

Package ‘nmfkc’

August 25, 2026

Type Package

Title Non-Negative Matrix Factorization with Kernel Covariates

Version 0.9.6

Date 2026-08-23

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 and Tokuda (2026) <[doi:10.48550/arXiv.2607.27474](https://doi.org/10.48550/arXiv.2607.27474)>;
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

ByteCompile true

VignetteBuilder knitr, rmarkdown

Suggests knitr, rmarkdown, testthat (>= 3.0.0), mclust,
palmerpenguins, quanteda, vars, DiagrammeR, MASS, nlme, lavaan,
ade4

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

RoxygenNote 7.2.3

NeedsCompilation no

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

Repository CRAN

Date/Publication 2026-08-25 11:00:24 UTC

Contents

coef.nmf	4
fitted.nmf	5
nmf.cluster.criteria	6
nmf.cluster.flow	7
nmf.ffb	8
nmf.ffb.cv	12
nmf.ffb.DOT	13
nmf.ffb.inference	15
nmf.ffb.split	18
nmf.rrr	20
nmf.rrr.cv	23
nmf.rrr.DOT	24
nmf.rrr.ecv	26
nmf.rrr.heatmap	27
nmf.rrr.inference	28
nmf.rrr.kernel.beta.cv	30
nmf.rrr.rank	31
nmf.rrr.rename	32
nmf.rrr.signed	33
nmf.rrr.signed.ecv	37
nmf.rrr.signed.heatmap	38
nmf.rrr.signed.inference	39
nmf.rrr.signed.rank	40
nmf.rrr.signed.rename	42
nmfkc	43
nmfkc.ar	47
nmfkc.ar.degree.cv	49
nmfkc.ar.DOT	50
nmfkc.ar.latent	51
nmfkc.ar.latent.inference	54
nmfkc.ar.predict	58
nmfkc.ar.stationarity	59

nmfkc.ard	60
nmfkc.bicv	62
nmfkc.class	63
nmfkc.consensus	64
nmfkc.criterion	66
nmfkc.cv	67
nmfkc.cv.methods	68
nmfkc.denormalize	69
nmfkc.DOT	70
nmfkc.ecv	72
nmfkc.inference	74
nmfkc.kernel	76
nmfkc.kernel.beta.cv	77
nmfkc.kernel.beta.nearest.med	78
nmfkc.kernel.gaussian	80
nmfkc.net	82
nmfkc.net.DOT	84
nmfkc.net.ecv	86
nmfkc.net.inference	87
nmfkc.net.rank	88
nmfkc.normalize	89
nmfkc.rank	90
nmfkc.residual.plot	92
nmfkc.signed	93
nmfkc.signed.cv	96
nmfkc.signed.ecv	97
nmfkc.signed.rank	98
nmfkc.signed.rff	99
nmfre	101
nmfre.ecv	106
nmfre.inference	107
plot.nmf.cluster.criteria	109
plot.nmf.cluster.flow	110
plot.nmf.rank	111
plot.nmfae	112
plot.nmfkc	112
plot.nmfkc.ar.latent	113
plot.nmfkc.ar.stationarity	115
plot.nmfkc.ard	116
plot.nmfkc.consensus	116
plot.nmfkc.DOT	117
plot.nmfkc.signed	118
plot.nmfre	119
plot.nmfre.ecv	120
predict.nmfkc	120
predict.nmfkc.signed	122
predict.nmfre	123
print.nmf	123

print.nmf.cluster.criteria	124
print.nmf.cluster.flow	125
print.nmf.inference	125
print.nmf.rank	126
print.nmfae.ecv	126
print.nmfkc.ar.latent	127
print.nmfkc.ar.latent.inference	127
print.nmfkc.ar.stationarity	128
print.nmfkc.ard	128
print.nmfkc.consensus	129
print.nmfkc.DOT	129
print.nmfre.ecv	130
print.summary.nmf.sem	130
print.summary.nmfkc	131
print.summary.nmfkc.inference	131
print.summary.nmfkc.net	132
print.summary.nmfkc.net.inference	133
print.summary.nmfkc.net.signed	134
print.summary.nmfkc.signed	134
residuals.nmf	135
summary.nmf.sem	136
summary.nmfkc	137
summary.nmfkc.inference	137
summary.nmfkc.net	138
summary.nmfkc.net.inference	139
summary.nmfkc.net.signed	139
summary.nmfkc.signed	140
summary.nmfre	141

Index 143

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 (Θ) 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](#), [nmf.rrr.inference](#), [nmfre.inference](#), [nmf.ffb.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	<i>Extract fitted values from NMF models</i>
------------	--

Description

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

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

Usage

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

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

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

## 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 Y1 and Y2.
type	For nmfre objects: "blup" (default, the fit $X(\Theta A + U)$ including random effects) or "fixed" ($X\Theta A$, fixed effects only). Chosen so that $Y - \text{fitted}(\text{object}, \text{type})$ equals $\text{residuals}(\text{object}, Y, \text{type})$.

Value

The fitted matrix XB .

See Also

[nmfkc](#), [nmf.rrr](#), [nmfre](#), [nmf.ffb](#), [residuals.nmf](#)

Examples

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

nmf.cluster.criteria *Sample-clustering quality across ranks*

Description

Computes the clustering-quality criteria silhouette, CPCC, and dist.cor for a list of models fitted at different ranks (or a single fit), returning one row per rank. These are **clustering-stability** diagnostics (how decisively and faithfully the samples cluster), conceptually separate from the rank-selection `*.rank` functions (which use `r.squared`, effective rank, and ECV) and complementary to [nmf.cluster.flow](#) (which shows how the hard clustering itself changes across ranks).

Hard sample clustering requires a non-negative coefficient/score matrix (so the columns form a membership simplex); when a model's coefficient is signed (e.g. `nmfkc.signed`, `nmfae.signed`, `nmfre` fits whose coefficient has negative entries) the hard-label silhouette is NA while the distance-based CPCC and `dist.cor` are still computed.

Usage

```
nmf.cluster.criteria(fits, Y, Y2 = NULL, names = NULL, plot = TRUE, ...)
```

Arguments

fits	A list of fitted models, one per rank, all over the same N individuals (a single fitted model is also accepted and wrapped automatically). Supported families: nmfkc , nmfkc.signed , nmfae , nmfae.signed , nmfkc.net , nmfre , and nmf.sem / nmf.ffb .
------	--

<code>Y</code>	The original data matrix used to fit the models (Y_1 for <code>nmf.ffb</code>); required for the data-space distances.
<code>Y2</code>	Exogenous block, required only for <code>nmf.ffb</code> / <code>nmf.sem</code> .
<code>names</code>	Optional character vector (length <code>length(fits)</code>) of x-axis tick labels. Defaults to each result's <code>\$rank</code> .
<code>plot</code>	Logical; draw the diagnostics plot immediately (default <code>TRUE</code>); see plot.nmf.cluster.criteria .
<code>...</code>	When <code>plot = TRUE</code> , graphical arguments forwarded to plot.nmf.cluster.criteria .

Value

An object of class `"nmf.cluster.criteria"` (returned invisibly): a list with `criteria` (a data frame with one row per result and columns `rank`, `silhouette`, `CPCC`, `dist.cor`, and `hard`) and `labels` (the x-axis labels). Results are kept in the given order (not sorted).

See Also

[nmf.cluster.flow](#), [nmfkc.rank](#)

Examples

```
Y <- t(as.matrix(iris[, 1:4]))
fits <- lapply(2:6, function(q) nmfkc(Y, Q = q, print.dims = FALSE))
cc <- nmf.cluster.criteria(fits, Y, plot = FALSE)
cc$criteria
plot(cc)
```

<code>nmf.cluster.flow</code>	<i>Cluster-flow (alluvial) diagram across a sequence of fits</i>
-------------------------------	--

Description

Visualizes how the hard sample clustering changes across a sequence of fitted models – typically the same model at increasing ranks, but also different models at the same rank. Each individual is a line flowing left-to-right across the results (x-axis); its vertical position at each result is determined by its cluster, and clusters are reordered (barycenter method) to reduce crossings. Lines are coloured by the individual's cluster in the reference result, so one can see how the reference clusters split or merge. The adjusted Rand index (ARI) between each pair of adjacent results is printed along the top of the figure. X-axis ticks default to each result's `$rank` and can be overridden with `names`.

Works for any non-negative multiplicative-update family (`nmfkc`, `nmfae`, `nmfkc.net`, `nmfre`, and the signed variants); the hard label is the `argmax` of the coefficient/score matrix.

Usage

```
nmf.cluster.flow(fits, reference = NULL, names = NULL, plot = TRUE, ...)
```

Arguments

<code>fits</code>	A list ($\text{length} \geq 2$) of fitted models, all over the same N individuals. The results are taken in the given order (not sorted), so they may be different ranks or different models at the same rank.
<code>reference</code>	The index (1-based position in <code>fits</code>) of the result whose clustering defines the line colours. Defaults to the central result, $\text{floor}(\text{length}(\text{fits}) / 2) + 1$ (e.g. the 2nd of 2 or 3 results).
<code>names</code>	Optional character vector ($\text{length}(\text{length}(\text{fits}))$) of x-axis tick labels. Defaults to each result's $\$rank$.
<code>plot</code>	Logical; draw the diagram immediately by calling <code>plot.nmf.cluster.flow</code> (default TRUE). Set FALSE to only build the object and plot it later.
<code>...</code>	When <code>plot = TRUE</code> , graphical arguments forwarded to <code>plot.nmf.cluster.flow</code> (e.g. <code>col</code> , <code>lwd</code> , <code>xlab</code> , <code>ylab</code> , <code>main</code>).

