Package: BPCells (via r-universe)

July 26, 2024

Title Single Cell Counts Matrices to PCA

Version 0.2.0

Description > Efficient operations for single cell ATAC-seq fragments and RNA counts matrices. Interoperable with standard file formats, and introduces efficient bit-packed formats that allow large storage savings and increased read speeds.

License Apache-2.0 or MIT

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

URL https://bnprks.github.io/BPCells, https://github.com/bnprks/BPCells

LinkingTo Rcpp, RcppEigen

Imports methods, grDevices, magrittr, Matrix, Rcpp, rlang, vctrs, lifecycle, stringr, tibble, dplyr (>= 1.0.0), tidyr, readr, ggplot2 (>= 3.4.0), scales, patchwork, scattermore, ggrepel, RColorBrewer, hexbin

Suggests IRanges, GenomicRanges, matrixStats, igraph

Depends R (>= 3.5.0)

Repository https://bnprks.r-universe.dev

RemoteUrl https://github.com/bnprks/BPCells

RemoteRef HEAD

RemoteSha b78bc8321db802c6ebf29edc2296394f2e8587c6

Contents

`add_rows` *Broadcasting vector arithmetic*

Description

Convenience functions for adding or multiplying each row / column of a matrix by a number.

Usage

```
add_rows(mat, vec)

add_cols(mat, vec)

multiply_rows(mat, vec)

multiply_cols(mat, vec)
```

Arguments

`mat`	Matrix-like object
`vec`	Numeric vector

Value

Matrix-like object

---

all_matrix_inputs *Get/set inputs to a matrix transform*

Description

A matrix object can either be an input (i.e. a file on disk or a raw matrix in memory), or it can represent a delayed operation on one or more matrices. The `all_matrix_inputs()` getter and setter functions allow accessing the base-level input matrices as a list, and changing them. This is useful if you want to re-locate data on disk without losing your transformed BPCells matrix. (Note: experimental API; potentially subject to revisions).

Usage

```
all_matrix_inputs(x)

all_matrix_inputs(x) <- value
```

Arguments

`x`	IterableMatrix
`value`	List of IterableMatrix objects

Value

List of IterableMatrix objects. If a matrix `m` is itself an input object, then `all_matrix_inputs(m)` will return `list(m)`.

---

apply_by_row *Apply a function to summarize rows/cols*

Description

Apply a custom R function to each row/col of a BPCells matrix. This will run slower than the builtin C++-backed functions, but will keep most of the memory benefits from disk-backed operations.

Usage

```
apply_by_row(mat, fun, ...)

apply_by_col(mat, fun, ...)
```

Arguments

`mat`	IterableMatrix object
`fun`	`function(val, row, col)` that takes in a row/col of values and returns a summary output. Argument details: 1. `val` - Vector length (# non-zero values) with the value for each non-zero matrix entry 2. `row` - one-based row index (`apply_by_col`: vector length (# non-zero values), `apply_by_row`: single integer) 3. `col` - one-based col index (`apply_by_col`: single integer, `apply_by_row`: vector length (# non-zero values)) 4. `...` - Optional additional arguments (should not be named row, col, or val)
`...`	Optional additional arguments passed to `fun`

Details

These functions require row-major matrix storage for apply_by_row and col-major storage for apply_by_col, so matrices stored in the wrong order may neeed a re-ordered copy created using `transpose_storage_order()` first. This is required to be able to keep memory-usage low and allow calculating the result with a single streaming pass of the input matrix.

If vector/matrix outputs are desired instead of lists, calling `unlist(x)` or `do.call(cbind, x)` or `do.call(rbind, x)` can convert the list output.

Value

apply_by_row - A list of length `nrow(matrix)` with the results returned by `fun()` on each row

apply_by_col - A list of length `ncol(matrix)` with the results returned by `fun()` on each row

See Also

For an interface more similar to `base::apply`, see the BPCellsArray project. For calculating colMeans on a sparse single cell RNA matrix it is about 8x slower than `apply_by_col`, due to the `base::apply` interface not being sparsity-aware. (See pull request #104 for benchmarking.)

`binarize` *Convert matrix elements to zeros and ones*

Description

Binarize compares the matrix element values to the threshold value and sets the output elements to either zero or one. By default, element values greater than the threshold are set to one; otherwise, set to zero. When strict_inequality is set to FALSE, element values greater than or equal to the threshold are set to one. As an alternative, the <, <=, >, and >= operators are also supported.

Usage

```
binarize(mat, threshold = 0, strict_inequality = TRUE)
```

Arguments

mat	IterableMatrix
threshold	A numeric value that determines whether the elements of x are set to zero or one.
strict_inequality	A logical value determining whether the comparison to the threshold is >= (strict_inequality=FALSE) or > (strict_inequality=TRUE).

Value

binarized IterableMatrix object

---

call_peaks_tile *Call peaks from tiles*

Description

Calling peaks from a pre-set list of tiles can be much faster than using dedicated peak-calling software like `macs3`. The resulting peaks are less precise in terms of exact coordinates, but should be sufficient for most analyses.

Usage

```
call_peaks_tile(
  fragments,
  chromosome_sizes,
  cell_groups = rep.int("all", length(cellNames(fragments))),
  effective_genome_size = NULL,
  peak_width = 200,
  peak_tiling = 3,
  fdr_cutoff = 0.01,
  merge_peaks = c("all", "group", "none")
)
```

Arguments

fragments	IterableFragments object
chromosome_sizes	Chromosome start and end coordinates given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position See `read_ucsc_chrom_sizes()`.
cell_groups	Grouping vector with one entry per cell in fragments, e.g. cluster IDs

`effective_genome_size`
: (Optional) effective genome size for poisson background rate estimation. See deeptools for values for common genomes. Defaults to sum of chromosome sizes, which overestimates peak significance

`peak_width`
: Width of candidate peaks

`peak_tiling`
: Number of candidate peaks overlapping each base of genome. E.g. peak_width = 300 and peak_tiling = 3 results in candidate peaks of 300bp spaced 100bp apart

`fdr_cutoff`
: Adjusted p-value significance cutoff

`merge_peaks`
: How to merge significant peaks with `merge_peaks_iterative()`

  - `"all"` Merge the full set of peaks
  - `"group"` Merge peaks within each group
  - `"none"` Don't perform any merging

Details

Peak calling steps:

1. Estimate the genome-wide expected insertions per tile based on `peak_width`, `effective_genome_size`, and per-group read counts
2. Tile the genome with nonoverlapping tiles of size peak_width
3. For each tile and group, calculate p_value based on a Poisson model
4. Compute adjusted p-values using BH method and using the total number of tiles as the number of hypotheses tested.
5. Repeat steps 2-4 `peak_tiling` times, with evenly spaced offsets
6. If `merge_peaks` is "all" or "group": use `merge_peaks_iterative()` within each group to keep only the most significant of the overlapping candidate peaks
7. If `merge_peaks` is "all", perform a final round of `merge_peaks_iterative()`, prioritizing each peak by its within-group significance rank

Value

tibble with peak calls and the following columns:

- `chr`, `start`, `end`: genome coordinates
- `group`: group ID that this peak was identified in
- `p_val`, `q_val`: Poission p-value and BH-corrected p-value
- `enrichment`: Enrichment of counts in this peak compared to a genome-wide background

checksum *Calculate the MD5 checksum of an IterableMatrix*

Description

Calculate the MD5 checksum of an IterableMatrix and return the checksum in hexidecimal format.

Usage

```
checksum(matrix)
```

Arguments

matrix	IterableMatrix object

Details

`checksum()` converts the non-zero elements of the sparse input matrix to double precision, concatenates each element value with the element row and column index words, and uses these 16-byte blocks along with the matrix dimensions and row and column names to calculate the checksum. The checksum value depends on the storage order so column- and row-order matrices with the same element values give different checksum values. `checksum()` uses element and index values in little-endian CPU storage order. It converts to little-endian order on big-endian architecture although this has not been tested.

Value

MD5 checksum string in hexidecimal format.

Examples

```
library(Matrix)
library(BPCells)
m1 <- matrix(seq(1,12), nrow=3)
m2 <- as(m1, 'dgCMatrix')
m3 <- as(m2, 'IterableMatrix')
checksum(m3)
```

cluster_graph_leiden *Cluster an adjacency matrix*

Description

Cluster an adjacency matrix

Usage

```
cluster_graph_leiden(snn, resolution = 0.001, seed = 12531, ...)

cluster_graph_louvain(snn, resolution = 1, seed = 12531)

cluster_graph_seurat(snn, resolution = 0.8, ...)
```

Arguments

snn	Symmetric adjacency matrix (dgCMatrix) output from e.g. knn_to_snn_graph. Only the lower triangle is used
resolution	Resolution parameter. Higher values result in more clusters
seed	Random seed for clustering initialization
...	Additional arguments to underlying clustering function

Details

cluster_graph_leiden: Leiden graph clustering algorithm `igraph::cluster_leiden()`

cluster_graph_louvain: Louvain graph clustering algorithm `igraph::cluster_louvain()`

cluster_graph_seurat: Seurat's clustering algorithm `Seurat::FindClusters()`

Value

Factor vector containing the cluster assignment for each cell.

cluster_membership_matrix *Convert grouping vector to sparse matrix*

Description

Converts a vector of membership IDs into a sparse matrix

Usage

```
cluster_membership_matrix(groups, group_order = NULL)
```

Arguments

groups	Vector with one entry per cell, specifying the cell's group
group_order	Optional vector listing ordering of groups

Value

cell x group matrix where an entry is 1 when a cell is in a given group

---

collect_features *Collect features for plotting*

Description

Helper function for data on features to plot from a diverse set of data sources.

Usage

```
collect_features(
  source,
  features = NULL,
  gene_mapping = human_gene_mapping,
  n = 1
)
```

Arguments

source	Matrix or data frame to pull features from, or a vector of feature values for a single feature. For a matrix, the features must be rows.
features	Character vector of features names to plot if source is not a vector.
gene_mapping	An optional vector for gene name matching with match_gene_symbol(). Ignored if source is a data frame.
n	Internal-use parameter marking the number of nested calls. This is used for finding the name of the "source" input variable from the caller's perspective

Details

If source is a data.frame, features will be drawn from the columns. If source is a matrix object (IterableMatrix, dgCMatrix, or matrix), features will be drawn from rows.

