

# Package ‘ADImpute’

February 24, 2025

**Type** Package

**Title** Adaptive Dropout Imputer (ADImpute)

**Version** 1.16.0

**Description** Single-cell RNA sequencing (scRNA-seq) methods are typically unable to quantify the expression levels of all genes in a cell, creating a need for the computational prediction of missing values (‘dropout imputation’). Most existing dropout imputation methods are limited in the sense that they exclusively use the scRNA-seq dataset at hand and do not exploit external gene-gene relationship information. Here we propose two novel methods: a gene regulatory network-based approach using gene-gene relationships learnt from external data and a baseline approach corresponding to a sample-wide average. ADImpute can implement these novel methods and also combine them with existing imputation methods (currently supported: DrImpute, SAVER). ADImpute can learn the best performing method per gene and combine the results from different methods into an ensemble.

**License** GPL-3 + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 4.0)

**Imports** checkmate, BiocParallel, data.table, DrImpute, kernlab, MASS, Matrix, methods, rsvd, S4Vectors, SAVER, SingleCellExperiment, stats, SummarizedExperiment, utils

**Suggests** BiocStyle, knitr, rmarkdown, testthat

**biocViews** GeneExpression, Network, Preprocessing, Sequencing, SingleCell, Transcriptomics

**RoxygenNote** 7.1.1

**VignetteBuilder** knitr

**BugReports** <https://github.com/anacarolinaleote/ADImpute/issues>

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

**git\_branch** RELEASE\_3\_20

**git\_last\_commit** fc250fa

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

**Repository** Bioconductor 3.20

**Date/Publication** 2025-02-24

**Author** Ana Carolina Leote [cre, aut] (<<https://orcid.org/0000-0003-0879-328X>>)

**Maintainer** Ana Carolina Leote <anacarolinaleote@gmail.com>

## Contents

ArrangeData . . . . .	3
CenterData . . . . .	3
CheckArguments_Impute . . . . .	4
ChooseMethod . . . . .	4
Combine . . . . .	5
ComputeMSEGenewise . . . . .	6
CreateArgCheck . . . . .	6
CreateTrainData . . . . .	7
DataCheck_Matrix . . . . .	7
DataCheck_Network . . . . .	8
DataCheck_SingleCellExperiment . . . . .	8
DataCheck_TrLength . . . . .	9
demo_data . . . . .	9
demo_net . . . . .	10
demo_sce . . . . .	10
EvaluateMethods . . . . .	11
GetDropoutProbabilities . . . . .	12
HandleBiologicalZeros . . . . .	13
Impute . . . . .	14
ImputeBaseline . . . . .	16
ImputeDrImpute . . . . .	17
ImputeNetParallel . . . . .	17
ImputeNetwork . . . . .	18
ImputeNPDropouts . . . . .	19
ImputePredictiveDropouts . . . . .	19
ImputeSAVER . . . . .	20
MaskData . . . . .	20
MaskerPerGene . . . . .	21
network.coefficients . . . . .	21
NormalizeRPM . . . . .	22
NormalizeTPM . . . . .	22
PseudoInverseSolution_percell . . . . .	23
ReadData . . . . .	24
ReturnChoice . . . . .	24
ReturnOut . . . . .	25
SetBiologicalZeros . . . . .	25
SplitData . . . . .	26
transcript_length . . . . .	26
WriteCSV . . . . .	27
WriteTXT . . . . .	27

**Index**

**28**

---

ArrangeData	<i>Data trimming</i>
-------------	----------------------

---

**Description**

ArrangeData finds common genes to the network and provided data and limits both datasets to these

**Usage**

```
ArrangeData(data, net.coef = NULL)
```

**Arguments**

data	matrix with entries equal to zero to be imputed (genes as rows and samples as columns)
net.coef	matrix; object containing network coefficients

**Value**

list; data matrix, network coefficients matrix and intercept for genes common between the data matrix and the network

---

CenterData	<i>Data centering</i>
------------	-----------------------

---

**Description**

CenterData centers expression of each gene at 0

**Usage**

```
CenterData(data)
```

**Arguments**

data	matrix of gene expression to be centered row-wise (genes as rows and samples as columns)
------	--

**Value**

list; row-wise centers and centered data

---

CheckArguments\_Impute *Argument check to Impute()*

---

### Description

CheckArguments\_Impute checks whether the arguments passed to Impute are correct.

