

# Package ‘nmfkc’

May 9, 2026

**Type** Package

**Title** Non-Negative Matrix Factorization with Kernel Covariates

**Version** 0.6.7

**Date** 2026-03-30

**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, and NMF-based Structural Equation Modeling (NMF-SEM) via `nmf.sem()`, which fits a two-block input-output model for blind source separation and path analysis.  
References: Satoh (2025) <[doi:10.48550/arXiv.2403.05359](https://doi.org/10.48550/arXiv.2403.05359)>;  
Satoh (2025) <[doi:10.48550/arXiv.2510.10375](https://doi.org/10.48550/arXiv.2510.10375)>;  
Satoh (2025) <[doi:10.48550/arXiv.2512.18250](https://doi.org/10.48550/arXiv.2512.18250)>;  
Satoh (2026) <[doi:10.48550/arXiv.2603.01468](https://doi.org/10.48550/arXiv.2603.01468)>;  
Satoh (2026) <[doi:10.1007/s42081-025-00314-0](https://doi.org/10.1007/s42081-025-00314-0)>.

**License** MIT + file LICENSE

**Imports** stats, graphics, utils, grDevices

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.2.3

**ByteCompile** true

**VignetteBuilder** knitr

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0), mclust,  
palmerpenguins, quanteda, vars, DiagrammeR, MASS, nlme, lavaan

**Config/testthat/edition** 3

**NeedsCompilation** no

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

**Repository** CRAN

**Date/Publication** 2026-04-15 13:00:14 UTC

## Contents

coef.nmf . . . . .	3
fitted.nmf . . . . .	4
nmf.sem . . . . .	5
nmf.sem.cv . . . . .	7
nmf.sem.DOT . . . . .	8
nmf.sem.inference . . . . .	10
nmf.sem.split . . . . .	12
nmfae . . . . .	13
nmfae.cv . . . . .	15
nmfae.DOT . . . . .	16
nmfae.ecv . . . . .	18
nmfae.heatmap . . . . .	19
nmfae.inference . . . . .	20
nmfae.kernel.beta.cv . . . . .	22
nmfae.rename . . . . .	23
nmfkc . . . . .	24
nmfkc.ar . . . . .	28
nmfkc.ar.degree.cv . . . . .	29
nmfkc.ar.DOT . . . . .	30
nmfkc.ar.predict . . . . .	32
nmfkc.ar.stationarity . . . . .	33
nmfkc.class . . . . .	34
nmfkc.criterion . . . . .	34
nmfkc.cv . . . . .	35
nmfkc.denormalize . . . . .	37
nmfkc.DOT . . . . .	38
nmfkc.ecv . . . . .	40
nmfkc.inference . . . . .	41
nmfkc.kernel . . . . .	42
nmfkc.kernel.beta.cv . . . . .	43
nmfkc.kernel.beta.nearest.med . . . . .	44
nmfkc.kernel.gaussian . . . . .	47
nmfkc.normalize . . . . .	48
nmfkc.rank . . . . .	49
nmfkc.residual.plot . . . . .	50

nmfre . . . . .	51
nmfre.dfU.scan . . . . .	55
nmfre.inference . . . . .	57
plot.nmfae . . . . .	59
plot.nmfae.cv . . . . .	59
plot.nmfae.ecv . . . . .	60
plot.nmfae.kernel.beta.cv . . . . .	61
plot.nmfkc . . . . .	61
plot.nmfkc.DOT . . . . .	62
plot.nmfre . . . . .	63
plot.predict.nmfae . . . . .	64
predict.nmfae . . . . .	65
predict.nmfkc . . . . .	66
print.nmf.inference . . . . .	67
print.summary.nmfae . . . . .	67
print.summary.nmfae.inference . . . . .	68
print.summary.nmfkc . . . . .	69
print.summary.nmfkc.inference . . . . .	69
residuals.nmf . . . . .	70
summary.nmf.sem . . . . .	71
summary.nmfae . . . . .	72
summary.nmfae.inference . . . . .	73
summary.nmfkc . . . . .	73
summary.nmfkc.inference . . . . .	74
summary.nmfre . . . . .	75

**Index****76**


---

coef.nmf	<i>Extract coefficients from NMF models</i>
----------	---

---

**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  $C$  ( $\Theta$ ) directly.

For `nmf.sem` objects, returns  $C_2$  (exogenous block) as fallback.

**Usage**

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

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

**Arguments**

object            A fitted model object of class "nmf", "nmfkc", "nmfae", "nmfre", or "nmf.sem".  
 ...                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](#), [nmf.sem.inference](#)

**Examples**

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

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

---

 fitted.nmf

---

*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  $Y_1$  and  $Y_2$  to get the direct reconstruction  $X(C_1Y_1 + C_2Y_2)$  instead.

**Usage**

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

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

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

**Arguments**

object            A fitted model object of class "nmf", "nmfkc", "nmfae", "nmfre", or "nmf.sem".  
 ...                For `nmf.sem`: optionally  $Y_1$  and  $Y_2$ .

**Value**

The fitted matrix  $XB$ .

**See Also**

[nmfkc](#), [nmfae](#), [nmfre](#), [nmf.sem](#), [residuals.nmf](#)

**Examples**

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

---

nmf.sem

*NMF-SEM Main Estimation Algorithm*


---

**Description**

Fits the NMF-SEM model

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

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  $\Theta_1$  and  $\Theta_2$ , 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 <b>rows = variables (P1), columns = samples (N)</b> .
Y2	A non-negative numeric matrix of exogenous variables with <b>rows = variables (P2), columns = samples (N)</b> . Must satisfy $\text{nrow}(Y1) == \text{nrow}(Y2)$ .
rank	Integer; number of latent factors $Q$ . If NULL, $Q$ is taken from a hidden argument in ... or defaults to $\text{nrow}(Y2)$ .
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 $\lambda_X \ X^\top X - \text{diag}(X^\top X)\ _F^2$ . Default: 100.
C1.L1	L1 sparsity penalty for C1 (i.e., $\Theta_1$ ). Default: 1.0.
C2.L1	L1 sparsity penalty for C2 (i.e., $\Theta_2$ ). 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 (reserved for future use).