Value

Data frame with one column for each feature requested

convert_matrix_type *Convert the type of a matrix*

Description

Convert the type of a matrix

Usage

```
convert_matrix_type(matrix, type = c("uint32_t", "double", "float"))
```

Arguments

matrix	IterableMatrix object input
type	One of uint32_t (unsigned 32-bit integer), float (32-bit real number), or double (64-bit real number)

Value

IterableMatrix object

convert_to_fragments *Convert between BPCells fragments and R objects.*

Description

BPCells fragments can be interconverted with GRanges and data.frame R objects. The main conversion method is R's builtin `as()` function, though the `convert_to_fragments()` helper is also available. For all R objects except GRanges, BPCells assumes a 0-based, end-exclusive coordinate system. (See genomic-ranges-like reference for details)

Usage

```
# Convert from R to BPCells
convert_to_fragments(x, zero_based_coords = !is(x, "GRanges"))
as(x, "IterableFragments")

# Convert from BPCells to R
as.data.frame(bpcells_fragments)
as(bpcells_fragments, "data.frame")
as(bpcells_fragments, "GRanges")
```

Arguments

x	Fragment coordinates given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position • `cell_id`: cell barcodes or unique identifiers as string or factor
zero_based_coords	Whether to convert the ranges from a 1-based end-inclusive coordinate system to a 0-based end-exclusive coordinate system. Defaults to true for GRanges and false for other formats (see this archived UCSC blogpost)

Value

convert_to_fragments(): IterableFragments object

---

discrete_palette *Color palettes*

Description

These color palettes are derived from the ArchR color palettes, and provide large sets of distinguishable colors

Usage

```
discrete_palette(name, n = 1)

continuous_palette(name)
```

Arguments

name	Name of the color palette. Valid discrete palettes are: `stallion`, `calm`, `kelly`, `bear`, `ironMan`, `circus`, `paired`, `grove`, `summerNight`, and `captain`. Valid continuous palettes are `bluePurpleDark`
n	Minimum number of colors needed

Details

If the requested number of colors is too large, a new palette will be constructed via interpolation from the requested palette

Value

Character vector of hex color codes

`extend_ranges` *Extend genome ranges in a strand-aware fashion.*

Description

Extend genome ranges in a strand-aware fashion.

Usage

```
extend_ranges(
  ranges,
  upstream = 0,
  downstream = 0,
  metadata_cols = c("strand"),
  chromosome_sizes = NULL,
  zero_based_coords = !is(ranges, "GRanges")
)
```

Arguments

`ranges`	Genomic regions given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position
`upstream`	Number of bases to extend each range upstream (negative to shrink width)
`downstream`	Number of bases to extend each range downstream (negative to shrink width)
`metadata_cols`	Optional list of metadata columns to require & extract
`chromosome_sizes`	(optional) Size of chromosomes as a genomic-ranges object
`zero_based_coords`	If true, coordinates start and 0 and the end coordinate is not included in the range. If false, coordinates start at 1 and the end coordinate is included in the range

Details

Note that ranges will be blocked from extending past the beginning of the chromosome (base 0), and if `chromosome_sizes` is given then they will also be blocked from extending past the end of the chromosome

`footprint` *Get footprints around a set of genomic coordinates*

Description

Get footprints around a set of genomic coordinates

Usage

```
footprint(
  fragments,
  ranges,
  zero_based_coords = !is(ranges, "GRanges"),
  cell_groups = rlang::rep_along(cellNames(fragments), "all"),
  cell_weights = rlang::rep_along(cell_groups, 1),
  flank = 125L,
  normalization_width = flank%/%10L
)
```

Arguments

`fragments`	IterableFragments object
`ranges`	Footprint centers given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position • `strand`: +/- or TRUE/FALSE for positive or negative strand "+" strand ranges will footprint around the start coordinate, and "-" strand ranges around the end coordinate.
`zero_based_coords`	If true, coordinates start and 0 and the end coordinate is not included in the range. If false, coordinates start at 1 and the end coordinate is included in the range
`cell_groups`	Character or factor assigning a group to each cell, in order of `cellNames(fragments)`
`cell_weights`	Numeric vector assigning weight factors (e.g. inverse of total reads) to each cell, in order of `cellNames(fragments)`
`flank`	Number of flanking basepairs to include on either side of the motif
`normalization_width`	Number of basepairs at the upstream + downstream extremes to use for calculating enrichment

Value

`tibble::tibble()` with columns `group`, `position`, and `count`, `enrichment`

fragments_identical *Check if two fragments objects are identical*

Description

Check if two fragments objects are identical

Usage

```
fragments_identical(fragments1, fragments2)
```

Arguments

fragments1	First IterableFragments to compare
fragments2	Second IterableFragments to compare

Value

boolean for whether the fragments objects are identical

gene_region *Find gene region*

Description

Conveniently look up the region of a gene by gene symbol. The value returned by this function can be used as the `region` argument for trackplot functions such as `trackplot_coverage()` or `trackplot_gene()`

Usage

```
gene_region(
  genes,
  gene_symbol,
  extend_bp = c(10000, 10000),
  gene_mapping = human_gene_mapping
)
```

Arguments

genes	Transcipt features given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position • `strand`: +/- or TRUE/FALSE for positive or negative strand • `gene_name`: Symbol or gene ID

gene_symbol	Name of gene symbol or ID
extend_bp	Bases to extend region upstream and downstream of gene. If length 1, extension is symmetric. If length 2, provide upstream extension then downstream extension as positive distances.
gene_mapping	Named vector where names are gene symbols or IDs and values are canonical gene symbols

Value

List of chr, start, end positions for use with trackplot functions.

`gene_score_tiles_archr` *Calculate gene-tile distances for ArchR gene activities*

Description

ArchR-style gene activity scores are based on a weighted sum of each tile according to the signed distance from the tile to a gene body. This function calculates the signed distances according to ArchR's default parameters.

Usage

```
gene_score_tiles_archr(
  genes,
  chromosome_sizes = NULL,
  tile_width = 500,
  addArchRBug = FALSE
)
```

Arguments

genes	Gene coordinates given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position • `strand`: +/- or TRUE/FALSE for positive or negative strand
chromosome_sizes	(optional) Size of chromosomes as a genomic-ranges object
tile_width	Size of tiles to consider
addArchRBug	Replicate ArchR bug in handling nested genes

Details

ArchR's tile distance algorithm works as follows

1. Genes are extended 5kb upstream
2. Genes are linked to any tiles 1kb-100kb upstream + downstream, but tiles beyond a neighboring gene are not considered

Value

Tibble with one range per tile, with additional metadata columns gene_idx (row index of the gene this tile corresponds to) and distance.

Distance is a signed distance calculated such that if the tile has a smaller start coordinate than the gene and the gene is on the + strand, distance will be negative. The distance of adjacent but non-overlapping regions is 1bp, counting up from there.

`gene_score_weights_archr` *Calculate GeneActivityScores*

Description

Gene activity scores can be calculated as a distance-weighted sum of per-tile accessibility. The tile weights for each gene can be represented as a sparse matrix of dimension genes x tiles. If we multiply this weight matrix by a corresponding tile matrix (tiles x cells), then we can get a gene activity score matrix of genes x cells. `gene_score_weights_archr()` calculates the weight matrix (best if you have a pre-computed tile matrix), while `gene_score_archr()` provides a easy-to-use wrapper.

Usage

```
gene_score_weights_archr(
  genes,
  chromosome_sizes,
  blacklist = NULL,
  tile_width = 500,
  gene_name_column = "gene_id",
  addArchRBug = FALSE
)

gene_score_archr(
  fragments,
  genes,
  chromosome_sizes,
  blacklist = NULL,
  tile_width = 500,
  gene_name_column = "gene_id",
  addArchRBug = FALSE,
  tile_max_count = 4,
  scale_factor = 10000,
  tile_matrix_path = tempfile(pattern = "gene_score_tile_mat")
)
```

Arguments

genes	Gene coordinates given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position • `strand`: +/- or TRUE/FALSE for positive or negative strand
chromosome_sizes	Chromosome start and end coordinates given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position See `read_ucsc_chrom_sizes()`.
blacklist	Regions to exclude from calculations, given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position
tile_width	Size of tiles to consider
gene_name_column	If not NULL, a column name of `genes` to use as row names
addArchRBug	Replicate ArchR bug in handling nested genes
fragments	Input fragments object
tile_max_count	Maximum value in the tile counts matrix. If not null, tile counts higher than this will be clipped to `tile_max_count`. Equivalent to `ceiling` argument of `ArchR::addGeneScoreMatrix()`
scale_factor	If not null, counts for each cell will be scaled to sum to `scale_factor`. Equivalent to `scaleTo` argument of `ArchR::addGeneScoreMatrix()`
tile_matrix_path	Path of a directory where the intermediate tile matrix will be saved

Details

gene_score_weights_archr:

Given a set of tile coordinates and distances returned by `gene_score_tiles_archr()`, calculate a weight matrix of dimensions genes x tiles. This matrix can be multiplied with a tile matrix to obtain ArchR-compatible gene activity scores.

Value

gene_score_weights_archr

Weight matrix of dimension genes x tiles

gene_score_archr

Gene score matrix of dimension genes x cells.

genomic-ranges-like *Genomic range formats*

Description

BPCells accepts a flexible set of genomic ranges-like objects as input, either GRanges, data.frame, lists, or character vectors. These objects must specify chromosome, start, and end coordinates along with optional metadata about each range. With the exception of `GenomicRanges::GRanges` objects, BPCells assumes all objects use a zero-based, end-exclusive coordinate system (see below for details).

Valid Range-like objects:

BPCells can interpret the following types as ranges:

- `list()`, `data.frame()`, with columns:
  - `chr`: Character or factor of chromosome names
  - `start`: Start coordinates (0-based)
  - `end`: End coordinates (exclusive)
  - (optional) `strand`: `"+"/"-"` or `TRUE/FALSE` for pos/neg strand
  - (optional) Additional metadata as named `list` entries or `data.frame` columns
- `GenomicRanges::GRanges`
  - `start(x)` is interpreted as a 1-based start coordinate
  - `end(x)` is interpreted as an inclusive end coordinate
  - `strand(x)`: `"*"` entries are interpeted as postive strand
  - (optional) `mcols(x)` holds additional metadata
- `character`
  - Given in format `"chr1:1000-2000"` or `"chr1:1,000-2,000"`
  - Uses 0-based, end-exclusive coordinate system
  - Cannot be used for ranges where additional metadata is required

Range coordinate systems:

There are two main conventions for the coordinate systems:

One-based, end-inclusive ranges

- The first base of a chromosome is numbered 1
- The last base in a range is equal to the end coordinate
- e.g. 1-5 describes the first 5 bases of the chromosome
- Used in formats such as SAM, GTF
- In BPCells, used when reading or writing `GenomicRanges::GRanges` objects

Zero-based, end-exclusive ranges

- The first base of a chromosome is numbered 0
- The last base in a range is one less than the end coordinate
- e.g. 0-5 describes the first 5 bases of the chromosome
- Used in formats such as BAM, BED
- In BPCells, used for all other range objects

human_gene_mapping *Gene Symbol Mapping data*

Description

Mapping of the canonical gene symbols corresponding to each unambiguous alias, previous symbol, ensembl ID, or entrez ID.

Usage

```
human_gene_mapping

mouse_gene_mapping
```

Format

human_gene_mapping

A named character vector. Names are aliases or IDs and values are the corresponding canonical gene symbol

mouse_gene_mapping

A named character vector. Names are aliases or IDs and values are the corresponding canonical gene symbol

Details

See the source code in `data-raw/human_gene_mapping.R` and `data-raw/mouse_gene_mapping.R` for exactly how these mappings were made.

Source

human_gene_mapping

http://ftp.ebi.ac.uk/pub/databases/genenames/hgnc/tsv/non_alt_loci_set.txt

mouse_gene_mapping

http://www.informatics.jax.org/downloads/reports/MGI_EntrezGene.rpt http://www.informatics.jax.org/downloads/reports/MRK_ENSEMBL.rpt

`import_matrix_market` *Import MatrixMarket files*

Description

Read a sparse matrix from a MatrixMarket file. This is a text-based format used by 10x, Parse, and others to store sparse matrices. Format details on the NIST website.