Value

An object of class `"nmf.cluster.flow"` (returned invisibly): a list with `clusters` (the $N \times R$ table: rows = individuals, columns = results, entries = cluster number = the dominant-factor index of each fit, so it matches the factor/basis numbering of `fits`; a factor that never dominates leaves an empty, unused cluster number), `ypos` (the layout positions), `ranks` (each result's rank), `labels` (the x-axis labels), `reference` (the reference index), `ref.cluster` (the reference hard labels), `ARI` (adjusted Rand index between each pair of adjacent results, length $R - 1$), and `colors` (the default per-individual reference colour). Call `plot` on it to (re)draw the diagram.

See Also

`plot.nmf.cluster.flow`, `nmf.cluster.criteria`, `nmfkc.rank`

Examples

```
Y <- t(as.matrix(iris[, 1:4]))
fits <- lapply(2:6, function(q) nmfkc(Y, Q = q, print.dims = FALSE))
f1 <- nmf.cluster.flow(fits, reference = 2, plot = FALSE) # 2nd result
head(f1$clusters)
plot(f1, lwd = 2, main = "iris cluster flow")
```

nmf.ffb

NMF-FFB Main Estimation Algorithm (formerly NMF-SEM)

Description

Fits the NMF-FFB 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

$$Y_1 \approx (I - X\Theta_1)^{-1}X\Theta_2Y_2 \equiv M_{\text{model}}Y_2,$$

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

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

Usage

```
nmf.ffb(
  Y1,
  Y2,
  rank = NULL,
  X.init = "nndsvd",
  X.L2.ortho = 100,
  C1.L1 = 1,
  C2.L1 = 0.1,
  epsilon = 1e-06,
  maxit = 5000,
  seed = 123,
  ...
)
```

Arguments

Y1	A non-negative numeric matrix of endogenous variables with rows = variables (P1) , columns = samples (N) .
Y2	A non-negative numeric matrix of exogenous variables with rows = variables (P2) , columns = samples (N) . Must satisfy <code>ncol(Y1) == ncol(Y2)</code> .
rank	Integer; number of latent factors Q . If NULL, Q is taken from a hidden argument in ... or defaults to <code>nrow(Y2)</code> .
X.init	Initialization strategy for the basis matrix X ($P_1 \times Q$). One of: "nndsvd" (default): Non-negative Double SVD with additive randomness (NNDSDar; Boutsidis & Gallopoulos 2008), computed internally via <code>.nndsvdar(Y1, Q)</code>. Requires $Q \leq \min(P_1, N)$ (over-rank case falls back to "runif"). Uses a full SVD of Y_1, so for very large Y_1 consider switching to "kmeans" to avoid SVD memory / compute cost. "kmeans": k-means on the columns of Y_1 (samples clustered into Q groups); the transposed cluster centers become X. Scales well for large Y_1; this is the default of <code>nmfkc</code>. "kmeansar": "kmeans" followed by filling zero entries of X with <code>Uniform(0, $\bar{Y}_1/100$)</code> (NNDSDar-style additive randomness to escape trivial stationary points). "runif": Uniform random entries in $[0, 1]$.

- A numeric $P_1 \times Q$ matrix supplied by the user; negative entries are projected to 0.
- NULL: backward-compatible alias for "nndsvd".

In all cases the result is column-normalized to $\text{colSums}(X) = 1$ before iteration. The menu mirrors `nmfkc`'s `X.init` option for consistency across the package.

<code>X.L2.ortho</code>	L2 orthogonality penalty for X . This controls the penalty term $\lambda_X \ X^\top X - \text{diag}(X^\top X)\ _F^2$. Default: 100.
<code>C1.L1</code>	L1 sparsity penalty for $C1$ (i.e., Θ_1). Default: 1.0. Scale note: this function adds <code>C1.L1</code> to the multiplicative denominator, whereas <code>nmfkc</code> , <code>nmfae</code> and <code>nmfkc.net</code> add <code>C.L1</code> / 2. The same nominal value is therefore twice as strong here. The difference is retained so that published <code>nmf.ffb</code> fits reproduce; halve the value to match the other models.
<code>C2.L1</code>	L1 sparsity penalty for $C2$ (i.e., Θ_2). Default: 0.1. Same scale note as <code>C1.L1</code> .
<code>epsilon</code>	Relative convergence threshold for the objective function. Iterations stop when the relative change in reconstruction loss falls below this value. Default: 1e-6. Note: the test is on the unpenalized loss (<code>objfunc</code>), not on the penalized objective the updates actually minimize (<code>objfunc.full</code>). Every other optimizer in the package tests the penalized value, so with a large <code>X.L2.ortho</code> or <code>C*.L1</code> this function can stop while the quantity being optimized is still moving. Both traces are returned; compare them if the penalties are strong.
<code>maxit</code>	Maximum number of iterations for the multiplicative updates. Default: 5000 (matches <code>nmfkc</code> and other MU functions in the package).
<code>seed</code>	Random seed used to initialize X , $C1$, and $C2$. Default: 123.
<code>...</code>	Additional hidden arguments controlling the optional feedforward baseline (used both as an X warm-start and as the reference for <code>SC.map</code> , the input-output structural fidelity defined in Satoh (2025) §4.SC.map):
<code>nmfkc.baseline</code>	Controls whether a feedforward <code>nmfkc(Y1, A = Y2)</code> fit is used as baseline. Possible values: • Default (not given) — <code>nmf.sem</code> runs <code>nmfkc</code> internally when <code>X.init</code> is a string method ("nndsvd", "kmeans", ...) or NULL, forwarding <code>X.init</code>, <code>X.L2.ortho</code>, <code>epsilon</code>, <code>maxit</code>, <code>seed</code>. The fitted X of the baseline is then used as warm-start for the <code>nmf.sem</code> MU iterations, and <code>SC.map</code> is computed. This means <code>nmf.sem(Y1, Y2, rank = Q)</code> runs end-to-end without a prior <code>nmfkc</code> call. • TRUE — same as above, but force the internal <code>nmfkc</code> call even when <code>X.init</code> is a user-supplied matrix (the matrix is overridden). • FALSE — opt out; no internal call, <code>SC.map</code> = NA (pre-v0.6.8 behavior). • An <code>nmfkc</code> result (list with <code>\$X</code> and <code>\$C</code>) — use as the baseline directly (no internal call); also adopted as <code>X.init</code> when the latter is a string / NULL.
<code>M.simple</code>	Optional $P_1 \times P_2$ pre-computed baseline mapping. Takes precedence over <code>nmfkc.baseline</code> for the <code>SC.map</code> calculation but does not affect warm-start.
<code>Q</code>	Backward-compat alias for rank.

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 $\ X\Theta_1\ _{1,\text{op}}$.
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 $\ M_{\text{model}}\ _{1,\text{op}}/\ X\Theta_2\ _{1,\text{op}}$.
amplification.bound	Geometric-series upper bound $1/(1 - \ X\Theta_1\ _{1,\text{op}})$ if $\ X\Theta_1\ _{1,\text{op}} < 1$, otherwise Inf.
Q	Effective latent dimension used in the fit.
SC.cov	Correlation between sample and model-implied covariance (flattened) of Y_1 . See <i>second-moment fidelity</i> in Satoh (2025).
SC.map	Correlation between the equilibrium operator M_{model} and a feedforward baseline mapping $M_{\text{simple}} = X_0\Theta_0$, computed only when the baseline is supplied via <code>M.simple</code> or <code>nmfkc.baseline</code> in <code>...</code> ; otherwise NA. See <i>input-output structural fidelity</i> in Satoh (2025).
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.ffb.inference](#), [nmf.ffb.cv](#), [nmf.ffb.split](#), [nmf.ffb.DOT](#), [summary.nmf.sem](#)

Examples

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

nmf.ffb.cv

Cross-Validation for NMF-FFB

Description

Performs K-fold cross-validation to evaluate the equilibrium mapping of the NMF-FFB 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.ffb.cv(
  Y1,
  Y2,
  rank = NULL,
  X.init = "nndsvd",
  X.L2.ortho = 100,
  C1.L1 = 1,
  C2.L1 = 0.1,
  epsilon = 1e-06,
  maxit = 5000,
  ...
)
```

Arguments

Y1	A non-negative numeric matrix of endogenous variables with rows = variables (P1) , columns = samples (N) .
Y2	A non-negative numeric matrix of exogenous variables with rows = variables (P2) , columns = samples (N) . Must satisfy <code>ncol(Y1) == ncol(Y2)</code> .
rank	Integer; rank (number of latent factors) passed to <code>nmf.sem</code> . If <code>NULL</code> , <code>nmf.sem</code> decides the effective rank (via <code>...</code> or <code>nrow(Y2)</code>).

<code>X.init</code>	Initialization strategy for <code>X</code> , forwarded to <code>nmf.sem</code> . One of "nndsvd" (default), "kmeans", "kmeansar", "runif", a numeric $P_1 \times Q$ matrix, or NULL (alias for "nndsvd"). See <code>nmf.sem</code> for details.
<code>X.L2.ortho</code>	L2 orthogonality penalty for <code>X</code> .
<code>C1.L1</code>	L1 sparsity penalty for <code>C1</code> (Θ_1).
<code>C2.L1</code>	L1 sparsity penalty for <code>C2</code> (Θ_2).
<code>epsilon</code>	Convergence threshold for <code>nmf.sem</code> .
<code>maxit</code>	Maximum number of iterations for <code>nmf.sem</code> .
<code>...</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), <code>cores</code> (integer; number of parallel workers for the fold loop, default <code>getOption("mc.cores", 1L)</code>). Fold partitions and per-fold seeds are drawn before the loop and each <code>nmf.ffb</code> fit self-seeds, so results are identical to the sequential run for any <code>cores</code> .

Value

A numeric scalar: mean MAE across CV folds.

See Also

`nmf.ffb`

Examples

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

nmf.ffb.DOT

Generate a Graphviz DOT Diagram for an NMF-FFB Model

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 C_2 (exogenous loadings),
- $F_q \rightarrow Y_1$: rows of X (factor-to-endogenous mappings),
- $Y_1 \rightarrow F_q$: entries of C_1 (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.ffb.DOT(
  result,
  weight_scale = 5,
  weight_scale_c2 = weight_scale,
  weight_scale_x1 = weight_scale,
  weight_scale_feedback = weight_scale,
  threshold = 0.01,
  sig.level = 0.1,
  rankdir = "LR",
  fill = TRUE,
  cluster.box = c("normal", "faint", "invisible", "none"),
  cluster.labels = NULL,
  hide.isolated = TRUE,
  ...
)
```

