Package ‘nmfkc’

March 29, 2026

Type Package

Title Non-Negative Matrix Factorization with Kernel Covariates

Version 0.6.3

Date 2026-03-28

Author Kenichi Satoh [aut, cre] (<https://orcid.org/0000-0003-4436-9347>)

Maintainer Kenichi Satoh <kenichi-satoh@biwako.shiga-u.ac.jp>

URL https://github.com/ksatohds/nmfkc, https://ksatohds.github.io/nmfkc/

BugReports https://github.com/ksatohds/nmfkc/issues

Description Performs Non-negative Matrix Factorization (NMF) with Kernel Covariates. Given an observation matrix and kernel covariates, it optimizes both a basis matrix and a parameter matrix. Notably, if the kernel matrix is an identity matrix, the method simplifies to standard NMF. Also provides NMF with Random Effects (NMF-RE) via nmfre(), which estimates a mixed-effects model combining covariate-driven scores with unit-specific random effects, together with wild bootstrap inference.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

ByteCompile true

VignetteBuilder knitr

Suggests knitr,
rmarkdown,
testthat (>= 3.0.0),
mclust,
palmerpenguins,
quanteda,
vars,
DiagrammeR,
fda,
NipponMap,
alluvial,

httr,
readxl,
RColorBrewer,
MASS,
nlme,
lavaan

Config/testthat/edition 3

Contents

`coef.nmfkc` *Extract coefficients from NMF models*

Description

Returns the `coefficients` data frame from a fitted NMF model that has been passed through an inference function (`nmfkc.inference`, `nmfae.inference`, `nmfre.inference`).

If inference has not been run, returns the parameter matrix C (Θ) directly.

Usage

```
## S3 method for class 'nmfkc'
coef(object, ...)

## S3 method for class 'nmfae'
coef(object, ...)

## S3 method for class 'nmfre'
coef(object, ...)

## S3 method for class 'nmf.sem'
coef(object, ...)
```

Arguments

`object`	A fitted model object.
`...`	Not used.

Value

A data frame of coefficients (if inference was performed), or the parameter matrix C.

See Also

nmfkc.inference, nmfae.inference, nmfre.inference

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
result <- nmfkc(Y, A, Q = 1)
coef(result)  # returns C matrix

result2 <- nmfkc.inference(result, Y, A)
coef(result2)  # returns coefficients data frame
```

`fitted.nmfkc` *Extract fitted values from NMF models*

Description

Returns the reconstructed matrix $\hat{Y} = XB$ from a fitted NMF model.

For `nmf.sem` objects, returns the equilibrium prediction $\hat{Y}_1 = M_{model}Y_2$ if available. Supply `Y1` and `Y2` to get the direct reconstruction $X(C_1Y_1 + C_2Y_2)$ instead.

Usage

```
## S3 method for class 'nmfkc'
fitted(object, ...)

## S3 method for class 'nmfae'
fitted(object, ...)

## S3 method for class 'nmfre'
fitted(object, ...)

## S3 method for class 'nmf.sem'
fitted(object, ...)
```

Arguments

`object`	A fitted model object.
`...`	For `nmf.sem`: optionally `Y1` and `Y2`.

Value

The fitted matrix XB.

Examples

```
result <- nmfkc(matrix(runif(50), 5, 10), Q = 2)
fitted(result)
```

`nmf.sem` *NMF-SEM Main Estimation Algorithm*

Description

Fits the NMF-SEM model

$$Y_1 \approx X\bigl(\Theta_1 Y_1 + \Theta_2 Y_2\bigr)$$

under non-negativity constraints with orthogonality and sparsity regularization. The function returns the estimated latent factors, structural coefficient matrices, and the implied equilibrium (input–output) mapping.

At equilibrium, the model can be written as

$$Y_1 \approx (I - X\Theta_1)^{-1} X\Theta_2 Y_2 \equiv M_{\text{model}} Y_2,$$

where $M_{\text{model}} = (I - X\Theta_1)^{-1} X\Theta_2$ is a Leontief-type cumulative-effect operator in latent space.

Internally, the latent feedback and exogenous loading matrices are stored as `C1` and `C2`, corresponding to Θ_1 and Θ_2, respectively.

Usage

```
nmf.sem(
  Y1,
  Y2,
  rank = NULL,
  X.init = NULL,
  X.L2.ortho = 100,
  C1.L1 = 1,
  C2.L1 = 0.1,
  epsilon = 1e-06,
  maxit = 20000,
  seed = 123,
  ...
)
```

Arguments

`Y1`	A non-negative numeric matrix of endogenous variables with **rows = variables (P1), columns = samples (N)**.
`Y2`	A non-negative numeric matrix of exogenous variables with **rows = variables (P2), columns = samples (N)**. Must satisfy `ncol(Y1) == ncol(Y2)`.
`rank`	Integer; number of latent factors Q. If `NULL`, Q is taken from a hidden argument in `...` or defaults to `nrow(Y2)`.
`X.init`	Optional non-negative initialization for the basis matrix `X` ($P_1 \times Q$). If supplied, it is projected to be non-negative and column-normalized.

`X.L2.ortho`	L2 orthogonality penalty for `X`. This controls the penalty term $\lambda_X \|X^\top X - \mathrm{diag}(X^\top X)\|_F^2$. Default: `100`.
`C1.L1`	L1 sparsity penalty for `C1` (i.e., Θ_1). Default: `1.0`.
`C2.L1`	L1 sparsity penalty for `C2` (i.e., Θ_2). Default: `0.1`.
`epsilon`	Relative convergence threshold for the objective function. Iterations stop when the relative change in reconstruction loss falls below this value. Default: `1e-6`.
`maxit`	Maximum number of iterations for the multiplicative updates. Default: `20000`.
`seed`	Random seed used to initialize `X`, `C1`, and `C2`. Default: `123`.
`...`	Additional arguments. Currently used to pass a hidden rank `Q` (e.g., via `Q = 3`) if `rank` is `NULL`.

Value

A list with components:

`X`	Estimated basis matrix ($P_1 \times Q$).
`C1`	Estimated latent feedback matrix (Θ_1, $Q \times P_1$).
`C2`	Estimated exogenous loading matrix (Θ_2, $Q \times P_2$).
`XC1`	Feedback matrix $X\Theta_1$.
`XC2`	Direct-effect matrix $X\Theta_2$.
`XC1.radius`	Spectral radius $\rho(X\Theta_1)$.
`XC1.norm1`	Induced 1-norm $\|X\Theta_1\|_{1,\mathrm{op}}$.
`Leontief.inv`	Leontief-type inverse $(I - X\Theta_1)^{-1}$.
`M.model`	Equilibrium mapping $M_{\mathrm{model}} = (I - X\Theta_1)^{-1} X\Theta_2$.
`amplification`	Latent amplification factor $\|M_{\mathrm{model}}\|_{1,\mathrm{op}} / \left\|X\Theta_2\right\|_{1,\mathrm{op}}$.
`amplification.bound`	Geometric-series upper bound $1/(1 - \|X\Theta_1\|_{1,\mathrm{op}})$ if $\|X\Theta_1\|_{1,\mathrm{op}} < 1$, otherwise `Inf`.
`Q`	Effective latent dimension used in the fit.
`SC.cov`	Correlation between sample and model-implied covariance (flattened) of Y_1.
`MAE`	Mean absolute error between Y_1 and its equilibrium prediction $\hat{Y}_1 = M_{\mathrm{model}} Y_2$.
`objfunc`	Vector of reconstruction losses per iteration.
`objfunc.full`	Vector of penalized objective values per iteration.
`iter`	Number of iterations actually performed.

References

Satoh, K. (2025). Applying non-negative matrix factorization with covariates to structural equation modeling for blind input-output analysis. arXiv:2512.18250. https://arxiv.org/abs/2512.18250

Examples

```
# Simple NMF-SEM with iris data (non-negative)
Y <- t(iris[, -5])
Y1 <- Y[1:2, ]  # Sepal
Y2 <- Y[3:4, ]  # Petal
result <- nmf.sem(Y1, Y2, rank = 2, maxit = 500)
result$MAE
```

nmf.sem.cv — *Cross-Validation for NMF-SEM*

Description

Performs K-fold cross-validation to evaluate the equilibrium mapping of the NMF-SEM model.

For each fold, `nmf.sem` is fitted on the training samples, yielding an equilibrium mapping $\hat{Y}_1 = M_{\text{model}} Y_2$. The held-out endogenous variables Y_1 are then predicted from Y_2 using this mapping, and the mean absolute error (MAE) over all entries in the test block is computed. The returned value is the average MAE across folds.

This implements the hyperparameter selection strategy described in the paper: hyperparameters are chosen by predictive cross-validation rather than direct inspection of the internal structural matrices.

Usage

```
nmf.sem.cv(
  Y1,
  Y2,
  rank = NULL,
  X.init = NULL,
  X.L2.ortho = 100,
  C1.L1 = 1,
  C2.L1 = 0.1,
  epsilon = 1e-06,
  maxit = 20000,
  seed = NULL,
  nfolds = 5,
  shuffle = TRUE,
  ...
)
```

Arguments

`Y1`	A non-negative numeric matrix of endogenous variables with **rows = variables (P1), columns = samples (N)**.
`Y2`	A non-negative numeric matrix of exogenous variables with **rows = variables (P2), columns = samples (N)**. Must satisfy `ncol(Y1) == ncol(Y2)`.
`rank`	Integer; rank (number of latent factors) passed to `nmf.sem`. If `NULL`, `nmf.sem` decides the effective rank (via `...` or `nrow(Y2)`).
`X.init`	Optional initialization for `X` (as in `nmf.sem`).
`X.L2.ortho`	L2 orthogonality penalty for `X`.
`C1.L1`	L1 sparsity penalty for `C1` (Θ_1).
`C2.L1`	L1 sparsity penalty for `C2` (Θ_2).
`epsilon`	Convergence threshold for `nmf.sem`.
`maxit`	Maximum number of iterations for `nmf.sem`.
`seed`	Master random seed for CV splitting and fold-specific calls to `nmf.sem`. If `NULL`, RNG is not controlled within folds.

`nfolds`	Number of CV folds. Default is 5. For backward compatibility, `div` is accepted via `...`.
`shuffle`	Logical; if `TRUE`, samples are randomly permuted before assigning to folds. (Default: `TRUE`)
`...`	Additional arguments passed to `nmf.sem` (except for `rank`, `seed`, `div`, `shuffle`, which are handled here).

Value

A numeric scalar: mean MAE across CV folds.

Examples

```
Y <- t(iris[, -5])
Y1 <- Y[1:2, ]
Y2 <- Y[3:4, ]
mae <- nmf.sem.cv(Y1, Y2, rank = 2, maxit = 500, div = 3)
mae
```

`nmf.sem.inference` *Statistical inference for the exogenous parameter matrix C2*

Description

`nmf.sem.inference` performs statistical inference on the exogenous parameter matrix C_2 from a fitted `nmf.sem` model, conditional on the estimated basis matrix $\hat{X}$ and the endogenous parameter matrix $\hat{C}_1$.

Under the working model $R = Y_1 - XC_1Y_1 \approx XC_2Y_2 + \varepsilon$, inference on C_2 is conducted via sandwich covariance estimation and one-step wild bootstrap with non-negative projection.

Usage

```
nmf.sem.inference(object, Y1, Y2, wild.bootstrap = TRUE, ...)
```

Arguments

`object`	A list returned by `nmf.sem`, containing at least `X`, `C1`, and `C2`.
`Y1`	Endogenous variable matrix (P1 x N). Must match the data used in `nmf.sem()`.
`Y2`	Exogenous variable matrix (P2 x N). Must match the data used in `nmf.sem()`.
`wild.bootstrap`	Logical. If `TRUE` (default), performs wild bootstrap for confidence intervals and bootstrap standard errors.
`...`	Additional arguments:

`wild.B` Number of bootstrap replicates. Default is 1000.

`wild.seed` Seed for bootstrap. Default is 42.

`wild.level` Confidence level for bootstrap CI. Default is 0.95.

`sandwich` Logical. Use sandwich covariance. Default is `TRUE`.

`C.p.side` P-value type: `"one.sided"` (default) or `"two.sided"`.

`cov.ridge` Ridge stabilization for information matrix inversion. Default is 1e-8.

`print.trace` Logical. If `TRUE`, prints progress. Default is `FALSE`.

Value

The input `object` with additional inference components:

`sigma2.used`	Estimated σ^2 used for inference.
`C2.se`	Sandwich standard errors for C_2 (Q x P2 matrix).
`C2.se.boot`	Bootstrap standard errors for C_2 (Q x P2 matrix).
`C2.ci.lower`	Lower CI bounds for C_2 (Q x P2 matrix).
`C2.ci.upper`	Upper CI bounds for C_2 (Q x P2 matrix).
`coefficients`	Data frame with Estimate, SE, BSE, z, p-value for each element of C_2.
`C2.p.side`	P-value type used.

References

Satoh, K. (2025). Applying non-negative matrix factorization with covariates to structural equation modeling for blind input-output analysis. arXiv:2512.18250. `https://arxiv.org/abs/2512.18250`

See Also

`nmf.sem`, `nmf.sem.DOT`

Examples

```
Y <- t(iris[, -5])
Y1 <- Y[1:2, ]; Y2 <- Y[3:4, ]
res <- nmf.sem(Y1, Y2, rank = 2)
res2 <- nmf.sem.inference(res, Y1, Y2)
res2$coefficients
```

`nmf.sem.split` *Heuristic Variable Splitting for NMF-SEM*

Description

Infers a heuristic partition of observed variables into exogenous (Y_2) and endogenous (Y_1) blocks for use in NMF-SEM. The method is based on positive-SEM logic, causal ordering, and optional sign alignment using the first principal component (PC1).

The procedure:

- internally standardizes variables (mean 0, sd 1),
- optionally flips signs so that most variables align positively with PC1,
- infers a causal ordering by repeatedly regressing each variable on the remaining ones and selecting the variable with the largest minimum standardized coefficient,
- determines an exogenous block by scanning the ordering from upstream and stopping at the first variable whose strongest parent coefficient exceeds `threshold`.

If `n.exogenous` is supplied, it overrides the automatic threshold rule.

Usage