Usage

```
import_matrix_market(
  mtx_path,
  outdir = tempfile("matrix_market"),
  row_names = NULL,
  col_names = NULL,
  row_major = FALSE,
  tmpdir = tempdir(),
  load_bytes = 4194304L,
  sort_bytes = 1073741824L
)

import_matrix_market_10x(
  mtx_dir,
  outdir = tempfile("matrix_market"),
  feature_type = NULL,
  row_major = FALSE,
  tmpdir = tempdir(),
  load_bytes = 4194304L,
  sort_bytes = 1073741824L
)
```

Arguments

`mtx_path`	Path of mtx or mtx.gz file
`outdir`	Directory to store the output
`row_names`	Character vector of row names
`col_names`	Character vector of col names
`row_major`	If true, store the matrix in row-major orientation
`tmpdir`	Temporary directory to use for intermediate storage
`load_bytes`	The minimum contiguous load size during the merge sort passes
`sort_bytes`	The amount of memory to allocate for re-sorting chunks of entries
`mtx_dir`	Directory holding matrix.mtx.gz, barcodes.tsv.gz, and features.tsv.gz
`feature_type`	String or vector of feature types to include. (cellranger 3.0 and newer)

Details

Import MatrixMarket mtx files to the BPCells format. This implementation ensures fixed memory usage even for very large inputs by doing on-disk sorts. It will be much slower than hdf5 inputs, so only use MatrixMarket format when absolutely necessary.

As a rough speed estimate, importing the 17GB Parse 1M PBMC `DGE_1M_PBMC.mtx` file takes about 4 minutes and 1.3GB of RAM, producing a compressed output matrix of 1.5GB. `mtx.gz` files will be slower to import due to gzip decompression.

When importing from 10x mtx files, the row and column names can be read automatically using the `import_matrix_market_10x()` convenience function.

Value

MatrixDir object with the imported matrix

`IterableFragments-methods`

IterableFragments methods

Description

Methods for IterableFragments objects

Usage

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

cellNames(x)

cellNames(x, ...) <- value

chrNames(x)

chrNames(x, ...) <- value
```

Arguments

`object`	IterableFragments object
`x`	an IterableFragments object
`value`	Character vector of new names

Details

- `cellNames<-` It is only possible to replace names, not add new names.
- `chrNames<-` It is only possible to replace names, not add new names.

Value

- `cellNames()` Character vector of cell names, or NULL if none are known
- `chrNames()`: Character vector of chromosome names, or NULL if none are known

Functions

- `show(IterableFragments)`: Print IterableFragments
- `cellNames()`: Get cell names
- `cellNames(x, ...) <- value`: Set cell names
- `chrNames()`: Set chromosome names
- `chrNames(x, ...) <- value`: Set chromosome names

IterableMatrix-methods

IterableMatrix methods

Description

Generic methods and built-in functions for IterableMatrix objects

Usage

```
matrix_type(x)

storage_order(x)

## S4 method for signature 'IterableMatrix'
show(object)

## S4 method for signature 'IterableMatrix'
t(x)

## S4 method for signature 'IterableMatrix,matrix'
x %*% y

## S4 method for signature 'IterableMatrix'
rowSums(x)

## S4 method for signature 'IterableMatrix'
colSums(x)

## S4 method for signature 'IterableMatrix'
rowMeans(x)

## S4 method for signature 'IterableMatrix'
```

```
colMeans(x)

colVars(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
  ...,
  useNames = TRUE
)

rowVars(
  x,
  rows = NULL,
  cols = NULL,
  na.rm = FALSE,
  center = NULL,
  ...,
  useNames = TRUE
)

rowMaxs(x, rows = NULL, cols = NULL, na.rm = FALSE, ..., useNames = TRUE)

colMaxs(x, rows = NULL, cols = NULL, na.rm = FALSE, ..., useNames = TRUE)

## S4 method for signature 'IterableMatrix'
log1p(x)

log1p_slow(x)

## S4 method for signature 'IterableMatrix'
expm1(x)

expm1_slow(x)

## S4 method for signature 'IterableMatrix,numeric'
e1 ^ e2

## S4 method for signature 'numeric,IterableMatrix'
e1 < e2

## S4 method for signature 'IterableMatrix,numeric'
e1 > e2

## S4 method for signature 'numeric,IterableMatrix'
e1 <= e2
```

```
## S4 method for signature 'IterableMatrix,numeric'
e1 >= e2

## S4 method for signature 'IterableMatrix'
round(x, digits = 0)

## S4 method for signature 'IterableMatrix,numeric'
e1 * e2

## S4 method for signature 'IterableMatrix,numeric'
e1 + e2

## S4 method for signature 'IterableMatrix,numeric'
e1 / e2

## S4 method for signature 'IterableMatrix,numeric'
e1 - e2
```

Arguments

x	IterableMatrix/dgCMatrix object
object	IterableMatrix object
y	matrix

Value

- `t()` Transposed object
- `x %*% y`: dense matrix result
- `rowSums()`: vector of row sums
- `colSums()`: vector of col sums
- `rowMeans()`: vector of row means
- `colMeans()`: vector of col means
- `colVars()`: vector of col variance
- `rowVars()`: vector of row variance
- `rowMaxs()`: vector of maxes for every row
- `colMaxs()`: vector of column maxes

Functions

- `matrix_type()`: Get the matrix data type (mat_uint32_t, mat_float, or mat_double for now)
- `storage_order()`: Get the matrix storage order ("row" or "col")
- `show(IterableMatrix)`: Display an IterableMatrix
- `t(IterableMatrix)`: Transpose an IterableMatrix
- `x %*% y`: Multiply by a dense matrix
- `rowSums(IterableMatrix)`: Calculate rowSums
- `colSums(IterableMatrix)`: Calculate colSums
- `rowMeans(IterableMatrix)`: Calculate rowMeans
- `colMeans(IterableMatrix)`: Calculate colMeans
- `colVars()`: Calculate colVars (replacement for `matrixStats::colVars()`)
- `rowVars()`: Calculate rowVars (replacement for `matrixStats::rowVars()`)
- `rowMaxs()`: Calculate rowMaxs (replacement for `matrixStats::rowMaxs()`)
- `colMaxs()`: Calculate colMax (replacement for `matrixStats::colMax()`)
- `log1p(IterableMatrix)`: Calculate log(x + 1)
- `log1p_slow()`: Calculate log(x + 1) (non-SIMD version)
- `expm1(IterableMatrix)`: Calculate exp(x) - 1
- `expm1_slow()`: Calculate exp(x) - 1 (non-SIMD version)
- `e1^e2`: Calculate x^y (elementwise)
- `e1 < e2`: Binarize matrix according to numeric < matrix comparison
- `e1 > e2`: Binarize matrix according to matrix > numeric comparison
- `e1 <= e2`: Binarize matrix according to numeric <= matrix comparison
- `e1 >= e2`: Binarize matrix according to matrix >= numeric comparison
- `round(IterableMatrix)`: round to nearest integer (digits must be 0)
- `e1 * e2`: Multiply by a constant, or multiply rows by a vector length nrow(mat)
- `e1 + e2`: Add a constant, or row-wise addition with a vector length nrow(mat)
- `e1 / e2`: Divide by a constant, or divide rows by a vector length nrow(mat)
- `e1 - e2`: Subtract a constant, or row-wise subtraction with a vector length nrow(mat)

knn_hnsw *Get a knn matrix from reduced dimensions*

Description

Search for approximate nearest neighbors between cells in the reduced dimensions (e.g. PCA), and return the k nearest neighbors (knn) for each cell. Optionally, we can find neighbors between two separate sets of cells by utilizing both data and query.

Usage

```
knn_hnsw(
  data,
  query = NULL,
  k = 10,
  metric = c("euclidean", "cosine"),
  verbose = TRUE,
  threads = 1,
  ef = 100
)

knn_annoy(
  data,
  query = data,
  k = 10,
  metric = c("euclidean", "cosine", "manhattan", "hamming"),
  n_trees = 50,
  search_k = -1
)
```

Arguments

data	cell x dims matrix for reference dataset
query	cell x dims matrix for query dataset (optional)
k	number of neighbors to calculate
metric	distance metric to use
verbose	whether to print progress information during search
threads	Number of threads to use. Note that result is non-deterministic if threads > 1
ef	ef parameter for RccppHNSW::hnsw_search. Increase for slower search but improved accuracy
n_trees	Number of trees during index build time. More trees gives higher accuracy
search_k	Number of nodes to inspect during the query, or -1 for default value. Higher number gives higher accuracy

Details

knn_hnsw: Use RcppHNSW as knn engine

knn_annoy: Use RcppAnnoy as knn engine

Value

List of 2 matrices – idx for cell x K neighbor indices, dist for cell x K neighbor distances. If no query is given, nearest neighbors are found mapping the data matrix to itself, prohibiting self-neighbors

---

knn_to_graph *K Nearest Neighbor (KNN) Graph*

Description

Convert a KNN object (e.g. returned by `knn_hnsw()` or `knn_annoy()`) into a graph. The graph is represented as a sparse adjacency matrix.

Usage

```
knn_to_graph(knn, use_weights = FALSE, self_loops = TRUE)

knn_to_snn_graph(
  knn,
  min_val = 1/15,
  self_loops = FALSE,
  return_type = c("matrix", "list")
)

knn_to_geodesic_graph(knn, return_type = c("matrix", "list"), threads = 0L)
```

Arguments

`knn`	List of 2 matrices – idx for cell x K neighbor indices, dist for cell x K neighbor distances
`use_weights`	boolean for whether to replace all distance weights with 1
`self_loops`	Whether to allow self-loops in the output graph
`min_val`	minimum jaccard index between neighbors. Values below this will round to 0
`return_type`	Whether to return a sparse adjacency matrix or an edge list
`threads`	Number of threads to use during calculations

Details

knn_to_graph Create a knn graph

knn_to_snn_graph Convert a knn object into a shared nearest neighbors adjacency matrix. This follows the algorithm that Seurat uses to compute SNN graphs

knn_to_geodesic_graph Convert a knn object into an undirected weighted graph, using the same geodesic distance estimation method as the UMAP package. This matches the output of `umap._umap.fuzzy_simplicial_se` from the `umap-learn` python package, used by default in `scanpy.pp.neighbors`. Because this only re-weights and symmetrizes the KNN graph, it will usually use less memory and return a sparser graph than `knn_to_snn_graph` which computes 2nd-order neighbors. Note: when cells don't have themselves listed as the nearest neighbor, results may differ slightly from `umap._umap.fuzzy_simplicial_set`, which assumes self is always successfully found in the approximate nearest neighbor search.

Value

knn_to_graph Sparse matrix (dgCMatrix) where `mat[i,j]` = distance from cell `i` to cell `j`, or 0 if cell `j` is not in the K nearest neighbors of `i`

knn_to_snn_graph

- `return_type == "matrix"`: Sparse matrix (dgCMatrix) where `mat[i,j]` = jaccard index of the overlap in nearest neigbors between cell `i` and cell `j`, or 0 if the jaccard index is < `min_val`. Only the lower triangle is filled in, which is compatible with the BPCells clustering methods
- `return_type == "list"`: List of 3 equal-length vectors `i`, `j`, and `weight`, along with an integer `dim`. These correspond to the rows, cols, and values of non-zero entries in the lower triangle adjacency matrix. `dim` is the total number of vertices (cells) in the graph

knn_to_geodesic_graph

- `return_type == "matrix"`: Sparse matrix (dgCMatrix) where `mat[i,j]` = normalized similarity between cell `i` and cell `j`. Only the lower triangle is filled in, which is compatible with the BPCells clustering methods
- `return_type == "list"`: List of 3 equal-length vectors `i`, `j`, and `weight`, along with an integer `dim`. These correspond to the rows, cols, and values of non-zero entries in the lower triangle adjacency matrix. `dim` is the total number of vertices (cells) in the graph

---

marker_features *Test for marker features*

Description

Given a features x cells matrix, perform one-vs-all differential tests to find markers.

Usage