### Usage

```
CheckArguments_Impute(data, method.choice, do, tr.length, labels,
cell.clusters, true.zero.thr, drop_thre)
```

### Arguments

data	matrix; raw counts (genes as rows and samples as columns)
method.choice	character; best performing method in training data for each gene
do	character; choice of methods to be used for imputation. Currently supported methods are 'Baseline', 'DrImpute', 'Network', and 'Ensemble'. Defaults to 'Ensemble'. Not case-sensitive. Can include one or more methods. Non-supported methods will be ignored.
tr.length	matrix with at least 2 columns: 'hgnc_symbol' and 'transcript_length'
labels	character; vector specifying the cell type of each column of data
cell.clusters	integer; number of cell subpopulations
true.zero.thr	if set to NULL (default), no true zero estimation is performed. Set to numeric value between 0 and 1 for estimation. Value corresponds to the threshold used to determine true zeros: if the probability of dropout is lower than true.zero.thr, the imputed entries are set to zero.
drop_thre	numeric; between 0 and 1 specifying the threshold to determine dropout values

### Value

NULL object

---

ChooseMethod *Method choice per gene*

---

### Description

ChooseMethod determines the method for dropout imputation based on performance on each gene in training data

### Usage

```
ChooseMethod(real, masked, imputed, write.to.file = TRUE)
```

**Arguments**

<code>real</code>	matrix; original gene expression data, i.e. before masking (genes as rows and samples as columns)
<code>masked</code>	matrix, logical indicating which entries were masked (genes as rows and samples as columns)
<code>imputed</code>	list; list of matrices with imputation results for all considered methods
<code>write.to.file</code>	logical; should the output be written to a file?

**Details**

The imputed values are compared to the real ones for every masked entry in `real`. The Mean Squared Error is computed for all masked entries per gene and the method with the best performance is chosen for each gene.

**Value**

character; best performing method in the training set for each gene

**See Also**

[ComputeMSEGenewise](#)

---

Combine

*Combine imputation methods*

---

**Description**

Combine imputation methods

**Usage**

```
Combine(data, imputed, method.choice, write = FALSE)
```

**Arguments**

<code>data</code>	matrix with entries equal to zero to be imputed, already normalized (genes as rows and samples as columns)
<code>imputed</code>	list; list of matrices with imputation results for all considered methods
<code>method.choice</code>	named character; vector with the best performing method per gene
<code>write</code>	logical; should a file with the imputation results be written?

**Details**

Combines imputation results from all methods according to training results provided in `method.choice`

**Value**

matrix; imputation results combining the best performing method per gene

---

ComputeMSEGenewise      *Computation of MSE per gene*

---

### Description

ComputeMSEGenewise computes the MSE of dropout imputation for a given gene.

### Usage

```
ComputeMSEGenewise(real, masked, imputed, baseline)
```

### Arguments

real	numeric; vector of original expression of a given gene (before masking)
masked	logical; vector indicating which entries were masked for a given gene
imputed	matrix; imputation results for a given imputation method
baseline	logical; is this baseline imputation?

### Value

MSE of all imputations indicated by masked

---

CreateArgCheck      *Argument check*

---

### Description

CreateArgCheck creates tests for argument correctness.

### Usage

```
CreateArgCheck(missing = NULL, match = NULL, acceptable = NULL,
null = NULL)
```

### Arguments

missing	named list; logical. Name corresponds to variable name, and corresponding entry to whether it was missing from the function call.
match	named list. Name corresponds to variable name, and corresponding entry to its value.
acceptable	named list. Name corresponds to variable name, and corresponding entry to its acceptable values.
null	named list; logical. Name corresponds to variable name, and corresponding entry to whether it was NULL in the function call.

### Value

argument check object.

---

CreateTrainData      *Preparation of training data for method evaluation*

---

### Description

CreateTrainingData selects a subset of cells to use as training set and sets a portion (mask) of the non-zero entries in each row of the subset to zero

### Usage

```
CreateTrainData(data, train.ratio = .7, train.only = TRUE, mask = .1,
write = FALSE)
```

### Arguments

data	matrix; raw counts (genes as rows and samples as columns)
train.ratio	numeric; ratio of the samples to be used for training
train.only	logical; if TRUE define only a training dataset, if FALSE writes both training and validation sets (defaults to TRUE)
mask	numeric; ratio of total non-zero samples to be masked per gene (defaults to .1)
write	logical; should the output be written to a file?

### Value

list with resulting matrix after subsetting and after masking

---

DataCheck\_Matrix      *Data check (matrix)*

---

### Description

DataCheck\_Matrix tests for potential format and storage issues with matrices. Helper function to ADImpute.

### Usage

```
DataCheck_Matrix(data)
```

### Arguments

data	data object to check
------	----------------------

### Value

data object with needed adjustments

---

DataCheck\_Network      *Data check (network)*

---

### Description

DataCheck\_Network tests for potential format and storage issues with the network coefficient matrix. Helper function to ADImpute.

### Usage

```
DataCheck_Network(network)
```

### Arguments

network                  data object containing matrix coefficients

### Value

network data object with needed adjustments

---

DataCheck\_SingleCellExperiment  
*Data check (SingleCellExperiment)*

---

### Description

DataCheck\_SingleCellExperiment tests for existence of the appropriate assays in sce. Helper function to ADImpute.

### Usage

```
DataCheck_SingleCellExperiment(sce, normalized = TRUE)
```

### Arguments

sce                        SingleCellExperiment; data for normalization or imputation  
normalized                logical; is the data expected to be normalized?