```
nmf.sem.split(
  x,
  n.exogenous = NULL,
  threshold = 0.1,
  auto.flipped = TRUE,
  verbose = TRUE
)
```

Arguments

`x`	A numeric matrix or data frame with **rows = samples** and **columns = observed variables**.
`n.exogenous`	Optional integer specifying the number of exogenous variables (Y_2). If `NULL`, the number is inferred automatically by the coefficient cut-off rule.
`threshold`	Standardized regression-coefficient threshold used in the automatic exogenous–endogenous split. A variable is treated as endogenous once its maximum standardized parent coefficient exceeds this value. (Default: `0.1`)
`auto.flipped`	Logical; if `TRUE`, applies PC1-based automatic sign flipping after standardization to ensure consistent orientation. (Default: `TRUE`)
`verbose`	Logical; if `TRUE`, prints progress messages and the resulting variable split. (Default: `TRUE`)

Value

A list with:

`endogenous.variables`	Character vector of variables selected as endogenous (Y_1).
`exogenous.variables`	Character vector of variables selected as exogenous (Y_2).
`ordered.variables`	Variables in inferred causal order (from exogenous to endogenous).
`is.flipped`	Logical vector indicating which variables were sign-flipped during processing.
`n.exogenous`	Integer giving the number of exogenous variables.

Examples

```
# Infer exogenous/endogenous split from iris
sp <- nmf.sem.split(iris[, -5], n.exogenous = 2)
sp$Y1.names
sp$Y2.names
```

nmfae *Three-Layer Non-negative Matrix Factorization (NMF-AE)*

Description

`nmfae` fits a three-layer nonnegative matrix factorization model $Y_1 \approx X_1 \Theta X_2 Y_2$, where X_1 is a decoder basis (column sum 1), Θ is a bottleneck parameter matrix, X_2 is an encoder basis (row sum 1), and Y_2 is the input matrix.

When `Y2 = Y1`, the model acts as a non-negative autoencoder. When `Y1 != Y2`, it acts as a heteroencoder.

Initialization uses a three-step NMF procedure via `nmfkc`: (1) `nmfkc(Y1, rank=Q)` to obtain X_1, (2) `nmfkc(Y1, A=Y2, rank=Q)` with fixed X_1 to obtain $C = \Theta X_2$, (3) `nmfkc(Y2, rank=R)` to factor C into Θ and X_2.

Usage

```
nmfae(
  Y1,
  Y2 = Y1,
  rank = 2,
  rank.encoder = rank,
  epsilon = 1e-04,
  maxit = 5000,
  ...
)
```

Arguments

`Y1`	Output matrix Y_1 (P1 x N). Non-negative. May contain `NA`s (handled via `Y1.weights`).
`Y2`	Input matrix Y_2 (P2 x N). Non-negative. Default is `Y1` (autoencoder).
`rank`	Integer. Rank of the decoder basis X_1 (P1 x Q). Default is 2. For backward compatibility, `Q` is accepted via `...`.
`rank.encoder`	Integer. Rank of the encoder basis X_2 (R x P2). Default is `rank`. For backward compatibility, `R` is accepted via `...`.
`epsilon`	Positive convergence tolerance. Default is `1e-4`.
`maxit`	Maximum number of multiplicative update iterations. Default is 5000.
`...`	Additional arguments:

`Y1.weights` Weight matrix (P1 x N) or vector for Y_1. 0 indicates missing/ignored elements. Default: auto-detect `NA`s.

`C.L1` L1 regularization parameter for C. Default is 0.

`X1.L2.ortho` L2 orthogonality regularization for X_1 columns. Default is 0.

`X2.L2.ortho` L2 orthogonality regularization for X_2 rows. Default is 0.

`seed` Integer seed for reproducibility. Default is 123.

`print.trace` Logical. If `TRUE`, prints progress. Default is `FALSE`.

Value

An object of class "nmfae", a list with components:

X1	Decoder basis matrix (P1 x Q), column sum 1.
C	Parameter matrix (Q x R).
X2	Encoder basis matrix (R x P2), row sum 1.
Y1hat	Fitted values $X_1 \Theta X_2 Y_2$ (P1 x N).
rank	Named integer vector c(Q, R).
objfunc	Final objective value.
objfunc.iter	Objective values by iteration.
r.squared	Coefficient of determination R^2.
niter	Number of iterations performed.
runtime	Elapsed time as a difftime object.
n.missing	Number of missing (or zero-weighted) elements in Y_1.
n.total	Total number of elements in Y_1 (P1 x N).

Lifecycle

This function is **experimental**. The interface may change in future versions.

Source

Satoh, K. (2025). Applying Non-negative Matrix Factorization with Covariates to Multivariate Time Series. *Japanese Journal of Statistics and Data Science*.

References

Lee, D. D. and Seung, H. S. (2001). Algorithms for Non-negative Matrix Factorization. *Advances in Neural Information Processing Systems*, 13.

Saha, S. et al. (2022). Hierarchical Deep Learning Neural Network (HiDeNN): An Artificial Intelligence (AI) Framework for Computational Science and Engineering. *Computer Methods in Applied Mechanics and Engineering*, 399.

See Also

nmfae.inference, predict.nmfae, nmfae.ecv, nmfae.DOT, nmfkc

Examples

```
# Autoencoder example
library(nmfkc)
Y <- matrix(c(1,0,1,0, 0,1,0,1, 1,1,0,0), nrow=3, byrow=TRUE)
res <- nmfae(Y, Q=2, R=2)
res$r.squared

# Heteroencoder example
Y1 <- matrix(c(1,0,0,1), nrow=2)
Y2 <- matrix(runif(8), nrow=4)
res2 <- nmfae(Y1, Y2, Q=2, R=2)
```

`nmfae.cv` *Sample-wise k-fold Cross-Validation for nmfae*

Description

`nmfae.cv` performs k-fold cross-validation by splitting columns (samples) of Y_1 and Y_2 into `div` folds. For each fold, the model $Y_1 \approx X_1 \Theta X_2 Y_2$ is fitted on the training samples and predictive performance is evaluated on the held-out samples.

When `Y2` is a kernel matrix created by `nmfkc.kernel` (detected via attributes), the symmetric kernel splitting convention is used: `Y2[train, train]` for training and `Y2[train, test]` for prediction.

Usage

```
nmfae.cv(
  Y1,
  Y2 = Y1,
  Q = 2,
  R = Q,
  nfolds = 5,
  seed = 123,
  shuffle = TRUE,
  ...
)
```

Arguments

`Y1`	Output matrix Y_1 (P1 x N). Non-negative.
`Y2`	Input matrix Y_2 (P2 x N), or a kernel matrix (N x N). Default is `Y1` (autoencoder).
`Q`	Integer. Rank of the decoder basis. Default is 2.
`R`	Integer. Rank of the encoder basis. Default is `Q`.
`nfolds`	Number of folds. Default is 5. For backward compatibility, `div` is accepted via `...`.
`seed`	Integer seed for reproducible fold partitioning. Default is 123.
`shuffle`	Logical. If `TRUE` (default), randomly shuffles samples; if `FALSE`, splits sequentially (block CV for time series).
`...`	Additional arguments passed to `nmfae` (e.g., `epsilon`, `maxit`, `Y1.weights`).

Value

A list with components:

`objfunc`	Mean squared error per valid element over all folds.
`sigma`	Residual standard error (RMSE), same scale as Y_1.
`objfunc.block`	Per-fold squared error totals.
`block`	Integer vector of fold assignments (1, ..., `div`) for each column.

See Also

`nmfae`, `nmfae.ecv`, `nmfae.kernel.beta.cv`, `nmfkc.cv`

Examples

```
library(nmfkc)
Y <- t(iris[1:30, 1:4])
res <- nmfae.cv(Y, Q = 2, R = 2, div = 5, maxit = 500)
res$sigma
```

`nmfae.DOT` *DOT graph visualization for nmfae objects*

Description

`nmfae.DOT` generates a DOT language string for visualizing the structure of a three-layer NMF model. Two graph types are supported: `"XCX"` shows encoder factors, Θ, and decoder factors; `"YXCXY"` shows the full structure from Y_2 through X_2, Θ, X_1 to Y_1.

Edge widths are proportional to matrix element values, and edges below `threshold` are omitted for clarity.

Usage

```
nmfae.DOT(
  x,
  type = c("XCX", "YXCXY"),
  threshold = 0.01,
  sig.level = 0.1,
  rankdir = "LR",
  fill = TRUE,
  weight_scale = 5,
  weight_scale_x1 = weight_scale,
  weight_scale_theta = weight_scale,
  weight_scale_x2 = weight_scale,
  Y1.label = NULL,
  X1.label = NULL,
  X2.label = NULL,
  Y2.label = NULL,
  Y1.title = "Output (Y1)",
  X1.title = "Decoder (X1)",
  X2.title = "Encoder (X2)",
  Y2.title = "Input (Y2)",
  hide.isolated = TRUE
)
```

Arguments

`x`	An object of class `"nmfae"` returned by `nmfae`.
`type`	Character. Graph type: `"XCX"` (default) or `"YXCXY"`.
`threshold`	Numeric. Edges with values below this are omitted. Default is 0.01.
`sig.level`	Numeric or `NULL`. Significance level for filtering C edges when inference results are available. Only edges with p-value below `sig.level` are shown, with significance stars. Set to `NULL` to disable. Default is 0.1.

`rankdir`	Character. Graph direction for DOT layout. Default is `"LR"` (left to right).
`fill`	Logical. If `TRUE`, nodes are filled with color. Default is `TRUE`.
`weight_scale`	Numeric. Base scale factor for edge widths. Default is 5.
`weight_scale_x1`	Numeric. Scale factor for X_1 edges.
`weight_scale_theta`	Numeric. Scale factor for Θ edges.
`weight_scale_x2`	Numeric. Scale factor for X_2 edges.
`Y1.label`	Character vector of output variable labels.
`X1.label`	Character vector of decoder basis labels.
`X2.label`	Character vector of encoder basis labels.
`Y2.label`	Character vector of input variable labels.
`Y1.title`	Character. Title for output node group. Default is `"Output (Y1)"`.
`X1.title`	Character. Title for decoder node group. Default is `"Decoder (X1)"`.
`X2.title`	Character. Title for encoder node group. Default is `"Encoder (X2)"`.
`Y2.title`	Character. Title for input node group. Default is `"Input (Y2)"`.
`hide.isolated`	Logical. If `TRUE` (default), Y1 and Y2 nodes that have no edges at or above `threshold` are excluded from the graph. Only applies when `type = "YXCXY"`.

Value

A character string containing the DOT graph specification.

See Also

`nmfae`

`nmfae.ecv` *Element-wise Cross-Validation for nmfae (Wold's CV)*

Description

`nmfae.ecv` performs k-fold element-wise cross-validation by randomly holding out individual elements of Y_1, assigning them a weight of 0 via `Y1.weights`, and evaluating the reconstruction error on those held-out elements.

This method (also known as Wold's CV) is suitable for determining the optimal rank pair (Q, R) in three-layer NMF. Both `Q` and `R` accept vector inputs. When `R = NULL` (default), `R` is set equal to `Q` and pairs are evaluated element-wise (i.e., $(Q_1, R_1), (Q_2, R_2), \ldots$). When `R` is explicitly specified, all combinations of `Q` and `R` are evaluated via `expand.grid`.

Usage

```
nmfae.ecv(Y1, Y2 = Y1, Q = 1:2, R = NULL, nfolds = 5, seed = 123, ...)
```

Arguments

`Y1`	Output matrix Y_1 (P1 x N).
`Y2`	Input matrix Y_2 (P2 x N). Default is `Y1`.
`Q`	Integer vector of decoder ranks to evaluate. Default is `1:2`.
`R`	Integer vector of encoder ranks to evaluate. Default is `NULL`, which sets `R = Q` and evaluates element-wise pairs. When explicitly specified, all combinations with `Q` are evaluated.
`nfolds`	Number of folds. Default is 5. For backward compatibility, `div` is accepted via `...`.
`seed`	Integer seed for reproducibility. Default is 123.
`...`	Additional arguments passed to `nmfae` (e.g., `epsilon`, `maxit`).

Value

A list with components:

`objfunc`	Named numeric vector of mean MSE for each (Q, R) pair.
`sigma`	Named numeric vector of RMSE (square root of MSE) for each pair.
`objfunc.fold`	Named list of per-fold MSE vectors for each pair.
`folds`	List of length `div` containing the held-out element indices for each fold.
`QR`	Data frame with columns `Q` and `R` listing the evaluated pairs.

See Also

`nmfae`, `nmfkc.ecv`

Examples

```
library(nmfkc)
Y <- t(iris[1:30, 1:4])
# Default: R=NULL -> paired Q=R
res <- nmfae.ecv(Y, Q = 1:3, div = 3, maxit = 500)
res$sigma
# Explicit R: full grid
res2 <- nmfae.ecv(Y, Q = 1:3, R = 1:3, div = 3, maxit = 500)
res2$sigma
```

`nmfae.heatmap` *Heatmap visualization of nmfae factor matrices*

Description

`nmfae.heatmap` displays the three factor matrices X_1, Θ, and X_2 as side-by-side heatmaps. This provides an alternative to DOT graph visualization, especially when Y_2 has many variables (e.g., kernel matrix).

Usage

```
nmfae.heatmap(
  x,
  Y1.label = NULL,
  X1.label = NULL,
  X2.label = NULL,
  Y2.label = NULL,
  palette = NULL,
  ...
)
```

Arguments

`x`	An object of class `"nmfae"` returned by `nmfae`.
`Y1.label`	Character vector of output variable names (rows of X_1).
`X1.label`	Character vector of decoder basis labels (columns of X_1).
`X2.label`	Character vector of encoder basis labels (rows of X_2).
`Y2.label`	Character vector of input variable names (columns of X_2).
`palette`	Color palette vector. Default is white-orange-red (64 colors).
`...`	Not used.

Value

Invisible `NULL`. Called for its side effect (plot).

See Also

`nmfae`, `plot.nmfae`, `nmfae.DOT`

`nmfae.inference` *Statistical Inference for NMF-AE Parameter Matrix*

Description

Performs post-estimation inference for Θ in the three-layer NMF model $Y_1 \approx X_1 \Theta X_2 Y_2$, conditional on $(\hat{X}_1, \hat{X}_2)$. Uses sandwich covariance estimation and one-step wild bootstrap with non-negative projection.

Usage