```
marker_features(mat, groups, method = "wilcoxon")
```

Arguments

`mat`	IterableMatrix object of dimensions features x cells
`groups`	Character/factor vector of cell groups/clusters. Length #cells
`method`	Test method to use. Current options are: • `wilcoxon`: Wilconxon rank-sum test a.k.a Mann-Whitney U test

Details

Tips for using the values from this function:

- Use `dplyr::mutate()` to add columns for e.g. adjusted p-value and log fold change.
- Use `dplyr::filter()` to get only differential genes above some given threshold
- To get adjusted p-values, use R `p.adjust()`, recommended method is "BH"
- To get log2 fold change: if your input matrix was already log-transformed, calculate `(foreground_mean - background_mean)/log(2)`. If your input matrix was not log-transformed, calculate `log2(forground_mean/backgr`

Value

tibble with the following columns:

- **foreground**: Group ID used for the foreground
- **background**: Group ID used for the background (or NA if comparing to rest of cells)
- **feature**: ID of the feature
- **p_val_raw**: Unadjusted p-value for differential test
- **foreground_mean**: Average value in the foreground group
- **background_mean**: Average value in the background group

`match_gene_symbol` *Gene symbol matching*

Description

Correct alias gene symbols, Ensembl IDs, and Entrez IDs to canonical gene symbols. This is useful for matching gene names between different datasets which might not always use the same gene naming conventions.

Usage

```
match_gene_symbol(query, subject, gene_mapping = human_gene_mapping)

canonical_gene_symbol(query, gene_mapping = human_gene_mapping)
```

Arguments

query	Character vector of gene symbols or IDs
subject	Vector of gene symbols or IDs to index into
gene_mapping	Named vector where names are gene symbols or IDs and values are canonical gene symbols

Value

match_gene_symbol

Integer vector of indices v such that subject[v] corresponds to the gene symbols in query

canonical_gene_symbol

Character vector of canonical gene symbols for each symbol in query

---

matrix_R_conversion *Convert between BPCells matrix and R objects.*

Description

BPCells matrices can be interconverted with Matrix package dgCMatrix sparse matrices, as well as base R dense matrices (though this may result in high memory usage for large matrices)

Usage

```
# Convert to R from BPCells
as(bpcells_mat, "dgCMatrix") # Sparse matrix conversion
as.matrix(bpcells_mat) # Dense matrix conversion

# Convert to BPCells from R
as(dgc_mat, "IterableMatrix")
```

---

matrix_stats *Calculate matrix stats*

Description

Calculate matrix stats

Usage

```
matrix_stats(
  matrix,
  row_stats = c("none", "nonzero", "mean", "variance"),
  col_stats = c("none", "nonzero", "mean", "variance"),
  threads = 0L
)
```

Arguments

matrix	Input matrix object
row_stats	Which row statistics to compute
col_stats	Which col statistics to compute
threads	Number of threads to use during execution

Details

The statistics will be calculated in a single pass over the matrix, so this method is desirable to use for efficiency purposes compared to the more standard rowMeans or colMeans if multiple statistics are needed. The stats are ordered by complexity: nonzero, mean, then variance. All less complex stats are calculated in the process of calculating a more complicated stat. So to calculate mean and variance simultaneously, just ask for variance, which will compute mean and nonzero counts as a side-effect

Value

List of row_stats: matrix of n_stats x n_rows, col_stats: matrix of n_stats x n_cols

---

merge_cells *Merge cells into pseudobulks*

Description

Peak and tile matrix calculations can be sped up by reducing the number of cells. For cases where the outputs are going to be added together afterwards, this can provide a performance improvement

Usage

```
merge_cells(fragments, cell_groups)
```

Arguments

fragments	Input fragments object
cell_groups	Character or factor vector providing a group for each cell. Ordering is the same as `cellNames(fragments)`

`merge_peaks_iterative` *Merge peaks*

Description

Merge peaks according to ArchR's iterative merging algorithm. More details on the ArchR website

Usage

```
merge_peaks_iterative(peaks)
```

Arguments

`peaks`	Peaks given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position Must be ordered by priority and have columns chr, start, end.

Details

Properties of merged peaks:

- No peaks in the merged set overlap
- Peaks are prioritized according to their order in the original input
- The output peaks are a subset of the input peaks, with no peak boundaries changed

Value

`tibble::tibble()` with a nonoverlapping subset of the rows in peaks. All metadata columns are preserved

`min_scalar` *Elementwise minimum*

Description

min_scalar: Take minumum with a global constant

min_by_row: Take the minimum with a per-row constant

min_by_col: Take the minimum with a per-col constant

Usage

```
min_scalar(mat, val)

min_by_row(mat, vals)

min_by_col(mat, vals)
```

Arguments

mat	IterableMatrix
val	Single positive numeric value

Details

Take the minimum value of a matrix with a per-row, per-col, or global constant. This constant must be >0 to preserve sparsity of the matrix. This has the effect of capping the maximum value in the matrix.

Value

IterableMatrix

---

normalize_ranges *Normalize an object representing genomic ranges*

Description

Normalize an object representing genomic ranges

Usage

```
normalize_ranges(
  ranges,
  metadata_cols = character(0),
  zero_based_coords = !is(ranges, "GRanges"),
  n = 1
)
```

Arguments

ranges	Genomic regions given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position
metadata_cols	Optional list of metadata columns to require & extract
zero_based_coords	If true, coordinates start and 0 and the end coordinate is not included in the range. If false, coordinates start at 1 and the end coordinate is included in the range

Value

data frame with zero-based coordinates, and elements chr (factor), start (int), and end (int). If `ranges` does not have chr level information, chr levels are the sorted unique values of chr.

If strand is in metadata_cols, then the output strand element will be TRUE for positive strand, and FALSE for negative strand. (Converted from a character vector of "+"/"-" if necessary)

`nucleosome_counts` *Count fragments by nucleosomal size*

Description

Count fragments by nucleosomal size

Usage

```
nucleosome_counts(fragments, nucleosome_width = 147)
```

Arguments

`fragments`	Fragments object
`nucleosome_width`	Integer cutoff to use as nucleosome width

Details

Shorter than `nucleosome_width` is `subNucleosomal`, `nucleosome_width` to `2*nucleosome_width-1` is `monoNucleosomal`, and anything longer is `multiNucleosomal`. The sum of all fragments is given as `nFrags`

Value

List with names `subNucleosomal`, `monoNucleosomal`, `multiNucleosomal`, and `nFrags`, containing the count vectors of fragments in each class per cell.

`open_fragments_10x` *Read/write a 10x fragments file*

Description

10x fragment files come in a bed-like format, with columns chr, start, end, cell_id, and pcr_duplicates. Unlike a standard bed format, the format from cellranger has an *inclusive* end-coordinate, meaning the end coordinate itself is what should be counted as the tagmentation site, rather than offset by 1.

Usage

```
open_fragments_10x(path, comment = "#", end_inclusive = TRUE)

write_fragments_10x(
  fragments,
  path,
  end_inclusive = TRUE,
  append_5th_column = FALSE
)
```

Arguments

path	File path (e.g. fragments.tsv or fragments.tsv.gz)
comment	Skip lines at beginning of file which start with comment string
end_inclusive	Whether the end coordinate of the bed is inclusive – i.e. there was an insertion at the end coordinate rather than the base before the end coordinate. This is the 10x default, though it's not quite standard for the bed file format.
fragments	Input fragments object
append_5th_column	Whether to include 5th column of all 0 for compatibility with 10x fragment file outputs (defaults to 4 columns chr,start,end,cell)

Details

open_fragments_10x

No disk operations will take place until the fragments are used in a function

write_fragments_10x

Fragments will be written to disk immediately, then returned in a readable object.

Value

10x fragments file object

---

open_matrix_10x_hdf5 *Read/write a 10x feature matrix*

Description

Read/write a 10x feature matrix

Usage

```
open_matrix_10x_hdf5(path, feature_type = NULL, buffer_size = 16384L)

write_matrix_10x_hdf5(
  mat,
  path,
  barcodes = colnames(mat),
  feature_ids = rownames(mat),
  feature_names = rownames(mat),
  feature_types = "Gene Expression",
  feature_metadata = list(),
  buffer_size = 16384L,
  chunk_size = 1024L,
  gzip_level = 0L,
  type = c("uint32_t", "double", "float", "auto")
)
```

Arguments

path	Path to the hdf5 file on disk
feature_type	Optional selection of feature types to include in output matrix. For multiome data, the options are "Gene Expression" and "Peaks". This option is only compatible with files from cellranger 3.0 and newer.
buffer_size	For performance tuning only. The number of items to be buffered in memory before calling writes to disk.
mat	IterableMatrix
barcodes	Vector of names for the cells
feature_ids	Vector of IDs for the features
feature_names	Vector of names for the features
feature_types	String or vector of feature types
feature_metadata	Named list of additional metadata vectors to store for each feature
chunk_size	For performance tuning only. The chunk size used for the HDF5 array storage.
gzip_level	Gzip compression level. Default is 0 (no compression)
type	Data type of the output matrix. Default is `uint32_t` to match a matrix of 10x UMI counts. Non-integer data types include `float` and `double`. If `auto`, will use the data type of `mat`.

Details

The 10x format makes use of gzip compression for the matrix data, which can slow down read performance. Consider writing into another format if the read performance is important to you.

Input matrices must be in column-major storage order, and if the rownames and colnames are not set, names must be provided for the relevant metadata parameters. Some of the metadata parameters are not read by default in BPCells, but it is possible to export them for use with other tools.

Value

BPCells matrix object

`open_matrix_anndata_hdf5` *Read/write AnnData matrix*

Description

Read or write a sparse matrix from an anndata hdf5 file. These functions will automatically transpose matrices when converting to/from the AnnData format. This is because the AnnData convention stores cells as rows, whereas the R convention stores cells as columns. If this behavior is undesired, call `t()` manually on the matrix inputs and outputs of these functions.

Usage

