

# Package ‘SimBu’

December 17, 2024

**Title** Simulate Bulk RNA-seq Datasets from Single-Cell Datasets

**Version** 1.9.0

**Description** SimBu can be used to simulate bulk RNA-seq datasets with known cell type fractions.

You can either use your own single-cell study for the simulation or the sfaira database.

Different pre-defined simulation scenarios exist, as are options to run custom simulations.

Additionally, expression values can be adapted by adding an mRNA bias, which produces more biologically relevant simulations.

**License** GPL-3 + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Imports** basilisk, BiocParallel, data.table, dplyr, ggplot2, tools,  
Matrix (>= 1.3.3), methods, phyloseq, proxyC, RColorBrewer,  
RCurl, reticulate, sparseMatrixStats, SummarizedExperiment,  
tidyr

**Suggests** curl, knitr, matrixStats, rmarkdown, Seurat (>= 5.0.0),  
SeuratObject (>= 5.0.0), testthat (>= 3.0.0)

**URL** <https://github.com/omnideconv/SimBu>

**BugReports** <https://github.com/omnideconv/SimBu/issues>

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**StagedInstall** no

**biocViews** Software, RNASeq, SingleCell

**git\_url** <https://git.bioconductor.org/packages/SimBu>

**git\_branch** devel

**git\_last\_commit** 3836594

**git\_last\_commit\_date** 2024-10-29

**Repository** Bioconductor 3.21

**Date/Publication** 2024-12-16

**Author** Alexander Dietrich [aut, cre]

**Maintainer** Alexander Dietrich <alex.dietrich@tum.de>

## Contents

|  |           |
|--|-----------|
| calc_scaling_vector . . . . .            | 2         |
| census . . . . .                         | 3         |
| census_monocle . . . . .                 | 4         |
| check_annotation . . . . .               | 5         |
| check_if_tpm . . . . .                   | 5         |
| compare_matrix_with_annotation . . . . . | 6         |
| dataset . . . . .                        | 6         |
| dataset_h5ad . . . . .                   | 8         |
| dataset_merge . . . . .                  | 9         |
| dataset_seurat . . . . .                 | 11        |
| dataset_sfaira . . . . .                 | 12        |
| dataset_sfaira_multiple . . . . .        | 14        |
| dmode . . . . .                          | 15        |
| download_sfaira . . . . .                | 16        |
| download_sfaira_multiple . . . . .       | 16        |
| filter_matrix . . . . .                  | 17        |
| generate_summarized_experiment . . . . . | 18        |
| h5ad_to_adata . . . . .                  | 19        |
| merge_scaling_factor . . . . .           | 19        |
| merge_simulations . . . . .              | 20        |
| plot_simulation . . . . .                | 21        |
| save_simulation . . . . .                | 22        |
| setup_sfaira . . . . .                   | 23        |
| sfaira_overview . . . . .                | 24        |
| SimBu . . . . .                          | 25        |
| simulate_bulk . . . . .                  | 25        |
| simulate_sample . . . . .                | 29        |
| <b>Index</b>                             | <b>31</b> |

---

calc\_scaling\_vector     *Calculate scaling factor for a dataset*

---

### Description

Each scaling factor has a default matrix it will try to use (counts or TPM). If the required matrix is not available, the other one is used and a warning is given.

### Usage

```
calc_scaling_vector(
  data,
  scaling_factor,
  custom_scaling_vector,
  scaling_factor_single_cell,
  BPPARAM,
```

```

    run_parallel
  )

```

### Arguments

**data** dataset object

**scaling\_factor** name of scaling factor; possible are: census, spike\_in, read\_number, custom or NONE for no scaling factor

**custom\_scaling\_vector** named vector with custom scaling values for cell-types. Cell-types that do not occur in this vector but are present in the dataset will be set to 1

**scaling\_factor\_single\_cell** boolean: decide if a scaling value for each single cell is calculated (default) or the median of all scaling values for each cell type is calculated

**BPPARAM** BiocParallel::bpparam() by default; if specific number of threads x want to be used, insert: BiocParallel::MulticoreParam(workers = x)

**run\_parallel** boolean, decide if multi-threaded calculation will be run. FALSE by default

### Value

a named vector with a scaling value for each cell in the dataset

---

|        |  |
|--------|--|
| census | <i>Applies the Census count transformation on a count matrix</i> |
|--------|--|

---

### Description

needs a sparse matrix with cells in columns and genes in rows. You can find the detailed explanation here: <http://cole-trapnell-lab.github.io/monocle-release/docs/#census>

### Usage

```

census(
  matrix,
  exp_capture_rate = 0.25,
  expr_threshold = 0,
  BPPARAM = BiocParallel::bpparam(),
  run_parallel = FALSE
)

```

### Arguments

**matrix** sparse count matrix; cells in columns, genes in rows

**exp\_capture\_rate** expected capture rate; default=0.25

**expr\_threshold** expression threshold; default=0

**BPPARAM** BiocParallel::bpparam() by default; if specific number of threads x want to be used, insert: BiocParallel::MulticoreParam(workers = x)

**run\_parallel** boolean, decide if multi-threaded calculation will be run. FALSE by default

### Value

a vector for each cell-type, with a scaling factor which can be used to transform the counts of the matrix

### Examples

```
tpm <- Matrix::Matrix(matrix(rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::t(1e6 * Matrix::t(tpm) / Matrix::colSums(tpm))
cen <- SimBu::census(tpm)
```

---

census\_monocle      *Census calculation as implemented in monocle*

---

### Description

Implementation taken from Monocle2: <https://github.com/cole-trapnell-lab/monocle-release/blob/master/R/normalization.R#>

