Package 'plaid'

May 26, 2026

Title PLAID ultrafast gene set enrichment scoring

Version 1.1.0

Description
PLAID (Pathway Level Average Intensity Detection) is an ultra-fast method to compute single-sample enrichment scores for gene expression or proteomics data. For each sample, plaid computes the gene set score as the average intensity of the genes/proteins in the gene set. The output is a gene set score matrix suitable for further analyses.

License GPL-3

URL https://github.com/bigomics/plaid, https://bigomics.github.io/plaid/

BugReports https://github.com/bigomics/plaid/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests BiocStyle, knitr, rmarkdown, sparseMatrixStats, testthat (>= 3.0.0)

Config/testthat/edition 3

biocViews GeneSetEnrichment, GeneExpression, Proteomics

Depends R (>= 4.3.3)

Imports Matrix, MatrixGenerics, matrixStats, methods, parallel, Rfast, qlcMatrix, GSVA, fgsea, SummarizedExperiment, BiocSet, stats, utils

VignetteBuilder knitr

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

git_branch devel

git_last_commit 6d2ad5c

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-25

Author Ivo Kwee [aut] (ORCID: <https://orcid.org/0000-0002-2751-4218>), Antonino Zito [cre] (ORCID: <https://orcid.org/0000-0003-1931-984X>)

Maintainer Antonino Zito <antonino.zito@bigomics.ch>

Contents

`chunked_crossprod` *Chunked computation of cross product*

Description

Compute crossprod (t(x) %*% y) for very large y by computing in chunks.

Usage

```
chunked_crossprod(x, y, chunk = NULL)
```

Arguments

x	Matrix First matrix for multiplication. Can be sparse.
y	Matrix Second matrix for multiplication. Can be sparse.
chunk	Integer Chunk size (max number of columns) for computation.

Value

Matrix. Result of matrix cross product.

`colranks` *Compute columnwise ranks of matrix*

Description

Computes columnwise rank of matrix. Can be sparse. Tries to call optimized functions from Rfast or matrixStats.

Usage

```
colranks(
  X,
  sparse = NULL,
  signed = FALSE,
  keep.zero = FALSE,
  ties.method = "average"
)
```

Arguments

`X`	Input matrix
`sparse`	Logical indicating to use sparse methods
`signed`	Logical indicating using signed ranks
`keep.zero`	Logical indicating whether to keep zero as ranked zero
`ties.method`	Character Choice of ties.method

Value

Matrix of columnwise ranks with same dimensions as input.

Examples

```
# Create example matrix
set.seed(123)
X <- matrix(rnorm(100), nrow = 10, ncol = 10)
rownames(X) <- paste0("Gene", 1:10)
colnames(X) <- paste0("Sample", 1:10)

# Compute column ranks
ranks <- colranks(X)
print(ranks[1:5, 1:5])

# Compute signed ranks
signed_ranks <- colranks(X, signed = TRUE)
print(signed_ranks[1:5, 1:5])
```

`cor_sparse_matrix` *Calculate sparse correlation matrix handling missing values*

Description

Calculate sparse correlation matrix handling missing values

Usage

```
cor_sparse_matrix(G, mat)
```

Arguments

`G`	Sparse matrix containing gene sets
`mat`	Matrix of values

Details

If mat has no missing values, calculates correlation directly using corSparse. Otherwise computes column-wise correlations only using non-missing values.

Value

Correlation matrix between G and mat

`dualGSEA` *Reimplementation of dualGSEA (Bull et al., 2024) but defaults with replaid backend. For the preranked test we still use fgsea. Should be much faster than original using fgsea + GSVA::ssGSEA.*

Description

Reimplementation of dualGSEA (Bull et al., 2024) but defaults with replaid backend. For the preranked test we still use fgsea. Should be much faster than original using fgsea + GSVA::ssGSEA.