**Value**

A list with components:

X	Estimated basis matrix ( $P_1 \times Q$ ).
C1	Estimated latent feedback matrix ( $\Theta_1, Q \times P_1$ ).
C2	Estimated exogenous loading matrix ( $\Theta_2, Q \times P_2$ ).
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 $\ X\Theta_1\ _{1,op}$ .
Leontief.inv	Leontief-type inverse $(I - X\Theta_1)^{-1}$ .
M.model	Equilibrium mapping $M_{\text{model}} = (I - X\Theta_1)^{-1}X\Theta_2$ .
amplification	Latent amplification factor $\ M_{\text{model}}\ _{1,op} / \ X\Theta_2\ _{1,op}$ .
amplification.bound	Geometric-series upper bound $1/(1 - \ X\Theta_1\ _{1,op})$ if $\ X\Theta_1\ _{1,op} < 1$ , 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 $\hat{Y}_1 = M_{\text{model}}Y_2$ .
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>

## See Also

[nmf.sem.inference](#), [nmf.sem.cv](#), [nmf.sem.split](#), [nmf.sem.DOT](#), [summary.nmf.sem](#)

## 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,  
  ...  
)
```

**Arguments**

Y1	A non-negative numeric matrix of endogenous variables with <b>rows = variables (P1)</b> , <b>columns = samples (N)</b> .
Y2	A non-negative numeric matrix of exogenous variables with <b>rows = variables (P2)</b> , <b>columns = samples (N)</b> . Must satisfy <code>ncol(Y1) == ncol(Y2)</code> .
rank	Integer; rank (number of latent factors) passed to <code>nmf.sem</code> . If NULL, <code>nmf.sem</code> decides the effective rank (via <code>...</code> or <code>nrow(Y2)</code> ).
X.init	Optional initialization for X (as in <code>nmf.sem</code> ).
X.L2.ortho	L2 orthogonality penalty for X.
C1.L1	L1 sparsity penalty for C1 ( $\Theta_1$ ).
C2.L1	L1 sparsity penalty for C2 ( $\Theta_2$ ).
epsilon	Convergence threshold for <code>nmf.sem</code> .
maxit	Maximum number of iterations for <code>nmf.sem</code> .
...	Additional arguments passed to <code>nmf.sem</code> (except for <code>rank</code> , <code>seed</code> , <code>div</code> , <code>shuffle</code> , which are handled here). Also accepts: <code>nfolds</code> (number of folds, default 5; <code>div</code> also accepted), <code>seed</code> (master random seed, default NULL), <code>shuffle</code> (logical, default TRUE).

**Value**

A numeric scalar: mean MAE across CV folds.

**See Also**

[nmf.sem](#)

**Examples**

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

## Description

Creates a Graphviz DOT script that visualizes the structural network estimated by `nmf.sem`. The resulting diagram displays:

- endogenous observed variables ( $Y_1$ ),
- exogenous observed variables ( $Y_2$ ),
- latent factors ( $F_1, \dots, F_Q$ ),

together with the non-negative path coefficients whose magnitudes exceed a user-specified threshold.

Directed edges represent estimated relationships:

- $Y_2 \rightarrow F_q$ : entries of C2 (exogenous loadings),
- $F_q \rightarrow Y_1$ : rows of X (factor-to-endogenous mappings),
- $Y_1 \rightarrow F_q$ : entries of C1 (feedback paths).

Edge widths are scaled by coefficient magnitude, and nodes are placed in optional visual clusters. Only variables participating in edges above the threshold are displayed, while latent factors are always shown.

## Usage

```
nmf.sem.DOT(
  result,
  weight_scale = 5,
  weight_scale_y2f = weight_scale,
  weight_scale_fy1 = weight_scale,
  weight_scale_feedback = weight_scale,
  threshold = 0.01,
  rankdir = "LR",
  fill = TRUE,
  cluster.box = c("normal", "faint", "invisible", "none"),
  cluster.labels = NULL,
  hide.isolated = TRUE,
  sig.level = 0.1
)
```

## Arguments

<code>result</code>	A list returned by <code>nmf.sem</code> , containing matrices X, C1, and C2.
<code>weight_scale</code>	Base scaling factor for edge widths.
<code>weight_scale_y2f</code>	Optional override for scaling edges $Y_2 \rightarrow F_q$ . Defaults to <code>weight_scale</code> .
<code>weight_scale_fy1</code>	Optional override for scaling edges $F_q \rightarrow Y_1$ . Defaults to <code>weight_scale</code> .
<code>weight_scale_feedback</code>	Optional override for scaling feedback edges $Y_1 \rightarrow F_q$ . Defaults to <code>weight_scale</code> .

threshold	Minimum coefficient value needed for an edge to be drawn.
rankdir	Graphviz rank direction (e.g., "LR", "TB").
fill	Logical; whether to use filled node shapes.
cluster.box	Character string controlling the visibility and style of cluster frames around Y2, factors, and Y1 blocks. One of "normal", "faint", "invisible", "none".
cluster.labels	Optional character vector of length 3 giving custom labels for the Y2, factor, and Y1 clusters.
hide.isolated	Logical. If TRUE (default), Y1 and Y2 nodes that have no edges at or above threshold are excluded from the graph.
sig.level	Significance level for filtering C2 edges when inference results are present. If result contains a coefficients data frame (from <code>nmf.sem.inference</code> ), only edges with <code>p_value &lt; sig.level</code> are drawn, with significance stars appended. Set to NULL to disable filtering. Default is 0.1.

**Value**

A character string representing a valid Graphviz DOT script.

**See Also**

[nmf.sem](#), [nmf.sem.inference](#), [plot.nmfkc.DOT](#)

**Examples**

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

---

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 <code>nmf.sem</code> , containing at least $X$ , $C_1$ , and $C_2$ .
Y1	Endogenous variable matrix ( $P_1 \times N$ ). Must match the data used in <code>nmf.sem()</code> .
Y2	Exogenous variable matrix ( $P_2 \times N$ ). Must match the data used in <code>nmf.sem()</code> .
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 $\sigma^2$ used for inference.
C2.se	Sandwich standard errors for $C_2$ ( $Q \times P_2$ matrix).
C2.se.boot	Bootstrap standard errors for $C_2$ ( $Q \times P_2$ matrix).
C2.ci.lower	Lower CI bounds for $C_2$ ( $Q \times P_2$ matrix).
C2.ci.upper	Upper CI bounds for $C_2$ ( $Q \times P_2$ 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 = FALSE
)
```