### Usage

```
census_monocle(expr_matrix, exp_capture_rate, expr_threshold)
```

### Arguments

**expr\_matrix**      TPM matrix

**exp\_capture\_rate**  
expected capture rate; default=0.25

**expr\_threshold**   expression threshold; default=0

### Value

vector with estimated mRNA values per cell in expr\_matrix

---

|                  |  |
|------------------|--|
| check_annotation | <i>check for correct column names in annotation file and replace them if necessary</i> |
|------------------|--|

---

**Description**

check for correct column names in annotation file and replace them if necessary

**Usage**

```
check_annotation(annotation, cell_column = "cell_type", id_column = 1)
```

**Arguments**

|             |   |
|-------------|---|
| annotation  | dataframe; annotation dataframe                               |
| cell_column | name of cell-type column; default is "cell_type"              |
| id_column   | name of cell ID column; default is 1, which uses the rownames |

**Value**

annotation dataframe with correct column names

---

|              |  |
|--------------|--|
| check_if_tpm | <i>Checks, if a matrix is TPM-like (columns sum up to 1e6)</i> |
|--------------|--|

---

**Description**

Checks, if a matrix is TPM-like (columns sum up to 1e6)

**Usage**

```
check_if_tpm(tpm_matrix, lower_limit = 7e+05)
```

**Arguments**

|             |                                       |
|-------------|---------------------------------------|
| tpm_matrix  | matrix to check                       |
| lower_limit | the lowest sum value, a cell may have |

**Value**

boolean

---

`compare_matrix_with_annotation`

*Check if annotation and matrix have same cells*

---

### Description

Otherwise intersection of both is used

### Usage

```
compare_matrix_with_annotation(m, annotation)
```

### Arguments

|                         |   |
|-------------------------|---|
| <code>m</code>          | matrix, column names are cells                              |
| <code>annotation</code> | data.frame, rownames are genes, cell names are in ID column |

### Value

intersected matrix

---

`dataset`

*Build [SummarizedExperiment](#) using local annotation and count matrix R objects*

---

### Description

Build [SummarizedExperiment](#) using local annotation and count matrix R objects

### Usage

```
dataset(
  annotation,
  count_matrix = NULL,
  tpm_matrix = NULL,
  name = "SimBu_dataset",
  spike_in_col = NULL,
  additional_cols = NULL,
  filter_genes = TRUE,
  variance_cutoff = 0,
  type_abundance_cutoff = 0,
  scale_tpm = TRUE
)
```

**Arguments**

|                       |   |
|-----------------------|---|
| annotation            | (mandatory) dataframe; needs columns 'ID' and 'cell_type'; 'ID' needs to be equal with cell_names in count_matrix   |
| count_matrix          | (mandatory) sparse count matrix; raw count data is expected with genes in rows, cells in columns  |
| tpm_matrix            | sparse count matrix; TPM like count data is expected with genes in rows, cells in columns   |
| name                  | name of the dataset; will be used for new unique IDs of cells   |
| spike_in_col          | which column in annotation contains information on spike_in counts, which can be used to re-scale counts; mandatory for spike_in scaling factor in simulation |
| additional_cols       | list of column names in annotation, that should be stored as well in dataset object   |
| filter_genes          | boolean, if TRUE, removes all genes with 0 expression over all samples & genes with variance below variance_cutoff  |
| variance_cutoff       | numeric, is only applied if filter_genes is TRUE: removes all genes with variance below the chosen cutoff (default = 0)                                       |
| type_abundance_cutoff | numeric, remove all cells, whose cell-type appears less then the given value. This removes low abundant cell-types  |
| scale_tpm             | boolean, if TRUE (default) the cells in tpm_matrix will be scaled to sum up to 1e6  |

**Value**

Return a [SummarizedExperiment](#) object

**Examples**

```
counts <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::t(1e6 * Matrix::t(tpm) / Matrix::colSums(tpm))

colnames(counts) <- paste0("cell_", rep(1:300))
colnames(tpm) <- paste0("cell_", rep(1:300))
rownames(counts) <- paste0("gene_", rep(1:1000))
rownames(tpm) <- paste0("gene_", rep(1:1000))

annotation <- data.frame(
  "ID" = paste0("cell_", rep(1:300)),
  "cell_type" = c(rep("T cells CD4", 300))
)

ds <- SimBu::dataset(annotation = annotation, count_matrix = counts, tpm_matrix = tpm, name = "test_dataset")
```

---

dataset\_h5ad

*Build [SummarizedExperiment](#) using a h5ad file for the counts*


---

## Description

Build [SummarizedExperiment](#) using a h5ad file for the counts

## Usage

```
dataset_h5ad(
  h5ad_file_counts,
  h5ad_file_tpm = NULL,
  cell_id_col = "ID",
  cell_type_col = "cell_type",
  cells_in_obs = TRUE,
  name = "SimBu_dataset",
  spike_in_col = NULL,
  additional_cols = NULL,
  filter_genes = TRUE,
  variance_cutoff = 0,
  type_abundance_cutoff = 0,
  scale_tpm = TRUE
)
```

## Arguments

|                  |   |
|------------------|---|
| h5ad_file_counts | (mandatory) h5ad file with raw count data   |
| h5ad_file_tpm    | h5ad file with TPM count data   |
| cell_id_col      | (mandatory) name of column in Seurat meta.data with unique cell ids; 0 for rownames   |
| cell_type_col    | (mandatory) name of column in Seurat meta.data with cell type name  |
| cells_in_obs     | boolean, if TRUE, cell identifiers are taken from obs layer in anndata object; if FALSE, they are taken from var  |
| name             | name of the dataset; will be used for new unique IDs of cells#' @param spike_in_col which column in annotation contains information on spike_in counts, which can be used to re-scale counts; mandatory for spike_in scaling factor in simulation |
| spike_in_col     | which column in annotation contains information on spike_in counts, which can be used to re-scale counts; mandatory for spike_in scaling factor in simulation   |
| additional_cols  | list of column names in annotation, that should be stored as well in dataset object   |
| filter_genes     | boolean, if TRUE, removes all genes with 0 expression over all samples & genes with variance below variance_cutoff  |



|                       |  |
|-----------------------|--|
| variance_cutoff       | numeric, is only applied if filter_genes is TRUE: removes all genes with variance below the chosen cutoff          |
| type_abundance_cutoff | numeric, remove all cells, whose cell-type appears less then the given value. This removes low abundant cell-types |
| scale_tpm             | boolean, if TRUE (default) the cells in tpm_matrix will be scaled to sum up to 1e6                                 |