```
nmfae.inference(object, Y1, Y2 = Y1, wild.bootstrap = TRUE, ...)
```

Arguments

`object`	An object of class `"nmfae"` returned by `nmfae`.
`Y1`	Output matrix Y_1 (P1 x N). Must match the data used in `nmfae()`.
`Y2`	Input matrix Y_2 (P2 x N). Default is `Y1` (autoencoder).

`wild.bootstrap`	Logical. If `TRUE` (default), performs wild bootstrap for bootstrap SE and confidence intervals. If `FALSE`, only sandwich SE and z-test p-values are computed (faster).
`...`	Additional arguments:

- `wild.B` Number of bootstrap replicates. Default is 1000.
- `wild.seed` Seed for bootstrap. Default is 42.
- `wild.level` Confidence level for bootstrap CI. Default is 0.95.
- `sandwich` Logical. Use sandwich covariance. Default is `TRUE`.
- `C.p.side` P-value type: `"one.sided"` (default) or `"two.sided"`.
- `cov.ridge` Ridge stabilization for information matrix inversion. Default is 1e-8.
- `print.trace` Logical. If `TRUE`, prints progress. Default is `FALSE`.

Value

An object of class `c("nmfae.inference", "nmfae")`, inheriting all components from the input `object`, with additional inference components:

`sigma2.used`	Estimated σ^2 used for inference.
`C.se`	Sandwich standard errors for Θ (Q x R matrix).
`C.se.boot`	Bootstrap standard errors for Θ (Q x R matrix).
`C.ci.lower`	Lower CI bounds for Θ (Q x R matrix).
`C.ci.upper`	Upper CI bounds for Θ (Q x R matrix).
`coefficients`	Data frame with Estimate, SE, BSE, z, p-value for each element of Θ.
`C.p.side`	P-value type used.

See Also

`nmfae`, `summary.nmfae.inference`

Examples

```
Y <- matrix(c(1,0,1,0, 0,1,0,1, 1,1,0,0), nrow=3, byrow=TRUE)
res <- nmfae(Y, Q=2, R=2)
res2 <- nmfae.inference(res, Y)
summary(res2)
```

`nmfae.kernel.beta.cv` *Optimize kernel beta for nmfae by cross-validation*

Description

`nmfae.kernel.beta.cv` selects the optimal `beta` parameter of the kernel function by evaluating `nmfae.cv` for each candidate value. The kernel matrix $A = K(U, V; \beta)$ replaces Y_2 in the three-layer NMF model.

When `beta = NULL`, candidate values are automatically generated via `nmfkc.kernel.beta.nearest.med`.

Usage

```
nmfae.kernel.beta.cv(
  Y1,
  Q = 2,
  R = Q,
  U,
  V = NULL,
  beta = NULL,
  plot = TRUE,
  ...
)
```

Arguments

`Y1`	Output matrix Y_1 (P1 x N). Non-negative.
`Q`	Integer. Rank of the decoder basis. Default is 2.
`R`	Integer. Rank of the encoder basis. Default is `Q`.
`U`	Covariate matrix U (K x M). Rows are features, columns are samples (or knot points for non-symmetric kernels).
`V`	Covariate matrix V (K x N). If `NULL` (default), `V` = `U` and a symmetric kernel is used.
`beta`	Numeric vector of candidate beta values. If `NULL`, automatically determined via `nmfkc.kernel.beta.nearest.med`.
`plot`	Logical. If `TRUE` (default), plots the objective function curve.
`...`	Additional arguments. Kernel-specific args (`kernel`, `degree`) are passed to `nmfkc.kernel`; all others (`div`, `seed`, `shuffle`, `epsilon`, `maxit`, etc.) are passed to `nmfae.cv`.

Value

A list with components:

`beta`	The beta value that minimizes the cross-validation objective.
`objfunc`	Named numeric vector of objective function values for each candidate beta.

See Also

`nmfae.cv`, `nmfkc.kernel`, `nmfkc.kernel.beta.cv`

Examples

```
library(nmfkc)
Y <- matrix(cars$dist, nrow = 1)
U <- matrix(cars$speed, nrow = 1)
res <- nmfae.kernel.beta.cv(Y, Q = 1, R = 1, U = U,
                             beta = c(0.01, 0.02, 0.05), div = 5)
res$beta
```

`nmfae.rename` *Rename decoder and encoder bases*

Description

Assigns user-specified names to the decoder (X1 columns) and encoder (X2 rows) bases of an `nmfae` object. The names propagate to Θ, the coefficients table, and all downstream displays such as `summary`, `nmfae.DOT`, and `nmfae.heatmap`.

Usage

```
nmfae.rename(x, X1.colnames = NULL, X2.rownames = NULL)
```

Arguments

`x`	An object of class `"nmfae"` returned by `nmfae`.
`X1.colnames`	Character vector of length Q for decoder bases (columns of X_1 / rows of Θ). If `NULL` (default), the decoder names are left unchanged.
`X2.rownames`	Character vector of length R for encoder bases (rows of X_2 / columns of Θ). If `NULL` (default), the encoder names are left unchanged.

Value

A modified copy of `x` with updated names.

See Also

`nmfae`

Examples

```
## Not run:
res <- nmfae(Y, Q = 3, R = 3)
res <- nmfae.rename(res,
  X1.colnames = c("Sprint", "Throw", "Endurance"),
  X2.rownames = c("Speed", "Power", "Stamina"))
summary(res)
nmfae.DOT(res)

## End(Not run)
```

`nmfkc` *Optimize NMF with kernel covariates (Full Support for Missing Values)*

Description

`nmfkc` fits a nonnegative matrix factorization with kernel covariates under the tri-factorization model $Y \approx XCA = XB$.

This function supports two major input modes:

1. **Matrix Mode (Existing)**: `nmfkc(Y=matrix, A=matrix, ...)`
2. **Formula Mode (New)**: `nmfkc(formula=Y_vars ~ A_vars, data=df, rank=Q, ...)`

The rank of the basis matrix can be specified using either the `rank` argument (preferred for formula mode) or the hidden `Q` argument (for backward compatibility).

Usage

```
nmfkc(Y, A = NULL, rank = NULL, data, epsilon = 1e-04, maxit = 5000, ...)
```

Arguments

`Y`	Observation matrix (P x N), OR a formula object for Formula Mode. In Formula Mode, use `Y1 + Y2 ~ A1 + A2` with `data`, or `Y_matrix ~ A_matrix` for direct matrix evaluation. Supports dot notation (`. ~ A1 + A2`) when `data` is supplied.
`A`	Covariate matrix. Default is `NULL` (no covariates). Ignored when `Y` is a formula.
`rank`	Integer. The rank of the basis matrix X (Q). Preferred over `Q`.
`data`	Optional. A data frame from which variables in the formula should be taken.
`epsilon`	Positive convergence tolerance.
`maxit`	Maximum number of iterations.
`...`	Additional arguments passed for fine-tuning regularization, initialization, constraints, and output control. This includes the backward-compatible arguments `Q` and `method`.

- `Y.weights`: Optional numeric matrix (P x N) or vector (length N). 0 indicates missing/ignored values. If NULL (default), weights are automatically set to 0 for NAs in Y, and 1 otherwise.
- `X.L2.ortho`: Nonnegative penalty parameter for the orthogonality of X (default: 0). It minimizes the off-diagonal elements of the Gram matrix $X^\top X$, reducing the correlation between basis vectors (conceptually minimizing $\|X^\top X - \mathrm{diag}(X^\top X)\|_F^2$). (Formerly `lambda.ortho`).
- `B.L1`: Nonnegative penalty parameter for L1 regularization on $B = CA$ (default: 0). Promotes **sparsity in the coefficients**. (Formerly `gamma`).
- `C.L1`: Nonnegative penalty parameter for L1 regularization on C (default: 0). Promotes **sparsity in the parameter matrix**. (Formerly `lambda`).
- `Q`: Backward-compatible name for the rank of the basis matrix (Q).
- `method`: Objective function: Euclidean distance `"EU"` (default) or Kullback–Leibler divergence `"KL"`.

- `X.restriction`: Constraint for columns of X. Options: `"colSums"` (default), `"colSqSums"`, `"totalSum"`, `"none"`, or `"fixed"`. `"none"` applies no normalization to X after each update, allowing it to absorb the scale freely. This is automatically set when `Y.symmetric = "bi"` or `"tri"`, because column normalization would prevent $XX^\top$ (or $XCX^\top$) from approximating Y at the correct scale.
- `X.init`: Method for initializing the basis matrix X. Options: `"kmeans"` (default), `"kmeansar"`, `"runif"`, `"nndsvd"`, or a user-specified matrix. `"kmeansar"` applies k-means initialization and then fills zero entries with `Uniform(0, mean(Y)/100)`, analogous to NNDSVDar.
- `nstart`: Number of random starts for `kmeans` when initializing X (default: 1).
- `seed`: Integer seed for reproducibility (default: 123).
- `C.init`: Optional numeric matrix giving the initial value of the parameter matrix C (i.e., Θ). If `A` is `NULL`, `C` has dimension $Q \times N$ (equivalently B); otherwise, `C` has dimension $Q \times K$ where $K = nrow(A)$. Default initializes all entries to 1.
- `Y.symmetric`: Character string specifying the type of symmetric NMF. `"none"` (default): standard NMF ($Y \approx XB$). `"bi"`: 2-factor symmetric NMF ($Y \approx XX^\top$). Internally implemented as the `"tri"` model with $C = I_Q$ (identity matrix) held fixed, so that X is updated freely without column normalization. The multiplicative update for X uses cube-root damping ($X \leftarrow X \circ (numerator/denominator)^{1/3}$) to prevent oscillation, since X appears in both factors of the decomposition (He et al., 2011, Proposition 1). `"tri"`: 3-factor symmetric NMF ($Y \approx XCX^\top$) where C is a $Q \times Q$ matrix representing cluster interactions (Ding et al., 2006). Both `"bi"` and `"tri"` require `Y` to be square and cannot be used with covariate matrix `A`. When `Y.symmetric = "bi"`, `X.restriction` is automatically set to `"none"` (no column normalization), because $C = I_Q$ is fixed and cannot absorb the scale. For `"tri"`, column normalization is retained (default `"colSums"`) because the free parameter matrix C absorbs the scale.
- `prefix`: Prefix for column names of X and row names of B (default: "Basis").
- `print.trace`: Logical. If `TRUE`, prints progress every 10 iterations (default: `FALSE`).
- `print.dims`: Logical. If `TRUE` (default), prints matrix dimensions and elapsed time.
- `detail`: Level of post-fit criterion computation. `"full"` computes all criteria including silhouette, CPCC, dist.cor; `"fast"` skips expensive distance-based criteria; `"minimal"` returns only information criteria. Default is `"full"`. For backward compatibility, `save.time = TRUE` maps to `"fast"` and `save.memory = TRUE` maps to `"minimal"`.

Value

A list with components:

`call`	The matched call, as captured by `match.call()`.
`dims`	A character string summarizing the matrix dimensions of the model.
`runtime`	A character string summarizing the computation time.
`X`	Basis matrix. Column normalization depends on `X.restriction`.

`B`	Coefficient matrix $B = CA$.
`XB`	Fitted values for Y.
`C`	Parameter matrix.
`B.prob`	Soft-clustering probabilities derived from columns of B.
`B.cluster`	Hard-clustering labels (argmax over $B.prob$ for each column).
`X.prob`	Row-wise soft-clustering probabilities derived from X.
`X.cluster`	Hard-clustering labels (argmax over $X.prob$ for each row).
`A.attr`	List of attributes of the input covariate matrix `A`, containing metadata like lag order and intercept status if created by `nmfkc.ar` or `nmfkc.kernel`.
`formula.meta`	If fitted via Formula Mode, a list with `formula`, `Y_cols`, and `A_cols`; otherwise `NULL`.
`objfunc`	Final objective value.
`objfunc.iter`	Objective values by iteration.
`r.squared`	Coefficient of determination R^2 between Y and XB.
`method`	Character string indicating the optimization method used (`"EU"` or `"KL"`).
`n.missing`	Number of missing (or zero-weighted) elements in Y.
`n.total`	Total number of elements in Y.
`rank`	The rank Q used in the factorization.
`sigma`	The residual standard error, representing the typical deviation of the observed values Y from the fitted values XB.
`mae`	Mean Absolute Error between Y and XB.
`criterion`	A list of selection criteria, including `ICp` (`ICp1`, `ICp2`, `ICp3`), `CPCC`, `silhouette`, `AIC`, `BIC`, `dist.cor`, `B.prob.sd.min`, `B.prob.max.mean`, and `B.prob.entropy.mean`.

References

Satoh, K. (2024). Applying Non-negative Matrix Factorization with Covariates to the Longitudinal Data as Growth Curve Model. arXiv:2403.05359. `https://arxiv.org/abs/2403.05359`

Satoh, K. (2025). Applying non-negative matrix factorization with covariates to multivariate time series data as a vector autoregression model. *Japanese Journal of Statistics and Data Science*. arXiv:2501.17446. doi:10.1007/s42081025003140

Satoh, K. (2025). Applying non-negative matrix factorization with covariates to label matrix for classification. arXiv:2510.10375. `https://arxiv.org/abs/2510.10375`

Ding, C., Li, T., Peng, W., & Park, H. (2006). Orthogonal Nonnegative Matrix Tri-Factorizations for Clustering. In *Proc. 12th ACM SIGKDD* (pp. 126–135). doi:10.1145/1150402.1150420

See Also

`nmfkc.cv`, `nmfkc.rank`, `nmfkc.kernel`, `nmfkc.ar`, `predict.nmfkc`

Examples