### Arguments

<code>x</code>	A numeric matrix or data frame with <b>rows = samples</b> and <b>columns = observed variables</b> .
<code>n.exogenous</code>	Optional integer specifying the number of exogenous variables ( $Y_2$ ). If NULL, the number is inferred automatically by the coefficient cut-off rule.
<code>threshold</code>	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)
<code>auto.flipped</code>	Logical; if TRUE, applies PC1-based automatic sign flipping after standardization to ensure consistent orientation. (Default: TRUE)
<code>verbose</code>	Logical; if TRUE, prints progress messages and the resulting variable split. (Default: FALSE)

**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.

**See Also**

[nmf.sem](#)

**Examples**

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

---

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),  $\Theta$  is a bottleneck parameter matrix,  $X_2$  is an encoder basis (row sum 1), and  $Y_2$  is the input matrix.

When  $Y_2 = Y_1$ , the model acts as a non-negative autoencoder. When  $Y_1 \neq Y_2$ , 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  $\Theta$  and  $X_2$ .

**Usage**

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

```

    maxit = 5000,
    verbose = FALSE,
    ...
)

```

### Arguments

Y1	Output matrix $Y_1$ ( $P1 \times N$ ). Non-negative. May contain NAs (handled via <code>Y1.weights</code> ).
Y2	Input matrix $Y_2$ ( $P2 \times N$ ). Non-negative. Default is Y1 (autoencoder).
rank	Integer. Rank of the decoder basis $X_1$ ( $P1 \times Q$ ). Default is 2. For backward compatibility, Q is accepted via <code>...</code>
rank.encoder	Integer. Rank of the encoder basis $X_2$ ( $R \times P2$ ). Default is rank. For backward compatibility, R is accepted via <code>...</code>
epsilon	Positive convergence tolerance. Default is $1e-4$ .
maxit	Maximum number of multiplicative update iterations. Default is 5000.
verbose	Logical. If TRUE, prints progress messages during fitting. Default is FALSE.
...	Additional arguments: <code>Y1.weights</code> Weight matrix ( $P1 \times N$ ) or vector for $Y_1$ . 0 indicates missing/ignored elements. Default: auto-detect NAs. <code>C.L1</code> L1 regularization parameter for $C$ . Default is 0. <code>X1.L2.ortho</code> L2 orthogonality regularization for $X_1$ columns. Default is 0. <code>X2.L2.ortho</code> L2 orthogonality regularization for $X_2$ rows. Default is 0. <code>seed</code> Integer seed for reproducibility. Default is 123. <code>print.trace</code> Logical. If TRUE, prints progress. Default is FALSE.

### Value

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

X1	Decoder basis matrix ( $P1 \times Q$ ), column sum 1.
C	Parameter matrix ( $Q \times R$ ).
X2	Encoder basis matrix ( $R \times P2$ ), row sum 1.
Y1hat	Fitted values $X_1 \Theta X_2 Y_2$ ( $P1 \times N$ ).
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 <code>difftime</code> object.
n.missing	Number of missing (or zero-weighted) elements in $Y_1$ .
n.total	Total number of elements in $Y_1$ ( $P1 \times N$ ).

## 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
Y <- matrix(c(1,0,1,0, 0,1,0,1, 1,1,0,0), nrow=3, byrow=TRUE)
res <- nmfae(Y, rank=2, rank.encoder=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, rank=2, rank.encoder=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  $Y_2$  is a kernel matrix created by [nmfkc.kernel](#) (detected via attributes), the symmetric kernel splitting convention is used:  $Y_2[\text{train}, \text{train}]$  for training and  $Y_2[\text{train}, \text{test}]$  for prediction.

## Usage

```
nmfae.cv(Y1, Y2 = Y1, rank = 2, rank.encoder = rank, ...)
```

**Arguments**

Y1	Output matrix $Y_1$ ( $P1 \times N$ ). Non-negative.
Y2	Input matrix $Y_2$ ( $P2 \times N$ ), or a kernel matrix ( $N \times N$ ). Default is Y1 (autoencoder).
rank	Integer. Rank of the decoder basis. Default is 2.
rank.encoder	Integer. Rank of the encoder basis. Default is rank.
...	Additional arguments passed to <code>nmfae</code> (e.g., <code>epsilon</code> , <code>maxit</code> , <code>Y1.weights</code> ). Also accepts: <code>nfolds</code> (number of folds, default 5; <code>div</code> also accepted), <code>seed</code> (integer seed, default 123), <code>shuffle</code> (logical, default TRUE). For backward compatibility, <code>Q</code> , <code>R</code> are accepted as aliases for <code>rank</code> , <code>rank.encoder</code> .

**Value**

A list with components:

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

**See Also**

[nmfae](#), [nmfae.ecv](#), [nmfae.kernel.beta.cv](#), [nmfkc.cv](#)

**Examples**

```
Y <- t(iris[1:30, 1:4])
res <- nmfae.cv(Y, rank = 2, rank.encoder = 2, nfolds = 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,  $\Theta$ , and decoder factors; "YXCXY" shows the full structure from  $Y_2$  through  $X_2$ ,  $\Theta$ ,  $X_1$  to  $Y_1$ .

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

**Usage**

```

nmfae.DOT(
  result,
  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**

result	An object of class "nmfae" returned by <code>nmfae</code> .
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 <code>sig.level</code> 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 $\Theta$ 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](#)

### Examples

```
set.seed(1)
Y <- matrix(runif(20), nrow = 4)
res <- nmfae(Y, rank = 2)
dot <- nmfae.DOT(res)
```

---

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 `rank` and `rank.encoder` accept vector inputs. When `rank.encoder = NULL` (default), `rank.encoder` is set equal to `rank` and pairs are evaluated element-wise (i.e.,  $(Q_1, R_1), (Q_2, R_2), \dots$ ). When `rank.encoder` is explicitly specified, all combinations of `rank` and `rank.encoder` are evaluated via `expand.grid`.