### Value

NULL object.



---

DataCheck_TrLength	<i>Data check (transcript length)</i>
--------------------	---------------------------------------

---

**Description**

DataCheck\_TrLength tests for potential format and storage issues with the object encoding transcript length, for e.g. TPM normalization. Helper function to ADImpute.

**Usage**

```
DataCheck_TrLength(trlength)
```

**Arguments**

trlength            data object containing transcript length information

**Value**

transcript length object with needed adjustments

---

demo_data	<i>Small dataset for example purposes</i>
-----------	---

---

**Description**

A small dataset to use on vignettes and examples (50 cells).

**Usage**

```
demo_data
```

**Format**

matrix; a subset of the Grun pancreas dataset, obtained with the scRNAseq R package, to use in the vignette and examples.

**References**

Grun D et al. (2016). De novo prediction of stem cell identity using single-cell transcriptome data. Cell Stem Cell 19(2), 266-277.

---

demo_net	<i>Small regulatory network for example purposes</i>
----------	--

---

**Description**

Subset of the Gene Regulatory Network used by ADImpute's Network imputation method.

**Usage**

demo\_net

**Format**

matrix; subset of the Gene Regulatory Network installed along with ADImpute.

---

demo_sce	<i>Small dataset for example purposes</i>
----------	---

---

**Description**

A small dataset to use on vignettes and examples (50 cells).

**Usage**

demo\_sce

**Format**

SingleCellExperiment; a subset of the Grun pancreas dataset, obtained with the scRNAseq R package, to use in the vignette and examples.

**References**

Grun D et al. (2016). De novo prediction of stem cell identity using single-cell transcriptome data. *Cell Stem Cell* 19(2), 266-277.

EvaluateMethods

*Imputation method evaluation on training set***Description**

EvaluateMethods returns the best-performing imputation method for each gene in the dataset

**Usage**

```
EvaluateMethods(data, sce = NULL, do = c('Baseline', 'DrImpute',
'Network'), write = FALSE, train.ratio = .7, train.only = TRUE,
mask.ratio = .1, outdir = getwd(), scale = 1, pseudo.count = 1,
labels = NULL, cell.clusters = 2, drop_thre = NULL, type = 'count',
cores = BiocParallel::bpworkers(BPPARAM),
BPPARAM = BiocParallel::SnowParam(type = "SOCK"),
net.coef = ADImpute::network.coefficients, net.implementation = 'iteration',
tr.length = ADImpute::transcript_length, bulk = NULL, ...)
```

**Arguments**

data	matrix; normalized counts, not logged (genes as rows and samples as columns)
sce	SingleCellExperiment; normalized counts and associated metadata.
do	character; choice of methods to be used for imputation. Currently supported methods are 'Baseline', 'DrImpute' and 'Network'. Not case-sensitive. Can include one or more methods. Non- supported methods will be ignored.
write	logical; write intermediary and imputed objects to files?
train.ratio	numeric; ratio of samples to be used for training
train.only	logical; if TRUE define only a training dataset, if FALSE writes and returns both training and validation sets (defaults to TRUE)
mask.ratio	numeric; ratio of samples to be masked per gene
outdir	character; path to directory where output files are written. Defaults to working directory
scale	integer; scaling factor to divide all expression levels by (defaults to 1)
pseudo.count	integer; pseudo-count to be added to expression levels to avoid log(0) (defaults to 1)
labels	character; vector specifying the cell type of each column of data
cell.clusters	integer; number of cell subpopulations
drop_thre	numeric; between 0 and 1 specifying the threshold to determine dropout values
type	A character specifying the type of values in the expression matrix. Can be 'count' or 'TPM'
cores	integer; number of cores used for parallel computation
BPPARAM	parallel back-end to be used during parallel computation. See <a href="#">BiocParallelParam-class</a> .
net.coef	matrix; network coefficients. Please provide if you don't want to use ADImpute's network model. Must contain one first column 'O' accounting for the intercept of the model and otherwise be an adjacency matrix with hgnc_symbols in rows and columns. Doesn't have to be squared. See <code>ADImpute::demo_net</code> for a small example.

<code>net.implementation</code>	character; either 'iteration', for an iterative solution, or 'pseudoinv', to use Moore-Penrose pseudo-inversion as a solution. 'pseudoinv' is not advised for big data.
<code>tr.length</code>	matrix with at least 2 columns: 'hgnc_symbol' and 'transcript_length'
<code>bulk</code>	vector of reference bulk RNA-seq, if available (average across samples)
<code>...</code>	additional parameters to pass to network-based imputation

### Details

For each gene, a fraction (`mask.ratio`) of the quantified expression values are set to zero and imputed according to 3 different methods: `scImpute`, `baseline` (average gene expression across all cells) or a network-based method. The imputation error is computed for each of the values in the original dataset that was set to 0, for each method. The method resulting in a lowest imputation error for each gene is chosen.

