrixSeed (ResidualMatrix), 6
rsvd, *2*, *11*, *12*
runExactSVD, *4*, *5*, 7, *13*
runIrlbaSVD, *3–5*, 8, *13*
runPCA, *6*, 10
runPCA,ANY-method (runPCA), 10
runRandomSVD, *3–5*, 11, *13*
runSVD, *4*, *5*, *10*, *11*, 12
runSVD,ExactParam-method (runSVD), 12
runSVD,FastAutoParam-method (runSVD), 12
runSVD,IrlbaParam-method (runSVD), 12
runSVD,missing-method (runSVD), 12
runSVD,RandomParam-method (runSVD), 12

scale, *2*
ScaledMatrix, *3*, *5*
show,BiocSingularParam-method (BiocSingularParam), 3
show,IrlbaParam-method (BiocSingularParam), 3
show,LowRankMatrixSeed-method (LowRankMatrix), 5
show,RandomParam-method (BiocSingularParam), 3
svd, *8*

t,LowRankMatrix-method (LowRankMatrix), 5