### Usage

```
nmfae.ecv(Y1, Y2 = Y1, rank = 1:2, rank.encoder = NULL, ...)
```

**Arguments**

Y1	Output matrix $Y_1$ ( $P_1 \times N$ ).
Y2	Input matrix $Y_2$ ( $P_2 \times N$ ). Default is Y1.
rank	Integer vector of decoder ranks to evaluate. Default is 1:2.
rank.encoder	Integer vector of encoder ranks to evaluate. Default is NULL, which sets rank.encoder = rank and evaluates element-wise pairs. When explicitly specified, all combinations with rank are evaluated.
...	Additional arguments passed to <code>nmfae</code> (e.g., <code>epsilon</code> , <code>maxit</code> ). Also accepts: <code>nfolds</code> (number of folds, default 5; <code>div</code> also accepted), <code>seed</code> (integer seed, default 123). For backward compatibility, <code>Q</code> and <code>R</code> are accepted as aliases for <code>rank</code> and <code>rank.encoder</code> .

**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 <code>div</code> containing the held-out element indices for each fold.
QR	Data frame with columns <code>Q</code> and <code>R</code> listing the evaluated pairs.

**See Also**

[nmfae](#), [nmfkc.ecv](#)

**Examples**

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

---

nmfae.heatmap

*Heatmap visualization of nmfae factor matrices*


---

**Description**

`nmfae.heatmap` displays the three factor matrices  $X_1$ ,  $\Theta$ , 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 <code>nmfae</code> .
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](#)

**Examples**

```
set.seed(1)
Y <- matrix(runif(20), nrow = 4)
res <- nmfae(Y, rank = 2)
nmfae.heatmap(res)
```

**Description**

Performs post-estimation inference for  $\Theta$  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 <code>nmfae</code> .
Y1	Output matrix $Y_1$ ( $P1 \times N$ ). Must match the data used in <code>nmfae()</code> .
Y2	Input matrix $Y_2$ ( $P2 \times N$ ). 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 $\sigma^2$ used for inference.
C.se	Sandwich standard errors for $\Theta$ ( $Q \times R$ matrix).
C.se.boot	Bootstrap standard errors for $\Theta$ ( $Q \times R$ matrix).
C.ci.lower	Lower CI bounds for $\Theta$ ( $Q \times R$ matrix).
C.ci.upper	Upper CI bounds for $\Theta$ ( $Q \times R$ matrix).
coefficients	Data frame with Estimate, SE, BSE, z, p-value for each element of $\Theta$ .
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, rank=2, rank.encoder=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,
  rank = 2,
  rank.encoder = rank,
  U,
  V = NULL,
  beta = NULL,
  plot = TRUE,
  ...
)
```

### Arguments

Y1	Output matrix $Y_1$ (P1 x N). Non-negative.
rank	Integer. Rank of the decoder basis. Default is 2.
rank.encoder	Integer. Rank of the encoder basis. Default is rank.
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 <a href="#">nmfkc.kernel.beta.nearest.med</a> .
plot	Logical. If TRUE (default), plots the objective function curve.
...	Additional arguments. Kernel-specific args (kernel, degree) are passed to <a href="#">nmfkc.kernel</a> ; all others (div, seed, shuffle, epsilon, maxit, etc.) are passed to <a href="#">nmfae.cv</a> . For backward compatibility, Q and R are accepted as aliases for rank and rank.encoder.

### 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**

```
Y <- matrix(cars$dist, nrow = 1)
U <- matrix(cars$speed, nrow = 1)
res <- nmfae.kernel.beta.cv(Y, rank = 1, rank.encoder = 1, U = U,
                           beta = c(0.01, 0.02, 0.05), nfolds = 5)
res$beta
```

---

nmfae.rename

*Rename decoder and encoder bases*


---

**Description**

Assigns user-specified names to the decoder ( $X_1$  columns) and encoder ( $X_2$  rows) bases of an `nmfae` object. The names propagate to  $\Theta$ , 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**

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

**Value**

A modified copy of `x` with updated names.

**See Also**

[nmfae](#)

## Examples

```
set.seed(1)
Y <- matrix(runif(15), nrow = 3)
res <- nmfae(Y, rank = 2, rank.encoder = 2)
res <- nmfae.rename(res,
  X1.colnames = c("Basis1", "Basis2"),
  X2.rownames = c("Enc1", "Enc2"))
summary(res)
```

---

nmfkc	<i>Optimize NMF with kernel covariates (Full Support for Missing Values)</i>
-------	--

---

## 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,
  verbose = TRUE,
  ...
)
```

## Arguments

Y	Observation matrix (P x N), OR a formula object for Formula Mode. In Formula Mode, use <code>Y1 + Y2 ~ A1 + A2</code> with data, or <code>Y_matrix ~ A_matrix</code> for direct matrix evaluation. Supports dot notation ( <code>. ~ A1 + A2</code> ) 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.
verbose	Logical. If TRUE (default), prints matrix dimensions and elapsed time.
...	Additional arguments passed for fine-tuning regularization, initialization, constraints, and output control. This includes the backward-compatible arguments Q and method. <ul style="list-style-type: none"> <li>• <code>Y.weights</code>: 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.</li> <li>• <code>X.L2.ortho</code>: Nonnegative penalty parameter for the orthogonality of X (default: 0). It minimizes the off-diagonal elements of the Gram matrix <math>X^T X</math>, reducing the correlation between basis vectors (conceptually minimizing <math>\ X^T X - \text{diag}(X^T X)\ _F^2</math>). (Formerly <code>lambda.ortho</code>).</li> <li>• <code>B.L1</code>: Nonnegative penalty parameter for L1 regularization on <math>B = CA</math> (default: 0). Promotes <b>sparsity in the coefficients</b>. (Formerly <code>gamma</code>).</li> <li>• <code>C.L1</code>: Nonnegative penalty parameter for L1 regularization on C (default: 0). Promotes <b>sparsity in the parameter matrix</b>. (Formerly <code>lambda</code>).</li> <li>• <code>Q</code>: Backward-compatible name for the rank of the basis matrix (Q).</li> <li>• <code>method</code>: Objective function: Euclidean distance "EU" (default) or Kullback–Leibler divergence "KL".</li> <li>• <code>X.restriction</code>: 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 <code>Y.symmetric = "bi" or "tri"</code>, because column normalization would prevent <math>XX^T</math> (or <math>XCX^T</math>) from approximating Y at the correct scale.</li> <li>• <code>X.init</code>: Method for initializing the basis matrix X. Options: "kmeans" (default), "kmeansr", "runif", "nndsvd", or a user-specified matrix. "kmeansr" applies k-means initialization and then fills zero entries with <code>Uniform(0, mean(Y)/100)</code>, analogous to NNDSVDar.</li> <li>• <code>nstart</code>: Number of random starts for kmeans when initializing X (default: 1).</li> <li>• <code>seed</code>: Integer seed for reproducibility (default: 123).</li> <li>• <code>C.init</code>: Optional numeric matrix giving the initial value of the parameter matrix C (i.e., <math>\Theta</math>). If A is NULL, C has dimension <math>Q \times N</math> (equivalently B); otherwise, C has dimension <math>Q \times K</math> where <math>K = \text{nrow}(A)</math>. Default initializes all entries to 1.</li> <li>• <code>Y.symmetric</code>: Character string specifying the type of symmetric NMF. "none" (default): standard NMF (<math>Y \approx XB</math>). "bi": 2-factor symmetric NMF (<math>Y \approx XX^T</math>). Internally implemented as the "tri" model with <math>C = I_Q</math> (identity matrix) held fixed, so that X is updated freely without column normalization. The multiplicative update for X uses cube-root damping (<math>X \leftarrow X \circ (\text{numerator}/\text{denominator})^{1/3}</math>) to prevent oscillation, since X appears in both factors of the decomposition (He et al., 2011,</li> </ul>