```
# install.packages("remotes")
# remotes::install_github("ksatohds/nmfkc")
# Example 1. Matrix Mode (Existing)
library(nmfkc)
X <- cbind(c(1,0,1),c(0,1,0))
```

```
B <- cbind(c(1,0),c(0,1),c(1,1))
Y <- X %*% B
rownames(Y) <- paste0("P",1:nrow(Y))
colnames(Y) <- paste0("N",1:ncol(Y))
print(X); print(B); print(Y)
library(nmfkc)
res <- nmfkc(Y,Q=2,epsilon=1e-6)
res$X
res$B

# Example 2. Formula Mode
set.seed(1)
dummy_data <- data.frame(Y1=rpois(10,5), Y2=rpois(10,10),
                         A1=abs(rnorm(10,5)), A2=abs(rnorm(10,3)))
res_f <- nmfkc(Y1 + Y2 ~ A1 + A2, data=dummy_data, rank=2)

# Example 3. Symmetric NMF (bi: Y ~ X X^T)
S <- matrix(c(3,0,2, 0,3,1, 2,1,2), nrow=3)
res_bi <- nmfkc(S, Q=2, Y.symmetric="bi")
res_bi$X   # basis matrix (no column normalization)
res_bi$XB  # reconstruction X %*% t(X)

# Example 4. Symmetric NMF (tri: Y ~ X C X^T)
res_tri <- nmfkc(S, Q=2, Y.symmetric="tri")
res_tri$C  # Q x Q cluster interaction matrix
res_tri$XB # reconstruction X %*% C %*% t(X)
```

nmfkc.ar — *Construct observation and covariate matrices for a vector autoregressive model*

Description

`nmfkc.ar` generates the observation matrix and covariate matrix corresponding to a specified autoregressive lag order.

If the input `Y` is a `ts` object, its time properties are preserved in the `"tsp_info"` attribute, adjusted for the lag. Additionally, the column names of `Y` and `A` are set to the corresponding time points.

Usage

```
nmfkc.ar(Y, degree = 1, intercept = TRUE)
```

Arguments

`Y`	An observation matrix (P x N) or a `ts` object. If `Y` is a `ts` object (typically N x P), it is automatically transposed to match the (P x N) format.
`degree`	The lag order of the autoregressive model. The default is 1.
`intercept`	Logical. If TRUE (default), an intercept term is added to the covariate matrix.

Value

A list containing:

`Y`	Observation matrix (P x N_A) used for NMF. Includes adjusted `"tsp_info"` attribute and time-based column names.
`A`	Covariate matrix (R x N_A) constructed according to the specified lag order. Includes adjusted `"tsp_info"` attribute and time-based column names.
`A.columns`	Index matrix used to generate `A`.
`degree.max`	Maximum lag order.

References

Satoh, K. (2025). Applying non-negative matrix factorization with covariates to multivariate time series data as a vector autoregression model. *Japanese Journal of Statistics and Data Science*. arXiv:2501.17446. doi:10.1007/s42081025003140

See Also

`nmfkc`, `nmfkc.ar.degree.cv`, `nmfkc.ar.stationarity`, `nmfkc.ar.DOT`

Examples

```
# Example using AirPassengers (ts object)
d <- AirPassengers
ar_data <- nmfkc.ar(d, degree = 2)
dim(ar_data$Y)
dim(ar_data$A)

# Example using matrix input
Y <- matrix(1:20, nrow = 2)
ar_data <- nmfkc.ar(Y, degree = 1)
ar_data$degree.max
```

`nmfkc.ar.degree.cv` *Optimize lag order for the autoregressive model*

Description

`nmfkc.ar.degree.cv` selects the optimal lag order for an autoregressive model by applying cross-validation over candidate degrees.

This function accepts both standard matrices (Variables x Time) and `ts` objects (Time x Variables). `ts` objects are automatically transposed internally.

Usage

```
nmfkc.ar.degree.cv(
  Y,
  rank = 1,
  degree = 1:2,
  intercept = TRUE,
  plot = TRUE,
  ...
)
```

Arguments

`Y`	Observation matrix $Y(P,N)$ or a `ts` object.
`rank`	Rank of the basis matrix. For backward compatibility, `Q` is accepted via `...`.
`degree`	A vector of candidate lag orders to be evaluated.
`intercept`	Logical. If TRUE (default), an intercept is added to the covariate matrix.
`plot`	Logical. If TRUE (default), a plot of the objective function values is drawn.
`...`	Additional arguments passed to `nmfkc.cv`.

Value

A list with components:

`degree`	The lag order that minimizes the cross-validation objective function.
`degree.max`	Maximum recommended lag order, computed as $10\log_{10}(N)$ following the `ar` function in the **stats** package.
`objfunc`	Objective function values for each candidate lag order.

See Also

`nmfkc.ar`, `nmfkc.cv`

Examples

```
# install.packages("remotes")
# remotes::install_github("ksatohds/nmfkc")
# Example using ts object directly
d <- AirPassengers

# Selection of degree (using ts object)
# Note: Y is automatically transposed if it is a ts object
nmfkc.ar.degree.cv(Y=d, Q=1, degree=11:14)
```

`nmfkc.ar.DOT` *Generate a Graphviz DOT Diagram for NMF-AR / NMF-VAR Models*

Description

Produces a Graphviz DOT script for visualizing autoregressive NMF-with-covariates models constructed via `nmfkc.ar` + `nmfkc`.

The diagram displays three types of directed relationships:

- Lagged predictors: $T_{t-k} \to X$,
- Current latent factors: $X \to T_t$,
- Optional intercept effects: `Const -> X`.

Importantly, *no direct edges from lagged variables to current outputs* ($T_{t-k} \to T_t$) are drawn, in accordance with the NMF-AR formulation.

Each block of lagged variables is displayed in its own DOT subgraph (e.g., “T-1”, “T-2”, ...), while latent factor nodes and current-time outputs are arranged in separate clusters.

Usage

```
nmfkc.ar.DOT(
  x,
  degree = 1,
  intercept = FALSE,
  threshold = 0.1,
  rankdir = "RL",
  fill = TRUE,
  weight_scale_xy = 5,
  weight_scale_lag = 5,
  weight_scale_int = 3,
  hide.isolated = TRUE
)
```

Arguments

`x`	A fitted `nmfkc` object representing the AR model. Must contain matrices `X` and `C`.
`degree`	Maximum AR lag to visualize.
`intercept`	Logical; if `TRUE`, draws intercept nodes for columns named "(Intercept)" in matrix `C`.
`threshold`	Minimum coefficient magnitude required to draw an edge.
`rankdir`	Graphviz rank direction (e.g., `"RL"`, `"LR"`, `"TB"`).
`fill`	Logical; whether nodes are filled with color.
`weight_scale_xy`	Scaling factor for edges $X \to T$.
`weight_scale_lag`	Scaling factor for lagged edges $T - k \to X$.
`weight_scale_int`	Scaling factor for intercept edges.
`hide.isolated`	Logical. If `TRUE` (default), Y nodes that have no edges at or above `threshold` are excluded from the graph.

Value

A character string representing a Graphviz DOT file.

Examples

```
d <- AirPassengers
ar_data <- nmfkc.ar(d, degree = 2)
result <- nmfkc(ar_data$Y, ar_data$A, Q = 1)
dot <- nmfkc.ar.DOT(result, degree = 2)
cat(dot)
```

`nmfkc.ar.predict` *Forecast future values for NMF-VAR model*

Description

`nmfkc.ar.predict` computes multi-step-ahead forecasts for a fitted NMF-VAR model using recursive forecasting.

If the fitted model contains time series property information (from `nmfkc.ar`), the forecasted values will have appropriate time-based column names.

Usage

```
nmfkc.ar.predict(x, Y, degree = NULL, n.ahead = 1)
```

Arguments

`x`	An object of class `nmfkc` (the fitted model).
`Y`	The historical observation matrix used for fitting (or at least the last `degree` columns).
`degree`	Optional integer. Lag order (D). If `NULL` (default), it is inferred from `x$A.attr` (when available) or from the dimensions of `x$C`.
`n.ahead`	Integer (>=1). Number of steps ahead to forecast.

Value

A list with components:

`pred`	A $P \times n.ahead$ matrix of predicted values. Column names are future time points if time information is available.
`time`	A numeric vector of future time points corresponding to the columns of `pred`.

See Also

`nmfkc`, `nmfkc.ar`

Examples

```
# Forecast AirPassengers
d <- AirPassengers
ar_data <- nmfkc.ar(d, degree = 2)
result <- nmfkc(ar_data$Y, ar_data$A, Q = 1)
pred <- nmfkc.ar.predict(result, Y = matrix(d, nrow = 1), degree = 2, n.ahead = 3)
pred$pred
```

`nmfkc.ar.stationarity` *Check stationarity of an NMF-VAR model*

Description

`nmfkc.ar.stationarity` assesses the dynamic stability of a VAR model by computing the spectral radius of its companion matrix. It returns both the spectral radius and a logical indicator of stationarity.

Usage

```
nmfkc.ar.stationarity(x)
```

Arguments

`x`	The return value of `nmfkc` for a VAR model.

Value

A list with components:

`spectral.radius`	Numeric. The spectral radius of the companion matrix. A value less than 1 indicates stationarity.
`stationary`	Logical. `TRUE` if the spectral radius is less than 1 (i.e., the system is stationary), `FALSE` otherwise.

See Also

`nmfkc`, `nmfkc.ar`

Examples

```
# Check stationarity of fitted AR model
d <- AirPassengers
ar_data <- nmfkc.ar(d, degree = 2)
result <- nmfkc(ar_data$Y, ar_data$A, Q = 1)
nmfkc.ar.stationarity(result)
```

`nmfkc.class` *Create a class (one-hot) matrix from a categorical vector*

Description

`nmfkc.class` converts a categorical or factor vector into a class matrix (one-hot encoded representation), where each row corresponds to a category and each column corresponds to an observation.

Usage

```
nmfkc.class(x)
```

Arguments

`x`	A categorical vector or a factor.

Value

A binary matrix with one row per unique category and one column per observation. Each column has exactly one entry equal to 1, indicating the category of the observation.

See Also

`nmfkc`

Examples

```
# install.packages("remotes")
# remotes::install_github("ksatohds/nmfkc")
# Example.
Y <- nmfkc.class(iris$Species)
Y[,1:6]
```

`nmfkc.criterion` *Compute model selection criteria for a fitted nmfkc model*

Description

`nmfkc.criterion` computes information criteria (ICp, AIC, BIC), clustering quality measures (silhouette, CPCC, dist.cor), and soft-clustering statistics (B.prob entropy, max, sd) from a fitted `nmfkc` model.

This function can be called on a model that was fitted with `detail = "fast"` or `detail = "minimal"` to compute the full set of criteria afterwards.

Usage

```
nmfkc.criterion(object, Y, detail = c("full", "fast", "minimal"), ...)
```

Arguments

`object`	An object of class `"nmfkc"` returned by `nmfkc`.
`Y`	The original observation matrix (P x N) used for fitting.
`detail`	Character string controlling the level of computation: `"full"` (default) computes all criteria including silhouette, CPCC and dist.cor; `"fast"` skips the expensive distance-based criteria; `"minimal"` returns only information criteria.
`...`	Additional arguments: `Y.weights` (weight matrix, default: all ones).

Value

A list with components:

r.squared R-squared between Y and XB.

sigma Residual standard deviation.

mae Mean absolute error.

B.prob Column-normalized coefficient matrix (soft-clustering probabilities).

B.cluster Hard clustering labels (argmax of B.prob per column).

X.prob Row-normalized basis matrix.

X.cluster Hard clustering labels per row of X.

criterion Named list: ICp, ICp1, ICp2, ICp3, AIC, BIC, B.prob.sd.min, B.prob.max.mean, B.prob.entropy.mean, silhouette, CPCC, dist.cor.

See Also

`nmfkc`, `nmfkc.rank`

Examples

```
Y <- t(iris[, -5])
res <- nmfkc(Y, Q = 3, detail = "fast")
crit <- nmfkc.criterion(res, Y)
crit$criterion$silhouette
```

`nmfkc.cv` *Perform k-fold cross-validation for NMF with kernel covariates*

Description

`nmfkc.cv` performs k-fold cross-validation for the tri-factorization model $Y \approx XCA = XB$, where

- $Y(P, N)$ is the observation matrix,
- $A(R, N)$ is the covariate (or kernel) matrix,
- $X(P, Q)$ is the basis matrix,
- $C(Q, R)$ is the parameter matrix, and
- $B(Q, N)$ is the coefficient matrix ($B = CA$).

Given Y (and optionally A), X and C are fitted on each training split and predictive performance is evaluated on the held-out split.

Usage

```
nmfkc.cv(Y, A = NULL, Q = 2, data, ...)
```

Arguments

`Y`	Observation matrix, or a formula (see `nmfkc` for Formula Mode).
`A`	Covariate matrix. If `NULL`, the identity matrix is used. Ignored when `Y` is a formula.
`Q`	Rank of the basis matrix X.
`data`	A data frame (required when `Y` is a formula with column names).
`...`	Additional arguments controlling CV and the internal `nmfkc` call:

- `Y.weights` Optional numeric matrix or vector; 0 indicates missing/ignored values.
- `div` Number of folds (k); default: `5`.
- `seed` Integer seed for reproducible partitioning; default: `123`.
- `shuffle` Logical. If `TRUE` (default), randomly shuffles samples (standard CV); if `FALSE`, splits sequentially (block CV; recommended for time series).
- ***Arguments passed to*** **`nmfkc`** e.g., `gamma` (`B.L1`), `epsilon`, `maxit`, `method` (`"EU"` or `"KL"`), `X.restriction`, `X.init`, etc.

Value

A list with components:

- `objfunc` Mean loss per valid entry over all folds (MSE for `method="EU"`).
- `sigma` Residual standard error (RMSE). Available only if `method="EU"`; on the same scale as `Y`.
- `objfunc.block` Loss for each fold.
- `block` Vector of fold indices (1, …, `div`) assigned to each column of Y.

See Also

`nmfkc`, `nmfkc.kernel.beta.cv`, `nmfkc.ar.degree.cv`

Examples