```
open_matrix_anndata_hdf5(path, group = "X", buffer_size = 16384L)

write_matrix_anndata_hdf5(
  mat,
  path,
  group = "X",
  buffer_size = 16384L,
  chunk_size = 1024L,
  gzip_level = 0L
)
```

Arguments

`path`	Path to the hdf5 file on disk
`group`	The group within the hdf5 file to write the data to. If writing to an existing hdf5 file this group must not already be in use
`buffer_size`	For performance tuning only. The number of items to be buffered in memory before calling writes to disk.
`chunk_size`	For performance tuning only. The chunk size used for the HDF5 array storage.
`gzip_level`	Gzip compression level. Default is 0 (no compression)

Value

AnnDataMatrixH5 object, with cells as the columns.

order_ranges *Get end-sorted ordering for genome ranges*

Description

Use this function to order regioins prior to calling `peak_matrix()` or `tile_matrix()`.

Usage

```
order_ranges(ranges, chr_levels, sort_by_end = TRUE)
```

Arguments

`ranges`	Genomic regions given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position
`chr_levels`	Ordering of chromosome names
`sort_by_end`	If TRUE (defualt), sort by (chr, end, start). Else sort by (chr, start, end)

Value

Numeric vector analagous to the `order` function. Provides an index selection that will reorder the input ranges to be sorted by chr, end, start

peak_matrix *Calculate ranges x cells overlap matrix*

Description

Calculate ranges x cells overlap matrix

Usage

```
peak_matrix(
  fragments,
  ranges,
  mode = c("insertions", "fragments", "overlaps"),
  zero_based_coords = !is(ranges, "GRanges"),
  explicit_peak_names = TRUE
)
```

Arguments

`fragments`	Input fragments object. Must have cell names and chromosome names defined
`ranges`	Peaks/ranges to overlap, given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position
`mode`	Mode for counting peak overlaps. (See "value" section for more details)
`zero_based_coords`	Whether to convert the ranges from a 1-based end-inclusive coordinate system to a 0-based end-exclusive coordinate system. Defaults to true for GRanges and false for other formats (see this archived UCSC blogpost)
`explicit_peak_names`	Boolean for whether to add rownames to the output matrix in format e.g chr1:500-1000, where start and end coords are given in a 0-based coordinate system. Note that either way, peak names will be written when the matrix is saved.

Value

Iterable matrix object with dimension ranges x cells. When saved, the column names of the output matrix will be in the format chr1:500-1000, where start and end coords are given in a 0-based coordinate system.

`mode` **options**

- `"insertions"`: Start and end coordinates are separately overlapped with each peak
- `"fragments"`: Like `"insertions"`, but each fragment can contribute at most 1 count to each peak, even if both the start and end coordinates overlap
- `"overlaps"`: Like `"fragments"`, but an overlap is also counted if the fragment fully spans the peak even if neither the start or end falls within the peak

Note

When calculating the matrix directly from a fragments tsv, it's necessary to first call `select_chromosomes()` in order to provide the ordering of chromosomes to expect while reading the tsv.

`plot_dot` *Dotplot*

Description

Plot feature levels per group or cluster as a grid of dots. Dots are colored by z-score normalized average expression, and sized by percent non-zero.

Usage

```
plot_dot(
  source,
  features,
  groups,
  group_order = NULL,
  gene_mapping = human_gene_mapping,
  colors = c("lightgrey", "#4682B4"),
  return_data = FALSE,
  apply_styling = TRUE
)
```

Arguments

source	Feature x cell matrix or data.frame with features. For best results, features should be sparse and log-normalized (e.g. run `log1p()` so zero raw counts map to zero)
features	Character vector of features to plot
groups	Vector with one entry per cell, specifying the cell's group
group_order	Optional vector listing ordering of groups
gene_mapping	An optional vector for gene name matching with match_gene_symbol().
colors	Color scale for plot
return_data	If true, return data from just before plotting rather than a plot.
apply_styling	If false, return a plot without pretty styling applied

---

plot_embedding *Plot UMAP or embeddings*

Description

Plot one or more features by coloring cells in a UMAP plot.

Usage

```
plot_embedding(
  source,
  embedding,
  features = NULL,
  quantile_range = c(0.01, 0.99),
  randomize_order = TRUE,
  smooth = NULL,
  smooth_rounds = 3,
  gene_mapping = human_gene_mapping,
  size = NULL,
```

```
  rasterize = FALSE,
  raster_pixels = 512,
  legend_continuous = c("auto", "quantile", "value"),
  labels_quantile_range = TRUE,
  colors_continuous = c("lightgrey", "#4682B4"),
  legend_discrete = TRUE,
  labels_discrete = TRUE,
  colors_discrete = discrete_palette("stallion"),
  return_data = FALSE,
  return_plot_list = FALSE,
  apply_styling = TRUE
)
```

Arguments

source	Matrix, or data frame to pull features from, or a vector of feature values for a single feature. For a matrix, the features must be rows.
embedding	A matrix of dimensions cells x 2 with embedding coordinates
features	Character vector of features to plot if source is not a vector.
quantile_range	(optional) Length 2 vector giving the quantiles to clip the minimum and maximum color scale values, as fractions between 0 and 1. NULL or NA values to skip clipping
randomize_order	If TRUE, shuffle cells to prevent overplotting biases. Can pass an integer instead to specify a random seed to use.
smooth	(optional) Sparse matrix of dimensions cells x cells with cell-cell distance weights for smoothing.
smooth_rounds	Number of multiplication rounds to apply when smoothing.
gene_mapping	An optional vector for gene name matching with match_gene_symbol(). Ignored if source is a data frame.
size	Point size for plotting
rasterize	Whether to rasterize the point drawing to speed up display in graphics programs.
raster_pixels	Number of pixels to use when rasterizing. Can provide one number for square dimensions, or two numbers for width x height.
legend_continuous	Whether to label continuous features by quantile or value. "auto" labels by quantile only when all features are continuous and quantile_range is not NULL. Quantile labeling adds text annotation listing the range of displayed values.
labels_quantile_range	Whether to add a text label with the value range of each feature when the legend is set to quantile
colors_continuous	Vector of colors to use for continuous color palette
legend_discrete	Whether to show the legend for discrete (categorical) features.

labels_discrete
: Whether to add text labels at the center of each group for discrete (categorical) features.

colors_discrete
: Vector of colors to use for discrete (categorical) features.

return_data
: If true, return data from just before plotting rather than a plot.

return_plot_list
: If TRUE, return multiple plots as a list, rather than a single plot combined using patchwork::wrap_plots()

apply_styling
: If false, return a plot without pretty styling applied

Details

Smoothing:

Smoothing is performed as follows: first, the smoothing matrix is normalized so the sum of incoming weights to every cell is 1. Then, the raw data values are repeatedly multiplied by the smoothing matrix and re-scaled so the average value stays the same.

Value

By default, returns a ggplot2 object with all the requested features plotted in a grid. If `return_data` or `return_plot_list` is called, the return value will match that argument.

---

plot_fragment_length *Fragment size distribution*

Description

Plot the distribution of fragment lengths, with length in basepairs on the x-axis, and proportion of fragments on the y-axis. Typical plots will show 10-basepair periodicity, as well as humps spaced at multiples of a nucleosome width (about 150bp).

Usage

```
plot_fragment_length(
  fragments,
  max_length = 500,
  return_data = FALSE,
  apply_styling = TRUE
)
```

Arguments

fragments
: Fragments object

max_length
: Maximum length to show on the plot

return_data
: If true, return data from just before plotting rather than a plot.

apply_styling
: If false, return a plot without pretty styling applied

Value

Numeric vector where index i contans the number of length-i fragments

---

`plot_read_count_knee` *Knee plot of single cell read counts*

Description

Plots read count rank vs. number of reads on a log-log scale.

Usage

```
plot_read_count_knee(
  read_counts,
  cutoff = NULL,
  return_data = FALSE,
  apply_styling = TRUE
)
```

Arguments

`read_counts`	Vector of read counts per cell
`cutoff`	(optional) Read cutoff to mark on the plot
`return_data`	If true, return data from just before plotting rather than a plot.
`apply_styling`	If false, return a plot without pretty styling applied

Details

Performs logarithmic downsampling to reduce the number of points plotted

Value

ggplot2 plot object

`plot_tf_footprint` *Plot TF footprint*

Description

Plot the footprinting around TF motif sites

Usage

```
plot_tf_footprint(
  fragments,
  motif_positions,
  cell_groups = rlang::rep_along(cellNames(fragments), "all"),
  flank = 250L,
  smooth = 0L,
  zero_based_coords = !is(genes, "GRanges"),
  colors = discrete_palette("stallion"),
  return_data = FALSE,
  apply_styling = TRUE
)
```

Arguments

`fragments`	IterableFragments object
`motif_positions`	Coordinate ranges for motifs (must include strand) and have constant width
`cell_groups`	Character or factor assigning a group to each cell, in order of `cellNames(fragments)`
`flank`	Number of flanking basepairs to include on either side of the motif
`smooth`	(optional) Sparse matrix of dimensions cells x cells with cell-cell distance weights for smoothing.
`zero_based_coords`	If true, coordinates start and 0 and the end coordinate is not included in the range. If false, coordinates start at 1 and the end coordinate is included in the range
`return_data`	If true, return data from just before plotting rather than a plot.
`apply_styling`	If false, return a plot without pretty styling applied

See Also

`footprint()`, `plot_tss_profile()`

`plot_tss_profile` *Plot TSS profile*

Description

Plot the enrichmment of insertions relative to transcription start sites (TSS). Typically, this plot shows strong enrichment of insertions near a TSS, and a small bump downstream around 220bp downstream of the TSS for the +1 nucleosome.

Usage

```
plot_tss_profile(
  fragments,
  genes,
  cell_groups = rlang::rep_along(cellNames(fragments), "all"),
  flank = 2000L,
  smooth = 0L,
  zero_based_coords = !is(genes, "GRanges"),
  colors = discrete_palette("stallion"),
  return_data = FALSE,
  apply_styling = TRUE
)
```

Arguments

`fragments`	IterableFragments object
`genes`	Coordinate ranges for genes (must include strand)
`cell_groups`	Character or factor assigning a group to each cell, in order of `cellNames(fragments)`
`flank`	Number of flanking basepairs to include on either side of the motif
`smooth`	Number of bases to smooth over (rolling average)
`zero_based_coords`	If true, coordinates start and 0 and the end coordinate is not included in the range. If false, coordinates start at 1 and the end coordinate is included in the range
`return_data`	If true, return data from just before plotting rather than a plot.
`apply_styling`	If false, return a plot without pretty styling applied

See Also

`footprint()`, `plot_tf_footprint()`

plot_tss_scatter *TSS Enrichment vs. Fragment Counts plot*

Description

Density scatter plot with log10(fragment_count) on the x-axis and TSS enrichment on the y-axis. This plot is most useful to select which cell barcodes in an experiment correspond to high-quality cells

Usage

```
plot_tss_scatter(
  atac_qc,
  min_frags = NULL,
  min_tss = NULL,
  bins = 100,
  apply_styling = TRUE
)
```

Arguments

atac_qc	Tibble as returned by qc_scATAC(). Must have columns nFrags and TSSEnrichment
min_frags	Minimum fragment count cutoff
min_tss	Minimum TSS Enrichment cutoff
bins	Number of bins for density calculation
apply_styling	If false, return a plot without pretty styling applied

prefix_cell_names *Add sample prefix to cell names*

Description

Rename cells by adding a prefix to the names. Most commonly this will be a sample name. All cells will recieve the exact text of prefix added to the beginning, so any separator characters like "_" must be included in the given prefix. Use this prior to merging fragments from different experiments with c() in order to help prevent cell name clashes.

Usage

