Package: symTensor (via r-universe)

May 14, 2026

Type Package

Title Symmetric Matrix and Tensor Decomposition

Version 0.1.0

Description Provides symmetric matrix and tensor operations and decomposition algorithms including symmetric NMF (symNMF), PageRank, Label Propagation, Higher-order Power Method, and TOPHITS. Designed to work with 'rTensor' objects. Methods are described in Kuang et al. (2012) <doi:10.1137/1.9781611972825.10>, Kolda and Mayo (2011) <doi:10.1137/100801482>, and Kolda and Bader (2006) <doi:10.1137/1.9781611972764.26>.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.5.0)

Imports rTensor, methods, stats

Suggests testthat

RoxygenNote 7.3.2

NeedsCompilation no

Author Koki Tsuyuzaki [aut, cre]

Maintainer Koki Tsuyuzaki <k.t.the-answer@hotmail.co.jp>

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

Date/Publication 2026-05-14 10:38:09 UTC

RemoteUrl https://github.com/cran/symTensor

RemoteRef HEAD

RemoteSha a0a8cb0e6939660c894948c156be9117ef01a411

Contents

`HigherOrderPower` *Higher-order Power Method for Symmetric Tensor Decomposition*

Description

Decomposes a symmetric tensor into a sum of rank-1 symmetric terms using the higher-order power method (S-HOPM) with deflation.

Usage

```
HigherOrderPower(
  x,
  R = 1L,
  max_iter = 1000L,
  tol = 1e-08,
  init = c("random", "svd")
)
```

Arguments

`x`	A symmetric rTensor Tensor object (all modes must be equal)
`R`	Number of rank-1 components to extract (default: 1)
`max_iter`	Maximum iterations per component (default: 1000)
`tol`	Convergence tolerance (default: 1e-8)
`init`	Initialization method: `"random"` (default) or `"svd"`

Details

For a d-th order symmetric tensor $T \in R^{n \times n \times \cdots \times n}$, finds $T \approx \sum_{r=1}^{R} \lambda_r v_r \otimes v_r \otimes \cdots \otimes v_r$.

Value

A list with components:

lambdas Numeric vector of length R with eigenvalues

vectors Matrix (n x R) where each column is a unit eigenvector

residual Residual tensor after deflation

converged Logical vector of length R

iters Integer vector of iterations per component

References

Kolda, T. G., & Mayo, J. R. (2011). Shifted Power Method for Computing Tensor Eigenpairs. SIAM Journal on Matrix Analysis and Applications.

De Lathauwer, L., De Moor, B., & Vandewalle, J. (2000). On the Best Rank-1 and Rank-(R1, R2, ..., RN) Approximation of Higher-Order Tensors. SIAM Journal on Matrix Analysis and Applications.

Examples

```
library(rTensor)
# Create a symmetric 3-mode tensor
n <- 5
v1 <- rnorm(n); v1 <- v1 / sqrt(sum(v1^2))
# T = 3.0 * v1 o v1 o v1
arr <- 3.0 * outer(outer(v1, v1), v1)
tnsr <- as.tensor(arr)
result <- HigherOrderPower(tnsr, R = 1)
result$lambdas  # should be close to 3.0
```

---

isSymmetricTensor *Check if a matrix or tensor is symmetric*

Description

For a matrix, checks if A[i,j] == A[j,i] for all i,j. For a tensor, checks if the tensor is invariant under all permutations of its indices (i.e., fully symmetric).

Usage

```
isSymmetricTensor(x, tol = 1e-10)
```

Arguments

x	A matrix or rTensor Tensor object
tol	Numeric tolerance for comparison (default: 1e-10)

Value

Logical indicating whether x is symmetric

Examples

```
# Matrix
A <- matrix(c(1,2,2,3), 2, 2)
isSymmetricTensor(A)  # TRUE

# 3-mode symmetric tensor
library(rTensor)
arr <- array(0, dim = c(3, 3, 3))
```

```
for (i in 1:3) for (j in 1:3) for (k in 1:3) {
  arr[i, j, k] <- i + j + k
}
isSymmetricTensor(as.tensor(arr))  # TRUE
```

`LabelPropagation` *Label Propagation via PageRank*

Description

Semi-supervised label propagation on a graph. Uses Personalized PageRank internally to propagate labels from seed nodes to the rest of the graph.