**Value**

Return a [SummarizedExperiment](#) object

**Examples**

```
# h5 <- system.file("extdata", "anndata.h5ad", package = "SimBu")
# ds_h5ad <- SimBu::dataset_h5ad(
#   h5ad_file_counts = h5,
#   name = "h5ad_dataset",
#   cell_id_col = "id", # this will use the 'id' column of the metadata as cell identifiers
#   cell_type_col = "group", # this will use the 'group' column of the metadata as cell type info
#   cells_in_obs = TRUE
# ) # in case your cell information is stored in the var layer, switch to FALSE
```

---

dataset\_merge

---

Merge multiple [SummarizedExperiment](#) datasets into one

---

**Description**

The objects need to have the same number of assays in order to work.

**Usage**

```
dataset_merge(
  dataset_list,
  name = "SimBu_dataset",
  spike_in_col = NULL,
  additional_cols = NULL,
  filter_genes = TRUE,
  variance_cutoff = 0,
  type_abundance_cutoff = 0,
  scale_tpm = TRUE
)
```

**Arguments**

|                       |   |
|-----------------------|---|
| dataset_list          | (mandatory) list of <a href="#">SummarizedExperiment</a> objects  |
| name                  | name of the new dataset   |
| spike_in_col          | which column in annotation contains information on spike_in counts, which can be used to re-scale counts; mandatory for spike_in scaling factor in simulation |
| additional_cols       | list of column names in annotation, that should be stored as well in dataset object   |
| filter_genes          | boolean, if TRUE, removes all genes with 0 expression over all samples & genes with variance below variance_cutoff  |
| variance_cutoff       | numeric, is only applied if filter_genes is TRUE: removes all genes with variance below the chosen cutoff   |
| type_abundance_cutoff | numeric, remove all cells, whose cell-type appears less then the given value. This removes low abundant cell-types  |
| scale_tpm             | boolean, if TRUE (default) the cells in tpm_matrix will be scaled to sum up to 1e6  |

**Value**

[SummarizedExperiment](#) object

**Examples**

```
counts <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::t(1e6 * Matrix::t(tpm) / Matrix::colSums(tpm))

colnames(counts) <- paste0("cell_", rep(1:300))
colnames(tpm) <- paste0("cell_", rep(1:300))
rownames(counts) <- paste0("gene_", rep(1:1000))
rownames(tpm) <- paste0("gene_", rep(1:1000))

annotation <- data.frame(
  "ID" = paste0("cell_", rep(1:300)),
  "cell_type" = c(rep("T cells CD4", 300))
)

ds1 <- SimBu::dataset(annotation = annotation, count_matrix = counts, tpm_matrix = tpm, name = "test_dataset1")
ds2 <- SimBu::dataset(annotation = annotation, count_matrix = counts, tpm_matrix = tpm, name = "test_dataset2")
ds_merged <- SimBu::dataset_merge(list(ds1, ds2))
```

---

dataset\_seurat      *Build SummarizedExperiment using a Seurat object*

---

## Description

Build [SummarizedExperiment](#) using a [Seurat](#) object

## Usage

```
dataset_seurat(
  seurat_obj,
  counts_layer,
  cell_id_col,
  cell_type_col,
  assay = NULL,
  tpm_layer = NULL,
  name = "SimBu_dataset",
  spike_in_col = NULL,
  additional_cols = NULL,
  filter_genes = TRUE,
  variance_cutoff = 0,
  type_abundance_cutoff = 0,
  scale_tpm = TRUE
)
```

## Arguments

|                 |   |
|-----------------|---|
| seurat_obj      | (mandatory) <a href="#">Seurat</a> object with TPM counts   |
| counts_layer    | (mandatory) name of assay in Seurat object which contains count data in 'counts' slot   |
| cell_id_col     | (mandatory) name of column in Seurat meta.data with unique cell ids   |
| cell_type_col   | (mandatory) name of column in Seurat meta.data with cell type name  |
| assay           | name of the Seurat objeey assay that should be used. If NULL (default), the currently active assay is used  |
| tpm_layer       | name of assay in Seurat object which contains TPM data in 'counts' slot   |
| name            | name of the dataset; will be used for new unique IDs of cells   |
| spike_in_col    | which column in annotation contains information on spike_in counts, which can be used to re-scale counts; mandatory for spike_in scaling factor in simulation |
| additional_cols | list of column names in annotation, that should be stored as well in dataset object   |
| filter_genes    | boolean, if TRUE, removes all genes with 0 expression over all samples & genes with variance below variance_cutoff  |
| variance_cutoff | numeric, is only applied if filter_genes is TRUE: removes all genes with variance below the chosen cutoff   |

type\_abundance\_cutoff  
                           numeric, remove all cells, whose cell-type appears less then the given value.  
                           This removes low abundant cell-types

scale\_tpm  
                           boolean, if TRUE (default) the cells in tpm\_matrix will be scaled to sum up to  
                           1e6