Usage

```
dualGSEA(
  X,
  y,
  G,
  gmt = NULL,
  gsetX = NULL,
  fc.method = c("fgsea", "rankcor", "ztest", "ttest", "cor")[2],
  ss.method = c("plaid", "replaid.ssgsea", "replaid.gsva", "ssgsea", "gsva")[1],
  metap.method = c("stouffer", "fisher", "maxp")[1],
  pv1 = NULL,
  pv2 = NULL,
  sort.by = "p.dual"
)
```

Arguments

X	Expression matrix with genes on rows and samples ont columns
y	Binary vector (0/1) indicating group membership
G	Sparse matrix of gene sets. Non-zero entry indicates gene/feature is part of gene sets. Features on rows, gene sets on columns.
gmt	List of gene sets in GMT format
gsetX	Optional pre-computed matrix of gene set enrichment scores with gene sets on rows and samples on columns. If NULL (default), scores will be computed using the method specified by ss.method. Providing pre-computed scores improves efficiency when running multiple analyses.
fc.method	Method for fold change testing ("fgsea", "ztest", "ttest", "rankcor", "cor")
ss.method	Method for single-sample enrichment ("plaid", "replaid.ssgsea", "replaid.gsva", "ssgsea", "gsva")
metap.method	Method for combining p-values ("stouffer", "fisher" or "maxp"). Default "stouffer".
pv1	Pre-computed p-values from fold change test. If NULL, will be computed based on fc.test.
pv2	Pre-computed p-values from single sample test. If NULL, will be computed using gset_ttest.
sort.by	Column name to sort results by ("p.dual", "gsetFC", "p.fc", "p.ss"). Default "p.dual".

Value

Data frame with columns: gsetFC (gene set fold change), size (gene set size), p.fc (p-value from fold change test), p.ss (p-value from single sample test), p.dual (combined p-value), and q.dual (FDR-adjusted combined p-value).

Examples

```
# Create example expression matrix
set.seed(123)
X <- matrix(rnorm(1000), nrow = 100, ncol = 20)
rownames(X) <- paste0("GENE", 1:100)
colnames(X) <- paste0("Sample", 1:20)

# Create binary group vector
y <- rep(c(0, 1), each = 10)

# Create example gene sets
gmt <- list(
  "Pathway1" = paste0("GENE", 1:20),
  "Pathway2" = paste0("GENE", 15:35),
  "Pathway3" = paste0("GENE", 30:50)
)

# Perform dualGSEA with correlation test (fast method)
results_cor <- dualGSEA(X, y, G = NULL, gmt = gmt, fc.method = "cor", ss.method = "replaid.gsva")
print(head(results_cor))
```

```
# Perform dualGSEA with fgsea (requires fgsea package)
if (requireNamespace("fgsea", quietly = TRUE)) {
 results <- dualGSEA(X, y, G = NULL, gmt = gmt, fc.method = "fgsea", ss.method = "replaid.ssgsea")
  print(head(results))
}
```

`fc_ttest` *T-test statistical testing of differentially enrichment*

Description

This function performs statistical testing for differential enrichment using plaid

Usage

```
fc_ttest(fc, G, sort.by = "pvalue")
```

Arguments

`fc`	Vector of logFC values
`G`	Sparse matrix of gene sets. Non-zero entry indicates gene/feature is part of gene sets. Features on rows, gene sets on columns.
`sort.by`	Column name to sort results by ("pvalue", "gsetFC", or "none")

Value

Data frame with columns: gsetFC (gene set fold change), pvalue (p-value from one-sample t-test), and qvalue (FDR-adjusted p-value).

`fc_ztest` *Z-test statistical testing of differentially enrichment*

Description

This function performs statistical testing for differential enrichment using plaid

Usage

```
fc_ztest(fc, G, zmat = FALSE, alpha = 0.5)
```

Arguments

`fc`	Vector of logFC values
`G`	Sparse matrix of gene sets. Non-zero entry indicates gene/feature is part of gene sets. Features on rows, gene sets on columns.
`zmat`	Logical indicating to return z-matrix
`alpha`	Scalar weight for SD estimation. Default 0.5.

Value

List with element: z_statistic (z-statistic from one-sample z-test), p_value (p-value from one-sample z-test), and zmat (z-matrix).

`gmt2mat` *Convert GMT to Binary Matrix*

Description

Convert a GMT file (Gene Matrix Transposed) to a binary matrix, where rows represent genes and columns represent gene sets. The binary matrix indicates presence or absence of genes in a gene set.

Usage

```
gmt2mat(
  gmt,
  max.genes = -1,
  ntop = -1,
  sparse = TRUE,
  bg = NULL,
  use.multicore = TRUE
)
```

Arguments

`gmt`	List representing the GMT file: each element is a character vector representing a gene set.
`max.genes`	Max number of genes to include in the binary matrix. Default = -1 to include all genes.
`ntop`	Number of top genes to consider for each gene set. Default = -1 to include all genes.
`sparse`	Logical: create a sparse matrix. Default `TRUE`. If `FALSE` creates a dense matrix.
`bg`	Character vector of background gene set. Default `NULL` to consider all unique genes.
`use.multicore`	Logical: use parallel processing ('parallel' R package). Default `TRUE`.