Proposition 1). "tri": 3-factor symmetric NMF ( $Y \approx X C X^T$ ) 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: Deprecated. Use verbose instead.
- 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 <code>match.call()</code> .
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 <code>X.restriction</code> .
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 <code>nmfkc.ar</code> or <code>nmfkc.kernel</code> .
formula.meta	If fitted via Formula Mode, a list with <code>formula</code> , <code>Y_cols</code> , and <code>A_cols</code> ; 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

```
# Example 1. Matrix Mode (Existing)
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)
res <- nmfkc(Y,rank=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, rank=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, rank=2, Y.symmetric="tri")
res_tri$C # Q x Q cluster interaction matrix
res_tri$XB # reconstruction X %*% C %*% t(X)
```

---

nmfkc.ar	<i>Construct observation and covariate matrices for a vector autoregressive model</i>
----------	---

---

## 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

<i>Y</i>	An observation matrix ( $P \times N$ ) or a <i>ts</i> object. If <i>Y</i> is a <i>ts</i> object (typically $N \times P$ ), it is automatically transposed to match the ( $P \times N$ ) 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:

<i>Y</i>	Observation matrix ( $P \times N_A$ ) used for NMF. Includes adjusted "tsp_info" attribute and time-based column names.
<i>A</i>	Covariate matrix ( $R \times N_A$ ) constructed according to the specified lag order. Includes adjusted "tsp_info" attribute and time-based column names.
A.columns	Index matrix used to generate <i>A</i> .
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 <b>stats</b> package.
objfunc	Objective function values for each candidate lag order.

**See Also**

[nmfkc.ar](#), [nmfkc.cv](#)

**Examples**

```
# 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, rank=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:  $T_{t-k} \rightarrow X$ ,
- Current latent factors:  $X \rightarrow T_t$ ,
- Optional intercept effects:  $\text{Const} \rightarrow X$ .

Importantly, *no direct edges from lagged variables to current outputs* ( $T_{t-k} \rightarrow T_t$ ) 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(
  result,
  degree = 1,
  intercept = any(colnames(result$C) == "(Intercept)"),
  threshold = 0.1,
```

```

    rankdir = "RL",
    fill = TRUE,
    weight_scale_xy = 5,
    weight_scale_lag = 5,
    weight_scale_int = 3,
    hide.isolated = TRUE
  )

```

### Arguments

result	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. Default is TRUE when an intercept column is detected in C, FALSE otherwise (auto-detected).
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 $X \rightarrow T$ .
weight_scale_lag	Scaling factor for lagged edges $T - k \rightarrow X$ .
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.

### See Also

[nmfkc.ar](#), [nmfkc](#), [plot.nmfkc.DOT](#)

### Examples

```

d <- AirPassengers
ar_data <- nmfkc.ar(d, degree = 2)
result <- nmfkc(ar_data$Y, ar_data$A, rank = 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.atr (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, rank = 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, rank = 1)
nmfkc.ar.stationarity(result)
```

---

nmfkc.class	<i>Create a class (one-hot) matrix from a categorical vector</i>
-------------	--

---

**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**

```
# Example.  
Y <- nmfkc.class(iris$Species)  
Y[,1:6]
```

---

nmfkc.criterion	<i>Compute model selection criteria for a fitted nmfkc model</i>
-----------------	--

---

**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**

<code>object</code>	An object of class "nmfkc" returned by <code>nmfkc</code> .
<code>Y</code>	The original observation matrix (P x N) used for fitting.
<code>detail</code>	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.
<code>...</code>	Additional arguments: <code>Y.weights</code> (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, rank = 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, rank = 2, data, ...)
```

### Arguments

<code>Y</code>	Observation matrix, or a formula (see <a href="#">nmfkc</a> for Formula Mode).
<code>A</code>	Covariate matrix. If NULL, the identity matrix is used. Ignored when <code>Y</code> is a formula.
<code>rank</code>	Rank of the basis matrix $X$ . Default is 2.
<code>data</code>	A data frame (required when <code>Y</code> is a formula with column names).
<code>...</code>	Additional arguments controlling CV and the internal <a href="#">nmfkc</a> call:
	<code>Y.weights</code> Optional numeric matrix or vector; 0 indicates missing/ignored values.
	<code>div</code> Number of folds ( $k$ ); default: 5.
	<code>seed</code> Integer seed for reproducible partitioning; default: 123.
	<code>shuffle</code> Logical. If TRUE (default), randomly shuffles samples (standard CV); if FALSE, splits sequentially (block CV; recommended for time series).
	<code>Q</code> (Deprecated) Alias for rank.
	<b>Arguments passed to <a href="#">nmfkc</a></b> e.g., <code>gamma(B.L1)</code> , <code>epsilon</code> , <code>maxit</code> , <code>method("EU" or "KL")</code> , <code>X.restriction</code> , <code>X.init</code> , etc.

### Value

A list with components:

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

### 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, rank = 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, rank = 1, nfolds = 10)$objfunc
}
betas[which.min(obj)]
```