Arguments

<code>result</code>	A list returned by <code>nmf.sem</code> , containing matrices X , C_1 , and C_2 .
<code>weight_scale</code>	Base scaling factor for edge widths.
<code>weight_scale_c2</code>	Scaling factor for edges $Y_2 \rightarrow F_q$ (C_2 matrix). Defaults to <code>weight_scale</code> .
<code>weight_scale_x1</code>	Scaling factor for edges $F_q \rightarrow Y_1$ (X matrix). Defaults to <code>weight_scale</code> .
<code>weight_scale_feedback</code>	Scaling factor for feedback edges $Y_1 \rightarrow F_q$ (C_1 matrix). Defaults to <code>weight_scale</code> .
<code>threshold</code>	Minimum coefficient value needed for an edge to be drawn.
<code>sig.level</code>	Significance level for filtering structural edges (C_1 feedback and C_2 exogenous loadings) when inference results are present. If <code>result</code> contains a <code>coefficients</code> data frame from <code>nmf.sem.inference</code> , only edges with <code>p_value < sig.level</code> are drawn, with significance stars (<code>* ** ***</code>) appended to the edge label. The X (factor-to- Y_1) edges are never starred since the basis is not the inference target. Set to <code>NULL</code> to disable significance filtering and fall back to the threshold magnitude filter for both C_1 and C_2 . Default is <code>0.1</code> .

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.
...	For backward compatibility: accepts deprecated names weight_scale_y2f (use weight_scale_c2) and weight_scale_fy1 (use weight_scale_x1).

Value

A character string representing a valid Graphviz DOT script.

See Also

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

Examples

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

nmf.ffb.inference

Statistical inference for NMF-FFB via X-fixed full pair bootstrap

Description

`nmf.sem.inference` performs statistical inference on the structural coefficient matrices C_1 (latent feedback, Θ_1) and C_2 (exogenous loading, Θ_2) from a fitted [nmf.sem](#) model.

The procedure is a **full pair bootstrap** that holds the basis matrix $\hat{X}$ from the original fit fixed across all replicates (which avoids label switching and gives a clean conditional interpretation: “uncertainty of the structural coefficients given the measurement model”):

1. For each replicate $b = 1, \dots, B$, resample column indices $(i_1, \dots, i_N)$ with replacement from $\{1, \dots, N\}$ and form $Y_1^{(b)} = Y_1[, i]$, $Y_2^{(b)} = Y_2[, i]$.
2. Re-estimate $(C_1^{(b)}, C_2^{(b)})$ by running the [nmf.sem](#) multiplicative updates *with* $X = \hat{X}$ held *fixed* (no X update; no centroid sort), using the same `C1.L1`, `C2.L1` as the original fit.
3. Discard replicates that violate stationarity ($\rho(XC_1^{(b)}) \geq 1$) or have an amplification ratio exceeding the geometric-series bound by more than 1%

Because $C_1, C_2 \geq 0$ are non-negative by construction, exact zeros are essentially never observed in the bootstrap distribution. Significance is assessed via a **support rate** at a small display threshold δ (default 0.01):

$$\text{sup}(c) = \frac{1}{|\text{valid}|} \sum_{b \in \text{valid}} \mathbf{1}(\hat{c}^{(b)} > \delta).$$

This is a one-sided counterpart of the classical p -value: large support_rate indicates strong evidence that the entry is meaningfully positive. Significance markers follow the lavaan convention with the natural correspondence $p = 1 - \text{sup}$: * (sup > 0.95), ** (sup > 0.99), *** (sup > 0.999). Cutoffs use strict greater-than so the rule mirrors the standard R convention for p -values ($p < 0.05 / 0.01 / 0.001 \rightarrow //$), translated to support_rate via $\text{sup} = 1 - p$.

Usage

```
nmf.ffb.inference(
  object,
  Y1,
  Y2,
  B = 1000L,
  threshold = 0.01,
  ci.level = 0.95,
  C1.L1 = 1,
  C2.L1 = 0.1,
  seed = 123L,
  ...
)
```

Arguments

object	A fitted object returned by <code>nmf.sem</code> . Must contain X, C1, C2.
Y1	Endogenous variable matrix (P1 x N). Must match the data used in <code>nmf.sem()</code> .
Y2	Exogenous variable matrix (P2 x N). Same.
B	Number of bootstrap replicates. Default 1000, the value the published analysis used; the other inference functions in the package default their <code>wild.B</code> to 500, and this one is deliberately left at 1000 so the manuscript results reproduce out of the box. Note that *** (sup > 0.999) is only reachable when <i>every</i> replicate clears threshold, at any B: raising B does not add a finer grade, it makes the same grade stronger evidence (all 1000 rather than all 500). Lowering it to 500 roughly halves the running time, since each replicate is a re-fit.
threshold	Display threshold δ for the support rate $\Pr_{\text{boot}}(\hat{c}^{(b)} > \delta)$. Default 0.01; entries below this magnitude are treated as effectively zero in the path diagram.
ci.level	Confidence level for the percentile bootstrap CI. Default 0.95.
C1.L1, C2.L1	L1 sparsity penalties used by the original <code>nmf.sem</code> fit. These must match the fit's hyperparameters for the bootstrap to estimate the correct model. Defaults (1.0, 0.1) match <code>nmf.sem</code> 's defaults but you should pass the actual values used.
seed	Base RNG seed for the bootstrap. Each replicate uses <code>seed + b</code> (resampling) and <code>seed + 1000 + b</code> (C_1, C_2 initialization). Default 123.

... Hidden options:

`epsilon` Convergence tolerance for the inner fixed-X MU loop. Default 1e-8 – deliberately tighter than a plain fit, because under multiplicative updates an entry heading for the non-negativity boundary approaches 0 slowly, so a loose tolerance stops every replicate while such an entry is still above threshold. The support rate is then inflated and significance over-declared: on a pure-noise design the support of null entries falls from 0.75 to 0.20 to 0.017 as `epsilon` goes 1e-6, 1e-8, 1e-10, while genuinely supported entries stay at 1. Tighten further when near-threshold entries matter; loosen only for speed.

`maxit` Maximum iterations for the inner MU loop. Default 100000 (the tighter tolerance needs the headroom).

`ncores` Number of parallel workers. Default 1 (serial). Cross-platform: uses `parallel::mclapply` on Linux/macOS and `parallel::parLapply` (PSOCK cluster) on Windows.

`print.trace` Logical, print progress. Default FALSE.

Value

The input object with additional bootstrap inference components:

`coefficients` Data frame with rows for every entry of C_1 and C_2 and columns Type ("C1" / "C2"), Basis, Covariate, Estimate, CI_low, CI_high, support_rate, p_value (= 1 – support_rate, for compatibility with downstream consumers such as [nmf.sem.DOT](#)), and sig.

`C1.support.rate, C2.support.rate`
Per-element support rates (Q x P1 and Q x P2 matrices).

`C1.ci.lower, C1.ci.upper, C2.ci.lower, C2.ci.upper`
Per-element percentile CI bounds.

`C1.array, C2.array`
Bootstrap distributions: 3D arrays of shape B x Q x P1 (and B x Q x P2). Invalid replicates contain NA.

`rho.boot, AR.boot, iter.boot`
Per-replicate spectral radius, amplification ratio, and inner-loop iteration count.

`bootstrap.B, bootstrap.threshold, bootstrap.ci.level`
Inputs recorded for reproducibility.

`bootstrap.n.valid, bootstrap.n.invalid`
Validity counts.

Lifecycle

This function's interface changed at v0.6.8: the legacy 1-step Newton wild bootstrap (with sandwich SE) has been replaced by the full pair bootstrap described above, following the paper revision. The fields `sigma2.used`, `C2.se`, `C2.se.boot`, `C2.p.side` that the previous implementation produced are no longer present.

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.ffb](#), [nmf.ffb.DOT](#)

Examples

```
Y <- t(iris[, -5])
Y1 <- Y[1:2, ]; Y2 <- Y[3:4, ]
res <- nmf.ffb(Y1, Y2, rank = 2)
res2 <- nmf.ffb.inference(res, Y1, Y2, B = 200) # quick demo
head(res2$coefficients)
```

nmf.ffb.split

Heuristic Variable Splitting for NMF-FFB

Description

Infers a heuristic partition of observed variables into exogenous (Y_2) and endogenous (Y_1) blocks for use in NMF-FFB. 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.ffb.split(
  x,
  n.exogenous = NULL,
  threshold = 0.1,
  auto.flipped = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

<code>x</code>	A numeric matrix or data frame with rows = samples and columns = observed variables .
<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)
<code>...</code>	Reserved for future use; currently unused (also accepted by the <code>nmf.ffb.split</code> alias for argument forwarding).