Value

A binary matrix representing the presence or absence of genes in each gene set. Rows correspond to genes, and columns correspond to gene sets.

Examples

```
# Create example GMT data
gmt <- list(
  "Pathway1" = c("GENE1", "GENE2", "GENE3"),
  "Pathway2" = c("GENE2", "GENE4", "GENE5"),
  "Pathway3" = c("GENE1", "GENE5", "GENE6")
)
```

```
# Convert to binary matrix
mat <- gmt2mat(gmt)
print(mat)

# Create dense matrix instead of sparse
mat_dense <- gmt2mat(gmt, sparse = FALSE)
print(mat_dense)
```

`gset.rankcor` *Calculate gene set rank correlation*

Description

Compute rank correlation between a gene rank vector/matrix and gene sets

Usage

```
gset.rankcor(rnk, gset, compute.p = FALSE, use.rank = TRUE)
```

Arguments

`rnk`	Numeric vector or matrix of gene ranks, with genes as row names
`gset`	Numeric matrix of gene sets, with genes as row/column names
`compute.p`	Logical indicating whether to compute p-values
`use.rank`	Logical indicating whether to rank transform rnk before correlation

Details

This function calculates sparse rank correlation between rnk and each column of gset using `qlcMatrix::corSparse()`. It handles missing values in rnk by computing column-wise correlations.

P-values are computed from statistical distribution

Value

Named list with components:

- rho - Matrix of correlation coefficients between rnk and gset
- p.value - Matrix of p-values for correlation (if compute.p = TRUE)
- q.value - Matrix of FDR adjusted p-values (if compute.p = TRUE)

Examples

```
# Create example rank vector
set.seed(123)
ranks <- rnorm(100)
names(ranks) <- paste0("GENE", 1:100)

# Create example gene sets as sparse matrix
gmt <- list(
  "Pathway1" = paste0("GENE", 1:20),
  "Pathway2" = paste0("GENE", 15:35),
```

```
  "Pathway3" = paste0("GENE", 30:50)
)
genesets <- gmt2mat(gmt)

# Calculate rank correlation
result <- gset.rankcor(ranks, genesets, compute.p = TRUE)
print(result$rho)
print(result$p.value)
```

`gset_averageCLR` *Compute geneset expression as the average log-ration of genes in the geneset. Requires log-expression matrix X and (sparse) geneset matrix matG.*

Description

Compute geneset expression as the average log-ration of genes in the geneset. Requires log-expression matrix X and (sparse) geneset matrix matG.

Usage

```
gset_averageCLR(X, matG, center = TRUE)
```

Arguments

`X`	Log-expression matrix with genes on rows and samples on columns
`matG`	Sparse gene set matrix with genes on rows and gene sets on columns
`center`	Logical indicating whether to center the results

Value

Matrix of gene set expression scores with gene sets on rows and samples on columns.

`gset_ttest` *Perform t-test on gene set scores*

Description

Perform t-test on gene set scores

Usage

```
gset_ttest(gsetX, y)
```

Arguments

`gsetX`	Matrix of gene set scores with gene sets on rows and samples on columns
`y`	Binary vector (0/1) indicating group membership

Value

Data frame with columns: diff (difference in means), statistic (t-statistic), pvalue (p-value), and other t-test results.

`mat.rowsds` *Calculate row standard deviations for matrix*

Description

Calculate row standard deviations for matrix

Usage

```
mat.rowsds(X)
```

Arguments

`X`	Input matrix (can be sparse or dense)

Value

Vector of row standard deviations.

`mat2gmt` *Convert Binary Matrix to GMT*

Description

Convert binary matrix to a GMT (Gene Matrix Transposed) list. The binary matrix indicates presence or absence of genes in each gene set. Rows represent genes and columns represent gene sets.

Usage

```
mat2gmt(mat)
```

Arguments

`mat`	Matrix with non-zero entries representing genes in each gene set. Rows represent genes and columns represent gene sets.

