# Package ‘RcppPlanc’

March 25, 2025

**Type** Package

**Title** Parallel Low-Rank Approximation with Nonnegativity Constraints

**Version** 2.0.0

**Date** 2025-03-21f

**Description** 'Rcpp' bindings for 'PLANC', a highly parallel
and extensible NMF/NTF (Non-negative Matrix/Tensor Factorization) library.
Wraps algorithms described in
Kannan et. al (2018) <doi:10.1109/TKDE.2017.2767592> and
Eswar et. al (2021) <doi:10.1145/3432185>.
Implements algorithms described in
Welch et al. (2019) <doi:10.1016/j.cell.2019.05.006>,
Gao et al. (2021) <doi:10.1038/s41587-021-00867-x>, and
Kriebel & Welch (2022) <doi:10.1038/s41467-022-28431-4>.

**Encoding** UTF-8

**Imports** methods, Rcpp, Matrix, hdf5r.Extra

**Suggests** knitr, withr, rmarkdown, testthat (>= 3.0.0)

**LinkingTo** Rcpp, RcppArmadillo, RcppProgress

**License** GPL (>= 2)

**RoxygenNote** 7.3.2

**SystemRequirements** C++17, cmake >= 3.21.0, hdf5, git, v8, patch,
gnumake, hwloc, GNU make

**Config/testthat/edition** 3

**Depends** R (>= 3.5)

**LazyData** true

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** Andrew Robbins [aut, cre] (<https://orcid.org/0009-0001-7961-7489>),
Yichen Wang [aut],
Joshua Welch [cph] (<https://orcid.org/0000-0002-5869-2391>),
Ramakrishnan Kannan [cph] (<https://orcid.org/0000-0002-5852-4806>),
UT-Batelle [cph] (The original PLANC code)

**Maintainer** Andrew Robbins <robbiand@umich.edu>

**Repository** CRAN

**Date/Publication** 2025-03-25 15:50:13 UTC

# Contents



---

`bppnnls` *Block Principal Pivoted Non-Negative Least Squares*

---

### Description

Use the BPP algorithm to get the nonnegative least squares solution. Regular NNLS problem is described as optimizing $\min_{x \geq 0} ||CX - B||_F^2$ where $C$ and $B$ are given and $X$ is to be solved. `bppnnls` takes $C$ and $B$ as input. `bppnnls_prod` takes $C^{\mathsf{T}}C$ and $C^{\mathsf{T}}B$ as input to directly go for the intermediate step of BPP algorithm. This can be useful when the dimensionality of $C$ and $B$ is large while pre-calculating $C^{\mathsf{T}}C$ and $C^{\mathsf{T}}B$ is cheap.

### Usage

```
bppnnls(C, B, nCores = 2L)

bppnnls_prod(CtC, CtB, nCores = 2L)
```

### Arguments

| | |
|---|---|
| `C` | Input dense $C$ matrix |
| `B` | Input $B$ matrix of either dense or sparse form |
| `nCores` | The number of parallel tasks that will be spawned. Default 2 |
| `CtC` | The $C^{\mathsf{T}}C$ matrix, see description. |
| `CtB` | The $C^{\mathsf{T}}B$ matrix, see description. |

## Value

The calculated solution matrix in dense form.

## Examples

```
set.seed(1)
C <- matrix(rnorm(250), nrow = 25)
B <- matrix(rnorm(375), nrow = 25)
res1 <- bppnnls(C, B)
dim(res1)
res2 <- bppnnls_prod(t(C) %*% C, t(C) %*% B)
all.equal(res1, res2)
```

---

`ctrl.sparse` *Example single-cell transcriptomic data in sparse form*

---

## Description

The two datasets, namingly `ctrl.sparse` and `stim.sparse`, are single-cell transcriptomic data preprocessed and subsampled from the study of Hyun Min Kang and et al., Nat Biotech., 2018. The raw datasets were two sparse matrices of integer values indicating the counts of genes (rows) per cell (columns). We normalized each column of both matrices by its library size (sum), and selected common variable genes across the datasets. Finally, we scaled the genes without centering them, in order to keep the non-negativity. The processed datasets were then randomly subsampled for a minimal example.

