Package: scTenifoldKnk (via r-universe)

May 23, 2026

Type Package

Title In-Silico Knockout Experiments from Single-Cell Gene Regulatory Networks

Version 2.1.0

Author Zaoqu Liu [ctb, cre], Daniel Osorio [aut], Yan Zhong [aut, ctb], Guanxun Li [aut, ctb], Qian Xu [aut, ctb], Andrew Hillhouse [aut, ctb], Jingshu Chen [aut, ctb], Laurie Davidson [aut, ctb], Yanan Tian [aut, ctb], Robert Chapkin [aut, ctb], Jianhua Huang [aut, ctb], James Cai [aut, ctb, ths]

Maintainer Zaoqu Liu <liuzaoqu@163.com>

Description A workflow based on 'scTenifoldNet' to perform in-silico knockout experiments using single-cell RNA sequencing (scRNA-seq) data from wild-type (WT) control samples as input. First, the package constructs a single-cell gene regulatory network (scGRN) and knocks out a target gene from the adjacency matrix of the WT scGRN by setting the gene's outdegree edges to zero. Then, it compares the knocked out scGRN with the WT scGRN to identify differentially regulated genes, called virtual-knockout perturbed genes, which are used to assess the impact of the gene knockout and reveal the gene's function in the analyzed cells. This version includes C++ acceleration with Eigen library, cross-platform compatibility (macOS, Linux, Windows), and Seurat v4/v5 support.

URL https://zaoqu-liu.github.io/scTenifoldKnk/, https://github.com/Zaoqu-Liu/scTenifoldKnk

BugReports https://github.com/Zaoqu-Liu/scTenifoldKnk/issues

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.3

Imports Matrix, RSpectra, MASS, stats, parallel, methods, Rcpp, RcppEigen, utils

LinkingTo Rcpp, RcppEigen

Suggests testthat (>= 2.1.0), SeuratObject, knitr, rmarkdown

VignetteBuilder knitr

SystemRequirements GNU make

NeedsCompilation yes

Config/pak/sysreqs make

Repository https://zaoqu-liu.r-universe.dev

Date/Publication 2026-01-23 08:35:24 UTC

RemoteUrl https://github.com/Zaoqu-Liu/scTenifoldKnk

RemoteRef HEAD

RemoteSha 08118337f6fc22cca5a22c443ccb78ef920dfa3b

Contents

`dRegulation` *Differential regulation analysis*

Description

Computes differential regulation between aligned manifolds using distance-based statistics. Identifies genes whose regulatory relationships are most affected by the virtual knockout.

Usage

```
dRegulation(manifoldOutput, gKO)
```

Arguments

manifoldOutput	Manifold alignment output matrix (2*nGenes x d dimensions)
gKO	Character vector of knocked out gene name(s)

Details

The fold change (FC) is computed as the squared distance of each gene divided by the mean squared distance of all OTHER genes (excluding the knocked out gene). P-values are computed using chi-square distribution (df=1) and adjusted using Benjamini-Hochberg FDR correction.

Value

Data frame with columns: gene, distance, Z, FC, p.value, p.adj

---

dRegulationCpp *Differential regulation analysis with C++ acceleration*

Description

Differential regulation analysis with C++ acceleration

Usage

```
dRegulationCpp(manifoldOutputSEXP, geneNames, gKO)
```

Arguments

geneNames	Vector of gene names
gKO	Name of knocked out gene
manifoldOutput	Matrix with WT and KO gene embeddings (2*nGenes x d)

Value

DataFrame with differential regulation statistics

knockoutGeneCpp *Perform virtual knockout on network*

Description

Perform virtual knockout on network

Usage

```
knockoutGeneCpp(networkSEXP, geneIdx)
```

Arguments

geneIdx	Index of gene to knockout (0-based in C++, but R passes 1-based)
network	Gene regulatory network matrix

Details

Sets all outgoing edges from the knocked out gene to zero, meaning this gene can no longer regulate other genes.