---

nmfkc.denormalize	<i>Denormalize a matrix from [0, 1] back to its original scale</i>
-------------------	--

---

## 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:  $A \rightarrow C \rightarrow X \rightarrow Y$ .

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

**Usage**

```
nmfkc.DOT(
  result,
  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**

result	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 $A \rightarrow X$ (type "YXA").
weight_scale_xy	Scaling factor for edges $X \rightarrow Y$ .
weight_scale_ay	Scaling factor for edges $A \rightarrow Y$ (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](#), [nmfae.DOT](#), [nmf.sem.DOT](#), [nmfkc.ar.DOT](#), [plot.nmfkc.DOT](#)

**Examples**

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
result <- nmfkc(Y, A, rank = 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, rank = 1:3, data, ...)
```

**Arguments**

<code>Y</code>	Observation matrix, or a formula (see <a href="#">nmfkc</a> for Formula Mode).
<code>A</code>	Covariate matrix. Ignored when <code>Y</code> is a formula.
<code>rank</code>	Vector of ranks to evaluate (e.g., 1:5). For backward compatibility, <code>Q</code> is accepted via <code>...</code>
<code>data</code>	A data frame (required when <code>Y</code> is a formula with column names).
<code>...</code>	Additional arguments passed to <a href="#">nmfkc</a> (e.g., <code>method="EU"</code> ). Also accepts: <code>nfol</code> (number of folds, default 5; <code>div</code> also accepted), <code>seed</code> (integer seed, default 123).

**Value**

A list with components:

<code>objfunc</code>	Numeric vector containing the Mean Squared Error (MSE) for each Q.
<code>sigma</code>	Numeric vector containing the Residual Standard Error (RMSE) for each Q. Only available if <code>method="EU"</code> .
<code>objfunc.fold</code>	List of length equal to Q vector. Each element contains the MSE values for the k folds.
<code>folds</code>	A list of length <code>div</code> , 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, rank = 1:2, nolds = 3)
res$objfunc
```

---

nmfkc.inference	<i>Statistical inference for the parameter matrix <math>C</math> (<math>\Theta</math>)</i>
-----------------	--

---

**Description**

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

Under the working model  $Y = XCA + \varepsilon$  where  $\varepsilon_{pn} \stackrel{iid}{\sim} N(0, \sigma^2)$ , 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:

<code>sigma2.used</code>	Estimated $\sigma^2$ used for inference.
<code>C.se</code>	Sandwich standard errors for $C$ (Q x K matrix).
<code>C.se.boot</code>	Bootstrap standard errors for $C$ (Q x K matrix).
<code>C.ci.lower</code>	Lower CI bounds for $C$ (Q x K matrix).
<code>C.ci.upper</code>	Upper CI bounds for $C$ (Q x K matrix).
<code>coefficients</code>	Data frame with Estimate, SE, BSE, z, p-value for each element of $C$ .
<code>C.p.side</code>	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, rank = 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.gaussian](#), [nmfkc.cv](#)

**Examples**

```
# 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, rank=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, rank = 2, U, V = NULL, beta = NULL, plot = TRUE, ...)
```

**Arguments**

Y	Observation matrix $Y(P, N)$ .
rank	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 <code>nmfkc.cv</code> .

**Value**

A list with components:

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

**See Also**

[nmfkc.kernel.gaussian](#), [nmfkc.kernel.beta.nearest.med](#), [nmfkc.kernel](#)

**Examples**

```
# 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,rank=1,U,V,beta=25:30/1000)
A <- nmfkc.kernel(U,V,beta=28/1000)
result <- nmfkc(Y,A,rank=1)
plot(as.vector(V),as.vector(result$XB))
lines(as.vector(V),as.vector(result$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  $\beta$ .

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  $\sigma_0$  by the median of distances, then sets  $\beta_0 = 1/(2\sigma_0^2)$ .

If `Uk` is NULL,  $\sigma_0$  is estimated as the median of nearest-neighbor distances within `U` (excluding self-distance). If `Uk` is provided,  $\sigma_0$  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

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

## 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  $\sigma$  around  $\sigma_0$  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  $\beta$  values (logarithmic grid) intended for cross-validation.
- `dist_median`: The estimated distance scale  $\sigma_0$  (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).

**See Also**

[nmfkc.kernel.gaussian](#), [nmfkc.kernel.beta.cv](#)

**Examples**

```
# Basic (nearest-neighbor within U)
U <- matrix(runif(20), nrow = 2)
beta_info <- nmfkc.kernel.beta.nearest.med(U)
beta0 <- beta_info$beta
betas <- beta_info$beta_candidates

# With landmarks (nearest-landmark distances)
Uk <- matrix(runif(10), nrow = 2)

beta_info2 <- nmfkc.kernel.beta.nearest.med(U, Uk)
```

---

 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 $U(K, N) = (u_1, \dots, u_N)$ . 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. 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**

```
# 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 `detail="full"`).

**Usage**

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

**Arguments**

Y	Observation matrix, or a formula (see <code>nmfkc</code> 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.
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 <code>nmfkc</code> and <code>nmfkc.ecv</code> . <ul style="list-style-type: none"> <li>• Q: (Deprecated) Alias for rank.</li> <li>• save.time: (Deprecated) TRUE maps to <code>detail = "fast"</code>.</li> </ul>

**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