## Usage

```
ctrl.sparse

stim.sparse
```

## Format

An object of class `dgCMatrix` with 173 rows and 300 columns.

An object of class `dgCMatrix` with 173 rows and 300 columns.

## Source

https://www.nature.com/articles/nbt.4042 dim.H5SpMat *Retrieve the dimension of H5SpMat argument list*

## Description

Retrieve the dimension of H5SpMat argument list

## Usage

```
## S3 method for class 'H5SpMat'
dim(x)

## S3 replacement method for class 'H5SpMat'
dim(x) <- value
```

## Arguments

| | |
|---|---|
| x | H5SpMat argument list object |
| value | Numeric vector of two, for number of rows and number of columns. |

## Value

Retriever returns a vector of two (nrow and ncol), setter sets the value of that in the argument list.

## Examples

```
h <- H5SpMat(system.file("extdata/ctrl_sparse.h5", package = "RcppPlanc"),
             "data", "indices", "indptr", 173, 300)
dim(h)
nrow(h)
ncol(h)
dim(h) <- c(200, 200)
h
```

format.H5Mat *Prepare character information of a H5Mat object*

## Description

Prepare character information of a H5Mat object

## Usage

```
## S3 method for class 'H5Mat'
format(x, ...)
```

### Arguments

| | |
|---|---|
| x | H5Mat argument list object |
| ... | Not used. |

### Value

A character scalar of the displayed message

### Examples

```
h <- H5Mat(system.file("extdata/ctrl_dense.h5", package = "RcppPlanc"),
           "data")
format(h)
```

---

## format.H5SpMat    *prepare character information of a H5SpMat object*

---

### Description

prepare character information of a H5SpMat object

### Usage

```
## S3 method for class 'H5SpMat'
format(x, ...)
```

### Arguments

| | |
|---|---|
| x | H5SpMat argument list object |
| ... | Not used. |

### Value

A character scalar of the displayed message

### Examples

```
h <- H5SpMat(system.file("extdata/ctrl_sparse.h5", package = "RcppPlanc"),
             "data", "indices", "indptr", 173, 300)
format(h)
```

H5Mat *Argument list object for using a dense matrix stored in HDF5 file*

## Description

For running inmf, onlineINMF or uinmf with dense matrix stored in HDF5 file, users will need to construct an argument list for the filename of the HDF5 file as well as the path in the file storing the matrix. H5Mat is provided as an instructed constructor. Meanwhile, since the INMF functions require that all datasets should be of the same type, as.H5Mat is provided for writing in-memory data into a new HDF5 file on disk and returning the constructed argument list.

## Usage

```
H5Mat(filename, dataPath)

as.H5Mat(x, filename, dataPath = "data", overwrite = FALSE, ...)

## S3 method for class 'matrix'
as.H5Mat(x, filename, dataPath = "data", overwrite, ...)

## S3 method for class 'dgCMatrix'
as.H5Mat(x, filename, dataPath = "data", overwrite = FALSE, ...)

## Default S3 method:
as.H5Mat(x, filename, dataPath = "data", ...)
```

## Arguments

| | |
|---|---|
| filename | Filename of the HDF5 file |
| dataPath | Path in the HDF5 file that points to a 2D dense matrix. Default "data" when using as.H5Mat. |
| x | For as.H5Mat, matrix of either dense or sparse type to be written; for print, a H5Mat argument list object |
| overwrite | Logical, whether to overwrite the file if already exists at the given path. Default FALSE. |
| ... | Passed down to hdf5r.Extra::h5Write |

## Value

H5Mat object, indeed a list object.