Value

Network with gene knocked out (outgoing edges set to zero)

makeNetworksCpp *Build multiple gene regulatory networks with C++ acceleration*

Description

Build multiple gene regulatory networks with C++ acceleration

Usage

```
makeNetworksCpp(
  countMatrixSEXP,
  nNet = 10L,
  nCells = 500L,
  nComp = 3L,
  q = 0.9,
  scaleScores = TRUE,
  symmetric = FALSE,
  nThreads = 0L
)
```

Arguments

nNet	Number of networks to generate
nCells	Number of cells to subsample for each network
nComp	Number of principal components
q	Quantile threshold for edge filtering
scaleScores	Whether to scale network weights to [-1, 1]
symmetric	Whether to make networks symmetric
nThreads	Number of threads (ignored, kept for API compatibility)
countMatrix	Gene expression matrix (genes x cells)

Details

Each network is built from a random subsample of cells using principal component regression. Networks are returned as sparse matrices for memory efficiency.

Value

List of sparse network matrices

---

makeNetworksFast *Construction of multiple gene regulatory networks*

Description

Computes multiple gene regulatory networks from random subsamples of cells using principal component regression. Supports both sequential and parallel processing.

Usage

```
makeNetworksFast(
  X,
  nNet = 10,
  nCells = 500,
  nComp = 3,
  scaleScores = TRUE,
  symmetric = FALSE,
  q = 0.95,
  nCores = NULL,
  verbose = TRUE,
  seed = 1
)
```

Arguments

X	A filtered and normalized gene expression matrix with cells as columns and genes as rows.
nNet	An integer value. The number of networks to generate (default: 10).
nCells	An integer value. The number of cells to subsample for each network (default: 500).
nComp	An integer value. The number of principal components (default: 3). Must be >= 2 and < number of genes.
scaleScores	Logical. If TRUE, normalize weights so max absolute value is 1.
symmetric	Logical. If TRUE, return symmetric network matrices.
q	A decimal value between 0 and 1. Quantile threshold for edge filtering.
nCores	An integer value. Number of cores for parallel processing. Set to 1 for sequential. Default: detectCores() - 1.
verbose	Logical. If TRUE, print progress information.
seed	An integer value. Random seed for reproducibility. If NULL, no seed is set.

Details

Each network is constructed independently from a random subsample of cells, making the process embarrassingly parallel. The function automatically chooses sequential processing for small datasets where parallel overhead exceeds benefits.

Value

A list of nNet gene regulatory networks in dgCMatrix format.

Examples

```
library(scTenifoldKnk)

# Simulating dataset
nCells <- 2000
nGenes <- 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- matrix(X, ncol = nCells)
rownames(X) <- paste0("gene", 1:nGenes)

# Quality control
X <- X[rowSums(X) > 0, ]

# Generate networks (sequential for small data)
networks <- makeNetworksFast(
  X = X, nNet = 10, nCells = 500, nComp = 3,
  scaleScores = TRUE, symmetric = FALSE, q = 0.95
)
```

manifoldAlignment — *Performs non-linear manifold alignment of two gene regulatory networks*

Description

Build comparable low-dimensional features for two weight-averaged denoised single-cell gene regulatory networks using non-linear network embedding

Usage

```
manifoldAlignment(X, Y, d = 30, nCores = parallel::detectCores())
```

Arguments

X	A gene regulatory network
Y	A gene regulatory network
d	The dimension of the low-dimensional feature space
nCores	An integer value. Defines the number of cores to be used (currently not used, kept for compatibility)

Value

A low-dimensional projection for the two gene regulatory networks used as input

pcNetFast — *Gene regulatory network construction using principal component regression*

Description

Constructs a gene regulatory network using principal component regression. Uses efficient matrix operations and supports both accurate (leave-one-out) and approximate (global PCA) methods.