```
# Example 1 (with explicit covariates):
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
res <- nmfkc.cv(Y, A, Q = 1)
res$objfunc

# Example 2 (kernel A and beta sweep):
Y <- matrix(cars$dist, nrow = 1)
U <- matrix(c(5, 10, 15, 20, 25), nrow = 1)
V <- matrix(cars$speed, nrow = 1)
betas <- 25:35/1000
obj <- numeric(length(betas))
for (i in seq_along(betas)) {
  A <- nmfkc.kernel(U, V, beta = betas[i])
  obj[i] <- nmfkc.cv(Y, A, Q = 1, div = 10)$objfunc
}
betas[which.min(obj)]
```

nmfkc.denormalize *Denormalize a matrix from $[0, 1]$ back to its original scale*

Description

`nmfkc.denormalize` rescales a matrix with values in $[0, 1]$ back to its original scale using the column-wise minima and maxima of a reference matrix.

Usage

```
nmfkc.denormalize(x, ref = x)
```

Arguments

`x`	A numeric matrix (or vector) with values in $[0, 1]$ to be denormalized.
`ref`	A reference matrix used to obtain the original column-wise minima and maxima. Must have the same number of columns as `x`.

Value

A numeric matrix with values transformed back to the original scale.

See Also

`nmfkc.normalize`

Examples

```
x <- nmfkc.normalize(iris[, -5])
x_recovered <- nmfkc.denormalize(x, iris[, -5])
apply(x_recovered - iris[, -5], 2, max)
```

nmfkc.DOT *Generate Graphviz DOT Scripts for NMF or NMF-with-Covariates Models*

Description

Produces a Graphviz DOT script visualizing the structure of an NMF model ($Y \approx XCA$) or its simplified forms.

Supported visualization types:

- `"YX"` — Standard NMF view: latent factors X map to observations Y.
- `"YA"` — Direct regression view: covariates A map directly to Y using the combined coefficient matrix XC.
- `"YXA"` — Full tri-factorization: $A \to C \to X \to Y$.

Edge widths are scaled by coefficient magnitude, and nodes with no edges above the threshold are omitted from the visualization.

Usage

```
nmfkc.DOT(
  x,
  type = c("YX", "YA", "YXA"),
  threshold = 0.01,
  sig.level = 0.1,
  rankdir = "LR",
  fill = TRUE,
  weight_scale = 5,
  weight_scale_ax = weight_scale,
  weight_scale_xy = weight_scale,
  weight_scale_ay = weight_scale,
  Y.label = NULL,
  X.label = NULL,
  A.label = NULL,
  Y.title = "Observation (Y)",
  X.title = "Basis (X)",
  A.title = "Covariates (A)",
  hide.isolated = TRUE
)
```

Arguments

`x`	The return value from `nmfkc`, containing matrices `X`, `B`, and optionally `C`.
`type`	Character string specifying the visualization style: one of `"YX"`, `"YA"`, `"YXA"`.
`threshold`	Minimum coefficient magnitude to display an edge.
`sig.level`	Significance level for filtering C edges when inference results are available (i.e., `x` is of class `"nmfkc.inference"`). Only edges with p-value below `sig.level` are shown, decorated with significance stars (`*`, `**`, `***`). Set to `NULL` to disable filtering and show all edges above `threshold`. Default is 0.1.
`rankdir`	Graphviz rank direction (e.g., `"LR"`, `"TB"`).
`fill`	Logical; whether nodes should be drawn with filled shapes.
`weight_scale`	Base scaling factor for edge widths.
`weight_scale_ax`	Scaling factor for edges $A \to X$ (type `"YXA"`).
`weight_scale_xy`	Scaling factor for edges $X \to Y$.
`weight_scale_ay`	Scaling factor for edges $A \to Y$ (type `"YA"`).
`Y.label`	Optional character vector for labels of Y nodes.
`X.label`	Optional character vector for labels of X (latent factor) nodes.
`A.label`	Optional character vector for labels of A (covariate) nodes.
`Y.title`	Cluster title for Y nodes.
`X.title`	Cluster title for X nodes.
`A.title`	Cluster title for A nodes.
`hide.isolated`	Logical. If `TRUE` (default), Y and A nodes that have no edges at or above `threshold` are excluded from the graph.

Value

A character string representing a Graphviz DOT script.

See Also

`nmfkc`

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
result <- nmfkc(Y, A, Q = 1)
dot <- nmfkc.DOT(result)
cat(dot)
```

`nmfkc.ecv` *Perform Element-wise Cross-Validation (Wold's CV)*

Description

`nmfkc.ecv` performs k-fold cross-validation by randomly holding out individual elements of the data matrix (element-wise), assigning them a weight of 0 via `Y.weights`, and evaluating the reconstruction error on those held-out elements.

This method (also known as Wold's CV) is theoretically robust for determining the optimal rank (Q) in NMF. This function supports vector input for `Q`, allowing simultaneous evaluation of multiple ranks on the same folds.

When `Y.symmetric = "bi"` or `"tri"` is passed via `...`, fold creation uses only the upper triangle (including the diagonal) to prevent information leakage through the symmetric entries $Y_{ij} = Y_{ji}$.

Usage

```
nmfkc.ecv(Y, A = NULL, Q = 1:3, nfolds = 5, seed = 123, data, ...)
```

Arguments

`Y`	Observation matrix, or a formula (see `nmfkc` for Formula Mode).
`A`	Covariate matrix. Ignored when `Y` is a formula.
`Q`	Vector of ranks to evaluate (e.g., 1:5).
`nfolds`	Number of folds (default: 5). For backward compatibility, `div` is accepted via `...`.
`seed`	Integer seed for reproducibility.
`data`	A data frame (required when `Y` is a formula with column names).
`...`	Additional arguments passed to `nmfkc` (e.g., `method="EU"`).

Value

A list with components:

`objfunc`	Numeric vector containing the Mean Squared Error (MSE) for each Q.
`sigma`	Numeric vector containing the Residual Standard Error (RMSE) for each Q. Only available if method="EU".
`objfunc.fold`	List of length equal to Q vector. Each element contains the MSE values for the k folds.
`folds`	A list of length `div`, containing the linear indices of held-out elements for each fold (shared across all Q).

See Also

`nmfkc`, `nmfkc.cv`

Examples

```
# Element-wise CV to select rank
Y <- t(iris[1:30, 1:4])
res <- nmfkc.ecv(Y, Q = 1:2, div = 3)
res$objfunc
```

`nmfkc.inference` *Statistical inference for the parameter matrix C (Theta)*

Description

`nmfkc.inference` performs statistical inference on the parameter matrix C (Θ) from a fitted `nmfkc` model, conditional on the estimated basis matrix $\hat{X}$.

Under the working model $Y = XCA + \varepsilon$ where $\varepsilon_{pn} \overset{iid}{\sim} N(0, \sigma^2)$, inference is conducted via sandwich covariance estimation and one-step wild bootstrap with non-negative projection.

Usage

```
nmfkc.inference(object, Y, A = NULL, wild.bootstrap = TRUE, ...)
```

Arguments

`object`	An object of class "`nmfkc`" returned by `nmfkc`.
`Y`	Observation matrix (P x N). Must match the data used in `nmfkc()`.
`A`	Covariate matrix (K x N). Default is `NULL` (same as identity; in this case $B = C$ and inference is on B directly).
`wild.bootstrap`	Logical. If `TRUE` (default), performs wild bootstrap for confidence intervals and bootstrap standard errors. Set to `FALSE` to skip bootstrap (faster, only sandwich SE is computed).
`...`	Additional arguments: `wild.B` Number of bootstrap replicates. Default is 1000. `wild.seed` Seed for bootstrap. Default is 42.

`wild.level`	Confidence level for bootstrap CI. Default is 0.95.
`sandwich`	Logical. Use sandwich covariance. Default is `TRUE`.
`C.p.side`	P-value type: `"one.sided"` (default) or `"two.sided"`.
`cov.ridge`	Ridge stabilization for information matrix inversion. Default is 1e-8.
`print.trace`	Logical. If `TRUE`, prints progress. Default is `FALSE`.

Value

An object of class `c("nmfkc.inference", "nmfkc")`, inheriting all components from the input `object`, with additional inference components:

`sigma2.used`	Estimated σ^2 used for inference.
`C.se`	Sandwich standard errors for C (Q x K matrix).
`C.se.boot`	Bootstrap standard errors for C (Q x K matrix).
`C.ci.lower`	Lower CI bounds for C (Q x K matrix).
`C.ci.upper`	Upper CI bounds for C (Q x K matrix).
`coefficients`	Data frame with Estimate, SE, BSE, z, p-value for each element of C.
`C.p.side`	P-value type used.

References

Satoh, K. (2026). Wild Bootstrap Inference for Non-Negative Matrix Factorization with Random Effects. arXiv:2603.01468. https://arxiv.org/abs/2603.01468

See Also

`nmfkc`, `summary.nmfkc.inference`

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(intercept = 1, speed = cars$speed)
result <- nmfkc(Y, A, Q = 1)
result2 <- nmfkc.inference(result, Y, A)
summary(result2)
```

`nmfkc.kernel` *Create a kernel matrix from covariates*

Description

`nmfkc.kernel` constructs a kernel matrix from covariate matrices. It supports Gaussian, Exponential, Periodic, Linear, Normalized Linear, and Polynomial kernels.

Usage

```
nmfkc.kernel(
  U,
  V = NULL,
 kernel = c("Gaussian", "Exponential", "Periodic", "Linear", "NormalizedLinear",
   "Polynomial"),
  ...
)
```

Arguments

`U`	Covariate matrix $U(K,N) = (u_1,\dots,u_N)$. Each row may be normalized in advance.
`V`	Covariate matrix $V(K,M) = (v_1,\dots,v_M)$, typically used for prediction. If `NULL`, the default is `U`.
`kernel`	Kernel function to use. Default is `"Gaussian"`. Options are `"Gaussian"`, `"Exponential"`, `"Periodic"`, `"Linear"`, `"NormalizedLinear"`, and `"Polynomial"`.
`...`	Additional arguments passed to the specific kernel function (e.g., `beta`, `degree`).

Value

Kernel matrix $A(N,M)$.

Source

Satoh, K. (2024). Applying Non-negative Matrix Factorization with Covariates to the Longitudinal Data as Growth Curve Model. arXiv preprint arXiv:2403.05359. https://arxiv.org/abs/2403.05359

See Also

`nmfkc.kernel`, `nmfkc.cv`

Examples

```
# install.packages("remotes")
# remotes::install_github("ksatohds/nmfkc")
# Example.
Y <- matrix(cars$dist,nrow=1)
U <- matrix(c(5,10,15,20,25),nrow=1)
V <- matrix(cars$speed,nrow=1)
A <- nmfkc.kernel(U,V,beta=28/1000)
dim(A)
result <- nmfkc(Y,A,Q=1)
plot(as.vector(V),as.vector(Y))
lines(as.vector(V),as.vector(result$XB),col=2,lwd=2)
```

`nmfkc.kernel.beta.cv` *Optimize beta of the Gaussian kernel function by cross-validation*

Description

`nmfkc.kernel.beta.cv` selects the optimal beta parameter of the kernel function by applying cross-validation over a set of candidate values.

Usage

```
nmfkc.kernel.beta.cv(Y, Q = 2, U, V = NULL, beta = NULL, plot = TRUE, ...)
```

Arguments

`Y`	Observation matrix $Y(P, N)$.
`Q`	Rank of the basis matrix.
`U`	Covariate matrix $U(K, N) = (u_1, \dots, u_N)$. Each row may be normalized in advance.
`V`	Covariate matrix $V(K, M) = (v_1, \dots, v_M)$, typically used for prediction. If `NULL`, the default is `U`.
`beta`	A numeric vector of candidate kernel parameters to evaluate via cross-validation.
`plot`	Logical. If TRUE (default), plots the objective function values for each candidate `beta`.
`...`	Additional arguments passed to `nmfkc.cv`.

Value

A list with components:

`beta`	The beta value that minimizes the cross-validation objective function.
`objfunc`	Objective function values for each candidate `beta`.

Examples

```
# install.packages("remotes")
# remotes::install_github("ksatohds/nmfkc")
# Example.
Y <- matrix(cars$dist,nrow=1)
U <- matrix(c(5,10,15,20,25),nrow=1)
V <- matrix(cars$speed,nrow=1)
nmfkc.kernel.beta.cv(Y,Q=1,U,V,beta=25:30/1000)
A <- nmfkc.kernel(U,V,beta=28/1000)
result <- nmfkc(Y,A,Q=1)
plot(as.vector(V),as.vector(Y))
lines(as.vector(V),as.vector(result$XB),col=2,lwd=2)
```

`nmfkc.kernel.beta.nearest.med` *Estimate Gaussian/RBF kernel parameter beta from covariates (supports landmarks)*

Description

Computes a data-driven reference scale for the Gaussian/RBF kernel from covariates using a robust "median nearest-neighbor (or nearest-landmark) distance" heuristic, and returns the corresponding kernel parameter β.

The Gaussian/RBF kernel is assumed to be written in the form

$$k(u, v) = \exp\{-\beta\|u - v\|^2\} = \exp\{-\|u - v\|^2/(2\sigma^2)\},$$

hence $\beta = 1/(2\sigma^2)$. This function first estimates a typical distance scale σ_0 by the median of distances, then sets $\beta_0 = 1/(2\sigma_0^2)$.

If `Uk` is `NULL`, σ_0 is estimated as the median of nearest-neighbor distances within `U` (excluding self-distance). If `Uk` is provided, σ_0 is estimated as the median of nearest-landmark distances from each sample in `U` to its closest landmark in `Uk`.

To control memory usage for large `N` (and `M`), distances are computed in blocks. Optionally, columns of `U` can be randomly subsampled via `sample.size` to reduce cost.

Usage