Value

A list with:

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

See Also

[nmf.ffb](#)

Examples

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

nmf.rrr

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

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

When $Y_2 = Y_1$, 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 Θ and X_2 .

Usage

```
nmf.rrr(
  Y1,
  Y2 = Y1,
  rank1 = 2,
  rank2 = NULL,
  epsilon = 1e-04,
  maxit = 5000,
  verbose = FALSE,
  ...
)
```

Arguments

Y1	Output matrix Y_1 (P1 x N). Non-negative. May contain NAs (handled via <code>Y1.weights</code>).
Y2	Input matrix Y_2 (P2 x N). Non-negative. Default is Y1 (autoencoder).
rank1	Integer. Rank of the response basis X_1 (P1 x Q). Default is 2.
rank2	Integer. Rank of the covariate basis X_2 (R x P2). Default (NULL) sets rank2 = rank1.
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:
	method Objective function: Euclidean distance "EU" (default) or Kullback-Leibler divergence "KL". Both use Lee-Seung multiplicative updates for the three factors X_1, Θ, X_2 of $Y_1 \approx X_1 \Theta X_2 Y_2$; for "KL" the residual SE sigma is NA (not on the data scale).

`Y1.weights` Optional non-negative weight matrix ($P1 \times N$) or vector for Y_1 , analogous to the `weights` argument of `lm`. Loss becomes $\sum W_{ij} (Y_{1,ij} - \hat{Y}_{1,ij})^2$ (`lm()`-style, **linear** in W). Logical matrices (TRUE / FALSE) are also accepted. Typical ECV / CV usage passes a binary mask $W \in \{0, 1\}$ for held-out elements; real-valued weights for importance weighting are also supported. Default: if $Y1$ has NA, a binary mask is auto-generated (0 for NA, 1 elsewhere).

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

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

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

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

`nstart` Number of random restarts for the `nmfkc()` initialisation steps (passed to the k-means initialiser of the X_1 and X_2 factorisations). Default 1 (single start; the historical behaviour). A larger value (e.g. 10-20) gives a more stable initialisation and is recommended before inference.

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

Rank aliases accepted here for backward compatibility: Q for rank1, R for rank2.

Value

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

<code>X1</code>	Decoder basis matrix ($P1 \times Q$), column sum 1.
<code>C</code>	Parameter matrix ($Q \times R$).
<code>X2</code>	Encoder basis matrix ($R \times P2$), row sum 1.
<code>Y1hat</code>	Fitted values $X_1 \Theta X_2 Y_2$ ($P1 \times N$).
<code>B1</code>	Decoder-side scores $B_1 = C X_2 Y_2$ ($Q \times N$), so that $\hat{Y}_1 = X_1 B_1$. Analogue of B in <code>nmfkc</code> .
<code>H</code>	Deprecated; identical to B1. Kept so that code written against earlier releases keeps working. Use B1.
<code>B2</code>	Encoder-side scores $B_2 = X_2 Y_2$ ($R \times N$), i.e. B_1 before the C map.
<code>B1.prob</code> , <code>B2.prob</code>	Column-normalized B_1 / B_2 : each SAMPLE's soft membership over the Q response groups and over the R covariate groups respectively.
<code>B1.cluster</code> , <code>B2.cluster</code>	Hard label per sample (argmax over the columns of the corresponding .prob).
<code>X1.prob</code>	Row-normalized X_1 ($P1 \times Q$): each RESPONSE VARIABLE's soft membership over the Q response groups.
<code>X1.cluster</code>	Hard response-group label per response variable (argmax over <code>X1.prob</code> rows).
<code>X2.prob</code>	Column-normalized X_2 ($R \times P2$): each COVARIATE VARIABLE's soft membership over the R covariate groups.
<code>X2.cluster</code>	Hard covariate-group label per covariate variable (argmax over <code>X2.prob</code> columns).
<code>rank</code>	Named integer vector c(Q, R).

method	Objective used ("EU" or "KL").
objfunc	Final objective value.
objfunc.iter	Objective values by iteration.
r.squared	$\text{cor}(Y, \hat{Y})^2$ (Pearson; in $[0, 1]$).
r.squared.uncentered	Uncentered $R^2 = 1 - \ Y - \hat{Y}\ _F^2 / \ Y\ _F^2$ (baseline = zero matrix).
r.squared.centered	Row-mean centered $1 - \ Y - \hat{Y}\ _F^2 / \ Y - \bar{Y}_p\ _F^2$.
niter	Number of iterations performed.
runtime	Elapsed time as a difftime object.
n.missing	Number of missing (or zero-weighted) elements in Y_1 .
n.total	Total number of elements in Y_1 ($P_1 \times N$).

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

- Satoh, K. and Tokuda, Y. (2026). Co-clustering of Response and Covariate Variables by Tri-Factorizing Their Non-negative Regression Coefficient Matrix. *arXiv preprint* arXiv:2607.27474. [doi:10.48550/arXiv.2607.27474](https://doi.org/10.48550/arXiv.2607.27474)
- 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

[nmf.rrr.inference](#), [predict.nmfae](#), [nmf.rrr.ecv](#), [nmf.rrr.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 <- nmf.rrr(Y, rank1=2, rank2=2)
res$r.squared

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

```
res2 <- nmf.rrr(Y1, Y2, rank1=2, rank2=2)
```

nmf.rrr.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

```
nmf.rrr.cv(Y1, Y2 = Y1, rank1 = 2, rank2 = NULL, ...)
```

Arguments

<code>Y1</code>	Output matrix Y_1 ($P_1 \times N$). Non-negative.
<code>Y2</code>	Input matrix Y_2 ($P_2 \times N$), or a kernel matrix ($N \times N$). Default is <code>Y1</code> (autoencoder).
<code>rank1</code>	Integer. Rank of the response basis. Default is 2.
<code>rank2</code>	Integer. Rank of the covariate basis. Default (<code>NULL</code>) = <code>rank1</code> .
<code>...</code>	Additional arguments passed to nmfae (e.g., <code>epsilon</code> , <code>maxit</code> , <code>Y1.weights</code>). Also accepts: <code>nfold</code> s (number of folds, default 5; <code>div</code> also accepted), <code>seed</code> (integer seed, default 123), <code>shuffle</code> (logical, default <code>TRUE</code>), and <code>cores</code> (fit the folds in parallel; default <code>getOption("mc.cores", 1L)</code> , <code>PSOCK</code> on Windows / forking elsewhere). Each fold is an independent self-seeded fit and the per-fold errors are summed in order, so the result is identical for any cores. For backward compatibility, <code>Q</code> , <code>R</code> are accepted as aliases for <code>rank</code> , <code>rank.encoder</code> . Rank aliases accepted here for backward compatibility: <code>Q</code> for <code>rank1</code> , <code>R</code> for <code>rank2</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.

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

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

Examples

```
Y <- t(iris[1:30, 1:4])
res <- nmf.rrr.cv(Y, rank1 = 2, rank2 = 2, nfolds = 5, maxit = 500)
res$sigma
```

nmf.rrr.DOT

DOT graph visualization for nmfae objects

Description

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

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

Usage

```
nmf.rrr.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 = "Response (X1)",
  X2.title = "Covariate (X2)",
  Y2.title = "Input (Y2)",
  hide.isolated = TRUE
)
```

Arguments

result	An object of class "nmfae" returned by nmfae .
type	Character. Graph type: "XCX" (default) or "YXCXY".
threshold	Numeric. Edges with values below this are omitted. Default is 0.01.
sig.level	Numeric or NULL. Significance level for filtering C edges when inference results are available. Only edges with p-value below sig.level are shown, with significance stars. Set to NULL to disable. Default is 0.1.
rankdir	Character. Graph direction for DOT layout. Default is "LR" (left to right).
fill	Logical. If TRUE, nodes are filled with color. Default is TRUE.
weight_scale	Numeric. Base scale factor for edge widths. Default is 5.
weight_scale_x1	Numeric. Scale factor for X_1 edges.
weight_scale_theta	Numeric. Scale factor for Θ edges.
weight_scale_x2	Numeric. Scale factor for X_2 edges.
Y1.label	Character vector of output variable labels.
X1.label	Character vector of decoder basis labels.
X2.label	Character vector of encoder basis labels.
Y2.label	Character vector of input variable labels.
Y1.title	Character. Title for output node group. Default is "Output (Y1)".
X1.title	Character. Title for the response-basis node group. Default is "Response (X1)".
X2.title	Character. Title for the covariate-basis node group. Default is "Covariate (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.

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

[nmf.rrr](#)

Examples

```
set.seed(1)
Y <- matrix(runif(20), nrow = 4)
res <- nmf.rrr(Y, rank1 = 2)
dot <- nmf.rrr.DOT(res)
```

nmf.rrr.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 `rank1` and `rank2` accept vector inputs. When `rank2 = NULL` (default), `rank2` is set equal to `rank1` 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

```
nmf.rrr.ecv(Y1, Y2 = Y1, rank1 = 1:2, rank2 = NULL, ...)
```

Arguments

<code>Y1</code>	Output matrix Y_1 ($P_1 \times N$).
<code>Y2</code>	Input matrix Y_2 ($P_2 \times N$). Default is <code>Y1</code> .
<code>rank1</code>	Integer vector of response-basis ranks to evaluate. Default is <code>1:2</code> .
<code>rank2</code>	Integer vector of covariate-basis ranks to evaluate. Default is <code>NULL</code> , which sets <code>rank2 = rank1</code> and evaluates element-wise pairs.
<code>...</code>	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), and <code>cores</code> (evaluate the (Q, R) -pair $\times$ fold grid in parallel; default <code>getOption("mc.cores", 1L)</code> , <code>PSOCK</code> on Windows / forking elsewhere). Each task is an independent self-seeded fit and results are aggregated in order, so the returned object is identical for any cores. For backward compatibility, <code>Q</code> and <code>R</code> are accepted as aliases for <code>rank</code> and <code>rank.encoder</code> . Rank aliases accepted here for backward compatibility: <code>Q</code> for <code>rank1</code> , <code>R</code> for <code>rank2</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 div containing the held-out element indices for each fold.
QR	Data frame with columns Q and R listing the evaluated pairs.