### Value

- if `sce` is provided: returns a `SingleCellExperiment` with the best performing method per gene stored as row-features. Access via `SingleCellExperiment::int_elementMetadata(sce)$ADImpute$methods`.
- if `sce` is not provided: returns a character with the best performing method in the training set for each gene

### See Also

[ImputeBaseline](#), [ImputeDrImpute](#), [ImputeNetwork](#)

### Examples

```
# Normalize demo data
norm_data <- NormalizeRPM(ADImpute::demo_data)
method_choice <- EvaluateMethods(norm_data, do = c('Baseline', 'DrImpute'),
cores = 2)
```

---

GetDropoutProbabilities

*Get dropout probabilities*

---

### Description

`GetDropoutProbabilities` computes dropout probabilities (probability of being a dropout that should be imputed rather than a true biological zero) using an adaptation of `scImpute`'s approach

### Usage

```
GetDropoutProbabilities(data, thre, cell.clusters, labels = NULL,
type = 'count', cores, BPPARAM, genelen = ADImpute::transcript_length)
```

**Arguments**

data	matrix; original data before imputation
thre	numeric; probability threshold to classify entries as biological zeros
cell.clusters	integer; number of cell subpopulations
labels	character; vector specifying the cell type of each column of data
type	A character specifying the type of values in the expression matrix. Can be 'count' or 'TPM'
cores	integer; number of cores used for parallel computation
BPPARAM	parallel back-end to be used during parallel computation. See <a href="#">BiocParallelParam-class</a> .
genelen	matrix with at least 2 columns: 'hgnc_symbol' and 'transcript_length'

**Details**

This function follows scImpute's model to distinguish between true biological zeros and dropouts, and is based on adapted code from the scImpute R package.

**Value**

matrix with same dimensions as data containing the dropout probabilities for the corresponding entries

---

HandleBiologicalZeros *Get dropout probabilities*

---

**Description**

GetDropoutProbabilities computes dropout probabilities (probability of being a dropout that should be imputed rather than a true biological zero) using an adaptation of scImpute's approach

**Usage**

```
HandleBiologicalZeros(data, imputed, thre = 0.5, cell.clusters,
  labels = NULL, type = 'count', cores = BiocParallel::bpworkers(BPPARAM),
  BPPARAM = BiocParallel::SnowParam(type = "SOCK"),
  genelen = ADImpute::transcript_length, prob.mat = NULL)
```

**Arguments**

data	matrix; original data before imputation
imputed	list; imputation results for considered methods
thre	numeric; between 0 and 1 specifying the threshold to determine dropout values
cell.clusters	integer; number of cell subpopulations
labels	character; vector specifying the cell type of each column of data
type	A character specifying the type of values in the expression matrix. Can be 'count' or 'TPM'
cores	integer; number of cores used for parallel computation
BPPARAM	parallel back-end to be used during parallel computation. See <a href="#">BiocParallelParam-class</a> .
genelen	matrix with at least 2 columns: 'hgnc_symbol' and 'transcript_length'
prob.mat	matrix with same dimensions as data containing the dropout probabilities for the corresponding entries

**Details**

This function follows scImpute's model to distinguish between true biological zeros and dropouts, and is based on adapted code from the scImpute R package.

**Value**

list with 2 components: `zerofiltered`, a list equivalent to `imputed` but with entries of imputed likely biological zeros set back to zero, and `dropoutprobabilities` matrix with same dimensions as data containing the dropout probabilities for the corresponding entries

---

Impute	<i>Dropout imputation using different methods</i>
--------	---

---

**Description**

Impute performs dropout imputation on normalized data, based on the choice of imputation methods.

**Usage**

```
Impute(data, sce = NULL, do = 'Ensemble', write = FALSE,
        outdir = getwd(), method.choice = NULL, scale = 1, pseudo.count = 1,
        labels = NULL, cell.clusters = 2, drop_thre = NULL, type = 'count',
        tr.length = ADImpute::transcript_length,
        cores = BiocParallel::bpworkers(BPPARAM),
        BPPARAM = BiocParallel::SnowParam(type = "SOCK"),
        net.coef = ADImpute::network.coefficients, net.implementation = 'iteration',
        bulk = NULL, true.zero.thr = NULL, prob.mat = NULL, ...)
```

**Arguments**

<code>data</code>	matrix; raw counts (genes as rows and samples as columns)
<code>sce</code>	SingleCellExperiment; normalized counts and associated metadata.
<code>do</code>	character; choice of methods to be used for imputation. Currently supported methods are 'Baseline', 'DrImpute', 'Network', and 'Ensemble'. Defaults to 'Ensemble'. Not case-sensitive. Can include one or more methods. Non-supported methods will be ignored.
<code>write</code>	logical; write intermediary and imputed objects to files?
<code>outdir</code>	character; path to directory where output files are written. Defaults to working directory
<code>method.choice</code>	character; best performing method in training data for each gene
<code>scale</code>	integer; scaling factor to divide all expression levels by (defaults to 1)
<code>pseudo.count</code>	integer; pseudo-count to be added to expression levels to avoid log(0) (defaults to 1)
<code>labels</code>	character; vector specifying the cell type of each column of data
<code>cell.clusters</code>	integer; number of cell subpopulations
<code>drop_thre</code>	numeric; between 0 and 1 specifying the threshold to determine dropout values