```
nmfkc.kernel.beta.nearest.med(
  U,
  Uk = NULL,
  block.size = 1000,
  block.size.Uk = 2000,
  sample.size = NULL,
  ...
)
```

Arguments

`U`	A numeric matrix of covariates (`K x N`); columns are samples.
`Uk`	An optional numeric matrix of landmarks (`K x M`); columns are landmark points. If provided, distances are computed from samples in `U` to landmarks in `Uk`.
`block.size`	Integer. Number of columns of `U` processed per block when computing distances (controls memory usage). If `N <= 1000`, it is automatically set to `N`.
`block.size.Uk`	Integer. Number of columns of `Uk` processed per block when `Uk` is not `NULL` (controls memory usage). If `M <= 2000`, it is automatically set to `M`.
`sample.size`	Integer or `NULL`. If not `NULL`, randomly subsamples this many columns of `U` (without replacement) before computing distances, to reduce computational cost.

Details

Candidate grid: Along with `beta`, the function returns `beta_candidates`, a small logarithmic grid suitable for cross-validation.

In the landmark case (`Uk` provided), the grid is designed to be symmetric on the bandwidth scale σ around σ_0 over one decade:

$$\sigma = \sigma_0 \times 10^t, \quad t \in \{-1, -2/3, -1/3, 0, 1/3, 2/3, 1\}.$$

Using $\beta = 1/(2\sigma^2)$, this corresponds to

$$\beta = \beta_0 \times 10^{-2t}.$$

When `Uk` is `NULL`, a shorter coarse grid may be returned (see `Value`).

Notes:

- When `Uk` is identical to `U`, the function detects this case and excludes self-distances (distance 0) to avoid $\sigma_0 = 0$.
- `sample.size` performs random subsampling without setting a seed. For reproducible results, set `set.seed()` before calling this function.

Value

A list with elements:

- `beta`: Estimated kernel parameter $\beta_0 = 1/(2\sigma_0^2)$.
- `beta_candidates`: Numeric vector of candidate β values (logarithmic grid) intended for cross-validation.
- `dist_median`: The estimated distance scale σ_0 (median of nearest-neighbor or nearest-landmark distances).
- `block.size.used`: The effective block size(s) used. Either a scalar (no `Uk`) or a named vector `c(U=..., Uk=...)` when `Uk` is provided.
- `sample.size.used`: The number of columns of `U` actually used (after subsampling).
- `uk_is_u`: Logical flag indicating whether `Uk` was detected as identical to `U` (only returned when `Uk` is provided).

Examples

```
# Basic (nearest-neighbor within U)
# beta_info <- nmfkc.kernel.beta.nearest.med(U)
# beta0 <- beta_info$beta
# betas <- beta_info$beta_candidates

# With landmarks (nearest-landmark distances)
# beta_info <- nmfkc.kernel.beta.nearest.med(U, Uk)
# beta0 <- beta_info$beta
# betas <- beta_info$beta_candidates
```

`nmfkc.kernel.gaussian` *Create a Gaussian kernel matrix from covariates*

Description

`nmfkc.kernel.gaussian` constructs a Gaussian (RBF) kernel matrix from covariate matrices. The kernel is defined as $K(u, v) = \exp(-\beta\|u - v\|^2)$. When `V` contains `NA` values, two methods are available via `na.method`:

`"pds"` Partial Distance Strategy. Computes the kernel using only observed (non-NA) rows, with beta adjusted by $\beta_{adj} = \beta \times K/K_{obs}$ where K is the total number of rows and K_{obs} is the number of observed rows.

`"egk"` Expected Gaussian Kernel (Mesquita et al., 2019). Uses a Gaussian Mixture Model (GMM) to estimate the conditional distribution of missing values given observed values, then computes the expected kernel value via a Gamma approximation. Requires `gmm.means`, `gmm.sigmas`, and `gmm.weights` passed through `...`.

Usage

```
nmfkc.kernel.gaussian(
  U,
  V = NULL,
  beta = 0.5,
  na.method = c("pds", "egk"),
  ...
)
```

Arguments

`U`	Covariate matrix $U(K, N) = (u_1, \ldots, u_N)$. Each row may be normalized in advance.
`V`	Covariate matrix $V(K, M) = (v_1, \ldots, v_M)$, typically used for prediction. If `NULL`, the default is `U`. May contain `NA` values.
`beta`	Bandwidth parameter for the Gaussian kernel. Default is `0.5`.
`na.method`	Method for handling `NA` values in `V`. Either `"pds"` or `"egk"`. Ignored if `V` has no `NA`.
`...`	Additional arguments for EGK method: `gmm.G` Number of GMM components for EGK. Default is `3` (Mesquita et al., 2019).

Value

Kernel matrix $A(N, M)$.

Source

Mesquita, D., Gomes, J. P., & Rodrigues, L. R. (2019). Gaussian kernels for incomplete data. *Applied Soft Computing*, 77, 356–365.

See Also

nmfkc.kernel, nmfkc.kernel.beta.cv, nmfkc.kernel.beta.nearest.med

Examples

```
U <- matrix(c(5,10,15,20,25),nrow=1)
V <- matrix(1:25,nrow=1)
A <- nmfkc.kernel.gaussian(U,V,beta=28/1000)
dim(A)

# PDS example: V with NA in first row
U2 <- matrix(rnorm(20), nrow=2)
V2 <- matrix(rnorm(10), nrow=2)
V2[1, c(2,4)] <- NA
A2 <- nmfkc.kernel.gaussian(U2, V2, beta=0.5, na.method="pds")
```

`nmfkc.normalize` *Normalize a matrix to the range* $[0, 1]$

Description

`nmfkc.normalize` rescales the values of a matrix to lie between 0 and 1 using the column-wise minimum and maximum values of a reference matrix.

Usage

```
nmfkc.normalize(x, ref = x)
```

Arguments

`x`	A numeric matrix (or vector) to be normalized.
`ref`	A reference matrix from which the column-wise minima and maxima are taken. Default is `x`.

Value

A matrix of the same dimensions as `x`, with each column rescaled to the $[0, 1]$ range.

See Also

nmfkc.denormalize

Examples

```
# install.packages("remotes")
# remotes::install_github("ksatohds/nmfkc")
# Example.
x <- nmfkc.normalize(iris[,-5])
apply(x,2,range)
```

nmfkc.rank — *Rank selection diagnostics with graphical output*

Description

`nmfkc.rank` provides diagnostic criteria for selecting the rank (Q) in NMF with kernel covariates. Several model selection measures are computed (e.g., R-squared, silhouette, CPCC, ARI), and results can be visualized in a plot.

By default (`save.time = FALSE`), this function also computes the Element-wise Cross-Validation error (Wold's CV Sigma) using `nmfkc.ecv`.

The plot explicitly marks the "BEST" rank based on two criteria:

1. **Elbow Method (Red)**: Based on the curvature of the R-squared values (always computed if Q > 2).
2. **Min RMSE (Blue)**: Based on the minimum Element-wise CV Sigma (only if `save.time=FALSE`).

Usage

```
nmfkc.rank(
  Y,
  A = NULL,
  rank = 1:2,
  detail = "full",
  save.time = FALSE,
  plot = TRUE,
  data,
  ...
)
```

Arguments

`Y`	Observation matrix, or a formula (see `nmfkc` for Formula Mode).
`A`	Covariate matrix. If `NULL`, the identity matrix is used. Ignored when `Y` is a formula.
`rank`	A vector of candidate ranks to be evaluated.
`detail`	Level of criterion computation: `"full"` (default) computes all criteria including ECV; `"fast"` skips ECV and distance-based criteria.
`save.time`	Logical. Backward-compatible alias: `TRUE` maps to `detail = "fast"`. Default is `FALSE`.
`plot`	Logical. If `TRUE` (default), draws a plot of the diagnostic criteria.
`data`	A data frame (required when `Y` is a formula with column names).
`...`	Additional arguments passed to `nmfkc` and `nmfkc.ecv`. • `Q`: (Deprecated) Alias for `rank`.

Value

A list containing:

`rank.best`	The estimated optimal rank. Prioritizes ECV minimum if available, otherwise R-squared Elbow.
`criteria`	A data frame containing diagnostic metrics for each rank.

References

Brunet, J.P., Tamayo, P., Golub, T.R., Mesirov, J.P. (2004). Metagenes and molecular pattern discovery using matrix factorization. *Proc. Natl. Acad. Sci. USA*, 101, 4164–4169. doi:10.1073/pnas.0308531101 Punera, K., & Ghosh, J. (2008). Consensus-based ensembles of soft clusterings. *Applied Artificial Intelligence*, 22(7–8), 780–810. doi:10.1080/08839510802170546

See Also

`nmfkc`, `nmfkc.ecv`

Examples

```
# install.packages("remotes")
# remotes::install_github("ksatohds/nmfkc")
# Example.
library(nmfkc)
Y <- t(iris[,-5])
# Full run (default)
nmfkc.rank(Y, rank=1:4)
# Fast run (skip ECV)
nmfkc.rank(Y, rank=1:4, save.time=TRUE)
```

`nmfkc.residual.plot` *Plot Diagnostics: Original, Fitted, and Residual Matrices as Heatmaps*

Description

This function generates a side-by-side plot of three heatmaps: the original observation matrix Y, the fitted matrix XB (from NMF), and the residual matrix E (Y - XB). This visualization aids in diagnosing whether the chosen rank Q is adequate by assessing if the residual matrix E appears to be random noise.

The axis labels (X-axis: Samples, Y-axis: Features) are integrated into the main title of each plot to maximize the plot area, reflecting the compact layout settings.

Usage

```
nmfkc.residual.plot(
  Y,
  result,
 fitted.palette = (grDevices::colorRampPalette(c("white", "orange", "red")))(256),
 residual.palette = (grDevices::colorRampPalette(c("blue", "white", "red")))(256),
  ...
)
```

Arguments

`Y`	The original observation matrix (P x N).
`result`	The result object returned by the nmfkc function.
`fitted.palette`	A vector of colors for Y and XB heatmaps. Defaults to white-orange-red. For backward compatibility, `Y_XB_palette` is accepted via `...`.

`residual.palette`	A vector of colors for the residuals heatmap. Defaults to blue-white-red. For backward compatibility, `E_palette` is accepted via `...`.
`...`	Additional graphical parameters passed to the internal image calls.

Value

NULL. The function generates a plot.

Examples

```
Y <- t(iris[1:30, 1:4])
result <- nmfkc(Y, Q = 2)
nmfkc.residual.plot(Y, result)
```

`nmfre` *Non-negative Matrix Factorization with Random Effects*

Description

Estimates the NMF-RE model

$$Y = X(\Theta A + U) + \mathcal{E}$$

where Y $(P \times N)$ is a non-negative observation matrix, X $(P \times Q)$ is a non-negative basis matrix learned from the data, Θ $(Q \times K)$ is a non-negative coefficient matrix capturing systematic covariate effects on latent scores, A $(K \times N)$ is a covariate matrix, and U $(Q \times N)$ is a random effects matrix capturing unit-specific deviations in the latent score space.

NMF-RE can be viewed as a mixed-effects latent-variable model defined on a reconstruction (mean) structure. The non-negativity constraint on X induces sparse, parts-based loadings, achieving measurement-side variable selection without an explicit sparsity penalty. Inference on Θ provides covariate-side variable selection by identifying which covariates significantly affect which components.

Estimation alternates ridge-type BLUP-like closed-form updates for U with multiplicative non-negative updates for X and Θ. The effective degrees of freedom consumed by U are monitored and a df-based cap can be enforced to prevent near-saturated fits.

When `wild.bootstrap = TRUE`, inference on Θ is performed conditional on $(\hat{X}, \hat{U})$ via asymptotic linearization, a one-step Newton update, and a multiplier (wild) bootstrap, yielding standard errors, z-values, p-values, and confidence intervals without repeated constrained re-optimization.

Usage