```
prefix_cell_names(fragments, prefix)
```

Arguments

fragments	Input fragments object.
prefix	String to add as the prefix

Value

Fragments object with prefixed names

`qc_scATAC` *Calculate ArchR-compatible per-cell QC statistics*

Description

Calculate ArchR-compatible per-cell QC statistics

Usage

```
qc_scATAC(fragments, genes, blacklist)
```

Arguments

`fragments`	IterableFragments object
`genes`	Gene coordinates given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position
`blacklist`	Blacklisted regions given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position

Details

This implementation mimics ArchR's default parameters. For uses requiring more flexibility to tweak default parameters, the best option is to re-implement this function with required changes. Output columns of data.frame:

- `cellName`: cell name for each cell
- `nFrags`: number of fragments per cell
- `subNucleosomal`, `monoNucleosomal`, `multiNucleosomal`: number of fragments of size 1-146bp, 147-254bp, and 255bp + respectively. equivalent to ArchR's nMonoFrags, nDiFrags, nMultiFrags respectively
- `TSSEnrichment`: `AvgInsertInTSS / max(AvgInsertFlankingTSS, 0.1)`, where `AvgInsertInTSS` is `ReadsInTSS / 101` (window size), and `AvgInsertFlankingTSS` is `ReadsFlankingTSS / (100*2)` (window size). The `max(0.1)` ensures that very low-read cells do not get assigned spuriously high TSSEnrichment.
- `ReadsInPromoter`: Number of reads from 2000bp upstream of TSS to 101bp downstream of TSS
- `ReadsInBlacklist`: Number of reads in the provided blacklist region
- `ReadsInTSS`: Number of reads overlapping the 101bp centered around each TSS
- `ReadsFlankingTSS`: Number of reads overlapping 1901-2000bp +/- each TSS

Differences from ArchR: Note that ArchR by default uses a different set of annotations to derive TSS sites and promoter sites. This function uses just one annotation for gene start+end sites, so must be called twice to exactly re-calculate the ArchR QC stats.

ArchR's `PromoterRatio` and `BlacklistRatio` are not included in the output, as they can be easily calculated from `ReadsInPromoter / nFrags` and `ReadsInBlacklist / nFrags`. Similarly, ArchR's `NucleosomeRatio` can be calculated as `(monoNucleosomal + multiNucleosomal) / subNucleosomal`.

Value

data.frame with QC data

`range_distance_to_nearest`

Find signed distance to nearest genomic ranges

Description

Given a set of genomic ranges, find the distance to the nearest neighbors both upstream and downstream.

Usage

```
range_distance_to_nearest(
  ranges,
  addArchRBug = FALSE,
  zero_based_coords = !is(ranges, "GRanges")
)
```

Arguments

`ranges`	Genomic regions given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position • `strand`: +/- or TRUE/FALSE for positive or negative strand
`addArchRBug`	boolean to reproduce ArchR's bug that incorrectly handles nested genes
`zero_based_coords`	If true, coordinates start and 0 and the end coordinate is not included in the range. If false, coordinates start at 1 and the end coordinate is included in the range

Value

A 2-column data.frame with columns upstream and downstream, containing the distances to the nearest neighbor in the respective directions. For ranges on + or * strand, distance is calculated as:

- upstream = `max(start(range) - end(upstreamNeighbor), 0)`

- downstream = `max(start(downstreamNeighbor) - end(range), 0)`

For ranges on - strand, the definition of upstream and downstream is flipped. Note that this definition of distance is one off from `GenomicRanges::distance()`, as ranges that neighbor but don't overlap are given a distance of 1 rather than 0.

`read_bed` *Read a bed file into a data frame*

Description

Bed files can contain peak or blacklist annotations. These utilities help read thos annotations

Usage

```
read_bed(
  path,
  additional_columns = character(0),
  backup_url = NULL,
  timeout = 300
)

read_encode_blacklist(
  dir,
  genome = c("hg38", "mm10", "hg19", "dm6", "dm3", "ce11", "ce10"),
  timeout = 300
)
```

Arguments

`path`	Path to file (or desired save location if backup_url is used)
`additional_columns`	Names for additional columns in the bed file
`backup_url`	If path does not exist, provides a URL to download the gtf from
`timeout`	Maximum time in seconds to wait for download from backup_url
`dir`	Output directory to cache the downloaded gtf file
`genome`	genome name

Details

read_bed

Read a bed file from disk or a url.

read_encode_blacklist

Downloads the Boyle Lab blacklist, as described in https://doi.org/10.1038/s41598-019-45839-z

Value

Data frame with coordinates using the 0-based convention.

See Also

read_gtf(), read_gencode_genes()

---

read_gtf *Read GTF gene annotations*

Description

Read gene annotations from gtf format into a data frame. The source can be a URL, a gtf file on disk, or a gencode release version.

Usage

```
read_gtf(
  path,
  attributes = c("gene_id"),
  tags = character(0),
  features = c("gene"),
  keep_attribute_column = FALSE,
  backup_url = NULL,
  timeout = 300
)

read_gencode_genes(
  dir,
  release = "latest",
  annotation_set = c("basic", "comprehensive"),
  gene_type = "lncRNA|protein_coding|IG_.*_gene|TR_.*_gene",
  attributes = c("gene_id", "gene_type", "gene_name"),
  tags = character(0),
  features = c("gene"),
  timeout = 300
)

read_gencode_transcripts(
  dir,
  release = "latest",
  transcript_choice = c("MANE_Select", "Ensembl_Canonical", "all"),
  annotation_set = c("basic", "comprehensive"),
  gene_type = "lncRNA|protein_coding|IG_.*_gene|TR_.*_gene",
  attributes = c("gene_id", "gene_type", "gene_name", "transcript_id"),
  features = c("transcript", "exon"),
  timeout = 300
)
```

Arguments

path	Path to file (or desired save location if backup_url is used)
attributes	Vector of GTF attribute names to parse out as columns
tags	Vector of tags to parse out as boolean presence/absence
features	List of features types to keep from the GTF (e.g. gene, transcript, exon, intron)
keep_attribute_column	Boolean for whether to preserve the raw attribute text column
backup_url	If path does not exist, provides a URL to download the gtf from
timeout	Maximum time in seconds to wait for download from backup_url
dir	Output directory to cache the downloaded gtf file
release	release version (prefix with M for mouse versions). For most recent version, use "latest" or "latest_mouse"
annotation_set	Either "basic" or "comprehensive" annotation sets (see details section).
gene_type	Regular expression with which gene types to keep. Defaults to protein_coding, lncRNA, and IG/TR genes
transcript_choice	Method for selecting representative transcripts. Choices are: • MANE_Select: human-only, most conservative • Ensembl_Canonical: human+mouse, superset of MANE_Select for human • all: Preserve all transcript models (not recommended for plotting)

Details

read_gtf

Read gtf from a file or URL

read_gencode_genes

Read gene annotations directly from GENCODE. The file name will vary depending on the release and annotation set requested, but will be of the format `gencode.v42.annotation.gtf.gz`. GENCODE currently recommends the basic set: https://www.gencodegenes.org/human/. In release 42, both the comprehensive and basic sets had identical gene-level annotations, but the comprehensive set had additional transcript variants annotated.

read_gencode_transcripts

Read transcript models from GENCODE, for use with trackplot_gene()

Value

Data frame with coordinates using the 0-based convention. Columns are:

- chr
- source
- feature
- start

- end
- score
- strand
- frame
- attributes (optional; named according to listed attributes)
- tags (named according to listed tags)

See Also

read_bed(), read_encode_blacklist()

---

read_ucsc_chrom_sizes *Read UCSC chromosome sizes*

Description

Read chromosome sizes from UCSC and return as a tibble with one row per chromosome. The underlying data is pulled from here: https://hgdownload.soe.ucsc.edu/downloads.html

Usage

```
read_ucsc_chrom_sizes(
  dir,
  genome = c("hg38", "mm39", "mm10", "mm9", "hg19"),
  keep_chromosomes = "chr[0-9]+|chrX|chrY",
  timeout = 300
)
```

---

regress_out *Regress out unwanted variation*

Description

Regress out the effects of confounding variables using a linear least squares regression model.

Usage

```
regress_out(mat, latent_data, prediction_axis = c("row", "col"))
```

Arguments

mat	Input IterableMatrix
latent_data	Data to regress out, as a data.frame where each column is a variable to regress out.
prediction_axis	Which axis corresponds to prediction outputs from the linear models (e.g. the gene axis in typical single cell analysis). Options include "row" (default) and "col".

Details

Conceptually, regress_out calculates a linear least squares best fit model for each row of the matrix. (Or column if prediction_axis is "col"). The input data for each regression model are the columns of latent_data, and each model tries to predict the values in the corresponding row (or column) of mat. After fitting each model, regress_out will subtract the model predictions from the input values, aiming to only retain effects that are not explained by the variables in latent_data.

These models can be fit efficiently since they all share the same input data and so most of the calculations for the closed-form best fit solution are shared. A QR factorization of the model matrix and a dense matrix-vector multiply are sufficient to fully calculate the residual values.

Efficiency considerations: As the output matrix is dense rather than sparse, mean and variance calculations may run comparatively slowly. However, PCA and matrix/vector multiply operations can be performed at nearly the same cost as the input matrix due to mathematical simplifications. Memory usage scales with n_features * ((nrow(mat) + ncol(mat)). Generally, n_features == ncol(latent_data), but for categorical variables in latent_data, each category will be expanded into its own indicator variable. Memory usage will therefore be higher when using categorical input variables with many (i.e. >100) distinct values.

Value

IterableMatrix

---

rotate_x_labels *Rotate ggplot x axis labels*

Description

Rotate ggplot x axis labels

Usage

```
rotate_x_labels(degrees = 45)
```

Arguments

degrees	Number of degrees to rotate by

`sctransform_pearson` *SCTransform Pearson Residuals*

Description

Calculate pearson residuals of a negative binomial sctransform model. Normalized values are calculated as `(X - mu) / sqrt(mu + mu^2/theta)`. mu is calculated as `cell_read_counts * gene_beta`.

Usage

```
sctransform_pearson(
  mat,
  gene_theta,
  gene_beta,
  cell_read_counts,
  min_var = -Inf,
  clip_range = c(-10, 10),
  columns_are_cells = TRUE,
  slow = FALSE
)
```

Arguments

`mat`	IterableMatrix (raw counts)
`gene_theta`	Vector of per-gene thetas (overdispersion values)
`gene_beta`	Vector of per-gene betas (expression level values)
`cell_read_counts`	Vector of total reads per (umi count for RNA)
`min_var`	Minimum value for clipping variance
`clip_range`	Length 2 vector of min and max clipping range
`columns_are_cells`	Whether the columns of the matrix correspond to cells (default) or genes
`slow`	If TRUE, use a 10x slower but more precise implementation (default FALSE)

Details

The parameterization used is somewhat simplified compared to the original SCTransform paper, in particular it uses a linear-scale rather than log-scale to represent the cell_read_counts and gene_beta variables. It also does not support the addition of arbitrary cell metadata (e.g. batch) to add to the negative binomial regression.

Value

IterableMatrix

select_cells *Subset, translate, or reorder cell IDs*

Description

Subset, translate, or reorder cell IDs

Usage