<code>type</code>	A character specifying the type of values in the expression matrix. Can be 'count' or 'TPM'
<code>tr.length</code>	matrix with at least 2 columns: 'hgnc_symbol' and 'transcript_length'
<code>cores</code>	integer; number of cores used for parallel computation
<code>BPPARAM</code>	parallel back-end to be used during parallel computation. See <a href="#">BiocParallelParam-class</a> .
<code>net.coef</code>	matrix; network coefficients. Please provide if you don't want to use ADImpute's network model. Must contain one first column 'O' accounting for the intercept of the model and otherwise be an adjacency matrix with hgnc_symbols in rows and columns. Doesn't have to be squared. See <code>ADImpute::demo_net</code> for a small example.
<code>net.implementation</code>	character; either 'iteration', for an iterative solution, or 'pseudoinv', to use Moore-Penrose pseudo-inversion as a solution. 'pseudoinv' is not advised for big data.
<code>bulk</code>	vector of reference bulk RNA-seq, if available (average across samples)
<code>true.zero.thr</code>	if set to NULL (default), no true zero estimation is performed. Set to numeric value between 0 and 1 for estimation. Value corresponds to the threshold used to determine true zeros: if the probability of dropout is lower than <code>true.zero.thr</code> , the imputed entries are set to zero.
<code>prob.mat</code>	matrix of the same size as data, filled with the dropout probabilities for each gene in each cell
<code>...</code>	additional parameters to pass to network-based imputation

## Details

Values that are 0 in data are imputed according to the best-performing methods indicated in `method.choice`. Currently supported methods are:

- **Baseline:** imputation with average expression across all cells in the dataset. See [ImputeBaseline](#).
- **Previously published approaches:** DrImpute and SAVER.
- **Network:** leverages information from a gene regulatory network to predicted expression of genes that are not quantified based on quantified interacting genes, in the same cell. See [ImputeNetwork](#).
- **Ensemble:** is based on results on a training subset of the data at hand, indicating which method best predicts the expression of each gene. These results are supplied via `method.choice`. Applies the imputation results of the best performing method to the zero entries of each gene.

If 'Ensemble' is included in `do`, `method.choice` has to be provided (use output from `EvaluateMethods()`). `Impute` can create a directory `imputation` containing the imputation results of all methods in `do`. If `true.zero.thr` is set, dropout probabilities are computed using `scImpute`'s framework. Expression values with dropout probabilities below `true.zero.thr` will be set back to 0 if imputed, as they likely correspond to true biological zeros (genes not expressed in cell) rather than technical dropouts (genes expressed but not captured). If `sce` is set, imputed values by the different methods are added as new assays to `sce`. Each assay corresponds to one imputation method. If `true.zero.thr` is set, only the values after filtering for biological zeros will be added. This is different from the output if `sce` is not set, where the original values before filtering and the dropout probability matrix are returned.

**Value**

- if `sce` is not set: returns a list of imputation results (normalized, log-transformed) for all selected methods in `do`. If `true.zero.thr` is defined, returns a list of 3 elements: 1) a list, `imputations`, containing the direct imputation results from each method; 2) a list, `zerofiltered`, containing the results of imputation in `imputations` after setting biological zeros back to zero; 3) a matrix, `dropoutprobabilities`, containing the dropout probability matrix used to set biological zeros.
- if `sce` is set: returns a `SingleCellExperiment` with new assays, each corresponding to one of the imputation methods applied. If `true.zero.thr` is defined, the assays will contain the results after imputation and setting biological zeros back to zero.

**See Also**

[EvaluateMethods](#), [ImputeBaseline](#), [ImputeDrImpute](#), [ImputeNetwork](#), [ImputeSAVER](#)

**Examples**

```
# Normalize demo data
norm_data <- NormalizeRPM(demo_data)
# Impute with particular method(s)
imputed_data <- Impute(do = 'Network', data = norm_data[,1:10],
net.coef = ADImpute::demo_net)
imputed_data <- Impute(do = 'Network', data = norm_data[,1:10],
net.implementation = 'pseudoinv', net.coef = ADImpute::demo_net)
```

---

ImputeBaseline

*Impute using average expression across all cells*

---

**Description**

ImputeBaseline imputes dropouts using gene averages across cells. Zero values are excluded from the mean computation.