## Examples

```
if (require("withr")) {
H5MatEx <- function(){
withr::local_dir(withr::local_tempdir())
```

```
h <- H5Mat(system.file("extdata/ctrl_dense.h5", package = "RcppPlanc"),
           "data")
print(h)

library(Matrix)
ctrl.dense <- as.matrix(ctrl.sparse)
h1 <- as.H5Mat(ctrl.dense, "ctrl_from_dense_to_dense.h5",
               dataPath = "data")
h1
h2 <- as.H5Mat(ctrl.sparse, "ctrl_from_sparse_to_dense.h5",
               dataPath = "data")
}
H5MatEx()
}
```

---

`H5SpMat` *Argument list object for using a sparse matrix stored in HDF5 file*

---

## Description

For running `inmf`, `onlineINMF` or `uinmf` with sparse matrix stored in HDF5 file, users will need to construct an argument list for the filename of the HDF5 file as well as the paths in the file storing the arrays that construct the CSC (compressed sparse column) matrix. `H5SpMat` is provided as an instructed constructor. Meanwhile, since the INMF functions require that all datasets should be of the same type, `as.H5SpMat` is provided for writing in-memory data into a new HDF5 file on disk and returning the constructed argument list.

## Usage

```
H5SpMat(filename, valuePath, rowindPath, colptrPath, nrow, ncol)

as.H5SpMat(x, filename, dataPath, overwrite = FALSE)

## S3 method for class 'matrix'
as.H5SpMat(x, filename, dataPath = "", overwrite = FALSE)

## S3 method for class 'dgCMatrix'
as.H5SpMat(x, filename, dataPath = "", overwrite = FALSE)

## Default S3 method:
as.H5SpMat(x, filename, dataPath = "", overwrite = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `filename` | Filename of the HDF5 file |
| `valuePath` | Path in the HDF5 file that points to a 1D array storing the non-zero values of the sparse matrix. Default `"data"` when using `as.H5SpMat`. |

| | |
|---|---|
| `rowindPath` | Path in the HDF5 file that points to a 1D integer array storing the row indices of non-zero values in each column of the sparse matrix. Default `"indices"` when using `as.H5SpMat`. |
| `colptrPath` | Path in the HDF5 file that points to a 1D integer array storing the number of non-zero values in each column of the sparse matrix. Default `"indptr"` when using `as.H5SpMat`. |
| `nrow, ncol` | Integer, the true dimensionality of the sparse matrix. |
| `x` | For `as.H5SpMat`, matrix of either dense or sparse type to be written; for `print`, a `H5SpMat` argument list object. |
| `dataPath` | For `as.H5SpMat` methods, the H5Group name for the sparse matrix. Default `""`. |
| `overwrite` | Logical, whether to overwrite the file if already exists at the given path. Default `FALSE`. |
| `...` | not used |

## Value

H5SpMat object, indeed a list object.

## Examples

```
if (require("withr")) {
  H5SpMatEx <- function() {
    withr::local_dir(withr::local_tempdir())
    h <- H5SpMat(system.file("extdata/ctrl_sparse.h5", package = "RcppPlanc"),
                 "data", "indices", "indptr", 173, 300)
    dim(h)

    library(Matrix)
    ctrl.dense <- as.matrix(ctrl.sparse)
    h1 <- as.H5SpMat(ctrl.sparse, "ctrl_from_sparse_to_sparse.h5", "matrix")
    h1
    h2 <- as.H5SpMat(ctrl.dense, "ctrl_from_dense_to_sparse.h5", "matrix")
    h2
  }
  H5SpMatEx()
}
```

---

`inmf` *Perform Integrative Non-negative Matrix Factorization*

---

## Description

Performs integrative non-negative matrix factorization (iNMF) (J.D. Welch, 2019) to return factorized $H$, $W$, and $V$ matrices. The objective function is stated as

$$\arg\min_{H\geq 0,W\geq 0,V\geq 0}\sum_{i}^{d}||E_i-(W+V_i)Hi||_F^2+\lambda\sum_{i}^{d}||V_iH_i||_F^2$$

where $E_i$ is the input non-negative matrix of the $i$'th dataset, $d$ is the total number of datasets. $E_i$ is of size $m \times n_i$ for $m$ features and $n_i$ sample points, $H_i$ is of size $k \times n_i$, $V_i$ is of size $m \times k$, and $W$ is of size $m \times k$.

`inmf` optimizes the objective with ANLS strategy, while `onlineINMF` optimizes the same objective with an online learning strategy.

## Usage