Value

A list of vector representing each gene set. Each list element correspond to a gene set and is a vector of genes

Examples

```
# Create example binary matrix
mat <- matrix(0, nrow = 6, ncol = 3)
rownames(mat) <- paste0("GENE", 1:6)
colnames(mat) <- paste0("Pathway", 1:3)
mat[1:3, 1] <- 1  # Pathway1: GENE1, GENE2, GENE3
mat[c(2,4,5), 2] <- 1  # Pathway2: GENE2, GENE4, GENE5
mat[c(1,5,6), 3] <- 1  # Pathway3: GENE1, GENE5, GENE6

# Convert to GMT list
gmt <- mat2gmt(mat)
print(gmt)
```

`matrix_metap` *Matrix version for combining p-values using fisher or stouffer method. Much faster than doing metap::sumlog() and metap::sumz()*

Description

Matrix version for combining p-values using fisher or stouffer method. Much faster than doing metap::sumlog() and metap::sumz()

Usage

```
matrix_metap(plist, method = "stouffer")
```

Arguments

`plist`	List of p-value vectors or matrix of p-values
`method`	Method for combining p-values ("fisher"/"sumlog" or "stouffer"/"sumz")

Value

Vector of combined p-values.

`matrix_onesample_ttest` *Perform one-sample t-test on matrix with gene sets*

Description

Perform one-sample t-test on matrix with gene sets

Usage

```
matrix_onesample_ttest(Fm, G)
```

Arguments

Fm	Vector of feature values (e.g., fold changes)
G	Sparse matrix of gene sets with genes on rows and gene sets on columns

Value

List containing mean, t-statistic, and p-value matrices.

---

normalize_medians *Normalize column medians of matrix*

Description

This function normalizes the column medians of matrix x. It calls optimized functions from the matrixStats package.

Usage

```
normalize_medians(x, ignore.zero = NULL)
```

Arguments

x	Input matrix
ignore.zero	Logical indicating whether to ignore zeros to exclude for median calculation

Value

Matrix with normalized column medians.

Examples

```
# Create example matrix
set.seed(123)
x <- matrix(rnorm(100), nrow = 10, ncol = 10)
x[1:3, 1:3] <- 0  # Add some zeros

# Normalize medians
x_norm <- normalize_medians(x)
head(x_norm)
```

plaid *Compute PLAID single-sample enrichment score*

Description

Compute single-sample geneset expression as the average log-expression f genes in the geneset. Requires log-expression matrix X and (sparse) geneset matrix matG. If you have gene sets as a gmt list, please convert it first using the function `gmt2mat()`.

Usage

```
plaid(
  X,
  matG,
  stats = c("mean", "sum"),
  chunk = NULL,
  normalize = TRUE,
  nsmooth = 3,
  assay = "logcounts",
  min.genes = 5,
  max.genes = 500
)
```

Arguments

`X`	Log-transformed expr. matrix. Genes on rows, samples on columns. Also accepts SummarizedExperiment or SingleCellExperiment objects.
`matG`	Gene sets sparse matrix. Genes on rows, gene sets on columns. Also accepts BiocSet objects or GMT lists (named list of gene vectors).
`stats`	Score computation stats: mean or sum of intensity. Default 'mean'.
`chunk`	Logical: use chunks for large matrices. Default 'NULL' for autodetect.
`normalize`	Logical: median normalize results or not. Default 'TRUE'.
`nsmooth`	Smoothing parameter for more stable average when stats="mean". Default 3.
`assay`	Character: assay name to extract from SummarizedExperiment/SingleCellExperiment. Default "logcounts".
`min.genes`	Integer: minimum genes per gene set (for BiocSet/GMT input). Default 5.
`max.genes`	Integer: maximum genes per gene set (for BiocSet/GMT input). Default 500.

Details

PLAID needs the gene sets as sparse matrix. If you have your collection of gene sets a a list, we need first to convert the gmt list to matrix format.

We recommend to run PLAID on the log transformed expression matrix, not on the counts, as the average in the logarithmic space is more robust and is in concordance to calculating the geometric mean.

It is not necessary to normalize your expression matrix before running PLAID because PLAID performs median normalization of the enrichment scores afterwards.

It is recommended to use sparse matrix as PLAID relies on sparse matrix computations. But, PLAID is also fast for dense matrices.

PLAID can also be run on the ranked matrix. This corresponds to the singscore (Fouratan et al., 2018). PLAID can also be run on the (non-logarithmic) counts which can be used to calculate the scSE score (Pont et al., 2019).