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

[nmf.rrr](#), [nmfkc.ecv](#)

Examples

```
Y <- t(iris[1:30, 1:4])
# Default: rank2=NULL -> paired rank1=rank2
res <- nmf.rrr.ecv(Y, rank1 = 1:2, nfolds = 3, maxit = 200)
res$sigma
# Explicit rank2: full grid (kept small so the example stays under CRAN's
# 10s budget -- a real sweep would use wider ranges and a larger maxit)
res2 <- nmf.rrr.ecv(Y, rank1 = 1:2, rank2 = 1:2, nfolds = 3, maxit = 200)
res2$sigma
```

nmf.rrr.heatmap

Heatmap visualization of nmfae factor matrices

Description

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

Usage

```
nmf.rrr.heatmap(
  x,
  Y1.label = NULL,
  X1.label = NULL,
  X2.label = NULL,
  Y2.label = NULL,
```

```

    palette = NULL,
    ...
  )

```

Arguments

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

Value

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

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

[nmf.rrr](#), [plot.nmfae](#), [nmf.rrr.DOT](#)

Examples

```

set.seed(1)
Y <- matrix(runif(20), nrow = 4)
res <- nmf.rrr(Y, rank1 = 2)
nmf.rrr.heatmap(res)

```

nmf.rrr.inference	<i>Statistical Inference for NMF-AE Parameter Matrix</i>
-------------------	--

Description

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

Usage

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

Arguments

<code>object</code>	An object of class "nmfae" returned by <code>nmfae</code> .
<code>Y1</code>	Output matrix Y_1 ($P1 \times N$). Must match the data used in <code>nmfae()</code> .
<code>Y2</code>	Input matrix Y_2 ($P2 \times N$). Default is <code>Y1</code> (autoencoder).
<code>wild.bootstrap</code>	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).
<code>...</code>	Additional arguments:
<code>wild.B</code>	Number of bootstrap replicates. Default is 500.
<code>wild.seed</code>	Seed for bootstrap. Default is 123.
<code>wild.level</code>	Confidence level for bootstrap CI. Default is 0.95.
<code>sandwich</code>	Logical. Use sandwich covariance. Default is TRUE.
<code>C.p.side</code>	P-value type: "one.sided" (default) or "two.sided".
<code>cov.ridge</code>	Ridge stabilization for information matrix inversion. Default is $1e-8$.
<code>print.trace</code>	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:

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

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

`nmf.rrr`, `summary.nmfae.inference`

Examples

```
Y <- matrix(c(1,0,1,0, 0,1,0,1, 1,1,0,0), nrow=3, byrow=TRUE)
res <- nmf.rrr(Y, rank1=2, rank2=2)
res2 <- nmf.rrr.inference(res, Y)
summary(res2)
```

nmf.rrr.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

```
nmf.rrr.kernel.beta.cv(
  Y1,
  rank1 = 2,
  rank2 = NULL,
  U,
  V = NULL,
  beta = NULL,
  plot = TRUE,
  ...
)
```

Arguments

Y1	Output matrix Y_1 (P1 x N). Non-negative.
rank1	Integer. Rank of the response basis. Default is 2.
rank2	Integer. Rank of the covariate basis. Default (NULL) = rank1.
U	Covariate matrix U (K x M). Rows are features, columns are samples (or knot points for non-symmetric kernels).
V	Covariate matrix V (K x N). If NULL (default), $V = U$ and a symmetric kernel is used.
beta	Numeric vector of candidate beta values. If NULL, automatically determined via nmfkc.kernel.beta.nearest.med .
plot	Logical. If TRUE (default), plots the objective function curve.
...	Additional arguments. Kernel-specific args (kernel, degree) are passed to nmfkc.kernel ; all others (div, seed, shuffle, epsilon, maxit, etc.) are passed to nmfae.cv . Also accepts cores to evaluate the candidate beta values in parallel (default <code>getOption("mc.cores", 1L)</code>); each inner CV then runs sequentially, and because results are gathered in order the selected beta is identical for any cores. For backward compatibility, Q and R are accepted as aliases for rank and rank.encoder. Rank aliases accepted here for backward compatibility: Q for rank1, R for rank2.

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

[nmf.rrr.cv](#), [nmfkc.kernel](#), [nmfkc.kernel.beta.cv](#)

Examples

```
Y <- matrix(cars$dist, nrow = 1)
U <- matrix(cars$speed, nrow = 1)
res <- nmf.rrr.kernel.beta.cv(Y, rank1 = 1, rank2 = 1, U = U,
                             beta = c(0.01, 0.02, 0.05), nfolds = 5)
res$beta
```

nmf.rrr.rank	<i>Rank selection for nmfae (paired rank, concise diagnostics)</i>
--------------	--

Description

Fits [nmfae](#) with a **paired** decoder/encoder rank ($Q = R$) across a range of ranks and reports `r.squared`, the effective rank (of the latent encoding H), and the element-wise CV error `sigma.ecv`, with the same concise plot as [nmfkc.rank](#). For a full (Q, R) grid use [nmfae.ecv](#) with `rank.encoder` and its heatmap.

Usage

```
nmf.rrr.rank(
  Y1,
  Y2 = Y1,
  rank1 = 1:5,
  detail = c("full", "fast"),
  plot = TRUE,
  ...
)
```

Arguments

Y1	Endogenous matrix ($P_1 \times N$).
Y2	Exogenous matrix; defaults to Y1 (autoencoder).
rank1	Integer vector of (paired) ranks to evaluate (both bases use the same value). Legacy Q accepted via ...

detail	"full" (default) also runs element-wise CV (sigma.ecv); "fast" skips it (plots r.squared and eff.rank only, and recommends the R-squared elbow).
plot	Logical; draw the diagnostics plot (default TRUE).
...	Passed on to nmfae and nmfae.ecv (e.g. \maxit, nfolds, seed). Also accepts cores to evaluate the rank sweep (and the element-wise CV) in parallel; default getOption("mc.cores", 1L). Each rank is an independent self-seeded fit and results are gathered in order, so the output is identical for any cores. Rank aliases accepted here for backward compatibility: Q for rank1.

Value

A list with rank.best and criteria (rank, effective.rank, effective.rank.ratio, r.squared, sigma.ecv).

References

Roy, O., & Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. *Proc. EUSIPCO*, 606–610. (effective.rank) Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4), 397–405. (sigma.ecv)

See Also

[nmf.rrr](#), [nmf.rrr.ecv](#), [nmfkc.rank](#)

nmf.rrr.rename

Renate 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 Θ , the coefficients table, and all downstream displays such as summary, nmfae.DOT, and nmfae.heatmap.

Usage

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

Arguments

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

Value

A modified copy of `x` with updated names.

See Also

`nmf.rrr`

Examples

```
set.seed(1)
Y <- matrix(runif(15), nrow = 3)
res <- nmf.rrr(Y, rank1 = 2, rank2 = 2)
res <- nmf.rrr.rename(res,
  X1.colnames = c("Resp1", "Resp2"),
  X2.rownames = c("Cov1", "Cov2"))
summary(res)
```

<code>nmf.rrr.signed</code>	<i>Signed-Bottleneck NMF-AE: Three-Layer NMF-AE with Signed Bottleneck</i>
-----------------------------	--

Description

`nmfae.signed` fits a three-layer non-negative matrix factorization autoencoder with a **signed** bottleneck, solving

$$Y_1 \approx X_1(C_+ - C_-)X_2Y_2, \quad X_1 \geq 0, C_+ \geq 0, C_- \geq 0, X_2 \geq 0,$$

where $\Theta = C_+ - C_-$ is the signed bottleneck. The basis matrices X_1 (columns sum to 1) and X_2 (rows sum to 1) retain their non-negative "parts-based" interpretability, while Θ can express anti-correlations (e.g., refractive index up vs. Abbe number down).

The algorithm uses **Direct Multiplicative Updates** derived from Ding et al. (2010) sign-splitting technique, applied block-wise to the four non-negative blocks (C_+, C_-, X_1, X_2) . Each block update monotonically decreases the true objective $\|Y_1 - X_1(C_+ - C_-)X_2Y_2\|_F^2$ (Lee-Seung auxiliary function method).

Relation to `nmfae`: When $\Theta \geq 0$ suffices (the `nmfae` case), `nmfae.signed` reduces to `nmfae` up to the $C_+ - C_-$ parameterization. Use `nmfae.signed` when the data exhibit negative cross-property correlations that tri-NMF-AE cannot express (e.g., high refractive index $\leftrightarrow$ low Abbe number trade-off).