```
inmf(
  objectList,
  k = 20,
  lambda = 5,
  niter = 30,
  nCores = 2,
  Hinit = NULL,
  Vinit = NULL,
  Winit = NULL,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `objectList` | list of input datasets. List elements should all be of the same class. Viable classes include: matrix, dgCMatrix, H5Mat, H5SpMat. |
| `k` | Integer. Inner dimensionality to factorize the datasets into. Default `20`. |
| `lambda` | Regularization parameter. Larger values penalize dataset-specific effects more strongly (i.e. alignment should increase as `lambda` increases). Default `5`. |
| `niter` | Integer. Total number of block coordinate descent iterations to perform. Default `30`. |
| `nCores` | The number of parallel tasks that will be spawned. Default `2` |
| `Hinit` | Initial values to use for $H$ matrices. A list object where each element is the initial $H$ matrix of each dataset. Each should be dense matrix of size $n_i \times k$. Default `NULL`. |
| `Vinit` | Similar to `Hinit`, but each should be of size $m \times k$. |
| `Winit` | Initial values to use for $W$ matrix. A matrix object of size $m \times k$. Default `NULL`. |
| `verbose` | Logical scalar. Whether to show information and progress. Default `FALSE`. |

## Value

A list of the following elements:

- `H` - a list of result $H_i$ matrices of size $n_i \times k$
- `V` - a list of result $V_i$ matrices
- `W` - the result $W$ matrix
- `objErr` - the final objective error value.

### Author(s)

Yichen Wang

### References

Joshua D. Welch and et al., Single-Cell Multi-omic Integration Compares and Contrasts Features of Brain Cell Identity, Cell, 2019

### Examples

```
library(Matrix)
set.seed(1)
result <- inmf(list(ctrl.sparse, stim.sparse), k = 10, niter = 10, verbose = FALSE)
```

---

`nmf` *Perform Non-negative Matrix Factorization*

---

### Description

Regularly, Non-negative Matrix Factorization (NMF) is factorizes input matrix $X$ into low rank matrices $W$ and $H$, so that $X \approx WH$. The objective function can be stated as $\arg\min_{W\geq 0,H\geq 0} ||X - WH||_F^2$. In practice, $X$ is usually regarded as a matrix of $m$ features by $n$ sample points. And the result matrix $W$ should have the dimensionality of $m \times k$ and H with $n \times k$ (transposed). This function wraps the algorithms implemented in PLANC library to solve NMF problems. Algorithms includes Alternating Non-negative Least Squares with Block Principal Pivoting (ANLS-BPP), Alternating Direction Method of Multipliers (ADMM), Hierarchical Alternating Least Squares (HALS), and Multiplicative Update (MU).

### Usage

```
nmf(
  x,
  k,
  niter = 30L,
  algo = "anlsbpp",
  nCores = 2L,
  Winit = NULL,
  Hinit = NULL
)
```

### Arguments

| | |
|---|---|
| `x` | Input matrix for factorization. Can be either dense or sparse. |
| `k` | Integer. Factor matrix rank. |
| `niter` | Integer. Maximum number of NMF interations. |
| `algo` | Algorithm to perform the factorization, choose from "anlsbpp", "admm", "hals" or "mu". See detailed sections. |

| | |
|---|---|
| nCores | The number of parallel tasks that will be spawned. Only applies to anlsbpp. Default 2 |
| Winit | Initial left-hand factor matrix, must be of size m x k. |
| Hinit | Initial right-hand factor matrix, must be of size n x k. |

## Value

A list with the following elements:

- W - the result left-hand factor matrix
- H - the result right hand matrix.
- objErr - the objective error of the factorization.