PLAID is fast and memery efficient because it uses efficient sparse matrix computation. When input matrix is very large, PLAID performs 'chunked' computation by splitting the matrix in chunks.

Although `X` and `matG` are generally sparse, the result matrix `gsetX` generally is dense and can thus be very large. Example: computing gene set scores for 10K gene sets on 1M cells will create a 10K x 1M dense matrix which requires ~75GB memory.

PLAID now automatically detects and handles Bioconductor objects. If X is a SummarizedExperiment or SingleCellExperiment, it will extract the appropriate assay. If matG is a BiocSet object or GMT list, it will be converted to sparse matrix format automatically.

Value

Matrix of single-sample enrichment scores. Gene sets on rows, samples on columns.

Examples

```
library(plaid)

# Create example expression matrix
set.seed(123)
X <- matrix(rnorm(1000), nrow = 100, ncol = 10)
rownames(X) <- paste0("GENE", 1:100)
colnames(X) <- paste0("Sample", 1:10)

# Create example gene sets
gmt <- list(
  "Pathway1" = paste0("GENE", 1:20),
  "Pathway2" = paste0("GENE", 15:35),
  "Pathway3" = paste0("GENE", 30:50)
)
matG <- gmt2mat(gmt)

# Compute PLAID scores
gsetX <- plaid(X, matG)
print(dim(gsetX))
print(gsetX[1:3, 1:5])

# Use sum statistics instead of mean
gsetX_sum <- plaid(X, matG, stats = "sum")


# Using real data (if available in package)
extdata_path <- system.file("extdata", "pbmc3k-50cells.rda", package = "plaid")
if (file.exists(extdata_path)) {
  load(extdata_path)
  hallmarks <- system.file("extdata", "hallmarks.gmt", package = "plaid")
  gmt <- read.gmt(hallmarks)
  matG <- gmt2mat(gmt)
  gsetX <- plaid(X, matG)
}
```

`read.gmt` *Read GMT File*

Description

Read data from a GMT file (Gene Matrix Transposed). The GMT format is commonly used to store gene sets or gene annotations.

Usage

```
read.gmt(gmt.file, dir = NULL, add.source = FALSE, nrows = -1)
```

Arguments

`gmt.file`	Path to GMT file.
`dir`	(Optional) The directory where the GMT file is located.
`add.source`	(optional) Include the source information in the gene sets' names.
`nrows`	(optional) Number of rows to read from the GMT file.

Value

A list of gene sets: each gene set is represented as a character vector of gene names.

Examples

```
# Read GMT file (requires file to exist)
gmt_file <- system.file("extdata", "hallmarks.gmt", package = "plaid")
if (file.exists(gmt_file)) {
  gmt <- read.gmt(gmt_file)
  print(names(gmt))
  print(head(gmt[[1]]))

  # Read with source information
  gmt_with_source <- read.gmt(gmt_file, add.source = TRUE)
  print(head(names(gmt_with_source)))
}
```

`replaid.aucell` *Fast calculation of AUCell*

Description

Calculates single-sample enrichment AUCell (Aibar et al., 2017) using plaid back-end. The computation is 10-100x faster than the original code.

Usage

```
replaid.aucell(
  X,
  matG,
  aucMaxRank = NULL,
  assay = "logcounts",
  min.genes = 5,
  max.genes = 500
)
```

Arguments

`X`	Gene or protein expression matrix. Generally log transformed. See details. Genes on rows, samples on columns. Also accepts SummarizedExperiment or SingleCellExperiment objects.
`matG`	Gene sets sparse matrix. Genes on rows, gene sets on columns. Also accepts BiocSet objects or GMT lists.
`aucMaxRank`	Rank threshold (see AUCell paper). Default aucMaxRank = 0.05*nrow(X).
`assay`	Character: assay name for Bioconductor objects. Default "logcounts".
`min.genes`	Integer: minimum genes per gene set. Default 5.
`max.genes`	Integer: maximum genes per gene set. Default 500.

Details

Computing the AUCell score requires to compute the ranks of the expression matrix and approximating the AUC of a gene set. We have wrapped this in a single convenience function.

We have extensively compared the results of `replaid.aucell` and from the original `AUCell` R package and we showed good concordance of results in the score, logFC and p-values.

Value

Matrix of single-sample AUCell enrichment scores. Gene sets on rows, samples on columns.