Usage

```
nmf.rrr.signed(
  Y1,
  Y2 = Y1,
  rank1 = 2,
```

```

rank2 = NULL,
epsilon = 1e-04,
maxit = 5000,
verbose = FALSE,
...
)

```

Arguments

Y1	Output matrix Y_1 (P1 x N). Negative entries allowed (Y.signed = TRUE is auto-detected).
Y2	Input matrix Y_2 (P2 x N). Must be non-negative. Default Y1 (autoencoder).
rank1	Integer. Response-basis rank Q. Default 2.
rank2	Integer. Covariate-basis rank R. Default (NULL) = rank1.
epsilon	Relative convergence tolerance on the objective. Default 1e-4.
maxit	Maximum iterations. Default 5000.
verbose	Logical. Print progress. Default FALSE.
...	Additional arguments:
warm.start	One of TRUE (default, hybrid : warm-start X_1, X_2 from nmfae but initialize C_+, C_- randomly), "full" (warm-start everything including $C_+ = C_{\text{tri}}, C_- = \delta$), or FALSE (random for all blocks). The hybrid default avoids the $C_- = 0$ local-minimum trap inherited from tri-NMF-AE while still benefiting from good X_1, X_2 initialization. Ignored when Y_1 has negative entries.
nstart	Integer, default 1 (cf. kmeans). Number of random restarts for C_+, C_- . Each restart uses seed $\text{seed} + 7919 * (s-1)$. Returns the best run by final objective. Signed models have more local minima than non-negative ones because the bottleneck $\Theta = C_+ - C_-$ can take both positive and negative values; during exploration a larger nstart (e.g., 10-50) reduces the chance of being trapped at an inferior stationary point (particularly the $C_- = 0$ trap from warm-start from non-negative tri-NMF-AE). Use the default 1 for fast development and raise for publication-grade runs.
cores	Run the nstart restarts in parallel (only when nstart > 1). Default <code>getOption("mc.cores", 1L)</code> (PSOCK cluster on Windows, forking elsewhere). Each restart is fully self-seeded and the best run is selected by first-minimum objective, so the returned fit is identical for any cores.
X1.L2.ortho, X2.L2.ortho	Non-negative L2 orthogonality penalties (default 0) on the columns of X_1 and the rows of X_2 , penalizing $(\lambda/2)\ \text{offdiag}(X_1^\top X_1)\ ^2$ and $(\lambda/2)\ \text{offdiag}(X_2 X_2^\top)\ ^2$ respectively. Same convention as nmfae ; encourage more distinct (less overlapping) response / covariate bases.
C.L2	Non-negative ridge penalty (default 0) on the signed bottleneck $C = C_+ - C_-$, adding $\lambda\ C_+ - C_-\ ^2$. Shrinks Θ toward zero (with zero gradient on the unidentified common mode $C_+ + C_-$), injected into both the unweighted and weighted C_+/C_- updates.

`Y1.weights` Optional non-negative weight matrix ($P1 \times N$) or vector (length N) for Y_1 , analogous to the `weights` argument of `lm`. Loss becomes $\sum W_{ij} (Y_{1,ij} - \hat{Y}_{1,ij})^2$ (`lm()`-style, **linear** in W). Logical matrices (TRUE / FALSE) are also accepted. Used by `nmfae.signed.ecv` to hold out test elements via a binary mask $W \in \{0, 1\}$; real-valued weights for importance weighting are also supported. Default: if $Y1$ has NA, a binary mask is auto-generated (0 for NA, 1 elsewhere).

`Cp.init`, `Cn.init` Explicit $Q \times R$ non-negative matrices for initialization. Overrides `warm.start`.

`C.init` Explicit signed $Q \times R$ matrix, internally split into (C_+, C_-) .

`X1.init`, `X2.init` Explicit basis matrices.

`seed` RNG seed. Default 123.

`print.trace` Logical. Print iteration trace. Default FALSE.

`prefix.dec`, `prefix.enc` Label prefixes for the response/covariate bases. Default "Resp", "Cov".

Rank aliases accepted here for backward compatibility: `Q` for `rank1`, `R` for `rank2`.

Value

An object of class `c("nmfae.signed", "nmfae", "nmf")` with:

<code>X1</code>	Decoder basis ($P1 \times Q$), column sum 1.
<code>Cp</code> , <code>Cn</code>	Non-negative parts of Θ (each $Q \times R$).
<code>C</code>	Signed bottleneck $\Theta = C_+ - C_-$ ($Q \times R$).
<code>X2</code>	Encoder basis ($R \times P2$), row sum 1.
<code>Y1hat</code>	Fitted values $X_1(C_+ - C_-)X_2Y_2$.
<code>B1</code>	Decoder-side scores $B_1 = (C_+ - C_-)X_2Y_2$ ($Q \times N$), so that $\hat{Y}_1 = X_1B_1$. Signed , since C is.
<code>B2</code>	Encoder-side scores $B_2 = X_2Y_2$ ($R \times N$), i.e. B_1 before the C map. Non-negative.
<code>H</code>	Deprecated ; identical to <code>B1</code> . Kept so that code written against earlier releases keeps working. Use <code>B1</code> .
<code>B.prob</code> , <code>B.cluster</code>	Sample-level soft/hard clustering from a non-negative clip of the (possibly signed) B_1 ; use with care since C may be signed. There are deliberately no <code>B1.prob</code> / <code>B2.prob</code> counterparts: a column-normalized membership is not a distribution when the scores can be negative.
<code>X1.prob</code>	Row-normalized X_1 ($P1 \times Q$): each RESPONSE VARIABLE's soft membership over the Q response groups. Well defined ($X_1 \geq 0$).
<code>X1.cluster</code>	Hard response-group label per response variable.
<code>X2.prob</code>	Column-normalized X_2 ($R \times P2$): each COVARIATE VARIABLE's soft membership over the R covariate groups. Well defined ($X_2 \geq 0$).
<code>X2.cluster</code>	Hard covariate-group label per covariate variable.

rank	c(Q = Q, R = R).
dims	c(P1, P2, N).
objfunc, objfunc.iter	Final and per-iteration objective values.
r.squared	$\text{cor}(Y_1, \hat{Y}_1)^2$ (Pearson; in $[0, 1]$).
r.squared.uncentered	Uncentered $R^2 = 1 - \ Y_1 - \hat{Y}_1\ _F^2 / \ Y_1\ _F^2$ (baseline = zero matrix).
r.squared.centered	Row-mean centered $1 - \ Y_1 - \hat{Y}_1\ _F^2 / \ Y_1 - \bar{Y}_p\ _F^2$.
sigma, mae	Residual SE and mean absolute error.
niter, runtime	Iterations and elapsed seconds.
Y.signed	Logical; whether Y_1 contained negative entries.
call	Matched call.

Lifecycle

This function is **experimental**; interface may change.

References

- Satoh, K. and Tokuda, Y. (2026). Co-clustering of Response and Covariate Variables by Tri-Factorizing Their Non-negative Regression Coefficient Matrix. *arXiv preprint* arXiv:2607.27474. [doi:10.48550/arXiv.2607.27474](https://doi.org/10.48550/arXiv.2607.27474)
- Ding, C.H.Q., Li, T., and Jordan, M.I. (2010). Convex and Semi-Nonnegative Matrix Factorizations. *IEEE TPAMI*, 32(1), 45-55.
- Satoh, K. (2026). Signed-Bottleneck NMF-AE: Signed-Bottleneck 3-Layer NMF (research memo, 2026-04-18).
- Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

[nmf.rrr](#), [predict.nmfae.signed](#), [summary.nmfae.signed](#), [nmf.rrr.signed.rename](#)

Examples

```
set.seed(1)
Y1 <- matrix(abs(rnorm(12))), 3, 4)
Y2 <- matrix(abs(rnorm(20))), 5, 4)
res <- nmf.rrr.signed(Y1, Y2, rank1 = 2, rank2 = 2, maxit = 500)
summary(res)
```

nmf.rrr.signed.ecv *Element-wise Cross-Validation for Signed-Bottleneck NMF-AE*

Description

Element-wise k-fold cross-validation for `nmfae.signed` to select the decoder / encoder ranks (Q, R). Mirrors `nmfae.ecv` but uses the **weighted** Signed-Bottleneck NMF-AE fit path (`Y1.weights`): test-fold elements are zero-weighted during fitting, and held-out MSE is computed on those elements.

Usage

```
nmf.rrr.signed.ecv(Y1, Y2 = Y1, rank1 = 1:2, rank2 = NULL, ...)
```

Arguments

Y1	Output matrix (P1 x N).
Y2	Input matrix (P2 x N). Default Y1.
rank1	Integer vector of candidate response-basis ranks. Default 1:2.
rank2	Integer vector of candidate covariate-basis ranks, or NULL (default: pair rank2 = rank1, diagonal grid).
...	Additional arguments: nolds / div Number of folds. Default 5. seed RNG seed for fold assignment. Default 123. cores Evaluate the (Q, R)-pair $\times$ fold grid in parallel. Default <code>getOption("mc.cores", 1L)</code> (PSOCK on Windows, forking elsewhere). Each task is an independent self-seeded fit and results are aggregated in order, so the returned object is identical for any cores. nstart Number of random restarts per fit. Default 1. Signed models have more local minima (the bottleneck can carry both signs), so <code>nstart >= 10</code> is recommended for reproducible rank selection. Other args <code>epsilon</code> , <code>maxit</code> , <code>warm.start</code> , etc.\ are passed to <code>nmfae.signed</code> . Rank aliases accepted here for backward compatibility: Q for rank1, R for rank2.

Value

An object of class `c("nmfae.signed.ecv", "nmfae.ecv")` with `objfunc` (MSE per pair), `sigma` (RMSE), `objfunc.fold` (per-fold MSE), `folds`, `QR`, `paired`.

Lifecycle

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

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

[nmf.rrr.signed](#), [nmf.rrr.ecv](#)

nmf.rrr.signed.heatmap

Heatmap visualization of nmfae.signed factor matrices

Description

Displays the factor blocks of a [nmfae.signed](#) fit as side-by-side heatmaps. Non-negative blocks (X_1, C_+, C_-, X_2) use the white-orange-red palette; the signed combined bottleneck $C = C_+ - C_-$ is rendered with a diverging blue-white-red palette so positive and negative weights are visually distinguishable.