Usage

```
LabelPropagation(A, seeds, damping = 0.85, max_iter = 1000L, tol = 1e-08)
```

Arguments

`A`	Square non-negative adjacency/weight matrix (n x n)
`seeds`	Named integer vector or factor. Names correspond to node indices (1-based), values are the class labels. Unlabeled nodes will be classified.
`damping`	Damping factor passed to PageRank (default: 0.85)
`max_iter`	Maximum iterations for PageRank (default: 1000)
`tol`	Convergence tolerance for PageRank (default: 1e-8)

Value

A list with components:

labels Integer vector of predicted labels for all nodes

scores Matrix (n x K) of PageRank-based scores per class

classes Unique class labels

Examples

```
# 6-node graph with 2 communities
A <- matrix(0, 6, 6)
A[1,2] <- A[2,1] <- 1; A[1,3] <- A[3,1] <- 1; A[2,3] <- A[3,2] <- 1
A[4,5] <- A[5,4] <- 1; A[4,6] <- A[6,4] <- 1; A[5,6] <- A[6,5] <- 1
A[3,4] <- A[4,3] <- 0.5  # weak bridge
seeds <- c("1" = 1, "6" = 2)  # node 1 -> class 1, node 6 -> class 2
result <- LabelPropagation(A, seeds)
result$labels
```

PageRank *(Personalized) PageRank*

Description

Computes the PageRank vector for a given adjacency or transition matrix using power iteration.

Usage

```
PageRank(
  A,
  damping = 0.85,
  personalization = NULL,
  max_iter = 1000L,
  tol = 1e-08
)
```

Arguments

A	Square non-negative matrix (adjacency or weight matrix). Columns are normalized internally to form a column-stochastic matrix.
damping	Damping factor (default: 0.85). Probability of following a link.
personalization	Optional personalization vector of length nrow(A). If NULL, uniform distribution is used (standard PageRank). Personalized PageRank biases the random jumps toward this distribution.
max_iter	Maximum number of power iterations (default: 1000)
tol	Convergence tolerance on L1 norm change (default: 1e-8)

Value

A list with components:

vector PageRank vector (sums to 1)

iter Number of iterations until convergence

converged Logical, whether the algorithm converged

Examples

```
# Simple web graph
A <- matrix(c(0,1,1,0, 1,0,0,1, 0,1,0,1, 1,0,1,0), 4, 4)
pr <- PageRank(A)
pr$vector
```

Symmetrize *Symmetrize a matrix or tensor*

Description

For a matrix, computes (A + A^T) / 2. For a tensor, averages over all permutations of indices.

Usage

```
Symmetrize(x)
```

Arguments

x	A matrix or rTensor Tensor object (must have equal modes)

Value

Symmetrized matrix or Tensor

Examples

```
A <- matrix(c(1, 2, 3, 4), 2, 2)
Symmetrize(A)

library(rTensor)
arr <- array(rnorm(27), dim = c(3, 3, 3))
S <- Symmetrize(as.tensor(arr))
isSymmetricTensor(S)  # TRUE
```

symNMF *Symmetric Non-negative Matrix Factorization (symNMF)*

Description

Decomposes a symmetric non-negative matrix $\Omega \approx MSM^\top$ using multiplicative update (MU) rules with Beta-divergence.