Examples

```
# Create example expression matrix
set.seed(123)
X <- matrix(rnorm(500), nrow = 50, ncol = 10)
rownames(X) <- paste0("GENE", 1:50)
colnames(X) <- paste0("Sample", 1:10)
```

```
# Create example gene sets
gmt <- list(
  "Pathway1" = paste0("GENE", 1:15),
  "Pathway2" = paste0("GENE", 10:25)
)
matG <- gmt2mat(gmt)

# Compute AUCell scores
scores <- replaid.aucell(X, matG)
print(scores[1:2, 1:5])
```

`replaid.gsva` *Fast approximation of GSVA*

Description

Calculates single-sample enrichment GSVA (Hänzelmann et al., 2013) using plaid back-end. The computation is 10-100x faster than the original code.

Usage

```
replaid.gsva(
  X,
  matG,
  tau = 0,
  rowtf = c("z", "ecdf")[1],
  assay = "logcounts",
  min.genes = 5,
  max.genes = 500
)
```

Arguments

`X`	Gene or protein expression matrix. Generally log transformed. See details. Genes on rows, samples on columns. Also accepts SummarizedExperiment or SingleCellExperiment objects.
`matG`	Gene sets sparse matrix. Genes on rows, gene sets on columns. Also accepts BiocSet objects or GMT lists.
`tau`	Rank weight parameter (see GSVA publication). Default tau=0.
`rowtf`	Row transformation method ("z" or "ecdf"). Default "z".
`assay`	Character: assay name for Bioconductor objects. Default "logcounts".
`min.genes`	Integer: minimum genes per gene set. Default 5.
`max.genes`	Integer: maximum genes per gene set. Default 500.

Details

Computing the GSVA score requires to compute the CDF of the expression matrix, ranking and scoring the genesets. We have wrapped this in a single convenience function.

We have extensively compared the results of `replaid.gsva` and from the original `GSVA` R package and we showed good concordance of results in the score, logFC and p-values.

In the original formulation, GSVA uses an emperical CDF to transform expression of each feature to a (0;1) relative expression value. For efficiency reasons, this is here approximated by a z-transform (center+scale) of each row.

Value

Matrix of single-sample GSVA enrichment scores. Gene sets on rows, samples on columns.

Examples

```
# Create example expression matrix
set.seed(123)
X <- matrix(rnorm(500), nrow = 50, ncol = 10)
rownames(X) <- paste0("GENE", 1:50)
colnames(X) <- paste0("Sample", 1:10)

# Create example gene sets
gmt <- list(
  "Pathway1" = paste0("GENE", 1:15),
  "Pathway2" = paste0("GENE", 10:25)
)
matG <- gmt2mat(gmt)

# Compute GSVA scores
scores <- replaid.gsva(X, matG)
print(scores[1:2, 1:5])
```

`replaid.scse` *Fast calculation of scSE score*

Description

Calculates Single-Cell Signature Explorer (Pont et al., 2019) scores using plaid back-end. The computation is 10-100x faster than the original code.

Usage

```
replaid.scse(
  X,
  matG,
  removeLog2 = NULL,
  scoreMean = FALSE,
  assay = "logcounts",
  min.genes = 5,
  max.genes = 500
)
```

Arguments

X	Gene or protein expression matrix. Generally log transformed. See details. Genes on rows, samples on columns. Also accepts SummarizedExperiment or SingleCellExperiment objects.
matG	Gene sets sparse matrix. Genes on rows, gene sets on columns. Also accepts BiocSet objects or GMT lists.
removeLog2	Logical for whether to remove the Log2, i.e. will apply power transform (base2) on input (default TRUE).
scoreMean	Logical for whether computing sum or mean as score (default FALSE).
assay	Character: assay name for Bioconductor objects. Default "logcounts".
min.genes	Integer: minimum genes per gene set. Default 5.
max.genes	Integer: maximum genes per gene set. Default 500.

Details

Computing the scSE requires running plaid on the linear (not logarithmic) score and perform additional normalization by the total UMI per sample. We have wrapped this in a single convenience function:

To replicate the original "sum-of-UMI" scSE score, set `removeLog2=TRUE` and `scoreMean=FALSE`. scSE and plaid scores become more similar for `removeLog2=FALSE` and `scoreMean=TRUE`.

We have extensively compared the results from `replaid.scse` and from the original scSE (implemented in GO lang) and we showed almost identical results in the score, logFC and p-values.