Usage

```
nmf.rrr.signed.heatmap(
  x,
  Y1.label = NULL,
  X1.label = NULL,
  X2.label = NULL,
  Y2.label = NULL,
  palette.pos = NULL,
  palette.signed = NULL,
  show.C = TRUE,
  ...
)
```

Arguments

x	An object of class "nmfae.signed".
Y1.label	Character vector for rows of X_1 .
X1.label	Decoder basis labels.
X2.label	Encoder basis labels.
Y2.label	Input variable labels.
palette.pos	Palette for non-negative blocks. Default white-orange-red.
palette.signed	Palette for signed C . Default blue-white-red.
show.C	Logical. If TRUE (default), shows the combined signed $C = C_+ - C_-$ as a separate panel.
...	Not used.

Value

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

Lifecycle

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

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

[nmf.rrr.signed](#), [nmf.rrr.heatmap](#)

Examples

```
set.seed(1)
Y1 <- matrix(abs(rnorm(12))), 3, 4)
Y2 <- matrix(abs(rnorm(20))), 5, 4)
res <- nmf.rrr.signed(Y1, Y2, rank1 = 2, rank2 = 2, maxit = 200)
nmf.rrr.signed.heatmap(res)
```

nmf.rrr.signed.inference

Statistical Inference for Signed-Bottleneck NMF-AE Signed Bottleneck

Description

Post-estimation inference for the **signed** bottleneck $\Theta = C_+ - C_-$ in the Signed-Bottleneck NMF-AE model $Y_1 \approx X_1 \Theta X_2 Y_2$, conditional on $(\hat{X}_1, \hat{X}_2)$. Uses sandwich covariance and wild bootstrap **without** the non-negativity projection that [nmfae.inference](#) applies (because Θ is unconstrained in sign here).

Usage

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

Arguments

object	A fitted "nmfae.signed" object.
Y1	Output matrix used during fitting.
Y2	Input matrix used during fitting. Default Y1.
wild.bootstrap	Logical. Default TRUE.
...	Additional arguments:

`wild.B` Bootstrap replicates. Default 500.
`wild.seed` RNG seed. Default 123.
`wild.level` CI confidence level. Default 0.95.
`sandwich` Use sandwich covariance. Default TRUE.
`C.p.side` P-value type: "two.sided" (default for Signed-Bottleneck NMF-AE) or "one.sided".
`cov.ridge` Ridge stabilization. Default 1e-8.
`print.trace` Logical. Default FALSE.

Value

An object of class `c("nmfae.signed.inference", "nmfae.inference", "nmfae.signed", "nmfae", "nmf")` with added fields:

`sigma2.used` Estimated σ^2 .
`C.se, C.se.boot` Sandwich / bootstrap SEs for Θ ($Q \times R$).
`C.ci.lower, C.ci.upper` Bootstrap CIs.
`coefficients` Data frame with Estimate, SE, BSE, z, p-value, CI.
`C.p.side` P-value side used.

Lifecycle

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

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

[nmf.rrr.signed](#), [nmf.rrr.inference](#)

`nmf.rrr.signed.rank` *Rank selection for nmfae.signed (paired rank, concise diagnostics)*

Description

Fits [nmfae.signed](#) with a **paired** decoder/encoder rank ($Q = R$) across a range of ranks and reports `r.squared`, the effective rank (of the latent encoding H), and the element-wise CV error `sigma.ecv`, with the same concise plot as [nmfkc.rank](#). For a full (Q, R) grid use [nmfae.signed.ecv](#).

Usage

```
nmf.rrr.signed.rank(
  Y1,
  Y2 = Y1,
  rank1 = 1:5,
  detail = c("full", "fast"),
  plot = TRUE,
  ...
)
```

Arguments

Y1	Endogenous matrix ($P_1 \times N$); may be signed.
Y2	Exogenous matrix; defaults to Y1.
rank1	Integer vector of (paired) ranks to evaluate (both bases use the same value). Legacy Q accepted via
detail	"full" (default) also runs element-wise CV (<code>sigma.ecv</code>); "fast" skips it (plots <code>r.squared</code> and <code>eff.rank</code> only, and recommends the R-squared elbow).
plot	Logical; draw the diagnostics plot (default TRUE).
...	Passed on to <code>nmfae.signed</code> and <code>nmfae.signed.ecv</code> . Also accepts <code>cores</code> to evaluate the rank sweep (and the element-wise CV) in parallel; default <code>getOption("mc.cores", 1L)</code> . Each rank is an independent self-seeded fit and results are gathered in order, so the output is identical for any cores. Rank aliases accepted here for backward compatibility: Q for rank1.

Value

A list with `rank.best` and `criteria` (`rank`, `effective.rank`, `effective.rank.ratio`, `r.squared`, `sigma.ecv`).

References

Roy, O., & Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. *Proc. EUSIPCO*, 606–610. (`effective.rank`) Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4), 397–405. (`sigma.ecv`)

See Also

`nmf.rrr.signed`, `nmf.rrr.signed.ecv`, `nmfkc.rank`

nmf.rrr.signed.rename *Rename Resp/Cov labels on nmfae.signed objects*

Description

Replaces the default "Resp1", "Resp2", ... (response basis / X1 columns and Cp/Cn/C rows) and "Cov1", "Cov2", ... (covariate basis / X2 rows and Cp/Cn/C columns) with user-supplied labels. Propagates to coefficients tables if present (e.g., from nmfae.inference).

Usage

```
nmf.rrr.signed.rename(x, X1.colnames = NULL, X2.rownames = NULL)
```

Arguments

x	An "nmfae.signed" object.
X1.colnames	Character vector of length Q for decoder labels.
X2.rownames	Character vector of length R for encoder labels.

Value

The renamed object (same class).

Lifecycle

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

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

[nmf.rrr.signed](#)

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.

- **Y.weights**: Optional weight matrix ($P \times N$) or vector (length N) with non-negative entries, analogous to the `weights` argument of `lm`. When supplied, the objective becomes $\sum W_{ij} (Y_{ij} - (XB)_{ij})^2$ (i.e. **linear** in W ; `lm()`-style weighted least squares). Logical matrices (TRUE / FALSE) are also accepted and coerced to 1 / 0. The primary use case is missing-value masking for ECV / CV, where $W_{ij} \in \{0, 1\}$ (FALSE / TRUE) indicates held-out vs. used elements; real-valued weights for observation-level importance weighting are also supported. Default NULL: if Y contains NA a binary mask is auto-constructed (0 for NA, 1 elsewhere); otherwise no weighting.
- **X.L2.ortho**: Nonnegative penalty parameter for the orthogonality of X (default: 0). It minimizes the off-diagonal elements of the Gram matrix $X^\top X$, reducing the correlation between basis vectors (conceptually minimizing $\|X^\top X - \text{diag}(X^\top X)\|_F^2$). (Formerly `lambda.ortho`).
- **X.L2.smooth**: Nonnegative penalty parameter for row-smoothness of X (default: 0). Adds $\lambda \text{tr}(X^\top L X)$ with L the path-graph Laplacian over the P rows, i.e. it penalizes squared differences between adjacent rows $\sum_q \sum_{j \geq 2} (x_{jq} - x_{j-1,q})^2$, yielding gently-varying (smooth) bases. Integrates with the multiplicative updates (non-negativity and monotone descent preserved). Useful when the rows of Y have a natural order (e.g. time points).
- **C.L1**: Nonnegative penalty parameter for L1 regularization on C (default: 0). Promotes **sparsity in the parameter matrix** Θ (variable selection over basis-covariate links). (Formerly `lambda`).
- **Q**: Backward-compatible name for the rank of the basis matrix (Q).
- **method**: Objective function: Euclidean distance "EU" (default) or Kullback–Leibler divergence "KL".
- **X.restriction**: Constraint for columns of X . Options: "colSums" (default), "colSqSums", "totalSum", "none", or "fixed". "none" applies no normalization to X after each update, allowing it to absorb the scale freely.
- **X.init**: Method for initializing the basis matrix X . Options: "kmeans" (default), "kmeansar", "kmeans++", "runif", "nnsvd", or a user-specified matrix. "kmeansar" applies k -means initialization and then fills zero entries with `Uniform(0, mean(Y)/100)`, analogous to `NNDSVDar`. "kmeans++" seeds the k -means centres by D^2 weighting (Arthur & Vassilvitskii, 2007) before Lloyd refinement, giving a more careful, stable initialization (`nstart` is not used in this case).
- **nstart**: Number of random starts for initialization of X (default: 1). Used by `kmeans` (when `X.init` = "kmeans" or "kmeansar") and by the multi-start evaluation (when `X.init` = "runif").
- **seed**: Integer seed for reproducibility (default: 123).
- **C.init**: Optional numeric matrix giving the initial value of the parameter matrix C (i.e., Θ). If A is NULL, C has dimension $Q \times N$ (equivalently B); otherwise, C has dimension $Q \times K$ where $K = \text{nrow}(A)$. Default initializes all entries to 1.
- **Y.symmetric**: **Removed**. Symmetric NMF ($Y \approx XX^\top$ or $XCX^\top$) has moved to the dedicated `nmfkc.net` function (types "tri", "bi", "signed"),

which uses the correct Frobenius bilateral-gradient updates. Passing `Y.symmetric` to `nmfkc()` now stops with a message pointing to `nmfkc.net()`.