```
nmfre(
  Y,
  A = NULL,
  rank = 2,
  df.rate = NULL,
  wild.bootstrap = TRUE,
  epsilon = 1e-05,
  maxit = 50000,
  ...
)
```

Arguments

`Y`	Observation matrix (P x N), non-negative.
`A`	Covariate matrix (K x N). Default is a row of ones (intercept only).
`rank`	Integer. Rank of the basis matrix X. Default is 2. For backward compatibility, `Q` is accepted via `...`.
`df.rate`	Rate for computing the dfU cap (`cap = rate * N * Q`). For backward compatibility, `dfU.cap.rate` is accepted via `...`. If `NULL` (default), runs `nmfre.dfU.scan` internally and selects the minimum rate where the cap is not binding. Use `nmfre.dfU.scan` beforehand to examine dfU behavior across rates and choose an appropriate value.
`wild.bootstrap`	Logical. If `TRUE` (default), perform wild bootstrap inference on Θ.
`epsilon`	Convergence tolerance for relative change in objective (default 1e-5).
`maxit`	Maximum number of iterations (default 50000).
`...`	Additional arguments for initialization, variance control, dfU control, optimization, and inference settings.

- `X.init`: Initial basis matrix (P x Q), or `NULL`. When `NULL`, `nmfkc` is called internally to generate initial values.
- `C.init`: Initial coefficient matrix (Q x K), or `NULL`. When `NULL`, `nmfkc` is called internally to generate initial values.
- `U.init`: Initial random effects matrix (Q x N), or `NULL` (all zeros).
- `prefix`: Prefix for basis names (default `"Basis"`).
- `sigma2`: Initial residual variance (default 1).
- `sigma2.update`: Logical. Update σ^2 during iterations (default `TRUE`).
- `tau2`: Initial random effect variance (default 1).
- `tau2.update`: Logical. Update τ^2 by moment matching (default `TRUE`). Disabled when dfU cap is active.
- `dfU.control`: Either `"cap"` (default) to enforce a cap on dfU, or `"off"` for no cap.
- `print.trace`: Logical. If `TRUE`, print progress every 100 iterations (default `FALSE`).
- `seed`: Integer seed for reproducibility (default 1).
- `C.p.side`: P-value sidedness: `"one.sided"` (default, for boundary null H0: C=0 vs H1: C>0) or `"two.sided"`.
- `wild.B`: Number of wild bootstrap replicates (default 500).
- `wild.seed`: Seed for wild bootstrap (default 123).

Value

A list of class `"nmfre"` with components. The model is $Y = X(\Theta A + U) + \mathcal{E}$.

Core matrices

`X` Basis matrix X ($P \times Q$), columns normalized to sum to 1.

`C` Coefficient matrix Θ ($Q \times K$).

`U` Random effects matrix U ($Q \times N$).

Variance components

`sigma2` Residual variance $\hat{\sigma}^2$.

`tau2` Random effect variance $\hat{\tau}^2$.

`lambda` Ridge penalty $\lambda = \sigma^2/\tau^2$.

Convergence diagnostics

`converged` Logical. Whether the algorithm converged.

`stop.reason` Character string describing why iteration stopped.

`iter` Number of iterations performed.

`maxit` Maximum iterations setting used.

`epsilon` Convergence tolerance used.

`objfunc` Final objective function value $\|Y - X(\Theta A + U)\|^2 + \lambda\|U\|^2$.

`rel.change.final` Final relative change in objective.

`objfunc.iter` Numeric vector of objective values per iteration.

`rss.trace` Numeric vector of $\|Y - X(\Theta A + U)\|^2$ per iteration.

Effective degrees of freedom (dfU) diagnostics

`dfU` Final effective degrees of freedom $\mathrm{df}_U = N\sum_q d_q/(d_q + \lambda)$, where d_q are eigenvalues of $X'X$.

`dfU.cap` Upper bound imposed on df_U.

`dfU.cap.rate` Rate used to compute the cap.

`dfU.cap.scan` Result of `nmfre.dfU.scan`, or `NULL`.

`lambda.enforced` Final λ enforced to satisfy the cap.

`dfU.hit.cap` Logical. Whether the cap was binding.

`dfU.hit.iter` Iteration at which the cap first bound.

`dfU.frac` $\mathrm{df}_U/(NQ)$, fraction of maximum df.

`dfU.cap.frac` $\mathrm{df}_U^{\mathrm{cap}}/(NQ)$.

Fitted matrices

`B` Fixed-effect scores ΘA $(Q \times N)$.

`B.prob` Column-normalized probabilities from $\max(\Theta A, 0)$.

`B.blup` BLUP scores $\Theta A + U$ $(Q \times N)$.

`B.blup.pos` Non-negative BLUP scores $\max(\Theta A + U, 0)$ $(Q \times N)$.

`B.blup.prob` Column-normalized probabilities from $\max(\Theta A + U, 0)$.

`XB` Fitted values from fixed effects $X\Theta A$ $(P \times N)$.

`XB.blup` Fitted values including random effects $X(\Theta A + U)$ $(P \times N)$.

Fit statistics

`r.squared` Coefficient of determination R^2 for Y vs $X(\Theta A + U)$ (BLUP).

`r.squared.fixed` Coefficient of determination R^2 for Y vs $X\Theta A$ (fixed effects only).

`ICC` Trace-based Intraclass Correlation Coefficient. In the NMF-RE model, the conditional covariance of the n-th observation column is $\mathrm{Var}(Y_n) = \tau^2 XX^\top + \sigma^2 I_P$, a $P \times P$ matrix. Unlike a standard random intercept model where the design matrix Z is a simple indicator (so the ICC reduces to $\tau^2/(\sigma^2 + \tau^2)$), the basis matrix X plays the role of Z in a random slopes model, making the variance contribution of U depend on X. To obtain a scalar summary, we take the trace of each component:

$$\mathrm{ICC} = \frac{\tau^2\,\mathrm{tr}(X^\top X)}{\tau^2\,\mathrm{tr}(X^\top X) + \sigma^2 P}.$$

This equals the average (over P dimensions) proportion of per-column variance attributable to the random effects.

Inference on Θ (wild bootstrap)

`sigma2.used` $\hat{\sigma}^2$ used for inference.

`C.vec.cov` Variance-covariance matrix for $\mathrm{vec}(\Theta)$ $(QK \times QK)$.

`C.se` Standard error matrix for Θ $(Q \times K)$.

`C.se.hess` Sandwich (Hessian-based) SE matrix for Θ.

`C.se.boot` Bootstrap SE matrix for Θ.

`coefficients` Data frame with columns Estimate, Std. Error, z value, Pr(>|z|), and confidence interval bounds for each element of Θ.

`C.ci.lower` Lower confidence interval matrix for Θ $(Q \times K)$.

`C.ci.upper` Upper confidence interval matrix for Θ $(Q \times K)$.

`C.boot.sd` Bootstrap standard deviation matrix for Θ $(Q \times K)$.

`C.p.side` P-value sidedness used: `"one.sided"` or `"two.sided"`.

References

Satoh, K. (2026). Wild Bootstrap Inference for Non-Negative Matrix Factorization with Random Effects. arXiv:2603.01468. https://arxiv.org/abs/2603.01468

Examples

```
# Example 1. cars data
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(intercept = 1, speed = cars$speed)
res <- nmfre(Y, A, Q = 1, maxit = 5000)
summary(res)


# Example 2. Orthodont data (nlme)
library(nlme)
Y <- matrix(Orthodont$distance, 4, 27)
male <- ifelse(Orthodont$Sex[seq(1, 108, 4)] == "Male", 1, 0)
A <- rbind(intercept = 1, male = male)

# Scan dfU cap rates to choose an appropriate value
nmfre.dfU.scan(1:10/10, Y, A, Q = 1)

# Fit with chosen rate
res <- nmfre(Y, A, Q = 1, dfU.cap.rate = 0.2)
summary(res)
```

nmfre.dfU.scan *Scan dfU cap rates for NMF-RE*

Description

Fits the NMF-RE model across a range of `dfU.cap.rate` values and returns a diagnostic table showing the resulting effective degrees of freedom, variance components, and convergence diagnostics for each rate.

The dfU cap limits the effective degrees of freedom consumed by the random effects U. The cap is computed as `rate * N * Q`, where N is the number of observations and Q is the rank. A suitable rate is one where the final df_U is below the cap (`safeguard = TRUE`) and the model has converged (`converged = TRUE`).

When called automatically by `nmfre` (i.e., `dfU.cap.rate = NULL`), the minimum rate satisfying both `safeguard = TRUE` and `converged = TRUE` is selected.

Usage

```
nmfre.dfU.scan(
  rates = (1:10)/10,
  Y,
  A,
  rank = NULL,
  X.init = NULL,
  C.init = NULL,
  U.init = NULL,
  print.trace = FALSE,
  ...
)
```

Arguments

`rates`	Numeric vector of cap rates to scan (default `(1:10)/10`).
`Y`	Observation matrix (P x N).
`A`	Covariate matrix (K x N).
`rank`	Integer. Rank of the basis matrix. For backward compatibility, `Q` is accepted via `...`.
`X.init`	Initial basis matrix, or `NULL`.
`C.init`	Initial coefficient matrix, or `NULL`.
`U.init`	Initial random effects matrix, or `NULL`.
`print.trace`	Logical. Print progress for each fit (default `FALSE`).
`...`	Additional arguments passed to `nmfre`.

Value

An object of class `"nmfre.dfU.scan"` with two components:

`table` A data frame with the following columns:

`rate` Cap rate used. The dfU cap is `rate * N * Q`.

`dfU.cap` The dfU cap value (upper bound on effective degrees of freedom).

`dfU` Final effective degrees of freedom for U at convergence.

`safeguard` Logical. `TRUE` if the dfU cap is functioning as a safeguard (`dfU / dfU.cap < 0.99`): the cap prevents random-effects saturation without over-constraining U. `FALSE` if dfU is at or near the cap, indicating the cap is binding and the rate may be too small.

`hit` Logical. `TRUE` if the cap was reached at least once during iteration, even if dfU later decreased below the cap.

`converged` Logical. `TRUE` if the algorithm converged within the maximum number of iterations.

`tau2` Final random effect variance $\hat{\tau}^2$.

`sigma2` Final residual variance $\hat{\sigma}^2$.

`ICC` Trace-based Intraclass Correlation Coefficient $\tau^2 \operatorname{tr}(X^\top X)/(\tau^2 \operatorname{tr}(X^\top X) + \sigma^2 P)$. See `nmfre` for details.

`cap.rate` Optimal cap rate selected automatically. If rows with `safeguard = TRUE` and `hit = TRUE` exist, the maximum rate among them is chosen (safeguard activated but giving U the most freedom). Otherwise, the minimum rate with `safeguard = TRUE` and `hit = FALSE` is chosen. `NA` if no suitable rate is found.

When printed, only the `table` is displayed. Access `cap.rate` directly from the returned object.

Examples

```
# Example 1. cars data (small maxit for speed)
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(intercept = 1, speed = cars$speed)
tab <- nmfre.dfU.scan(rates = c(0.1, 0.2), Y = Y, A = A, Q = 1, maxit = 1000)
print(tab)


# Example 2. Orthodont data (nlme)
library(nlme)
Y <- matrix(Orthodont$distance, 4, 27)
male <- ifelse(Orthodont$Sex[seq(1, 108, 4)] == "Male", 1, 0)
A <- rbind(intercept = 1, male = male)

nmfre.dfU.scan(1:10/10, Y, A, Q = 1)
```

`nmfre.inference` *Statistical inference for the coefficient matrix C from NMF-RE*

Description

`nmfre.inference` performs statistical inference on the coefficient matrix C (Θ) from a fitted `nmfre` model, conditional on the estimated basis matrix $\hat{X}$ and random effects $\hat{U}$.

Under the working model $Y^* = Y - X\hat{U} \approx XCA + \varepsilon$, inference is conducted via sandwich covariance estimation and one-step wild bootstrap with non-negative projection.

The result is compatible with `nmfkc.DOT` for visualization (pass the result directly as `x` with `type = "YXA"`).

Usage

```
nmfre.inference(object, Y, A = NULL, wild.bootstrap = TRUE, ...)
```

Arguments

`object`	An object of class `"nmfre"` returned by `nmfre`.
`Y`	Observation matrix (P x N). Must match the data used in `nmfre()`.
`A`	Covariate matrix (K x N). Default is `NULL` (intercept only).
`wild.bootstrap`	Logical. If `TRUE` (default), performs wild bootstrap for confidence intervals and bootstrap standard errors.
`...`	Additional arguments:

`wild.B`	Number of bootstrap replicates. Default is 500.
`wild.seed`	Seed for bootstrap. Default is 123.
`wild.level`	Confidence level for bootstrap CI. Default is 0.95.
`C.p.side`	P-value type: `"one.sided"` (default) or `"two.sided"`.
`cov.ridge`	Ridge stabilization. Default is 1e-8.
`print.trace`	Logical. Default is `FALSE`.

Value

The input `object` with additional inference components:

`sigma2.used`	Estimated σ^2 used for inference.
`C.vec.cov`	Full covariance matrix for $vec(C)$.
`C.se`	Sandwich standard errors for C.
`C.se.boot`	Bootstrap standard errors for C.
`C.ci.lower`	Lower CI bounds for C.
`C.ci.upper`	Upper CI bounds for C.
`coefficients`	Data frame with Basis, Covariate, Estimate, SE, BSE, z_value, p_value, CI_low, CI_high.
`C.p.side`	P-value type used.

References

Satoh, K. (2026). Wild Bootstrap Inference for Non-Negative Matrix Factorization with Random Effects. arXiv:2603.01468. https://arxiv.org/abs/2603.01468

See Also

`nmfre`, `nmfkc.DOT`, `summary.nmfre`

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(intercept = 1, speed = cars$speed)
res <- nmfre(Y, A, Q = 1, wild.bootstrap = FALSE)
res2 <- nmfre.inference(res, Y, A)
res2$coefficients
```

`plot.nmfae` — `plot.nmfae` *displays the convergence trajectory of the objective function across iterations. The title shows the achieved $R\hat{}2$.*

Description

`plot.nmfae` displays the convergence trajectory of the objective function across iterations. The title shows the achieved R^2.

Usage

```
## S3 method for class 'nmfae'
plot(x, ...)
```

Arguments

`x`	An object of class `"nmfae"` returned by `nmfae`.
`...`	Additional graphical parameters passed to `plot`.

Value

Invisible `NULL`. Called for its side effect (plot).

See Also

`nmfae`, `nmfae.heatmap`

`plot.nmfae.cv` — *Plot method for nmfae.cv objects*

Description

Displays a bar chart of per-fold cross-validation errors from `nmfae.cv`. The overall RMSE (sigma) is shown in the title.

Usage

```
## S3 method for class 'nmfae.cv'
plot(x, ...)
```

Arguments

`x`	An object of class `"nmfae.cv"` returned by `nmfae.cv`.
`...`	Additional graphical parameters passed to `barplot`.

Value

Invisible `NULL`. Called for its side effect (plot).

See Also

`nmfae.cv`

plot.nmfae.DOT *Plot method for nmfae.DOT objects*

Description

Renders a DOT graph string using `DiagrammeR::grViz`. If the **DiagrammeR** package is not installed, prints the DOT source to the console instead.

Usage

```
## S3 method for class 'nmfae.DOT'
plot(x, ...)
```

Arguments

`x`	An object of class `"nmfae.DOT"` returned by `nmfae.DOT`.
`...`	Not used.

Value

The `grViz` widget (invisibly), or invisible `NULL` if **DiagrammeR** is not available.

See Also

`nmfae.DOT`

plot.nmfae.ecv *Plot method for nmfae.ecv objects*

Description

Visualizes element-wise cross-validation results. When `R` was `NULL` (paired Q=R), a line plot of sigma vs Q is drawn. When `R` was explicitly specified (grid), a heatmap of sigma over the (Q, R) grid is drawn.

Usage

```
## S3 method for class 'nmfae.ecv'
plot(x, ...)
```

Arguments

`x`	An object of class `"nmfae.ecv"` returned by `nmfae.ecv`.
`...`	Additional graphical parameters (currently unused).

See Also

`nmfae.ecv`

`plot.nmfae.kernel.beta.cv` *Plot method for nmfae.kernel.beta.cv objects*

Description

Displays the cross-validation objective function across candidate `beta` values (log scale). The optimal beta is highlighted in red.