## References

Ramakrishnan Kannan and et al., A High-Performance Parallel Algorithm for Nonnegative Matrix Factorization, PPoPP '16, 2016, 10.1145/2851141.2851152

---

# onlineINMF *Perform Integrative Non-negative Matrix Factorization Using Online Learning*

---

## Description

Performs integrative non-negative matrix factorization (iNMF) (J.D. Welch, 2019, C. Gao, 2021) using online learning approach to return factorized $H$, $W$, and $V$ matrices. The objective function is stated as

$$\arg\min_{H\geq 0,W\geq 0,V\geq 0}\sum_{i}^{d}||E_i-(W+V_i)Hi||^2_F+\lambda\sum_{i}^{d}||V_iH_i||^2_F$$

where $E_i$ is the input non-negative matrix of the $i$'th dataset, $d$ is the total number of datasets. $E_i$ is of size $m \times n_i$ for $m$ features and $n_i$ sample points, $H_i$ is of size $k \times n_i$, $V_i$ is of size $m \times k$, and $W$ is of size $m \times k$.

Different from inmf which optimizes the objective with ANLS approach, onlineINMF optimizes the same objective with online learning strategy, where it updates mini-batches of $H_i$ solving the NNLS problem, and updates $V_i$ and $W$ with HALS multiplicative method.

This function allows online learning in 3 scenarios:

1. Fully observed datasets;
2. Iterative refinement using continually arriving datasets;
3. Projection of new datasets without updating the existing factorization

## Usage

```
onlineINMF(
  objectList,
  newDatasets = NULL,
  project = FALSE,
  k = 20,
  lambda = 5,
  maxEpoch = 5,
  minibatchSize = 5000,
  maxHALSIter = 1,
  permuteChunkSize = 1000,
  nCores = 2,
  Hinit = NULL,
  Vinit = NULL,
  Winit = NULL,
  Ainit = NULL,
  Binit = NULL,
  verbose = FALSE
)
```

## Arguments

`objectList` list of input datasets. List elements should all be of the same class. Viable classes include: matrix, dgCMatrix, H5Mat, H5SpMat.

`newDatasets` Same requirements as for new arriving datasets. Default `NULL` for scenario 1, specify for scenario 2 or 3.

`project` Logical scalar, whether to run scenario 3. See description. Default `FALSE`.

`k` Integer. Inner dimensionality to factorize the datasets into. Default `20`.

`lambda` Regularization parameter. Larger values penalize dataset-specific effects more strongly (i.e. alignment should increase as `lambda` increases). Default `5`.

`maxEpoch` The number of epochs to iterate through. Default `5`.

`minibatchSize` Total number of cells in each mini-batch. Default `5000`.

`maxHALSIter` Maximum number of block coordinate descent (HALS algorithm) iterations to perform for each update of $W$ and $V$. Default `1`. Changing this parameter is not recommended.

`permuteChunkSize` Number of cells in a chunk being shuffled before subsetting to minibatches. Only appliable to in-memory data and for Scenario 1 and 2. Default `1000`.

`nCores` The number of parallel tasks that will be spawned. Default `2`

`Hinit, Vinit, Winit, Ainit, Binit` Pass the previous factorization result for datasets existing in `objectList`, in order to run scenario 2 or 3. All should have `length(objectList)` matrices inside. See description for dimensionality of $H_i$, $V_i$ and $W_i$. $A_i$ should be of size $k \times k$ and $B_i$ should be of size $m \times k$

`verbose` Logical scalar. Whether to show information and progress. Default `FALSE`.

## Value

A list of the following elements:

- `H` - a list of result $H_i$ matrices of size $n_i \times k$
- `V` - a list of result $V_i$ matrices
- `W` - the result $W$ matrix
- `A` - a list of result $A_i$ matrices, $k \times k$
- `B` - a list of result $B_i$ matrices, $m \times k$
- `objErr` - the final objective error value.

## Author(s)

Yichen Wang

## References

Joshua D. Welch and et al., Single-Cell Multi-omic Integration Compares and Contrasts Features of Brain Cell Identity, Cell, 2019

Chao Gao and et al., Iterative single-cell multi-omic integration using online learning, Nat Biotechnol., 2021