```
# Example.
Y <- t(iris[,-5])
# Full run (default)
nmfkc.rank(Y, rank=1:4)
# Fast run (skip ECV)
nmfkc.rank(Y, rank=1:4, detail="fast")
```

---

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.

**See Also**

[nmfkc](#), [residuals.nmf](#)

**Examples**

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

**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,  $\Theta$  ( $Q \times K$ ) 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  $\Theta$  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  $\Theta$ . 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  $\Theta$  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 ( $\text{cap} = \text{rate} * N * Q$ ). For backward compatibility, <code>dfU.cap.rate</code> is accepted via . . . . If NULL (default), runs <code>nmfre.dfU.scan</code> internally and selects the minimum rate where the cap is not binding. Use <code>nmfre.dfU.scan</code> beforehand to examine dfU behavior across rates and choose an appropriate value.
wild.bootstrap	Logical. If TRUE (default), perform wild bootstrap inference on $\Theta$ .
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. <ul style="list-style-type: none"> <li>• <code>X.init</code>: Initial basis matrix (P x Q), or NULL. When NULL, <code>nmfkc</code> is called internally to generate initial values.</li> <li>• <code>C.init</code>: Initial coefficient matrix (Q x K), or NULL. When NULL, <code>nmfkc</code> is called internally to generate initial values.</li> <li>• <code>U.init</code>: Initial random effects matrix (Q x N), or NULL (all zeros).</li> <li>• <code>prefix</code>: Prefix for basis names (default "Basis").</li> <li>• <code>sigma2</code>: Initial residual variance (default 1).</li> <li>• <code>sigma2.update</code>: Logical. Update <math>\sigma^2</math> during iterations (default TRUE).</li> <li>• <code>tau2</code>: Initial random effect variance (default 1).</li> <li>• <code>tau2.update</code>: Logical. Update <math>\tau^2</math> by moment matching (default TRUE). Disabled when dfU cap is active.</li> <li>• <code>dfU.control</code>: Either "cap" (default) to enforce a cap on dfU, or "off" for no cap.</li> <li>• <code>print.trace</code>: Logical. If TRUE, print progress every 100 iterations (default FALSE).</li> <li>• <code>seed</code>: Integer seed for reproducibility (default 1).</li> </ul>

- `C.p.side`: P-value sidedness: "one.sided" (default, for boundary null  $H_0: C=0$  vs  $H_1: C>0$ ) 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  $\Theta$  ( $Q \times K$ ).

`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  $df_U = N \sum_q d_q / (d_q + \lambda)$ , 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  $\lambda$  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`  $df_U / (NQ)$ , fraction of maximum df.

`dfU.cap.frac`  $df_U^{\text{cap}} / (NQ)$ .

**Fitted matrices**

- B Fixed-effect scores  $\Theta A$  ( $Q \times N$ ).
- 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  $\text{Var}(Y_n) = \tau^2 X X^\top + \sigma^2 I_P$ , 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:

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

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

**Inference on  $\Theta$  (wild bootstrap)**

- sigma2.used  $\hat{\sigma}^2$  used for inference.
- C.vec.cov Variance-covariance matrix for  $\text{vec}(\Theta)$  ( $QK \times QK$ ).
- C.se Standard error matrix for  $\Theta$  ( $Q \times K$ ).
- C.se.hess Sandwich (Hessian-based) SE matrix for  $\Theta$ .
- C.se.boot Bootstrap SE matrix for  $\Theta$ .
- coefficients Data frame with columns Estimate, Std. Error, z value,  $\text{Pr}(>|z|)$ , and confidence interval bounds for each element of  $\Theta$ .
- C.ci.lower Lower confidence interval matrix for  $\Theta$  ( $Q \times K$ ).
- C.ci.upper Upper confidence interval matrix for  $\Theta$  ( $Q \times K$ ).
- C.boot.sd Bootstrap standard deviation matrix for  $\Theta$  ( $Q \times K$ ).
- 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>

**See Also**

[nmfre.inference](#), [nmfre.dfU.scan](#), [nmfkc.DOT](#), [summary.nmfre](#)

**Examples**

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

# Example 2. Orthodont data (nlme)
if (requireNamespace("nlme", quietly = TRUE)) {
  Y <- matrix(nlme::Orthodont$distance, 4, 27)
  male <- ifelse(nlme::Orthodont$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, rank = 1)

  # Fit with chosen rate
  res <- nmfre(Y, A, rank = 1, df.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  $\text{rate} * N * Q$ , where  $N$  is the number of observations and  $Q$  is the rank. A suitable rate is one where the final  $\text{df}_U$  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 <a href="#">nmfre</a> .

### 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 $\text{rate} * N * Q$ .
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 ( $\text{dfU} / \text{dfU.cap} < 0.99$ ): 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 $\tau^2 \text{tr}(X^\top X) / (\tau^2 \text{tr}(X^\top X) + \sigma^2 P)$ . See <a href="#">nmfre</a> 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.

**See Also**[nmfre](#)**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, rank = 1, maxit = 1000)
print(tab)
```

```
# Example 2. Orthodont data (nlme)
if (requireNamespace("nlme", quietly = TRUE)) {
  Y <- matrix(nlme::Orthodont$distance, 4, 27)
  male <- ifelse(nlme::Orthodont$Sex[seq(1, 108, 4)] == "Male", 1, 0)
  A <- rbind(intercept = 1, male = male)
  nmfre.dfU.scan(1:10/10, Y, A, rank = 1)
}
```

nmfre.inference

*Statistical inference for the coefficient matrix C from NMF-RE***Description**

nmfre.inference performs statistical inference on the coefficient matrix  $C$  ( $\Theta$ ) 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 <a href="#">nmfre</a> .
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 $\sigma^2$ 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, rank = 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 $R^2$ .
------------	--

---

### 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 <a href="#">nmfae</a> .
...	Additional graphical parameters passed to plot.

### Value

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

### See Also

[nmfae](#), [nmfae.heatmap](#)

### Examples

```
set.seed(1)  
Y <- matrix(runif(20), nrow = 4)  
res <- nmfae(Y, rank = 2)  
plot(res)
```

---

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 "nmfæ.cv" returned by [nmfæ.cv](#).  
...                  Additional graphical parameters passed to barplot.

**Value**

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

**See Also**

[nmfæ.cv](#)

---

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

---

**Description**

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

**Usage**

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

**Arguments**

x                    An object of class "nmfæ.ecv" returned by [nmfæ.ecv](#).  
...                  Additional graphical parameters (currently unused).

**Value**

Invisible NULL. Called for its side effect of producing a plot.

**See Also**

[nmfæ.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.

**See Also**

[nmfkc](#), [summary.nmfkc](#)

**Examples**

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

This method handles all DOT objects produced by the nmfkc package: [nmfkc.DOT](#), [nmfae.DOT](#), [nmf.sem.DOT](#), and [nmfkc.ar.DOT](#).