**Usage**

```
ImputeBaseline(data, write = FALSE, ...)
```

**Arguments**

<code>data</code>	matrix with entries equal to zero to be imputed, normalized and log2-transformed (genes as rows and samples as columns)
<code>write</code>	logical; should a file with the imputation results be written?
<code>...</code>	additional arguments to <code>saveRDS</code>

**Value**

matrix; imputation results considering the average expression values of genes



---

ImputeDrImpute	<i>Use DrImpute</i>
----------------	---------------------

---

**Description**

ImputeDrImpute uses the DrImpute package for dropout imputation

**Usage**

```
ImputeDrImpute(data, write = FALSE)
```

**Arguments**

data	matrix with entries equal to zero to be imputed, normalized and log2-transformed (genes as rows and samples as columns)
write	logical; should a file with the imputation results be written?

**Value**

matrix; imputation results from DrImpute

**See Also**

[DrImpute](#)

---

ImputeNetParallel	<i>Network-based parallel imputation</i>
-------------------	--

---

**Description**

ImputeNetParallel implements network-based imputation in parallel

**Usage**

```
ImputeNetParallel(drop.mat, arranged, cores =
  BiocParallel::bpworkers(BPPARAM), type = 'iteration', max.iter = 50,
  BPPARAM = BiocParallel::SnowParam(type = "SOCK"))
#'
```

**Arguments**

drop.mat	matrix, logical; dropout entries in the data matrix (genes as rows and samples as columns)
arranged	list; output of <a href="#">ArrangeData</a>
cores	integer; number of cores used for parallel computation
type	character; either 'iteration', for an iterative solution, or 'pseudoinv', to use Moore-Penrose pseudo-inversion as a solution.
max.iter	numeric; maximum number of iterations for network imputation. Set to -1 to remove limit (not recommended)
BPPARAM	parallel back-end to be used during parallel computation. See <a href="#">BiocParallelParam-class</a> .

**Value**

matrix; imputation results incorporating network information

---

ImputeNetwork	<i>Network-based imputation</i>
---------------	---------------------------------

---

**Description**

Network-based imputation

**Usage**

```
ImputeNetwork(data, net.coef = NULL,
              cores = BiocParallel::bpworkers(BPPARAM),
              BPPARAM = BiocParallel::SnowParam(type = "SOCK"),
              type = 'iteration', write = FALSE, ...)
```

**Arguments**

data	matrix with entries equal to zero to be imputed, normalized and log2-transformed (genes as rows and samples as columns)
net.coef	matrix; network coefficients.
cores	integer; number of cores to use
BPPARAM	parallel back-end to be used during parallel computation. See <a href="#">BiocParallelParam-class</a> .
type	character; either 'iteration', for an iterative solution, or 'pseudoinv', to use Moore-Penrose pseudo-inversion as a solution.
write	logical; should a file with the imputation results be written?
...	additional arguments to <code>ImputeNetParallel</code>

**Details**

Imputes dropouts using a gene regulatory network trained on external data, as provided in `net.coef`. Dropout expression values are estimated from the expression of their predictor genes and the network coefficients.

**Value**

matrix; imputation results incorporating network information

**See Also**

[ImputeNetParallel](#)

---

ImputeNPDropouts      *Helper function to PseudoInverseSolution\_percell*

---

**Description**

ImputeNPDropouts computes the non-dropout- dependent solution of network imputation for each cell

**Usage**

```
ImputeNPDropouts(net, expr)
```

**Arguments**

net	matrix, logical; network coefficients for all dropout (to be imputed) genes that are predictive of the expression of other dropout genes
expr	numeric; vector of gene expression for all genes in the cell at hand

**Value**

vector; imputation results for the non-dropout-dependent genes

---

ImputePredictiveDropouts  
*Helper function to PseudoInverseSolution\_percell*

---

**Description**

ImputePredictiveDropouts applies Moore-Penrose pseudo-inversion to compute the dropout-dependent solution of network imputation for each cell

**Usage**

```
ImputePredictiveDropouts(net, thr = 0.01, expr)
```

**Arguments**

net	matrix, logical; network coefficients for all dropout (to be imputed) genes that are predictive of the expression of other dropout genes
thr	numeric; tolerance threshold to detect zero singular values
expr	numeric; vector of gene expression for all genes in the cell at hand

**Value**

vector; imputation results for the dropout-dependent genes

---

ImputeSAVER	<i>Use SAVER</i>
-------------	------------------

---

**Description**

ImputeSAVER uses the SAVER package for dropout imputation

**Usage**

```
ImputeSAVER(data, cores, try.mean = FALSE, write = FALSE)
```

**Arguments**

data	matrix with entries equal to zero to be imputed, normalized (genes as rows and samples as columns)
cores	integer; number of cores to use
try.mean	logical; whether to additionally use mean gene expression as prediction
write	logical; should a file with the imputation results be written?

**Value**

matrix; imputation results from SAVER

**See Also**

[saver](#)

---

MaskData	<i>Masking of entries for performance evaluation</i>
----------	--

---

**Description**

MaskData sets a portion (mask) of the non-zero entries of each row of data to zero

**Usage**

```
MaskData(data, write.to.file = FALSE, mask = .1)
```

**Arguments**

data	matrix; raw counts (genes as rows and samples as columns)
write.to.file	logical; should the output be written to a file?
mask	numeric; ratio of total non-zero samples to be masked per gene (defaults to .1)

**Details**

Sets a portion (mask) of the non-zero entries of each row of data to zero. Result is written to filename.