Value

Matrix of single-sample scSE enrichment scores. Gene sets on rows, samples on columns.

Examples

```
# Create example expression matrix (log-transformed)
set.seed(123)
X <- log2(matrix(rpois(500, lambda = 10) + 1, nrow = 50, ncol = 10))
rownames(X) <- paste0("GENE", 1:50)
colnames(X) <- paste0("Sample", 1:10)

# Create example gene sets
gmt <- list(
  "Pathway1" = paste0("GENE", 1:15),
  "Pathway2" = paste0("GENE", 10:25)
)
matG <- gmt2mat(gmt)

# Compute scSE scores (original method)
scores <- replaid.scse(X, matG, removeLog2 = TRUE, scoreMean = FALSE)
print(scores[1:2, 1:5])

# Compute scSE scores (mean method)
scores_mean <- replaid.scse(X, matG, removeLog2 = TRUE, scoreMean = TRUE)
print(scores_mean[1:2, 1:5])
```

`replaid.sing` *Fast calculation of singscore*

Description

Calculates single-sample enrichment singscore (Fouratan et al., 2018) using plaid back-end. The computation is 10-100x faster than the original code.

Usage

```
replaid.sing(X, matG, assay = "logcounts", min.genes = 5, max.genes = 500)
```

Arguments

`X`	Gene or protein expression matrix. Generally log transformed. See details. Genes on rows, samples on columns. Also accepts SummarizedExperiment or SingleCellExperiment objects.
`matG`	Gene sets sparse matrix. Genes on rows, gene sets on columns. Also accepts BiocSet objects or GMT lists.
`assay`	Character: assay name for Bioconductor objects. Default "logcounts".
`min.genes`	Integer: minimum genes per gene set. Default 5.
`max.genes`	Integer: maximum genes per gene set. Default 500.

Details

Computing the singscore requires to compute the ranks of the expression matrix. We have wrapped this in a single convenience function.

We have extensively compared the results of `replaid.sing` and from the original `singscore` R package and we showed identical result in the score, logFC and p-values.

Value

Matrix of single-sample singscore enrichment scores. Gene sets on rows, samples on columns.

Examples

```
# Create example expression matrix
set.seed(123)
X <- matrix(rnorm(500), nrow = 50, ncol = 10)
rownames(X) <- paste0("GENE", 1:50)
colnames(X) <- paste0("Sample", 1:10)

# Create example gene sets
gmt <- list(
  "Pathway1" = paste0("GENE", 1:15),
  "Pathway2" = paste0("GENE", 10:25)
)
matG <- gmt2mat(gmt)

# Compute singscore
scores <- replaid.sing(X, matG)
print(scores[1:2, 1:5])
```

`replaid.ssgsea` *Fast calculation of ssGSEA*

Description

Calculates single-sample enrichment singscore (Barbie et al., 2009; Hänzelmann et al., 2013) using plaid back-end. The computation is 10-100x faster than the original code.

Usage

```
replaid.ssgsea(
  X,
  matG,
  alpha = 0,
  assay = "logcounts",
  min.genes = 5,
  max.genes = 500
)
```

Arguments

`X`	Gene or protein expression matrix. Generally log transformed. See details. Genes on rows, samples on columns. Also accepts SummarizedExperiment or SingleCellExperiment objects.
`matG`	Gene sets sparse matrix. Genes on rows, gene sets on columns. Also accepts BiocSet objects or GMT lists.
`alpha`	Weighting factor for exponential weighting of ranks
`assay`	Character: assay name for Bioconductor objects. Default "logcounts".
`min.genes`	Integer: minimum genes per gene set. Default 5.
`max.genes`	Integer: maximum genes per gene set. Default 500.

Details

Computing ssGSEA score requires to compute the ranks of the expression matrix and weighting of the ranks. We have wrapped this in a single convenience function.

We have extensively compared the results of `replaid.ssgsea` and from the original `GSVA` R package and we showed highly similar results in the score, logFC and p-values. For alpha=0 we obtain exact results, for alpha>0 the results are highly similar but not exactly the same.

Value

Matrix of single-sample ssGSEA enrichment scores. Gene sets on rows, samples on columns.