**Usage**

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

**Arguments**

`x` An object of class "nmfkc.DOT" (or a subclass thereof).  
`...` Not used.

**Value**

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

**See Also**

[nmfkc.DOT](#), [nmfae.DOT](#), [nmf.sem.DOT](#), [nmfkc.ar.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.

## See Also

[nmfre](#), [nmf.sem](#)

## Examples

```
set.seed(1)
Y <- matrix(runif(20), nrow = 4)
A <- diag(5)
res <- nmfre(Y, A, rank = 2, wild.bootstrap = FALSE)
plot(res)
```

---

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](#)

### Examples

```
set.seed(1)  
Y <- matrix(runif(20), nrow = 4)  
res <- nmfae(Y, rank = 2)  
pred <- predict(res)  
plot(pred)
```

---

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  $X_1 \Theta X_2 \cdot \text{newY2}$ .

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 <a href="#">nmfae</a> .
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](#)

### Examples

```
set.seed(1)
Y <- matrix(runif(20), nrow = 4)
res <- nmfae(Y, rank = 2)
pred <- predict(res)
```

---

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").

### See Also

[nmfkc](#), [nmfkc.cv](#)

### Examples

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

---

```
print.nmf.inference
```

*Print method for NMF inference objects*

---

**Description**

Prints a summary of any NMF inference result object ("nmfkc.inference" or "nmfae.inference").

**Usage**

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

**Arguments**

x                    An object of class "nmf.inference".  
 ...                  Additional arguments passed to the corresponding print.summary.\* method.

**Value**

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

**See Also**

[nmfkc.inference](#), [nmfae.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**

<code>x</code>	An object of class "summary.nmfae.inference".
<code>digits</code>	Minimum number of significant digits.
<code>max.coef</code>	Maximum coefficient rows to display. Default is 20.
<code>...</code>	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 <code>summary.nmfkc</code> .
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, rank = 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.nmf

*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 'nmf'
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.

**See Also**

[nmfkc](#), [nmfae](#), [nmfre](#), [nmf.sem](#), [fitted.nmf](#)

**Examples**

```
Y <- matrix(runif(50), 5, 10)
result <- nmfkc(Y, rank = 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  $\Theta$ .

**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.

**See Also**

[nmfkc](#), [nmfkc.inference](#), [plot.nmfkc](#)

**Examples**

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(1, cars$speed)
result <- nmfkc(Y, A, rank = 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  $C(\Theta)$ .

**Usage**

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

**Arguments**

<code>object</code>	An object of class <code>"nmfkc.inference"</code> .
<code>...</code>	Additional arguments (currently unused).

**Value**

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

**See Also**

[nmfkc.inference](#), [summary.nmfkc](#)

---

summary.nmfre	<i>Summary method for objects of class nmfre</i>
---------------	--

---

### 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 <a href="#">nmfre</a> .
show_ci	Logical. If TRUE, show confidence interval columns (default FALSE).
...	Additional arguments (currently unused).

### Value

The input object, invisibly.

### See Also

[nmfre](#), [nmfre.inference](#)

### Examples

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

# Index

coef.nmf, 3

fitted.nmf, 4, 71

fitted.nmfae (fitted.nmf), 4

nmf.sem, 5, 5, 8, 10, 11, 13, 63, 71

nmf.sem.cv, 7, 7

nmf.sem.DOT, 7, 8, 11, 39, 62

nmf.sem.inference, 4, 7, 10, 10, 71

nmf.sem.split, 7, 12

nmfae, 5, 13, 16–21, 23, 59, 65, 71, 72

nmfae.cv, 15, 22, 23, 59, 60

nmfae.DOT, 15, 16, 20, 39, 62

nmfae.ecv, 15, 16, 18, 60

nmfae.heatmap, 19, 59

nmfae.inference, 3, 4, 15, 20, 67, 73

nmfae.kernel.beta.cv, 16, 22, 61

nmfae.rename, 23

nmfkc, 5, 13, 15, 24, 28, 31–36, 39–42, 49–52, 62, 66, 71, 74

nmfkc.ar, 27, 28, 30–33

nmfkc.ar.degree.cv, 28, 29, 36

nmfkc.ar.DOT, 28, 30, 39, 62

nmfkc.ar.predict, 32

nmfkc.ar.stationarity, 28, 33

nmfkc.class, 34, 65

nmfkc.criterion, 34

nmfkc.cv, 16, 27, 30, 35, 40, 43, 66

nmfkc.denormalize, 37, 48

nmfkc.DOT, 38, 55, 57, 58, 62

nmfkc.ecv, 19, 40, 49, 50

nmfkc.inference, 3, 4, 41, 67, 74

nmfkc.kernel, 15, 22, 23, 27, 42, 44, 48

nmfkc.kernel.beta.cv, 23, 36, 43, 46, 48

nmfkc.kernel.beta.nearest.med, 22, 44, 44, 48

nmfkc.kernel.gaussian, 43, 44, 46, 47

nmfkc.normalize, 37, 48

nmfkc.rank, 27, 35, 49

nmfkc.residual.plot, 50

nmfre, 5, 51, 55–58, 63, 71, 75

nmfre.dfU.scan, 52, 53, 55, 55

nmfre.inference, 3, 4, 55, 57, 75

plot, 61, 63

plot.nmf.sem (plot.nmfre), 63

plot.nmfae, 20, 59

plot.nmfae.cv, 59

plot.nmfae.ecv, 60

plot.nmfae.kernel.beta.cv, 61

plot.nmfkc, 61, 74

plot.nmfkc.DOT, 10, 31, 39, 62

plot.nmfre, 63

plot.predict.nmfae, 64, 65

predict.nmfae, 15, 64, 65

predict.nmfkc, 27, 66

print.nmf.inference, 67

print.summary.nmfae, 67, 72

print.summary.nmfae.inference, 68

print.summary.nmfkc, 69

print.summary.nmfkc.inference, 69

residuals.nmf, 5, 51, 70

residuals.nmfae (residuals.nmf), 70

residuals.nmfre (residuals.nmf), 70

summary.nmf.sem, 7, 71

summary.nmfae, 68, 72, 73

summary.nmfae.inference, 21, 68, 73

summary.nmfkc, 62, 73, 74

summary.nmfkc.inference, 42, 70, 74

summary.nmfre, 55, 58, 75