Usage

```
pcNetFast(
  X,
  nComp = 3,
  scaleScores = TRUE,
  symmetric = FALSE,
  q = 0,
  verbose = TRUE,
  nCores = 1,
  use_approximate = FALSE
)
```

Arguments

X	A filtered and normalized gene expression matrix with cells as columns and genes as rows.
nComp	An integer value. The number of principal components in PCA to generate the networks.
scaleScores	A boolean value (TRUE/FALSE), if TRUE, the weights will be normalized such that the maximum absolute value is 1.
symmetric	A boolean value (TRUE/FALSE), if TRUE, the weights matrix returned will be symmetric.
q	A decimal value between 0 and 1. Defines the cut-off threshold of top q percent relationships to be returned.
verbose	A boolean value (TRUE/FALSE), if TRUE, progress information is shown.
nCores	An integer value. Defines the number of cores for BLAS operations (not for parallelization).
use_approximate	Logical. If TRUE, uses approximate method (faster but less accurate). Default FALSE.

Details

When use_approximate=FALSE (default), the function performs leave-one-out principal component regression for each gene, which provides accurate network inference. When use_approximate=TRUE, it uses a global PCA approach which is faster but less accurate.

Value

A gene regulatory network in dgCMatrix format.

Examples

```
library(scTenifoldKnk)

# Simulating dataset
nCells <- 2000
nGenes <- 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- matrix(X, ncol = nCells)
rownames(X) <- paste0("gene", 1:nGenes)

# Quality control
X <- X[rowSums(X) > 0, ]

# Compute network (accurate, default)
network <- pcNetFast(X, nComp = 3, use_approximate = FALSE)

# Compute network (fast approximation)
network_fast <- pcNetFast(X, nComp = 3, use_approximate = TRUE)
```

`scQC` *Single-cell quality control*

Description

Performs quality control filtering on single-cell RNA-seq data based on library size, gene count, and mitochondrial content.

Usage

```
scQC(X, mtThreshold = 0.1, minLSize = 1000)
```

Arguments

`X`	A count matrix (genes x cells) or Seurat object (v4 or v5 compatible)
`mtThreshold`	Maximum mitochondrial read ratio threshold (0-1)
`minLSize`	Minimum library size threshold (total UMI counts per cell)

Details

Cells are filtered based on:

1. Library size >= minLSize
2. Number of detected genes within expected range (linear model prediction)
3. Mitochondrial proportion <= mtThreshold (if MT genes present)
4. Library size < 2x mean (removes potential doublets)

Value

Filtered count matrix or Seurat object (same type as input)

`scTenifoldKnk` *Virtual Knockout Experiment*

Description

Performs in-silico gene knockout experiments on single-cell gene regulatory networks

Usage

```
scTenifoldKnk(
  countMatrix,
  gKO = NULL,
  qc_mtThreshold = 0.1,
  qc_minLSize = 1000,
  nc_lambda = 0,
  nc_nNet = 10,
  nc_nCells = 500,
  nc_nComp = 3,
  nc_scaleScores = TRUE,
  nc_symmetric = FALSE,
  nc_q = 0.9,
  td_K = 3,
  td_maxIter = 1000,
  td_maxError = 1e-05,
  td_nDecimal = 3,
  ma_nDim = 2,
  nCores = NULL,
  verbose = TRUE
)
```

Arguments

countMatrix	Gene expression count matrix (genes x cells)
gKO	Gene(s) to knockout
qc_mtThreshold	Maximum mitochondrial read ratio
qc_minLSize	Minimum library size
nc_lambda	Lambda parameter for strict directionality
nc_nNet	Number of networks to generate
nc_nCells	Number of cells per network
nc_nComp	Number of principal components
nc_scaleScores	Whether to scale network scores
nc_symmetric	Whether to make network symmetric
nc_q	Quantile threshold for edge filtering
td_K	Number of tensor components
td_maxIter	Maximum tensor decomposition iterations
td_maxError	Tensor decomposition error tolerance
td_nDecimal	Number of decimal places
ma_nDim	Number of manifold dimensions
nCores	Number of cores for parallel processing
verbose	Whether to print progress information