## Value

Return a [SummarizedExperiment](#) object

## Examples

```
counts <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::t(1e6 * Matrix::t(tpm) / Matrix::colSums(tpm))

colnames(counts) <- paste0("cell-", rep(1:300))
colnames(tpm) <- paste0("cell-", rep(1:300))
rownames(counts) <- paste0("gene-", rep(1:1000))
rownames(tpm) <- paste0("gene-", rep(1:1000))

annotation <- data.frame(
  "ID" = paste0("cell-", rep(1:300)),
  "cell_type" = c(
    rep("T cells CD4", 50),
    rep("T cells CD8", 50),
    rep("Macrophages", 100),
    rep("NK cells", 10),
    rep("B cells", 70),
    rep("Monocytes", 20)
  ),
  row.names = paste0("cell-", rep(1:300))
)

seurat_obj <- Seurat::CreateSeuratObject(counts = counts, assay = "gene_expression", meta.data = annotation)
SeuratObject::LayerData(seurat_obj, assay = "gene_expression", layer = "data") <- tpm

ds_seurat <- SimBu::dataset_seurat(
  seurat_obj = seurat_obj,
  counts_layer = "counts",
  cell_id_col = "ID",
  cell_type_col = "cell_type",
  tpm_layer = "data",
  name = "seurat_dataset"
)
```

**Description**

Build [SummarizedExperiment](#) using a single sfaira entry ID

**Usage**

```
dataset_sfaira(
  sfaira_id,
  sfaira_setup,
  name = "SimBu_dataset",
  spike_in_col = NULL,
  additional_cols = NULL,
  force = FALSE,
  filter_genes = TRUE,
  variance_cutoff = 0,
  type_abundance_cutoff = 0,
  scale_tpm = TRUE
)
```

**Arguments**

|                                    |   |
|------------------------------------|---|
| <code>sfaira_id</code>             | (mandatory) ID of a sfaira dataset  |
| <code>sfaira_setup</code>          | (mandatory) the sfaira setup; given by <a href="#">setup_sfaira</a>   |
| <code>name</code>                  | name of the dataset; will be used for new unique IDs of cells   |
| <code>spike_in_col</code>          | which column in annotation contains information on spike_in counts, which can be used to re-scale counts                        |
| <code>additional_cols</code>       | list of column names in annotation, that should be stored as well in dataset object   |
| <code>force</code>                 | boolean, if TRUE, datasets without annotation will be downloaded, FALSE otherwise (default)                                     |
| <code>filter_genes</code>          | boolean, if TRUE, removes all genes with 0 expression over all samples & genes with variance below <code>variance_cutoff</code> |
| <code>variance_cutoff</code>       | numeric, is only applied if <code>filter_genes</code> is TRUE: removes all genes with variance below the chosen cutoff          |
| <code>type_abundance_cutoff</code> | numeric, remove all cells, whose cell-type appears less then the given value. This removes low abundant cell-types              |
| <code>scale_tpm</code>             | boolean, if TRUE (default) the cells in <code>tpm_matrix</code> will be scaled to sum up to 1e6                                 |

**Value**

dataset object

**Examples**

```

setup_list <- SimBu::setup_sfaira(tempdir())
ds <- SimBu::dataset_sfaira(
  sfaira_id = "homo sapiens_lungparenchyma_2019_10x3v2_madissoon_001_10.1186/s13059-019-1906-x",
  sfaira_setup = setup_list,
  name = "test_dataset"
)

```

---

dataset\_sfaira\_multiple

*Build [SummarizedExperiment](#) using multiple sfaira entries*

---

**Description**

You can apply different filters on the whole data-zoo of sfaria; the resulting single-cell datasets will be combined into a single dataset which you can use for simulation Note: only datasets in sfaira with annotation are considered!

**Usage**

```

dataset_sfaira_multiple(
  organisms = NULL,
  tissues = NULL,
  assays = NULL,
  sfaira_setup,
  name = "SimBu_dataset",
  spike_in_col = NULL,
  additional_cols = NULL,
  filter_genes = TRUE,
  variance_cutoff = 0,
  type_abundance_cutoff = 0,
  scale_tpm = TRUE
)

```

**Arguments**

|                 |  |
|-----------------|--|
| organisms       | (mandatory) list of organisms (only human and mouse available)   |
| tissues         | (mandatory) list of tissues  |
| assays          | (mandatory) list of assays   |
| sfaira_setup    | (mandatory) the sfaira setup; given by <a href="#">setup_sfaira</a>                                      |
| name            | name of the dataset; will be used for new unique IDs of cells  |
| spike_in_col    | which column in annotation contains information on spike_in counts, which can be used to re-scale counts |
| additional_cols | list of column names in annotation, that should be stored as well in dataset object                      |

**filter\_genes**     boolean, if TRUE, removes all genes with 0 expression over all samples & genes with variance below **variance\_cutoff**  
**variance\_cutoff**     numeric, is only applied if **filter\_genes** is TRUE: removes all genes with variance below the chosen cutoff  
**type\_abundance\_cutoff**     numeric, remove all cells, whose cell-type appears less then the given value. This removes low abundant cell-types  
**scale\_tpm**     boolean, if TRUE (default) the cells in **tpm\_matrix** will be scaled to sum up to 1e6

**Value**

dataset object

**Examples**

```

setup_list <- SimBu::setup_sfaira(tempdir())
ds_human_lung <- SimBu::dataset_sfaira_multiple(
  sfaira_setup = setup_list,
  organisms = "Homo sapiens",
  tissues = "lung parenchyma",
  assay = "10x 3' v2",
  name = "human_lung"
)
  
```

---

dmode

*use gaussian kernel to calculate the mode of transcript counts*


---

**Description**

use gaussian kernel to calculate the mode of transcript counts

**Usage**

```
dmode(x)
```

**Arguments**

**x**                   vector of numeric values

**Value**

most commonly occurring (log-transformed) TPM value

---

|                 |   |
|-----------------|---|
| download_sfaira | <i>download a specific dataset from sfaira by an ID</i> |
|-----------------|---|