Usage

```
## S3 method for class 'nmfae.kernel.beta.cv'
plot(x, ...)
```

Arguments

`x`	An object of class `"nmfae.kernel.beta.cv"` returned by `nmfae.kernel.beta.cv`.
`...`	Additional graphical parameters passed to `plot`.

Value

Invisible `NULL`. Called for its side effect (plot).

See Also

`nmfae.kernel.beta.cv`

`plot.nmfkc` *Plot method for objects of class* `nmfkc`

Description

`plot.nmfkc` produces a diagnostic plot for the return value of `nmfkc`, showing the objective function across iterations.

Usage

```
## S3 method for class 'nmfkc'
plot(x, ...)
```

Arguments

`x`	An object of class `nmfkc`, i.e., the return value of `nmfkc`.
`...`	Additional arguments passed to the base `plot` function.

Value

Called for its side effect (a plot). Returns `NULL` invisibly.

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
result <- nmfkc(Y, A, Q = 1)
plot(result)
```

---

plot.nmfkc.DOT *Plot method for nmfkc.DOT objects*

Description

Renders a DOT graph string using `DiagrammeR::grViz`. If the **DiagrammeR** package is not installed, prints the DOT source to the console instead.

Usage

```
## S3 method for class 'nmfkc.DOT'
plot(x, ...)
```

Arguments

`x`	An object of class `"nmfkc.DOT"` returned by `nmfkc.DOT`.
`...`	Not used.

Value

Called for its side effect (rendering). Returns `x` invisibly.

See Also

`nmfkc.DOT`

---

plot.nmfre *Plot convergence diagnostics for NMF models*

Description

Plots the objective function value over iterations for `nmfre` and `nmf.sem` objects. (For `nmfkc` and `nmfae`, plot methods are defined in their respective source files.)

Usage

```
## S3 method for class 'nmfre'
plot(x, ...)

## S3 method for class 'nmf.sem'
plot(x, ...)
```

Arguments

`x`	A fitted model object.
`...`	Additional graphical arguments passed to `plot`.

Value

Invisible `NULL`.

Examples

```
## Not run:
res <- nmfre(Y, A, Q = 2, wild.bootstrap = FALSE)
plot(res)

## End(Not run)
```

`plot.predict.nmfae` *Plot method for predict.nmfae objects*

Description

For `type = "response"`: if actual values Y_1 were stored, displays an observed-vs-predicted scatter plot with R^2 in the title. Otherwise, displays the predicted matrix as a heatmap.

For `type = "class"`: if actual classes were stored, displays a confusion matrix heatmap with accuracy (ACC) in the title.

Usage

```
## S3 method for class 'predict.nmfae'
plot(x, ...)
```

Arguments

`x`	An object of class `"predict.nmfae"` returned by `predict.nmfae`.
`...`	Additional graphical parameters passed to `plot` or `image`.

Value

Invisible `NULL`. Called for its side effect (plot).

See Also

`predict.nmfae`

`predict.nmfae` *Predict method for nmfae objects*

Description

`predict.nmfae` computes fitted or predicted values from a three-layer NMF model. Without `newY2`, returns the in-sample fitted values $X_1 \Theta X_2 Y_2$. With `newY2`, computes out-of-sample predictions $X_1 \Theta X_2 \cdot \mathrm{newY2}$.

When `type = "class"`, each column is classified to the row with the maximum predicted value (useful when Y_1 is a one-hot class matrix from `nmfkc.class`).

If `Y1` (actual values) is provided, it is stored as an attribute so that `plot.predict.nmfae` can produce an observed-vs-predicted scatter plot (for `type = "response"`) or a confusion matrix heatmap (for `type = "class"`).

Usage

```
## S3 method for class 'nmfae'
predict(object, newY2 = NULL, Y1 = NULL, type = c("response", "class"), ...)
```

Arguments

`object`	An object of class `"nmfae"` returned by `nmfae`.
`newY2`	Optional new input matrix (P2 x M) for prediction. If `NULL`, returns in-sample fitted values.
`Y1`	Optional actual output matrix for comparison plotting.
`type`	Character. `"response"` (default) returns the predicted matrix. `"class"` returns a factor of predicted class labels (row with max value).
`...`	Not used.

Value

For `type = "response"`: a matrix of class `"predict.nmfae"`. For `type = "class"`: a factor of class `"predict.nmfae"` with predicted class labels. If `Y1` was provided, actual classes are stored in `attr(result, "actual")`.

See Also

`nmfae`, `plot.predict.nmfae`, `nmfkc.class`

predict.nmfkc *Prediction method for objects of class* nmfkc

Description

predict.nmfkc generates predictions from an object of class nmfkc, either using the fitted covariates or a new covariate matrix.

When the model was fitted using a formula (Formula Mode), a newdata data frame can be supplied instead of newA; the covariate matrix is then constructed automatically from the stored formula metadata.

Usage

```
## S3 method for class 'nmfkc'
predict(object, newA = NULL, newdata = NULL, type = "response", ...)
```

Arguments

object	An object of class nmfkc, i.e., the return value of nmfkc.
newA	Optional. A new covariate matrix to be used for prediction.
newdata	Optional data frame. Only available when the model was fitted using a formula. Covariate columns are extracted automatically using the stored formula metadata. If both newdata and newA are supplied, newdata takes precedence (with a warning).
type	Type of prediction to return. Options are "response" (fitted values matrix), "prob" (soft-clustering probabilities), or "class" (hard-clustering labels based on row names of X).
...	Further arguments passed to or from other methods.

Value

Depending on type: a numeric matrix ("response" or "prob") or a character vector of class labels ("class").

Examples

```
# Prediction with newA
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
result <- nmfkc(Y, A, Q = 1)
newA <- rbind(1, c(10, 20, 30))
predict(result, newA = newA)
```

`print.nmfae.inference` *Print method for nmfae.inference objects*

Description

Prints a summary of the NMF-AE model with inference results.

Usage

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

Arguments

`x`	An object of class `"nmfae.inference"`.
`...`	Additional arguments passed to `print.summary.nmfae.inference`.

Value

Called for its side effect (printing). Returns `x` invisibly.

See Also

`nmfae.inference`, `summary.nmfae.inference`

`print.nmfkc.inference` *Print method for nmfkc.inference objects*

Description

Prints a summary of the NMF model with inference results.

Usage

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

Arguments

`x`	An object of class `"nmfkc.inference"`.
`...`	Additional arguments passed to `print.summary.nmfkc.inference`.

Value

Called for its side effect (printing). Returns `x` invisibly.

See Also

`nmfkc.inference`, `summary.nmfkc.inference`

`print.summary.nmfae` *Print method for summary.nmfae objects*

Description

Prints a formatted summary of an NMF-AE model fit.

Usage

```
## S3 method for class 'summary.nmfae'
print(x, digits = max(3L, getOption("digits") - 3L), max.coef = 20, ...)
```

Arguments

`x`	An object of class `"summary.nmfae"`.
`digits`	Minimum number of significant digits to be used.
`max.coef`	Maximum number of coefficient rows to display. If the table has more rows, only significant rows ($p < 0.05$) are shown. Default is 20.
`...`	Additional arguments (currently unused).

Value

Called for its side effect (printing). Returns `x` invisibly.

See Also

`summary.nmfae`

`print.summary.nmfae.inference` *Print method for summary.nmfae.inference objects*

Description

Prints a formatted summary including the coefficients table.

Usage

```
## S3 method for class 'summary.nmfae.inference'
print(x, digits = max(3L, getOption("digits") - 3L), max.coef = 20, ...)
```

Arguments

`x`	An object of class `"summary.nmfae.inference"`.
`digits`	Minimum number of significant digits.
`max.coef`	Maximum coefficient rows to display. Default is 20.
`...`	Additional arguments (currently unused).

Value

Called for its side effect (printing). Returns x invisibly.

See Also

summary.nmfae.inference

---

print.summary.nmfkc *Print method for* summary.nmfkc *objects*

Description

Prints a formatted summary of an nmfkc model fit.

Usage

```
## S3 method for class 'summary.nmfkc'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of class summary.nmfkc.
digits	Minimum number of significant digits to be used.
...	Additional arguments (currently unused).

Value

Called for its side effect (printing). Returns x invisibly.

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
result <- nmfkc(Y, A, Q = 1)
print(summary(result))
```

---

print.summary.nmfkc.inference
Print method for summary.nmfkc.inference objects

Description

Prints a formatted summary including the coefficients table.

Usage

```
## S3 method for class 'summary.nmfkc.inference'
print(x, digits = max(3L, getOption("digits") - 3L), max.coef = 20, ...)
```

Arguments

`x`	An object of class `"summary.nmfkc.inference"`.
`digits`	Minimum number of significant digits.
`max.coef`	Maximum coefficient rows to display. Default is 20.
`...`	Additional arguments (currently unused).

Value

Called for its side effect (printing). Returns `x` invisibly.

See Also

`summary.nmfkc.inference`

`residuals.nmfkc` *Extract residuals from NMF models*

Description

Returns the residual matrix $Y - \hat{Y}$ from a fitted NMF model. Requires the original observation matrix `Y` to be supplied.

For `nmfre` objects, residuals are computed from the BLUP reconstruction ($Y - X(B_{blup})$) by default. Set `type = "fixed"` to use fixed-effects only.

Usage

```
## S3 method for class 'nmfkc'
residuals(object, Y, ...)

## S3 method for class 'nmfae'
residuals(object, Y, ...)

## S3 method for class 'nmfre'
residuals(object, Y, type = c("blup", "fixed"), ...)

## S3 method for class 'nmf.sem'
residuals(object, Y, ...)
```

Arguments

`object`	A fitted model object.
`Y`	The original observation matrix used for fitting.
`...`	Not used.
`type`	For `nmfre` objects: `"blup"` (default) or `"fixed"`.

Value

The residual matrix.

Examples

```
Y <- matrix(runif(50), 5, 10)
result <- nmfkc(Y, Q = 2)
residuals(result, Y)
```

summary.nmf.sem *Summary method for nmf.sem objects*

Description

Produces a formatted summary of a fitted NMF-SEM model, including matrix dimensions, convergence, stability diagnostics, fit statistics, and inference results (if available).

Usage

```
## S3 method for class 'nmf.sem'
summary(object, ...)
```

Arguments

`object`	An object of class `"nmf.sem"` returned by `nmf.sem`.
`...`	Not used.

Value

Invisible `object`.

See Also

`nmf.sem`, `nmf.sem.inference`

Examples

```
Y <- t(iris[, -5])
Y1 <- Y[1:2, ]; Y2 <- Y[3:4, ]
result <- nmf.sem(Y1, Y2, rank = 2, maxit = 500)
summary(result)
```

`summary.nmfae` *Summary method for nmfae objects*

Description

`summary.nmfae` produces a summary of a fitted NMF-AE model, including dimensions, convergence status, goodness-of-fit statistics, and structure diagnostics (sparsity of factor matrices).

Usage

```
## S3 method for class 'nmfae'
summary(object, ...)
```

Arguments

`object`	An object of class `"nmfae"` returned by `nmfae`.
`...`	Additional arguments (currently unused).

Value

An object of class `"summary.nmfae"`, a list with components:

`call`	The matched call.
`dims`	Named vector `c(P1, P2, N)`.
`Q`	Decoder rank.
`R`	Encoder rank.
`n.params`	Total number of parameters (P1Q + QR + R*P2).
`autoencoder`	Logical; TRUE if P1 == P2 and Y1 was used as Y2.
`niter`	Number of iterations.
`runtime`	Elapsed time.
`objfunc`	Final objective value.
`r.squared`	R-squared.
`sigma`	Residual standard error (RMSE).
`mae`	Mean absolute error.
`n.missing`	Number of missing elements.
`prop.missing`	Percentage of missing elements.
`X1.sparsity`	Proportion of near-zero elements in X1.
`C.sparsity`	Proportion of near-zero elements in C.
`X2.sparsity`	Proportion of near-zero elements in X2.

See Also

`nmfae`, `print.summary.nmfae`

summary.nmfae.inference — *Summary method for nmfae.inference objects*

Description

Produces a summary of a fitted NMF-AE model with inference results, including the coefficients table for Θ.

Usage

```
## S3 method for class 'nmfae.inference'
summary(object, ...)
```

Arguments

`object`	An object of class `"nmfae.inference"`.
`...`	Additional arguments (currently unused).

Value

An object of class `"summary.nmfae.inference"`.

See Also

`nmfae.inference`, `summary.nmfae`

summary.nmfkc — *Summary method for objects of class* `nmfkc`

Description

Produces a summary of an `nmfkc` object, including matrix dimensions, runtime, fit statistics, and diagnostics.

Usage

```
## S3 method for class 'nmfkc'
summary(object, ...)
```

Arguments

`object`	An object of class `nmfkc`, i.e., the return value of `nmfkc`.
`...`	Additional arguments (currently unused).

Value

An object of class `summary.nmfkc`, containing summary statistics.

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
result <- nmfkc(Y, A, Q = 1)
summary(result)
```

`summary.nmfkc.inference` *Summary method for nmfkc.inference objects*

Description

Produces a summary of a fitted NMF model with inference results, including the coefficients table for C (Θ).

Usage

```
## S3 method for class 'nmfkc.inference'
summary(object, ...)
```

Arguments

`object`	An object of class `"nmfkc.inference"`.
`...`	Additional arguments (currently unused).

Value

An object of class `"summary.nmfkc.inference"`.

See Also

`nmfkc.inference`, `summary.nmfkc`

`summary.nmfre` *Summary method for objects of class* `nmfre`

Description

Displays a concise summary of an NMF-RE model fit, including dimensions, convergence, variance components, and a coefficient table following standard R regression output conventions.

Usage

```
## S3 method for class 'nmfre'
summary(object, show_ci = FALSE, ...)
```

Arguments

object	An object of class nmfre, returned by nmfre.
show_ci	Logical. If TRUE, show confidence interval columns (default FALSE).
...	Additional arguments (currently unused).

Value

The input object, invisibly.

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(intercept = 1, speed = cars$speed)
res <- nmfre(Y, A, Q = 1, maxit = 5000)
summary(res)
```

Index