**Value**

matrix containing masked raw counts (genes as rows and samples as columns)

---

MaskerPerGene	<i>Helper mask function</i>
---------------	-----------------------------

---

**Description**

Helper mask function, per feature.

**Usage**

```
MaskerPerGene(x, rowmask)
```

**Arguments**

x	logical; data to mask
rowmask	numeric; number of samples to be masked per gene

**Value**

logical containing positions to mask

---

network.coefficients	<i>Transcriptome wide gene regulatory network</i>
----------------------	---

---

**Description**

Gene Regulatory Network used by ADImpute's Network imputation method. First column, 0, corresponds to the intercept of a gene-specific prediction model. The remaining rows and columns correspond to the adjacency matrix of the inferred network, where rows are target genes and columns are predictors. Genes are identified by their hgnc\_symbol.

**Usage**

```
network.coefficients
```

**Format**

```
dgCMatrix
```

---

NormalizeRPM                      *RPM normalization*

---

### Description

NormalizeRPM performs RPM normalization, with possibility to log the result

### Usage

```
NormalizeRPM(data, sce = NULL, log = FALSE, scale = 1,
pseudo.count = 1)
```

### Arguments

data	matrix; raw data (genes as rows and samples as columns)
sce	SingleCellExperiment; raw data
log	logical; log RPMs?
scale	integer; scale factor to divide RPMs by
pseudo.count	numeric; if log = TRUE, value to add to RPMs in order to avoid taking $\log(0)$

### Value

matrix; library size normalized data

### Examples

```
demo <- NormalizeRPM(ADImpute::demo_data)
```

---

NormalizeTPM                      *TPM normalization*

---

### Description

NormalizeTPM performs TPM normalization, with possibility to log the result

### Usage

```
NormalizeTPM(data, sce = NULL, tr_length = NULL, log = FALSE,
scale = 1, pseudo.count = 1)
```

### Arguments

data	matrix; raw data (genes as rows and samples as columns)
sce	SingleCellExperiment; raw data
tr_length	data.frame with at least 2 columns: 'hgnc_symbol' and 'transcript_length'
log	logical; log TPMs?
scale	integer; scale factor to divide TPMs by
pseudo.count	numeric; if log = T, value to add to TPMs in order to avoid taking $\log(0)$

**Details**

Gene length is estimated as the median of the lengths of all transcripts for each gene, as obtained from biomaRt. Genes for which length information cannot be found in biomaRt are dropped.

**Value**

matrix; normalized data (for transcript length and library size)

**Examples**

```
demo <- NormalizeTPM(ADImpute::demo_data)
```

---

PseudoInverseSolution\_percell

*Network-based parallel imputation - Moore-Penrose pseudoinversion*

---

**Description**

PseudoInverseSolution\_percell applies Moore-Penrose pseudo-inversion to compute the solution of network imputation for each cell

**Usage**

```
PseudoInverseSolution_percell(expr, net, drop_ind, thr = 0.01)
```

**Arguments**

expr	numeric; expression vector for cell at hand
net	matrix; network coefficients
drop_ind	logical; dropout entries in the cell at hand
thr	numeric; tolerance threshold to detect zero singular values

**Value**

matrix; imputation results incorporating network information

---

 ReadData
*Data read***Description**

ReadData reads data from raw input file (.txt or .csv)

**Usage**

```
ReadData(path, ...)
```

**Arguments**

path	character; path to input file
...	additional arguments to <code>data.table::fread()</code>

**Value**

matrix; raw counts (genes as rows and samples as columns)

---

ReturnChoice

*Wrapper for return of EvaluateMethods()***Description**

ReturnChoice Adjusts the output of EvaluateMethods to a character vector or a SingleCellExperiment object. Helper function to ADImpute.

**Usage**

```
ReturnChoice(sce, choice)
```

**Arguments**

sce	SingleCellExperiment; a SingleCellExperiment object if available; NULL otherwise
choice	character; best performing method in the training set for each gene

**Value**

- if sce is provided: returns a SingleCellExperiment with the best performing method per gene stored as row-features. Access via `SingleCellExperiment::int_elementMetadata(sce)$ADImpute$methods`.
- if sce is not provided: returns a character with the best performing method in the training set for each gene



---

ReturnOut	<i>Wrapper for return of Impute()</i>
-----------	---------------------------------------

---

**Description**

ReturnOut Adjusts the output of Impute to a list of matrices or a SingleCellExperiment object. Helper function to ADImpute.

**Usage**

```
ReturnOut(result, sce)
```

**Arguments**

result	list; imputation result
sce	SingleCellExperiment; a SingleCellExperiment object if available; NULL otherwise

**Value**

imputation results. A SingleCellExperiment if !is.null(sce), or a list with imputed results in matrix format otherwise.