Usage

```
symNMF(
  Omega,
  K,
  model = c("MSM", "MM"),
  Beta = 2,
  initM = NULL,
  initS = NULL,
```

```
  lambda = 0,
  num.iter = 100L,
  thr = 1e-10,
  verbose = FALSE
)
```

Arguments

`Omega`	Symmetric non-negative matrix (N x N)
`K`	Number of components (rank)
`model`	Character, either `"MSM"` (default) or `"MM"`
`Beta`	Beta parameter for Beta-divergence (default: 2). 2 = Frobenius, 1 = KL, 0 = IS.
`initM`	Initial M matrix (N x K). If NULL, random non-negative initialization.
`initS`	Initial S matrix (K x K). If NULL, random non-negative initialization. Ignored when `model = "MM"`.
`lambda`	Regularization parameter for $\lvert\det(S)\rvert$ penalty (default: 0). Only used with `model = "MSM"`.
`num.iter`	Maximum number of iterations (default: 100)
`thr`	Convergence threshold on relative change (default: 1e-10)
`verbose`	Logical, print iteration info (default: FALSE)

Details

Two models are supported:

- **MSM**: $\Omega \approx MSM^\top$ where M is N x K and S is K x K
- **MM**: $\Omega \approx MM^\top$ where M is N x K (special case with S = I)

The loss function is the Beta-divergence $D_\beta(\Omega \| \hat{\Omega})$:

- $\beta = 2$: Frobenius (Euclidean) distance
- $\beta = 1$: Kullback-Leibler divergence
- $\beta = 0$: Itakura-Saito divergence
- Other values of β interpolate/extrapolate these cases

Value

A list with components:

M Factor matrix (N x K)

S Coefficient matrix (K x K). Identity matrix when `model = "MM"`.

RecError Vector of reconstruction errors at each iteration

RelChange Vector of relative changes at each iteration

converged Logical

iter Number of iterations performed

References

Kuang, D., Park, H., & Ding, C. H. Q. (2012). Symmetric Nonnegative Matrix Factorization for Graph Clustering. SDM 2012.

Fevotte, C. & Idier, J. (2011). Algorithms for Nonnegative Matrix Factorization with the Beta-Divergence. Neural Computation, 23(9).

Examples

```
set.seed(42)
M_true <- matrix(c(rep(c(1,0), 5), rep(c(0,1), 5)), 10, 2)
Omega <- M_true %*% t(M_true) + matrix(runif(100, 0, 0.1), 10, 10)
Omega <- (Omega + t(Omega)) / 2

# Frobenius (Beta=2, default)
res_fro <- symNMF(Omega, K = 2, Beta = 2)

# KL divergence (Beta=1)
res_kl <- symNMF(Omega, K = 2, Beta = 1)

# Itakura-Saito (Beta=0)
res_is <- symNMF(Omega, K = 2, Beta = 0)
```

`TOPHITS` *TOPHITS: Tensor HITS Algorithm*

Description

Computes hub and authority scores for higher-order link analysis using Tucker decomposition of a tensor. Generalization of HITS (Hyperlink-Induced Topic Search) to tensors.

Usage

```
TOPHITS(x, R = 2L, max_iter = 100L, tol = 1e-08)
```

Arguments

`x`	An rTensor Tensor object (2-mode or 3-mode)
`R`	Integer or integer vector specifying the number of components per mode. If scalar, same rank is used for all modes.
`max_iter`	Maximum iterations for Tucker decomposition (default: 100)
`tol`	Convergence tolerance (default: 1e-8)

Details

For a 3-mode tensor $T \in R^{I \times J \times K}$, TOPHITS computes Tucker decomposition $T \approx C \times_1 U_1 \times_2 U_2 \times_3 U_3$ and derives hub/authority scores from the factor matrices.

For a symmetric tensor (all modes equal), a single set of scores is returned.

Value

A list with components:

scores List of score matrices, one per mode (each n_i x R_i). For symmetric tensors, a single matrix.

core Core tensor from Tucker decomposition

rankings List of integer vectors giving node rankings per mode (ordered by dominant score)

decomp Full Tucker decomposition result from rTensor

References

Kolda, T. G., & Bader, B. W. (2006). The TOPHITS Model for Higher-Order Web Link Analysis. Workshop on Link Analysis, Counterterrorism and Security.

Examples

```
library(rTensor)
# Create a 3-mode tensor
arr <- array(abs(rnorm(125)), dim = c(5, 5, 5))
tnsr <- as.tensor(arr)
result <- TOPHITS(tnsr, R = 2)
result$rankings
```

Index