Examples

```
# Create example expression matrix
set.seed(123)
X <- matrix(rnorm(500), nrow = 50, ncol = 10)
rownames(X) <- paste0("GENE", 1:50)
colnames(X) <- paste0("Sample", 1:10)
```

```
# Create example gene sets
gmt <- list(
  "Pathway1" = paste0("GENE", 1:15),
  "Pathway2" = paste0("GENE", 10:25)
)
matG <- gmt2mat(gmt)

# Compute ssGSEA scores (alpha = 0)
scores <- replaid.ssgsea(X, matG, alpha = 0)
print(scores[1:2, 1:5])

# Compute ssGSEA scores with weighting (alpha = 0.25)
scores_weighted <- replaid.ssgsea(X, matG, alpha = 0.25)
print(scores_weighted[1:2, 1:5])
```

`replaid.ucell` *Fast calculation of UCell*

Description

Calculates single-sample enrichment UCell (Andreatta et al., 2021) using plaid back-end. The computation is 10-100x faster than the original code.

Usage

```
replaid.ucell(
  X,
  matG,
  rmax = 1500,
  assay = "logcounts",
  min.genes = 5,
  max.genes = 500
)
```

Arguments

`X`	Gene or protein expression matrix. Generally log transformed. See details. Genes on rows, samples on columns. Also accepts SummarizedExperiment or SingleCellExperiment objects.
`matG`	Gene sets sparse matrix. Genes on rows, gene sets on columns. Also accepts BiocSet objects or GMT lists.
`rmax`	Rank threshold (see Ucell paper). Default rmax = 1500.
`assay`	Character: assay name for Bioconductor objects. Default "logcounts".
`min.genes`	Integer: minimum genes per gene set. Default 5.
`max.genes`	Integer: maximum genes per gene set. Default 500.

Details

Computing ssGSEA score requires to compute the ranks of the expression matrix and truncation of the ranks. We have wrapped this in a single convenience function.

We have extensively compared the results of `replaid.ucell` and from the original `UCell` R package and we showed near exacct results in the score, logFC and p-values.

Value

Matrix of single-sample UCell enrichment scores. Gene sets on rows, samples on columns.

Examples

```
# Create example expression matrix
set.seed(123)
X <- matrix(rnorm(500), nrow = 50, ncol = 10)
rownames(X) <- paste0("GENE", 1:50)
colnames(X) <- paste0("Sample", 1:10)

# Create example gene sets
gmt <- list(
  "Pathway1" = paste0("GENE", 1:15),
  "Pathway2" = paste0("GENE", 10:25)
)
matG <- gmt2mat(gmt)

# Compute UCell scores (default rmax = 1500)
scores <- replaid.ucell(X, matG)
print(scores[1:2, 1:5])

# Compute UCell scores with custom rmax
scores_custom <- replaid.ucell(X, matG, rmax = 1000)
print(scores_custom[1:2, 1:5])
```

---

sparse_colranks *Compute columm ranks for sparse matrix. Internally used by colranks()*

Description

Compute columm ranks for sparse matrix. Internally used by colranks()

Usage

```
sparse_colranks(X, signed = FALSE, ties.method = "average")
```

Arguments

`X`	Input matrix
`signed`	Logical: use or not signed ranks
`ties.method`	Character Choice of ties.method

Value

Sparse matrix of columnwise ranks with same dimensions as input.

`write.gmt` *Write GMT File*

Description

Write gene sets to GMT file (Gene Matrix Transposed). The GMT format is commonly used to store gene sets or gene annotations.

Usage

```
write.gmt(gmt, file, source = NA)
```

Arguments

`gmt`	A list of gene sets in GMT format: each gene set is represented as a vector of gene names.
`file`	The file path to write the GMT file.
`source`	A character vector specifying the source of each gene set. If not provided, the names of the gene sets are used as the source.

Value

Does not return anything.

Examples

```
# Create example GMT data
gmt <- list(
  "Pathway1" = c("GENE1", "GENE2", "GENE3"),
  "Pathway2" = c("GENE2", "GENE4", "GENE5"),
  "Pathway3" = c("GENE1", "GENE5", "GENE6")
)


# Write to GMT file (creates file in temp directory)
temp_file <- tempfile(fileext = ".gmt")
write.gmt(gmt, temp_file)

# Write with custom source information
temp_file2 <- tempfile(fileext = ".gmt")
write.gmt(gmt, temp_file2, source = c("DB1", "DB2", "DB3"))

# Clean up
unlink(c(temp_file, temp_file2))
```

Index