```
select_cells(fragments, cell_selection)
```

Arguments

fragments	Input fragments object
cell_selection	List of chromosme IDs (numeric), or names (character). The output cell ID n will be taken from the input cell with ID/name cell_selection[n].

select_chromosomes *Subset, translate, or reorder chromosome IDs*

Description

Subset, translate, or reorder chromosome IDs

Usage

```
select_chromosomes(fragments, chromosome_selection)
```

Arguments

fragments	Input fragments object
chromosome_selection	List of chromosme IDs (numeric), or names (character). The output chromosome ID n will be taken from the input fragments chromosome with ID/name chromosome_selection[n].

select_regions *Subset fragments by genomic region*

Description

Fragments can be subset based on overlapping (or not overlapping) a set of regions

Usage

```
select_regions(
  fragments,
  ranges,
  invert_selection = FALSE,
  zero_based_coords = !is(ranges, "GRanges")
)
```

Arguments

`fragments`	Input fragments object.
`ranges`	Peaks/ranges to overlap, given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position
`invert_selection`	If TRUE, select fragments *not* overlapping selected regions instead of only fragments overlapping the selected regions.
`zero_based_coords`	Whether to convert the ranges from a 1-based end-inclusive coordinate system to a 0-based end-exclusive coordinate system. Defaults to true for GRanges and false for other formats (see this archived UCSC blogpost)

Value

Fragments object filtered according to the selected regions

set_trackplot_label *Adjust trackplot properties*

Description

Adjust labels and heights on trackplots. Labels are set as facet labels in ggplot2, and heights are additional properties read by `trackplot_combine()` to determine relative height of input plots.

Usage

```
set_trackplot_label(plot, labels)

set_trackplot_height(plot, height)

get_trackplot_height(plot)
```

Arguments

plot	ggplot object
labels	character vector of labels – must match existing number of facets in plot
height	New height. If numeric, adjusts relative height. If ggplot2::unit or grid::unit sets absolute height in specified units. "null" units are interpreted as relative height.

Value

set_trackplot_label: ggplot object with adjusted facet labels

set_trackplot_height: ggplot object with adjusted trackplot height

get_trackplot_height: ggplot2::unit object with height setting

---

shift_fragments *Shift start or end coordinates*

Description

Shifts start or end of fragments by a fixed amount, which can be useful to correct the Tn5 offset.

Usage

```
shift_fragments(fragments, shift_start = 0L, shift_end = 0L)
```

Arguments

fragments	Input fragments object
shift_start	How many basepairs to shift the start coords
shift_end	How many basepairs to shift the end coords

Details

The correct Tn5 offset is +/- 4bp since the Tn5 cut sites on opposite strands are offset by 9bp. However, +4/-5 bp is often applied to bed-format files, since the end coordinate in bed files is 1 past the last basepair of the sequenced DNA fragment. This results in a bed-like format except with inclusive end coordinates.

Value

Shifted fragments object

`subset_lengths` *Subset fragments by length*

Description

Subset fragments by length

Usage

```
subset_lengths(fragments, min_len = 0L, max_len = NA_integer_)
```

Arguments

`fragments`	Input fragments object
`min_len`	Minimum bases in fragment (inclusive)
`max_len`	Maximum bases in fragment (inclusive)

Details

Fragment length is calculated as `end-start`

Value

Fragments object

`svds` *Calculate svds*

Description

Use the C++ Spectra solver (same as RSpectra package), in order to compute the largest k values and corresponding singular vectors. Empirically, memory usage is much lower than using `irlba::irlba()`, likely due to avoiding R garbage creation while solving due to the pure-C++ solver. This documentation is a slightly-edited version of the `RSpectra::svds()` documentation.

Usage

```
svds(A, k, nu = k, nv = k, opts = list(), threads=0L, ...)
```

Arguments

A	The matrix whose truncated SVD is to be computed.
k	Number of singular values requested.
nu	Number of right singular vectors to be computed. This must be between 0 and 'k'. (Must be equal to 'k' for BPCells IterableMatrix)
opts	Control parameters related to computing algorithm. See *Details* below
threads	Control threads to use calculating mat-vec producs (BPCells specific)

Details

When RSpectra is installed, this function will just add a method to `RSpectra::svds()` for the `IterableMatrix` class.

The `opts` argument is a list that can supply any of the following parameters:

`ncv` Number of Lanzcos basis vectors to use. More vectors will result in faster convergence, but with greater memory use. `ncv` must be satisfy $k < ncv \leq p$ where `p = min(m, n)`. Default is `min(p, max(2*k+1, 20))`.

`tol` Precision parameter. Default is 1e-10.

`maxitr` Maximum number of iterations. Default is 1000.

`center` Either a logical value (`TRUE`/`FALSE`), or a numeric vector of length n. If a vector c is supplied, then SVD is computed on the matrix $A - 1c'$, in an implicit way without actually forming this matrix. `center = TRUE` has the same effect as `center = colMeans(A)`. Default is `FALSE`. Ignored in BPCells

`scale` Either a logical value (`TRUE`/`FALSE`), or a numeric vector of length n. If a vector s is supplied, then SVD is computed on the matrix $(A - 1c')S$, where c is the centering vector and $S = diag(1/s)$. If `scale = TRUE`, then the vector s is computed as the column norm of $A - 1c'$. Default is `FALSE`. Ignored in BPCells

Value

A list with the following components:

d	A vector of the computed singular values.
u	An `m` by `nu` matrix whose columns contain the left singular vectors. If `nu == 0`, `NULL` will be returned.
v	An `n` by `nv` matrix whose columns contain the right singular vectors. If `nv == 0`, `NULL` will be returned.
nconv	Number of converged singular values.
niter	Number of iterations used.
nops	Number of matrix-vector multiplications used.

References

Qiu Y, Mei J (2022). *RSpectra: Solvers for Large-Scale Eigenvalue and SVD Problems*. R package version 0.16-1, https://CRAN.R-project.org/package=RSpectra.

`tile_matrix` *Calculate ranges x cells tile overlap matrix*

Description

Calculate ranges x cells tile overlap matrix

Usage

```
tile_matrix(
  fragments,
  ranges,
  mode = c("insertions", "fragments"),
  zero_based_coords = !is(ranges, "GRanges"),
  explicit_tile_names = FALSE
)
```

Arguments

`fragments`	Input fragments object
`ranges`	Tiled regions given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position • `tile_width`: Size of each tile in this region in basepairs Must be non-overlapping and sorted by (chr, start), with chromosomes ordered according to the chromosome names of `fragments`
`mode`	Mode for counting tile overlaps. (See "value" section for more detail)
`zero_based_coords`	Whether to convert the ranges from a 1-based end-inclusive coordinate system to a 0-based end-exclusive coordinate system. Defaults to true for GRanges and false for other formats (see this archived UCSC blogpost)
`explicit_tile_names`	Boolean for whether to add rownames to the output matrix in format e.g chr1:500-1000, where start and end coords are given in a 0-based coordinate system. For whole-genome Tile matrices the names will take ~5 seconds to generate and take up 400MB of memory. Note that either way, tile names will be written when the matrix is saved.

Value

Iterable matrix object with dimension ranges x cells. When saved, the column names will be in the format chr1:500-1000, where start and end coords are given in a 0-based coordinate system.

`mode` **options**

- `"insertions"`: Start and end coordinates are separately overlapped with each tile
- `"fragments"`: Like `"insertions"`, but each fragment can contribute at most 1 count to each tile, even if both the start and end coordinates overlap

Note

When calculating the matrix directly from a fragments tsv, it's necessary to first call `select_chromosomes()` in order to provide the ordering of chromosomes to expect while reading the tsv.

`trackplot_combine` *Combine track plots*

Description

Combines multiple track plots of the same region into a single grid. Uses the `patchwork` package to perform the alignment.

Usage

```
trackplot_combine(
  tracks,
  side_plot = NULL,
  title = NULL,
  side_plot_width = 0.3
)
```

Arguments

`tracks`	List of tracks in order from top to bottom, generally ggplots as output from the other `trackplot_*()` functions.
`side_plot`	Optional plot to align to the right (e.g. RNA expression per cluster). Will be aligned to a `trackplot_coverage()` output if present, or else the first generic ggplot in the alignment. Should be in horizontal orientation and in the same cluster ordering as the coverage plots.
`title`	Text for overarching title of the plot
`side_plot_width`	Fraction of width that should be used for the side plot relative to the main track area

Value

A plot object with aligned genome plots. Each aligned row has the text label, y-axis, and plot body. The relative height of each row is given by heights. A shared title and x-axis are put at the top.

See Also

`trackplot_coverage()`, `trackplot_gene()`, `trackplot_loop()`, `trackplot_scalebar()`

`trackplot_coverage` *Pseudobulk coverage trackplot*

Description

Plot a pseudobulk genome track, showing the number of fragment insertions across a region for each cell type or group.

Usage

```
trackplot_coverage(
  fragments,
  region,
  groups,
  cell_read_counts,
  group_order = NULL,
  bins = 500,
  clip_quantile = 0.999,
  colors = discrete_palette("stallion"),
  legend_label = "group",
  zero_based_coords = !is(region, "GRanges"),
  return_data = FALSE
)
```

Arguments

`fragments`	Fragments object
`region`	Region to plot, e.g. output from `gene_region()`. String of format "chr1:100-200", or list/data.frame/GRanges of length 1 specifying chr, start, end. See `help("genomic-ranges-like")` for details
`groups`	Vector with one entry per cell, specifying the cell's group
`cell_read_counts`	Numeric vector of read counts for each cell (used for normalization)
`group_order`	Optional vector listing ordering of groups
`bins`	Number of bins to plot across the region
`clip_quantile`	(optional) Quantile of values for clipping y-axis limits. Default of 0.999 will crop out just the most extreme outliers across the region. NULL to disable clipping
`colors`	Character vector of color values (optionally named by group)
`legend_label`	Custom label to put on the legend
`zero_based_coords`	Whether to convert the ranges from a 1-based end-inclusive coordinate system to a 0-based end-exclusive coordinate system. Defaults to true for GRanges and false for other formats (see this archived UCSC blogpost)
`return_data`	If true, return data from just before plotting rather than a plot.
`scale_bar`	Whether to include a scale bar in the top track (`TRUE` or `FALSE`)

Value

Returns a combined plot of pseudobulk genome tracks. For compatability with `draw_trackplot_grid()`, the extra attribute `$patches$labels` will be added to specify the labels for each track. If `return_data` or `return_plot_list` is `TRUE`, the return value will be modified accordingly.

See Also

`trackplot_combine()`, `trackplot_gene()`, `trackplot_loop()`, `trackplot_scalebar()`

---

trackplot_gene *Plot transcript models*

Description

Plot transcript models

Usage