Value

A list containing tensor networks, manifold alignment, and differential regulation results

Examples

```
# Loading single-cell data
scRNAseq <- system.file("single-cell/example.csv", package = "scTenifoldKnk")
scRNAseq <- read.csv(scRNAseq, row.names = 1)

# Running scTenifoldKnk
result <- scTenifoldKnk(countMatrix = scRNAseq, gKO = "G100", qc_minLSize = 0)
```

---

sparseToDenseTranspose *Convert sparse matrix to dense and transpose*

Description

Convert sparse matrix to dense and transpose

Usage

```
sparseToDenseTranspose(XSEXP)
```

Arguments

X	Sparse matrix

Value

Dense transposed matrix

---

strictDirection *Apply directional penalty to network*

Description

Applies lambda penalty to enforce edge directionality. For each pair of genes (i,j), keeps only the stronger edge direction.

Usage

```
strictDirection(X, lambda = 1)
```

Arguments

X	Network matrix (can be dense or sparse)
lambda	Penalty parameter between 0 and 1. lambda=1 means full directionality (keep only stronger edge), lambda=0 means no change.

Details

For each pair (i,j), if |X[i,j]| < |X[j,i]|, then X[i,j] is set to 0. This enforces that regulatory relationships are directional - only the stronger direction is retained.

Value

Modified network matrix as sparse dgCMatrix

---

strictDirectionCpp *Apply directional penalty to network*

Description

Apply directional penalty to network

Usage

```
strictDirectionCpp(XSEXP, lambda = 1)
```

Arguments

lambda	Directionality parameter (0 to 1)
X	Network adjacency matrix

Details

For each pair (i,j), if |X[i,j]| < |X[j,i]|, then X[i,j] is set to 0. The final result is: (1-lambda)*X + lambda*S where S is the directional matrix.

Value

Directional network matrix

`tensorDecomposition` *Performs CANDECOMP/PARAFAC (CP) Tensor Decomposition*

Description

Generate weight-averaged denoised gene regulatory networks using CANDECOMP/PARAFAC (CP) Tensor Decomposition with Alternating Least Squares.

Usage

```
tensorDecomposition(
  xList,
  yList = NULL,
  nDecimal = 1,
  K = 5,
  maxError = 1e-05,
  maxIter = 1000,
  useCpp = TRUE
)
```

Arguments

`xList`	A list of gene regulatory networks (sparse or dense matrices)
`yList`	Optional. A second list of gene regulatory networks for comparison
`nDecimal`	An integer value for decimal places in output (default: 1)
`K`	The CP rank (number of components, default: 5)
`maxError`	Convergence threshold for relative Frobenius norm (default: 1e-5)
`maxIter`	Maximum number of ALS iterations (default: 1000)
`useCpp`	Logical. If TRUE (default), use faster C++ implementation

Details

The function stacks input networks into a 3-way tensor and performs CP decomposition to extract the underlying low-rank structure. The reconstructed network is a weighted average that reduces noise.

Value

A list containing denoised network(s): $X and optionally $Y

`tensorDecompositionCpp` *CP Tensor Decomposition with C++ acceleration*

Description

CP Tensor Decomposition with C++ acceleration

Usage

```
tensorDecompositionCpp(
  networkList,
  K = 3L,
  maxIter = 1000L,
  maxError = 1e-05,
  nDecimal = 3L
)
```

Arguments

`networkList`	List of network matrices (can be sparse or dense)
`K`	Rank of CP decomposition (number of components)
`maxIter`	Maximum number of ALS iterations
`maxError`	Convergence threshold (relative Frobenius norm)
`nDecimal`	Decimal places for rounding output (0 = no rounding)

Details

Uses Alternating Least Squares (ALS) to compute CP decomposition of the 3-way tensor formed by stacking network matrices. The reconstructed network is a weighted average of the network slices.

Value

List containing: - X: reconstructed average network - A, B, C: factor matrices - iterations: number of iterations performed - error: final reconstruction error

Index