---

SetBiologicalZeros	<i>Set biological zeros</i>
--------------------	-----------------------------

---

**Description**

SetBiologicalZeros sets some of the entries back to zero after dropout imputation, as they likely correspond to true biological zeros (genes not expressed in given cell)

**Usage**

```
SetBiologicalZeros(imputation, drop_probs, thre = .2, was_zero)
```

**Arguments**

imputation	matrix; imputed values
drop_probs	matrix; dropout probabilities for each entry in imputation. 0 means certain biological zero, while 1 means certain dropout to be imputed
thre	numeric; probability threshold to classify entries as biological zeros
was_zero	matrix; logical matrix: was the corresponding entry of imputation originally a zero?

**Details**

Entries which were originally zero and have dropout probability below thre are considered biological zeros and, if they were imputed, are set back to 0.

**Value**

matrix containing likely biological zeros set back to 0.

---

SplitData	<i>Selection of samples for training</i>
-----------	--

---

**Description**

SplitData selects a portion (ratio) of samples (columns in data) to be used as training set

**Usage**

```
SplitData(data, ratio = .7, write.to.file = FALSE, train.only = TRUE)
```

**Arguments**

data	matrix; raw counts (genes as rows and samples as columns)
ratio	numeric; ratio of the samples to be used for training
write.to.file	logical; should the output be written to a file?
train.only	logical; if TRUE define only a training dataset, if FALSE writes both training and validation sets (defaults to TRUE)

**Details**

Selects a portion (ratio) of samples (columns in data) to be used as training set and writes to file 'training\_raw.txt'.

**Value**

matrix containing raw counts (genes as rows and samples as columns)

---

transcript_length	<i>Table for transcript length calculations</i>
-------------------	---

---

**Description**

A data.frame to be used for transcript length computations. May be necessary upon TPM normalization, or as input to scImpute. All data was retrieved from biomaRt.

**Usage**

```
transcript_length
```

**Format**

A data.frame with 2 columns:

**hgnc\_symbol** Gene symbol identifier  
**transcript length** Length of transcript

---

WriteCSV	<i>Write csv file</i>
----------	-----------------------

---

**Description**

WriteCSV writes data to a comma-delimited output file

**Usage**

```
WriteCSV(object, file)
```

**Arguments**

object	R object to write
file	character; path to output file

**Value**

Returns NULL

**Examples**

```
file <- tempfile()
WriteCSV(iris, file = file)
```

---

WriteTXT	<i>Write txt file</i>
----------	-----------------------

---

**Description**

WriteTXT writes data to a tab-delimited output file

**Usage**

```
WriteTXT(object, file)
```

**Arguments**

object	R object to write
file	character; path to output file

**Value**

Returns NULL

**Examples**

```
file <- tempfile()
WriteTXT(iris, file = file)
```

# Index

- \* **datasets**
  - demo\_data, [9](#)
  - demo\_net, [10](#)
  - demo\_sce, [10](#)
  - network.coefficients, [21](#)
  - transcript\_length, [26](#)
- ArrangeData, [3](#), [17](#)
- CenterData, [3](#)
- CheckArguments\_Impute, [4](#)
- ChooseMethod, [4](#)
- Combine, [5](#)
- ComputeMSEGenewise, [5](#), [6](#)
- CreateArgCheck, [6](#)
- CreateTrainData, [7](#)
- DataCheck\_Matrix, [7](#)
- DataCheck\_Network, [8](#)
- DataCheck\_SingleCellExperiment, [8](#)
- DataCheck\_TrLength, [9](#)
- demo\_data, [9](#)
- demo\_net, [10](#)
- demo\_sce, [10](#)
- DrImpute, [17](#)
- EvaluateMethods, [11](#), [16](#)
- GetDropoutProbabilities, [12](#)
- HandleBiologicalZeros, [13](#)
- Impute, [14](#)
- ImputeBaseline, [12](#), [15](#), [16](#), [16](#)
- ImputeDrImpute, [12](#), [16](#), [17](#)
- ImputeNetParallel, [17](#), [18](#)
- ImputeNetwork, [12](#), [15](#), [16](#), [18](#)
- ImputeNPDropouts, [19](#)
- ImputePredictiveDropouts, [19](#)
- ImputeSAVER, [16](#), [20](#)
- MaskData, [20](#)
- MaskerPerGene, [21](#)
- network.coefficients, [21](#)
- NormalizeRPM, [22](#)
- NormalizeTPM, [22](#)
- PseudoInverseSolution\_perCell, [23](#)
- ReadData, [24](#)
- ReturnChoice, [24](#)
- ReturnOut, [25](#)
- saver, [20](#)
- SetBiologicalZeros, [25](#)
- SplitData, [26](#)
- transcript\_length, [26](#)
- WriteCSV, [27](#)
- WriteTXT, [27](#)