## Examples

```
library(Matrix)

# Scenario 1 with sparse matrices
set.seed(1)
res1 <- onlineINMF(list(ctrl.sparse, stim.sparse),
                   minibatchSize = 50, k = 10, verbose = FALSE)

# Scenario 2 with H5 dense matrices
h5dense1 <- H5Mat(filename = system.file("extdata", "ctrl_dense.h5",
                             package = "RcppPlanc", mustWork = TRUE),
                                          dataPath = "scaleData")
h5dense2 <- H5Mat(filename = system.file("extdata", "stim_dense.h5",
                             package = "RcppPlanc", mustWork = TRUE),
                                          dataPath = "scaleData")
res2 <- onlineINMF(list(ctrl = h5dense1), minibatchSize = 50, k = 10, verbose = FALSE)
res3 <- onlineINMF(list(ctrl = h5dense1),
                   newDatasets = list(stim = h5dense2),
                   Hinit = res2$H, Vinit = res2$V, Winit = res2$W,
                   Ainit = res2$A, Binit = res2$B,
                   minibatchSize = 50, k = 10, verbose = FALSE)

# Scenario 3 with H5 sparse matrices
h5sparse1 <- H5SpMat(filename = system.file("extdata", "ctrl_sparse.h5",
                                package = "RcppPlanc", mustWork = TRUE),
                                valuePath = "scaleDataSparse/data",
                                rowindPath = "scaleDataSparse/indices",
```

```
                                colptrPath = "scaleDataSparse/indptr",
                                nrow = nrow(ctrl.sparse),
                                ncol = ncol(ctrl.sparse))
h5sparse2 <- H5SpMat(filename = system.file("extdata", "stim_sparse.h5",
                                package = "RcppPlanc", mustWork = TRUE),
                                valuePath = "scaleDataSparse/data",
                                rowindPath = "scaleDataSparse/indices",
                                colptrPath = "scaleDataSparse/indptr",
                                nrow = nrow(stim.sparse),
                                ncol = nrow(stim.sparse))
res4 <- onlineINMF(list(ctrl = h5sparse1), minibatchSize = 50, k = 10, verbose = FALSE)
res5 <- onlineINMF(list(ctrl = h5sparse1),
                   newDatasets = list(stim = h5sparse2), project = TRUE,
                   Hinit = res4$H, Vinit = res4$V, Winit = res4$W,
                   Ainit = res4$A, Binit = res4$B,
                   minibatchSize = 50, k = 10, verbose = FALSE)
```

---

`print.H5Mat` *Show information of a H5Mat object*

---

## Description

Show information of a H5Mat object

## Usage

```
## S3 method for class 'H5Mat'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | H5Mat argument list object |
| ... | Not used. |

## Value

NULL. Information displayed.

## Examples

```
h <- H5Mat(system.file("extdata/ctrl_dense.h5", package = "RcppPlanc"),
           "data")
print(h)
```

## print.H5SpMat *Show information of a H5SpMat object*

### Description

Show information of a H5SpMat object

### Usage

```
## S3 method for class 'H5SpMat'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | H5SpMat argument list object |
| ... | Not used. |

### Value

NULL. Information displayed.

### Examples

```
h <- H5SpMat(system.file("extdata/ctrl_sparse.h5", package = "RcppPlanc"),
             "data", "indices", "indptr", 173, 300)
print(h)
```

## symNMF *Perform Symmetric Non-negative Matrix Factorization*

### Description

Symmetric input matrix $X$ of size $n \times n$ is required. Two approaches are provided. Alternating Non-negative Least Squares Block Principal Pivoting algorithm (ANLSBPP) with symmetric regularization, where the objective function is set to be $\arg\min_{H\geq0,W\geq0}||X-WH||_F^2+\lambda||W-H||_F^2$, can be run with `algo = "anlsbpp"`. Gaussian-Newton algorithm, where the objective function is set to be $\arg\min_{H\geq0}||X-H^{\mathsf{T}}H||_F^2$, can be run with `algo = "gnsym"`. In the objectives, $W$ is of size $n \times k$ and $H$ is of size $k \times n$. The returned results will all be $n \times k$.