```
trackplot_gene(
  transcripts,
  region,
  exon_size = 2.5,
  gene_size = 0.5,
  label_size = 11 * 0.8/ggplot2::.pt,
  track_label = "Genes",
  return_data = FALSE
)
```

Arguments

`transcripts`	Transcipt features given as GRanges, data.frame, or list. See `help("genomic-ranges-like")` for details on format and coordinate systems. Required attributes: • `chr`, `start`, `end`: genomic position • `strand`: +/- or TRUE/FALSE for positive or negative strand • `feature`: Only entries marked as `"transcript"` or `"exon"` will be considered • `gene_name`: Symbol or gene ID to display • `transcript_id`: Transcritp identifier to link transcripts and exons Usually given as the output from `read_gencode_transcripts()`
`region`	Region to plot, e.g. output from `gene_region()`. String of format "chr1:100-200", or list/data.frame/GRanges of length 1 specifying chr, start, end. See `help("genomic-ranges-like")` for details
`exon_size`	size for exon lines in units of mm
`gene_size`	size for intron/gene lines in units of mm

label_size	size for transcript labels in units of mm
return_data	If true, return data from just before plotting rather than a plot.
labels	Character vector with labels for each item in transcripts. NA for items that should not be labeled
transcript_size	size for transcript lines in units of mm

Value

Plot of gene locations

See Also

trackplot_combine(), trackplot_coverage(), trackplot_loop(), trackplot_scalebar()

---

trackplot_genome_annotation

Plot range-based annotation tracks (e.g. peaks)

Description

Plot range-based annotation tracks (e.g. peaks)

Usage

```
trackplot_genome_annotation(
  loci,
  region,
  color_by = NULL,
  colors = NULL,
  label_by = NULL,
  label_size = 11 * 0.8/ggplot2::.pt,
  show_strand = FALSE,
  annotation_size = 2.5,
  track_label = "Peaks",
  return_data = FALSE
)
```

Arguments

loci	Genomic loci given as GRanges, data.frame, or list. See help("genomic-ranges-like") for details on format and coordinate systems. Required attributes: • chr, start, end: genomic position
region	Region to plot, e.g. output from gene_region(). String of format "chr1:100-200", or list/data.frame/GRanges of length 1 specifying chr, start, end. See help("genomic-ranges-like") for details

color_by	Name of a metadata column in loci to use for coloring, or a data vector with same length as loci. Column must be numeric or convertible to a factor.
colors	Vector of hex color codes to use for the color scale. For numeric color_by data, this is passed to ggplot2::scale_color_gradientn(), otherwise it is interpreted as a discrete color palette in ggplot2::scale_color_manual()
label_by	Name of a metadata column in loci to use for labeling, or a data vector with same length as loci. Column must hold string data.
label_size	size for labels in units of mm
show_strand	If TRUE, show strand direction as arrows
annotation_size	size for annotation lines in mm
return_data	If true, return data from just before plotting rather than a plot.

Value

Plot of genomic loci if return_data is FALSE, otherwise returns the data frame used to generate the plot

See Also

trackplot_combine(), trackplot_coverage(), trackplot_loop(), trackplot_scalebar(), trackplot_gene()

---

trackplot_loop *Plot loops*

Description

Plot loops

Usage

```
trackplot_loop(
  loops,
  region,
  color_by = NULL,
  colors = NULL,
  allow_truncated = TRUE,
  curvature = 0.75,
  track_label = "Links",
  return_data = FALSE
)
```

Arguments

loops	Genomic regions given as GRanges, data.frame, or list. See help("genomic-ranges-like") for details on format and coordinate systems. Required attributes: • chr, start, end: genomic position
region	Region to plot, e.g. output from gene_region(). String of format "chr1:100-200", or list/data.frame/GRanges of length 1 specifying chr, start, end. See help("genomic-ranges-like") for details
color_by	Name of a metadata column in loops to use for coloring, or a data vector with same length as loci. Column must be numeric or convertible to a factor.
colors	Vector of hex color codes to use for the color scale. For numeric color_by data, this is passed to ggplot2::scale_color_gradientn(), otherwise it is interpreted as a discrete color palette in ggplot2::scale_color_manual()
allow_truncated	If FALSE, remove any loops that are not fully contained within region
curvature	Curvature value between 0 and 1. 1 is a 180-degree arc, and 0 is flat lines.
return_data	If true, return data from just before plotting rather than a plot.

Value

Plot of loops connecting genomic coordinates

See Also

trackplot_combine(), trackplot_coverage(), trackplot_gene(), trackplot_scalebar(), trackplot_genome_annotation()

---

trackplot_scalebar *Plot scale bar*

Description

Plots a human-readable scale bar and coordinates of the region being plotted

Usage

```
trackplot_scalebar(region, font_pt = 11)
```

Arguments

region	Region to plot, e.g. output from gene_region(). String of format "chr1:100-200", or list/data.frame/GRanges of length 1 specifying chr, start, end. See help("genomic-ranges-like") for details
font_pt	Font size for scale bar labels in units of pt.

Value

Plot with coordinates and scalebar for plotted genomic region

See Also

`trackplot_combine()`, `trackplot_coverage()`, `trackplot_gene()`, `trackplot_loop()`

`transpose_storage_order` *Transpose the storage order for a matrix*

Description

Transpose the storage order for a matrix

Usage

```
transpose_storage_order(
  matrix,
  outdir = tempfile("transpose"),
  tmpdir = tempdir(),
  load_bytes = 4194304L,
  sort_bytes = 1073741824L
)
```

Arguments

`matrix`	Input matrix
`outdir`	Directory to store the output
`tmpdir`	Temporary directory to use for intermediate storage
`load_bytes`	The minimum contiguous load size during the merge sort passes
`sort_bytes`	The amount of memory to allocate for re-sorting chunks of entries

Details

This re-sorts the entries of a matrix to change the storage order from row-major to col-major. For large matrices, this can be slow – around 2 minutes to transpose a 500k cell RNA-seq matrix The default load_bytes (4MiB) and sort_bytes (1GiB) parameters allow ~85GB of data to be sorted with two passes through the data, and ~7.3TB of data to be sorted in three passes through the data.

Value

MatrixDir object with a copy of the input matrix, but the storage order flipped

write_fragments_memory *Read/write BPCells fragment objects*

Description

BPCells fragments can be read/written in compressed (bitpacked) or uncompressed form in a variety of storage locations: in memory (as an R object), in an hdf5 file, or in a directory on disk (containing binary files).

Usage

```
write_fragments_memory(fragments, compress = TRUE)

write_fragments_dir(
  fragments,
  dir,
  compress = TRUE,
  buffer_size = 1024L,
  overwrite = FALSE
)

open_fragments_dir(dir, buffer_size = 1024L)

write_fragments_hdf5(
  fragments,
  path,
  group = "fragments",
  compress = TRUE,
  buffer_size = 8192L,
  chunk_size = 1024L,
  overwrite = FALSE,
  gzip_level = 0L
)

open_fragments_hdf5(path, group = "fragments", buffer_size = 16384L)
```

Arguments

fragments	Input fragments object
compress	Whether or not to compress the data. With compression, storage size is be about half the size of a gzip-compressed 10x fragments file.
dir	Directory to read/write the data from
buffer_size	For performance tuning only. The number of items to be bufferred in memory before calling writes to disk.

overwrite	If TRUE, write to a temp dir then overwrite existing data. Alternatively, pass a temp path as a string to customize the temp dir location.
path	Path to the hdf5 file on disk
group	The group within the hdf5 file to write the data to. If writing to an existing hdf5 file this group must not already be in use
chunk_size	For performance tuning only. The chunk size used for the HDF5 array storage.
gzip_level	Gzip compression level. Default is 0 (no compression). This is recommended when both compression and compatibility with outside programs is required. Otherwise, using compress=TRUE is recommended as it is >10x faster with often similar compression levels.

Details

Saving in a directory on disk is a good default for local analysis, as it provides the best I/O performance and lowest memory usage. The HDF5 format allows saving within existing hdf5 files to group data together, and the in memory format provides the fastest performance in the event memory usage is unimportant.

Value

Fragment object

`write_insertion_bedgraph`

Write insertion counts to bedgraph file

Description

Write insertion counts data for one or more pseudobulks to bedgraph format. This reports the total number insertions at each basepair for each group listed in `cell_groups`.

Usage

```
write_insertion_bedgraph(
  fragments,
  path,
  cell_groups = NULL,
  insertion_mode = c("both", "start_only", "end_only")
)
```

Arguments

fragments	IterableFragments object
path	Path(s) to save bedgraph to, optionally ending in ".gz" to add gzip compression. If `cell_groups` is provided, `path` must be a named character vector, with one name for each level in `cell_groups`

cell_groups	Character or factor assigning a group to each cell, in order of cellNames(fragments)
insertion_mode	Which fragment ends to use for insertion counts calculation. One of "both", "start_only", or "end_only"

---

write_matrix_memory *Read/write sparse matrices*

Description

BPCells matrices are stored in sparse format, meaning only the non-zero entries are stored. Matrices can store integer counts data or decimal numbers (float or double). See details for more information.

Usage

```
write_matrix_memory(mat, compress = TRUE)

write_matrix_dir(
  mat,
  dir,
  compress = TRUE,
  buffer_size = 8192L,
  overwrite = FALSE
)

open_matrix_dir(dir, buffer_size = 8192L)

write_matrix_hdf5(
  mat,
  path,
  group,
  compress = TRUE,
  buffer_size = 8192L,
  chunk_size = 1024L,
  overwrite = FALSE,
  gzip_level = 0L
)

open_matrix_hdf5(path, group, buffer_size = 16384L)
```

Arguments

compress	Whether or not to compress the data.
dir	Directory to save the data into
buffer_size	For performance tuning only. The number of items to be buffered in memory before calling writes to disk.

overwrite	If TRUE, write to a temp dir then overwrite existing data. Alternatively, pass a temp path as a string to customize the temp dir location.
path	Path to the hdf5 file on disk
group	The group within the hdf5 file to write the data to. If writing to an existing hdf5 file this group must not already be in use
chunk_size	For performance tuning only. The chunk size used for the HDF5 array storage.
gzip_level	Gzip compression level. Default is 0 (no compression). This is recommended when both compression and compatibility with outside programs is required. Otherwise, using compress=TRUE is recommended as it is >10x faster with often similar compression levels.
matrix	Input matrix, either IterableMatrix or dgCMatrix

Details

Storage locations:

Matrices can be stored in a directory on disk, in memory, or in an HDF5 file. Saving in a directory on disk is a good default for local analysis, as it provides the best I/O performance and lowest memory usage. The HDF5 format allows saving within existing hdf5 files to group data together, and the in memory format provides the fastest performance in the event memory usage is unimportant.

Bitpacking Compression:

For typical RNA counts matrices holding integer counts, this bitpacking compression will result in 6-8x less space than an R dgCMatrix, and 4-6x smaller than a scipy csc_matrix. The compression will be more effective when the count values in the matrix are small, and when the rows of the matrix are sorted by rowMeans. In tests on RNA-seq data optimal ordering could save up to 40% of storage space. On non-integer data only the row indices are compressed, not the values themselves so space savings will be smaller.

For non-integer data matrices, bitpacking compression is much less effective, as it can only be applied to the indexes of each entry but not the values. There will still be some space savings, but far less than for counts matrices.

Value

BPCells matrix object

Index