---

### Description

download a specific dataset from sfaira by an ID

### Usage

```
download_sfaira(
  setup_list,
  ids,
  force = FALSE,
  synapse_user = NULL,
  synapse_pw = NULL
)
```

### Arguments

|              |  |
|--------------|--|
| setup_list   | the sfaira setup; given by <a href="#">setup_sfaira</a>  |
| ids          | the IDs of the datasets  |
| force        | logical; TRUE if you want to force the download, even though no cell-type annotation exists for this dataset. Default if FALSE |
| synapse_user | character; username for synapse portal ( <a href="https://www.synapse.org">https://www.synapse.org</a> )                       |
| synapse_pw   | character; password for synapse portal ( <a href="https://www.synapse.org">https://www.synapse.org</a> )                       |

### Value

matrix, gene names and cell IDs

---

|                          |   |
|--------------------------|---|
| download_sfaira_multiple | <i>download multiple datasets from sfaira using filters for organism, tissue and/or assay</i> |
|--------------------------|---|

---

### Description

similar to the filters on the sfaira website (<https://theislab.github.io/sfaira-portal/Datasets>)



**Usage**

```
download_sfaira_multiple(
  setup_list,
  organisms = NULL,
  tissues = NULL,
  assays = NULL,
  force = FALSE
)
```

**Arguments**

|            |   |
|------------|---|
| setup_list | the sfaira setup; given by <a href="#">setup_sfaira</a>   |
| organisms  | list of organisms (only human and mouse available)  |
| tissues    | list of tissues   |
| assays     | list of assays  |
| force      | logical; TRUE if you want to force to download all datasets, otherwise only the ones with cell-type annotation will be returned. Default if FALSE |

**Value**

annotated data object, contains count matrix and annotation

---

|               |   |
|---------------|---|
| filter_matrix | <i>filter one (or two) expression matrix by genes</i> |
|---------------|---|

---

**Description**

filter one (or two) expression matrix by genes

**Usage**

```
filter_matrix(m1, m2 = NULL, filter_genes = TRUE, variance_cutoff = 0)
```

**Arguments**

|                 |  |
|-----------------|--|
| m1              | Matrix 1   |
| m2              | Matrix 2 (optional)                                  |
| filter_genes    | boolean  |
| variance_cutoff | numeric, genes below this variance value are removed |

**Value**

filtered matrix

---

```
generate_summarized_experiment
```

*Generate SummarizedExperiment using multiple parameters*

---

## Description

Generate SummarizedExperiment using multiple parameters

## Usage

```
generate_summarized_experiment(  
  annotation,  
  count_matrix,  
  tpm_matrix,  
  name,  
  spike_in_col,  
  additional_cols,  
  filter_genes,  
  variance_cutoff,  
  type_abundance_cutoff,  
  scale_tpm  
)
```

## Arguments

|                       |   |
|-----------------------|---|
| annotation            | (mandatory) dataframe; needs columns 'ID' and 'cell_type'; 'ID' needs to be equal with cell_names in count_matrix   |
| count_matrix          | (mandatory) sparse count matrix; raw count data is expected with genes in rows, cells in columns  |
| tpm_matrix            | sparse count matrix; TPM like count data is expected with genes in rows, cells in columns   |
| name                  | name of the dataset; will be used for new unique IDs of cells   |
| spike_in_col          | which column in annotation contains information on spike_in counts, which can be used to re-scale counts; mandatory for spike_in scaling factor in simulation |
| additional_cols       | list of column names in annotation, that should be stored as well in dataset object   |
| filter_genes          | boolean, if TRUE, removes all genes with 0 expression over all samples & genes with variance below variance_cutoff  |
| variance_cutoff       | numeric, is only applied if filter_genes is TRUE: removes all genes with variance below the chosen cutoff   |
| type_abundance_cutoff | numeric, remove all cells, whose cell-type appears less then the given value. This removes low abundant cell-types  |
| scale_tpm             | boolean, if TRUE (default) the cells in tpm_matrix will be scaled to sum up to 1e6  |

**Value**

Return a [SummarizedExperiment](#) object

---

|               |   |
|---------------|---|
| h5ad_to_adata | <i>Use basilisk environment to read h5ad file and access anndata object</i> |
|---------------|---|

---

**Description**

Use basilisk environment to read h5ad file and access anndata object

**Usage**

```
h5ad_to_adata(h5ad_path, cells_in_obs)
```

**Arguments**

|              |  |
|--------------|--|
| h5ad_path    | path to h5ad file  |
| cells_in_obs | boolean, if TRUE, cell identifiers are taken from obs layer in anndata object; if FALSE, they are taken from var |

**Value**

matrix contained on h5ad file as dgCMatrix

---

|                      |  |
|----------------------|--|
| merge_scaling_factor | <i>Create scaling vector from custom or pre-defined scaling factor</i> |
|----------------------|--|

---

**Description**

Create scaling vector from custom or pre-defined scaling factor

**Usage**

```
merge_scaling_factor(data, scaling_factor_values, scaling_factor_name)
```

**Arguments**

|                       |                               |
|-----------------------|-------------------------------|
| data                  | dataset                       |
| scaling_factor_values | named list of scaling values  |
| scaling_factor_name   | name of scaling factor method |

**Value**

scaling vector

---

|                   |   |
|-------------------|---|
| merge_simulations | <i>Combine multiple simulations into one result</i> |
|-------------------|---|

---

### Description

we recommend to only merge simulations from the same dataset object, otherwise the count matrices might not correspond on the gene level

### Usage

```
merge_simulations(simulation_list)
```

### Arguments

```
simulation_list
  a list of simulations
```

### Value

named list; bulk a [SummarizedExperiment](#) object, where the assays store the simulated bulk RNAseq datasets. Can hold either one or two assays, depending on how many matrices were present in the dataset cell-fractions is a dataframe with the simulated cell-fractions per sample; scaling\_vector scaling value for each cell in dataset