## Usage

```
symNMF(
  x,
  k,
  niter = 30L,
  lambda = 0,
  algo = "gnsym",
  nCores = 2L,
  Hinit = NULL
)
```

## Arguments

| | |
|---|---|
| `x` | Input matrix for factorization. Must be symmetric. Can be either dense or sparse. |
| `k` | Integer. Factor matrix rank. |
| `niter` | Integer. Maximum number of symNMF interations. Default `30` |
| `lambda` | Symmetric regularization parameter. Must be non-negative. Default `0.0` uses the square of the maximum value in `x`. |
| `algo` | Algorithm to perform the factorization, choose from "gnsym" or "anlsbpp". Default `"gnsym"` |
| `nCores` | The number of parallel tasks that will be spawned. Only applies to anlsbpp. Default `2` |
| `Hinit` | Initial right-hand factor matrix, must be of size n x k. Default `NULL`. |

## Value

A list with the following elements:

- `W` - the result left-hand factor matrix, non-empty when using `"anlsbpp"`
- `H` - the result right hand matrix.
- `objErr` - the objective error of the factorization.

## References

Srinivas Eswar and et al., Distributed-Memory Parallel Symmetric Nonnegative Matrix Factorization, SC '20, 2020, 10.5555/3433701.3433799

uinmf *Perform Mosaic Integrative Non-negative Matrix Factorization with Unshared Features*

## Description

Performs mosaic integrative non-negative matrix factorization (UINMF) (A.R. Kriebel, 2022) to return factorized $H$, $W$, $V$ and $U$ matrices. The objective function is stated as

$$\arg\min_{H\geq 0,W\geq 0,V\geq 0,U\geq 0}\sum_{i}^{d}||\begin{bmatrix}E_i\\P_i\end{bmatrix}-(\begin{bmatrix}W\\0\end{bmatrix}+\begin{bmatrix}V_i\\U_i\end{bmatrix})Hi||^2_F+\lambda_i\sum_{i}^{d}||\begin{bmatrix}V_i\\U_i\end{bmatrix}H_i||^2_F$$

where $E_i$ is the input non-negative matrix of the $i$'th dataset, $P_i$ is the input non-negative matrix for the unshared features, $d$ is the total number of datasets. $E_i$ is of size $m \times n_i$ for $m$ shared features and $n_i$ sample points, $P_i$ is of size $u_i \times n_i$ for $u_i$ unshared feaetures, $H_i$ is of size $k \times n_i$, $V_i$ is of size $m \times k$, $W$ is of size $m \times k$ and $U_i$ is of size $u_i \times k$.

Similar to `inmf`, `uinmf` also optimizes the objective with ANLS algorithm.

## Usage

```
uinmf(
  objectList,
  unsharedList,
  k = 20,
  lambda = 5,
  niter = 30,
  nCores = 2,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `objectList` | list of input datasets. List elements should all be of the same class. Viable classes include: matrix, dgCMatrix, H5Mat, H5SpMat. |
| `unsharedList` | List of input unshared feature matrices, with the same requirement as `objectList`. |
| `k` | Integer. Inner dimensionality to factorize the datasets into. Default `20`. |
| `lambda` | Regularization parameter. Use one number for all datasets or a vector to specify for each dataset. Larger values penalize dataset-specific effects more strongly (i.e. alignment should increase as `lambda` increases). Default `5`. |
| `niter` | Integer. Total number of block coordinate descent iterations to perform. Default `30`. |
| `nCores` | The number of parallel tasks that will be spawned. Default `2`. |
| `verbose` | Logical scalar. Whether to show information and progress. Default `FALSE`. |

## Value

A list of the following elements:

- H - a list of result $H_i$ matrices of size $n_i \times k$
- V - a list of result $V_i$ matrices
- W - the result $W$ matrix
- U - a list of result $A_i$ matrices
- objErr - the final objective error value.

## Author(s)

Yichen Wang

## References

April R. Kriebel and Joshua D. Welch, UINMF performs mosaic integration of single-cell multi-omic datasets using nonnegative matrix factorization, Nat. Comm., 2022

## Examples

```
# Fake matrices representing unshared features of the given datasets
# Real-life use should have features that are not presented in the
# intersection of features of all datasets involved.
ctrl.unshared <- ctrl.sparse[1:10,]
stim.unshared <- stim.sparse[11:30,]
set.seed(1)
result <- uinmf(list(ctrl.sparse, stim.sparse),
                list(ctrl.unshared, stim.unshared), verbose = FALSE)
```

# Index