### Examples

```
counts <- Matrix::Matrix(matrix(rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::Matrix(matrix(rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::t(1e6 * Matrix::t(tpm) / Matrix::colSums(tpm))

colnames(counts) <- paste0("cell_", rep(1:300))
colnames(tpm) <- paste0("cell_", rep(1:300))
rownames(counts) <- paste0("gene_", rep(1:1000))
rownames(tpm) <- paste0("gene_", rep(1:1000))

annotation <- data.frame(
  "ID" = paste0("cell_", rep(1:300)),
  "cell_type" = c(
    rep("T cells CD4", 50),
    rep("T cells CD8", 50),
    rep("Macrophages", 100),
    rep("NK cells", 10),
    rep("B cells", 70),
    rep("Monocytes", 20)
  )
)

dataset <- SimBu::dataset(
  annotation = annotation,
```

```
count_matrix = counts,
tpm_matrix = tpm,
name = "test_dataset"
)

s1 <- SimBu::simulate_bulk(dataset,
  scenario = "even",
  scaling_factor = "NONE",
  nsamples = 10,
  ncells = 100
)

s2 <- SimBu::simulate_bulk(dataset,
  scenario = "even",
  scaling_factor = "NONE",
  nsamples = 10,
  ncells = 100
)

s <- SimBu::merge_simulations(list(s1, s2))
```

---

plot\_simulation

*Plot the cell-type fractions in your simulated dataset*

---

## Description

Plot the cell-type fractions in your simulated dataset

## Usage

```
plot_simulation(simulation)
```

## Arguments

simulation      a simulation object generated by simulate\_bulk

## Value

a gplot2 barplot

## Examples

```
counts <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::t(1e6 * Matrix::t(tpm) / Matrix::colSums(tpm))

colnames(counts) <- paste0("cell_", rep(1:300))
colnames(tpm) <- paste0("cell_", rep(1:300))
rownames(counts) <- paste0("gene_", rep(1:1000))
rownames(tpm) <- paste0("gene_", rep(1:1000))
```

```

annotation <- data.frame(
  "ID" = paste0("cell_", rep(1:300)),
  "cell_type" = c(
    rep("T cells CD4", 50),
    rep("T cells CD8", 50),
    rep("Macrophages", 100),
    rep("NK cells", 10),
    rep("B cells", 70),
    rep("Monocytes", 20)
  )
)

dataset <- SimBu::dataset(
  annotation = annotation,
  count_matrix = counts,
  tpm_matrix = tpm,
  name = "test_dataset"
)

s <- SimBu::simulate_bulk(dataset,
  scenario = "even",
  scaling_factor = "NONE",
  nsamples = 10,
  ncells = 100
)

SimBu::plot_simulation(s)

```

---

save\_simulation      *Save the expression matrix of a simulated pseudo-bulk dataset to a file*

---

### Description

Save the expression matrix of a simulated pseudo-bulk dataset to a file

### Usage

```
save_simulation(simulation, filename, assay = "bulk_counts")
```

### Arguments

|            |   |
|------------|---|
| simulation | the result of simulate_bulk()                                   |
| filename   | the filename where to save the expression matrix to             |
| assay      | name of the assay in simulation to save, default to bulk_counts |

### Value

write a file

**Examples**

```

counts <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::t(1e6 * Matrix::t(tpm) / Matrix::colSums(tpm))

colnames(counts) <- paste0("cell_", rep(1:300))
colnames(tpm) <- paste0("cell_", rep(1:300))
rownames(counts) <- paste0("gene_", rep(1:1000))
rownames(tpm) <- paste0("gene_", rep(1:1000))

annotation <- data.frame(
  "ID" = paste0("cell_", rep(1:300)),
  "cell_type" = c(
    rep("T cells CD4", 50),
    rep("T cells CD8", 50),
    rep("Macrophages", 100),
    rep("NK cells", 10),
    rep("B cells", 70),
    rep("Monocytes", 20)
  )
)

dataset <- SimBu::dataset(
  annotation = annotation,
  count_matrix = counts,
  tpm_matrix = tpm,
  name = "test_dataset"
)

s <- SimBu::simulate_bulk(dataset,
  scenario = "even",
  scaling_factor = "NONE",
  nsamples = 10,
  ncells = 100
)

save_simulation(s, tempfile())

```

---

 setup\_sfaira

*setup the sfaira package*


---

**Description**

If you want to download datasets from Sfaira, you need to specify a directory where the datasets are saved into. Additionally, when this function is called for the first time, a conda environment will be established and sfaira along all of its dependencies are installed. This can take some time but will be only performed one single time, as the environment can be re-used.

**Usage**

```
setup_sfaira(basedir)
```

**Arguments**

`basedir` name of the directory, where the raw files will be downloaded into

**Value**

list with sfaira file directories; must be used as input for other sfaira based functions

**Examples**

```
setup_list <- setup_sfaira(basedir = tempdir())
```

---

|                              |  |
|------------------------------|--|
| <code>sfaira_overview</code> | <i>Gives an overview of the possible datasets you can use from the sfaira database</i> |
|------------------------------|--|

---

**Description**

Gives an overview of the possible datasets you can use from the sfaira database

**Usage**

```
sfaira_overview(setup_list)
```

**Arguments**

`setup_list` the sfaira setup; given by [setup\\_sfaira](#)

**Value**

a dataframe with information on each dataset

**Examples**

```
setup_list <- setup_sfaira(basedir = tempdir())  
# all_datasets <- sfaira_overview(setup_list)
```



---

|       |  |
|-------|--|
| SimBu | <i>SimBu: Bias-aware simulation of bulk RNA-seq data with variable cell type composition</i> |
|-------|--|

---

## Description

As complex tissues are typically composed of various cell types, deconvolution tools have been developed to computationally infer their cellular composition from bulk RNA sequencing (RNA-seq) data. To comprehensively assess deconvolution performance, gold-standard datasets are indispensable. The simulation of ‘pseudo-bulk’ data, generated by aggregating single-cell RNA-seq (scRNA-seq) expression profiles in pre-defined proportions, offers a scalable and cost-effective way of generating these gold-standard datasets. SimBu was developed to simulate pseudo-bulk samples based on various simulation scenarios, designed to test specific features of deconvolution methods. A unique feature of SimBu is the modelling of cell-type-specific mRNA bias using experimentally-derived or data-driven scaling factors.

## Dataset generation

You will need an annotated scRNA-seq dataset (as matrix file, h5ad file, Seurat object), which is the baseline for the simulations. Use the `dataset_*` functions to generate a `SummarizedExperiment`, that holds all important information. It is also possible to access scRNA-seq datasets through the public database Sfaira, by using the functions `dataset_sfaira()` and `dataset_sfaira_multiple()`.

## Simulation

Use the `simulate_bulk()` function to generate multiple pseudo-bulk samples, which will be returned as a `SummarizedExperiment`. You can adapt the cell type fractions in each sample by changing the `scenario` parameter.

## Visualization

Inspect the cell type composition of your simulations with the `plot_simulation()` function.

---

|                            |  |
|----------------------------|--|
| <code>simulate_bulk</code> | <i>Simulate whole pseudo-bulk RNAseq dataset</i> |
|----------------------------|--|

---

## Description

This function allows you to create a full pseudo-bulk RNAseq dataset. You need to provide a [SummarizedExperiment](#) from which the cells will be sampled for the simulation. Also a `scenario` has to be selected, where you can choose how the cells will be sampled and a `scaling_factor` on how the read counts will be transformed prior to the simulation.

**Usage**

```
simulate_bulk(
  data,
  scenario = c("even", "random", "mirror_db", "weighted", "pure", "custom"),
  scaling_factor = c("NONE", "census", "spike_in", "custom", "read_number",
    "expressed_genes", "annotation_column", "epic", "abis", "quantiseq"),
  scaling_factor_single_cell = TRUE,
  weighted_cell_type = NULL,
  weighted_amount = NULL,
  pure_cell_type = NULL,
  custom_scenario_data = NULL,
  custom_scaling_vector = NULL,
  balance_even_mirror_scenario = 0.01,
  remove_bias_in_counts = FALSE,
  remove_bias_in_counts_method = "read-number",
  norm_counts = FALSE,
  nsamples = 100,
  ncells = 1000,
  total_read_counts = NULL,
  whitelist = NULL,
  blacklist = NULL,
  seed = NA,
  BPPARAM = BiocParallel::bpparam(),
  run_parallel = FALSE
)
```

**Arguments**

**data** (mandatory) [SummarizedExperiment](#) object

**scenario** (mandatory) select one of the pre-defined cell-type fraction scenarios; possible are: even,random,mirror\_db,pure,weighted; you can also use the custom scenario, where you need to set the custom\_scenario\_data parameter.

**scaling\_factor** (mandatory) name of scaling factor; possible are: census, spike\_in, read\_number, expressed\_genes, custom, epic, abis, quantiseq or NONE for no scaling factor

**scaling\_factor\_single\_cell** boolean: decide if a scaling value for each single cell is calculated (default) or the median of all scaling values for each cell type is calculated

**weighted\_cell\_type** name of cell-type used for weighted scenario

**weighted\_amount** fraction of cell-type used for weighted scenario; must be between 0 and 0.99

**pure\_cell\_type** name of cell-type for pure scenario

**custom\_scenario\_data** dataframe; needs to be of size nsamples x number\_of\_cell\_types, where each sample is a row and each entry is the cell-type fraction. Rows need to sum up to 1.

|                              |  |
|------------------------------|--|
| custom_scaling_vector        | named vector with custom scaling values for cell-types. Cell-types that do not occur in this vector but are present in the dataset will be set to 1; mandatory for custom scaling factor   |
| balance_even_mirror_scenario | balancing value for the uniform and mirror_db scenarios: increasing it will result in more diverse simulated fractions. To get the same fractions in each sample, set to 0. Default is 0.01.   |
| remove_bias_in_counts        | boolean; if TRUE the internal mRNA bias that is present in count data will be <i>removed</i> using the number of reads mapped to each cell. Default to FALSE   |
| remove_bias_in_counts_method | 'read-number' (default) or 'gene-number'; method with which the mRNA bias in counts will be removed  |
| norm_counts                  | boolean; if TRUE the samples simulated with counts will be normalized to CPMs, default is FALSE  |
| nsamples                     | numeric; number of samples in pseudo-bulk RNAseq dataset (default = 100)   |
| ncells                       | numeric; number of cells in each dataset (default = 1000)  |
| total_read_counts            | numeric; sets the total read count value for each sample   |
| whitelist                    | list; give a list of cell-types you want to keep for the simulation; if NULL, all are used   |
| blacklist                    | list; give a list of cell-types you want to remove for the simulation; if NULL, all are used; is applied after whitelist   |
| seed                         | numeric; specify a seed for RNG. This effects cell sampling; with a fixed seed you will always sample the same cells for each sample (seed value is increased by 1 for each sample). Default = NA (two simulation runs will sample different cells). |
| BPPARAM                      | BiocParallel::bpparam() by default; if specific number of threads x want to be used, insert: BiocParallel::MulticoreParam(workers = x)   |
| run_parallel                 | boolean, decide if multi-threaded calculation will be run. FALSE by default  |

## Value

named list; bulk a [SummarizedExperiment](#) object, where the assays store the simulated bulk RNAseq datasets. Can hold either one or two assays, depending on how many matrices were present in the dataset `cell-fractions` is a dataframe with the simulated cell-fractions per sample; `scaling_vector` scaling value for each cell in dataset

## Examples

```
# generate sample single-cell data to work with:

counts <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::Matrix(matrix(stats::rpois(3e5, 5), ncol = 300), sparse = TRUE)
tpm <- Matrix::t(1e6 * Matrix::t(tpm) / Matrix::colSums(tpm))
```

```
colnames(counts) <- paste0("cell_", rep(1:300))
colnames(tpm) <- paste0("cell_", rep(1:300))
rownames(counts) <- paste0("gene_", rep(1:1000))
rownames(tpm) <- paste0("gene_", rep(1:1000))

annotation <- data.frame(
  "ID" = paste0("cell_", rep(1:300)),
  "cell_type" = c(
    rep("T cells CD4", 50),
    rep("T cells CD8", 50),
    rep("Macrophages", 100),
    rep("NK cells", 10),
    rep("B cells", 70),
    rep("Monocytes", 20)
  )
)

dataset <- SimBu::dataset(
  annotation = annotation,
  count_matrix = counts,
  tpm_matrix = tpm,
  name = "test_dataset"
)

# this creates a basic pseudo-bulk dataset with uniform cell-type distribution
# and no additional transformation of the data with 10 samples and 2000 cells each

s <- SimBu::simulate_bulk(dataset,
  scenario = "even",
  scaling_factor = "NONE",
  nsamples = 10,
  ncells = 100
)

# use a blacklist to exclude certain cell-types for the simulation
s <- SimBu::simulate_bulk(dataset,
  scenario = "even",
  scaling_factor = "NONE",
  nsamples = 10,
  ncells = 2000,
  blacklist = c("Monocytes", "Macrophages")
)

# use the pure scenario to only have B cells
s <- SimBu::simulate_bulk(dataset,
  scenario = "pure",
  scaling_factor = "NONE",
  nsamples = 10,
  ncells = 100,
  pure_cell_type = "B cells"
)
```

```

# simulate a dataset with custom cell-type fraction for each of the 3 samples
fractions <- data.frame(
  "B cells" = c(0.2, 0.4, 0.2),
  "T cells CD4" = c(0.4, 0.2, 0.1),
  "Macrophages" = c(0.4, 0.4, 0.7), check.names = FALSE
)
s <- SimBu::simulate_bulk(dataset,
  scenario = "custom",
  scaling_factor = "NONE",
  nsamples = 3,
  ncells = 2000,
  custom_scenario_data = fractions
)

```

---

simulate\_sample

*simulate single pseudo-bulk sample*


---

### Description

function to sample cells according to given cell-type fractions. This creates a single pseudo-bulk sample by calculating the mean expression value per gene over all sampled cells. Note: if `total_read_counts` is used, the cell-fractions are applied to the number of counts, not the number of cells!

### Usage

```

simulate_sample(
  data,
  scaling_vector,
  simulation_vector,
  total_cells,
  total_read_counts,
  remove_bias_in_counts,
  remove_bias_in_counts_method,
  norm_counts,
  seed
)

```

### Arguments

|                                |   |
|--------------------------------|---|
| <code>data</code>              | <a href="#">SummarizedExperiment</a> object   |
| <code>scaling_vector</code>    | vector with scaling values for each cell; calculated by the <code>calc_scaling_vector</code> function |
| <code>simulation_vector</code> | named vector with wanted cell-types and their fractions   |
| <code>total_cells</code>       | numeric; number of total cells for this simulation  |

total\_read\_counts        numeric; sets the total read count value for each sample

remove\_bias\_in\_counts    boolean; if TRUE (default) the internal mRNA bias that is present in count data will be *removed* using the number of reads mapped to each cell

remove\_bias\_in\_counts\_method    'read-number' (default) or 'gene-number'; method with which the mRNA bias in counts will be removed

norm\_counts            boolean; if TRUE the samples simulated with counts will be normalized to CPMs, default is FALSE

seed                    numeric; fix this value if you want the same cells to be sampled

**Value**

returns two vectors (one based on counts, one based on tpm; depends on which matrices are present in data) with expression values for all genes in the provided dataset

# Index

## \* **internal**

- calc\_scaling\_vector, 2
- census\_monocle, 4
- check\_annotation, 5
- check\_if\_tpm, 5
- compare\_matrix\_with\_annotation, 6
- dmode, 15
- download\_sfaira, 16
- download\_sfaira\_multiple, 16
- filter\_matrix, 17
- generate\_summarized\_experiment, 18
- h5ad\_to\_adata, 19
- merge\_scaling\_factor, 19
- simulate\_sample, 29

- calc\_scaling\_vector, 2
- census, 3
- census\_monocle, 4
- check\_annotation, 5
- check\_if\_tpm, 5
- compare\_matrix\_with\_annotation, 6

- dataset, 6
- dataset\_h5ad, 8
- dataset\_merge, 9
- dataset\_seurat, 11
- dataset\_sfaira, 12
- dataset\_sfaira\_multiple, 14
- dmode, 15
- download\_sfaira, 16
- download\_sfaira\_multiple, 16

- filter\_matrix, 17

- generate\_summarized\_experiment, 18

- h5ad\_to\_adata, 19

- merge\_scaling\_factor, 19
- merge\_simulations, 20

- plot\_simulation, 21

- save\_simulation, 22

- setup\_sfaira, 13, 14, 16, 17, 23, 24

- Seurat, 11

- sfaira\_overview, 24

- SimBu, 25

- simulate\_bulk, 25

- simulate\_sample, 29

- SummarizedExperiment, 6–14, 19, 20, 25–27, 29