- `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. "fast" (**default**) computes everything except the $O(N^2)$ sample-clustering criteria; "full" adds silhouette, CPCC and `dist.cor`, which need two $N \times N$ distance matrices and a cophenetic correlation — at $N = 2000$ that was 26 times the cost of the rest of the call; "minimal" returns only information criteria and skips XB . The default changed from "full" in 0.8.9: nothing consumed those three (see the `criterion` entry under `Value`), so every call was paying for them. For backward compatibility, `save.time = TRUE` maps to "fast" and `save.memory = TRUE` maps to "minimal".

Value

A list with components:

<code>call</code>	The matched call, as captured by <code>match.call()</code> .
<code>dims</code>	A character string summarizing the matrix dimensions of the model.
<code>runtime</code>	A character string summarizing the computation time.
<code>X</code>	Basis matrix. Column normalization depends on <code>X.restriction</code> .
<code>B</code>	Coefficient matrix $B = CA$.
<code>XB</code>	Fitted values for Y .
<code>C</code>	Parameter matrix.
<code>B.prob</code>	Soft-clustering probabilities derived from columns of B .
<code>B.cluster</code>	Hard-clustering labels (argmax over $B.prob$ for each column).
<code>X.prob</code>	Row-wise soft-clustering probabilities derived from X .
<code>X.cluster</code>	Hard-clustering labels (argmax over $X.prob$ for each row).
<code>X.restriction</code>	The constraint that was applied to the columns of X . It decides how much of the $(X, C) \rightarrow (XT, T^{-1}C)$ freedom is pinned down, which inference on the latent parameters needs to know.
<code>A.attr</code>	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> .
<code>formula.meta</code>	If fitted via Formula Mode, a list with <code>formula</code> , <code>Y_cols</code> , and <code>A_cols</code> ; otherwise NULL.
<code>objfunc</code>	Final objective value.
<code>objfunc.iter</code>	Objective values by iteration.
<code>r.squared</code>	$R^2 = \text{cor}(Y, XB)^2$ (Pearson; scale-invariant; $[0, 1]$).

<code>r.squared.uncentered</code>	Uncentered $R^2 = 1 - \ Y - XB\ _F^2 / \ Y\ _F^2$ (baseline = zero matrix; natural for non-negative factorizations without an intercept).
<code>r.squared.centered</code>	Row-mean centered $R^2 = 1 - \ Y - XB\ _F^2 / \ Y - \bar{Y}_{p\cdot}\ _F^2$, the multivariate regression R^2 .
<code>method</code>	Character string indicating the optimization method used ("EU" or "KL").
<code>n.missing</code>	Number of missing (or zero-weighted) elements in Y .
<code>n.total</code>	Total number of elements in Y .
<code>rank</code>	The rank Q used in the factorization.
<code>sigma</code>	The residual standard error, representing the typical deviation of the observed values Y from the fitted values XB .
<code>mae</code>	Mean Absolute Error between Y and XB .
<code>criterion</code>	A list with <code>effective.rank</code> and <code>effective.rank.index</code>. <code>effective.rank.index</code> is <code>effective.rank</code> rescaled onto $[0, 1]$ by the broken-stick correction $(\hat{r}_Q - E_Q)/(Q - E_Q)$ with $E_Q = \exp(H_Q - 1)$, the effective rank a random split of the variance would produce; it is the quantity <code>nmfkc.rank</code> plots, and summary prints it in place of the crispness. Read it as how evenly the across-sample coefficient variance is shared, which is not the same as how useful the factors are: a factor whose coefficient is large but constant carries no variance and pulls the index down (correctly — the raw value can go negative and is clamped at 0, meaning the variance is spread over fewer factors than chance would give), while two duplicated factors split the variance evenly and score near 1. NA at $Q = 1$, where $E_1 = Q$. With <code>detail = "full"</code> the list additionally carries the sample-clustering criteria <code>silhouette</code> (mean silhouette width of the hard clustering, computed in the original data space <code>dist(t(Y))</code> with the per-sample labels), <code>CPCC</code> (cophenetic correlation of a hierarchical clustering of the coefficient distances <code>dist(t(B))</code>) and <code>dist.cor</code> (correlation between original-data and coefficient distances). They are absent by default because they cost $O(N^2)$ and nothing consumes them: they are no longer part of rank selection, <code>summary.nmfkc</code> does not print them, and <code>nmf.cluster.criteria</code> — the right entry point when several ranks are to be compared — recomputes them from the fits it is given. <code>effective.rank</code> is the effective rank: $\exp$ of the Shannon entropy of the explained-variance distribution $p_k = \text{var}(B_{k\cdot}) / \sum_j \text{var}(B_{j\cdot})$. By the trace identity $\sum_k \text{var}(B_{k\cdot}) = \text{tr}(\text{Cov}(B))$, p_k is the exact fraction of the total coefficient variance carried by factor k, so the entropy measures how that variance is spread across factors. It ranges in $[1, Q]$ (1 when one factor carries all the variance, Q when all contribute equally) and counts the number of latent factors that actively shape across-sample variation. This is the PCA-style explained-variance / effective-dimensionality measure and reuses the $\exp(\text{entropy})$ functional form of Roy & Vetterli (2007).

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

Roy, O., & Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. In *15th European Signal Processing Conference (EUSIPCO)* (pp. 606–610).

See Also

[nmfkc.inference](#) (standard errors and p-values for C), [summary.nmfkc](#), [plot.nmfkc](#), [nmfkc.cv](#), [nmfkc.ecv](#), [nmfkc.rank](#), [nmfkc.kernel](#), [nmfkc.ar](#), [nmfkc.DOT](#), [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)

# For symmetric NMF (Y approximated by  $X X^T$  or  $X C X^T$ ),
# use \link{nmfkc.net}() instead.
```

nmfkc.ar

Construct observation and covariate matrices for a vector autoregressive model

Description

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

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

Usage

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

Arguments

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

Value

A list containing:

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

References

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

See Also

[nmfkc](#), [nmfkc.ar.degree.cv](#), [nmfkc.ar.stationarity](#), [nmfkc.ar.DOT](#)

Examples

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

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

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

Description

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

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

Usage

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

Arguments

Y	Observation matrix $Y(P, N)$ or a ts object.
rank	Rank of the basis matrix. For backward compatibility, Q is accepted via . . .
degree	A vector of candidate lag orders to be evaluated.
intercept	Logical. If TRUE (default), an intercept is added to the covariate matrix.
plot	Logical. If TRUE (default), a plot of the objective function values is drawn.
...	Additional arguments passed to nmfkc.cv. Also accepts cores (getOption("mc.cores", 1L)) to evaluate the candidate degrees in parallel; results are identical to the sequential loop for any cores (PSOCK cluster on Windows, forking elsewhere).

Value

A list with components:

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

See Also

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

Examples

```
# 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

<code>result</code>	A fitted <code>nmfkc</code> object representing the AR model. Must contain matrices <code>X</code> and <code>C</code> .
<code>degree</code>	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.latent

Latent transition matrices of an NMF-VAR model

Description

This function is **experimental**. The interface may change in future versions: argument names, defaults and the contents of the returned object are not yet stable.

Reduces a fitted NMF-VAR to the $Q \times Q$ operators that drive its coefficient vector. Writing $\mathbf{b}_t = \Theta \mathbf{a}_t$ for the fitted scores, the residual-free recursion is

$$\mathbf{b}_t = \sum_{d=1}^D G_d \mathbf{b}_{t-d} + \boldsymbol{\theta}, \quad G_d = \Theta_d X \quad (Q \times Q),$$

and the G_d carry the model's whole deterministic structure — transition weights, propagation, roots, the long-run mean. They stay readable where the $P \times P$ matrices $\Xi_d = X\Theta_d$ do not: 4×4 instead of 47×47 .

Use `plot.nmfkc.ar.latent` to draw them, `nmfkc.ar.latent.inference` to attach standard errors, and `nmfkc.ar.stationarity` if the stationarity verdict is all that is wanted.

Usage

```
nmfkc.ar.latent(x)
```

Arguments

`x` A fitted "nmfkc" object obtained from an `nmfkc.ar` design (`nmfkc(ar$Y, ar$A, rank = Q)`).

Details

The latent process is VARMA, not VAR. The display above drops the observation residual. With $y_t = Xb_t + e_t$ the exact identity is

$$b_t = \sum_{d=1}^D G_d b_{t-d} + \theta + \sum_{d=1}^D \Theta_d e_{t-d},$$

i.e. a VARMA(D, D) in general (verified to machine precision; dropping the moving-average term leaves an error of order the residuals themselves). G_d is therefore the **autoregressive operator of the fitted VAR reduced to the latent space**, not the coefficient matrix of a latent VAR that the scores obey exactly. In particular an entry of G_d should not be read as Granger causality between latent conditions: the moving-average term is common to all coordinates and is not conditioned on.

Direction convention: $(G_d)_{qq'}$ is the effect of $b_{q', t-d}$ on $b_{q, t}$ — **row = effect** (at t), **column = cause** (at $t - d$).

Identification. The factorization is unique only up to $(X, \Theta) \rightarrow (XT, T^{-1}\Theta)$, under which $G_d \rightarrow T^{-1}G_dT$. The eigenvalues, ρ , Ξ_d and μ_y are therefore invariant, while individual entries of G_d are not. Column normalization removes only the scale part of T , and a separable fitted X does not certify the rest: with $X = I_2$, $\Theta = \begin{pmatrix} 2 & 3 \\ 1 & 1 \end{pmatrix}$ and $T = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$ the basis is perfectly separable and column-normalized, yet $XT \geq 0$, $T^{-1}\Theta \geq 0$, the product is unchanged and T is no permutation. Read the entries of G_d as descriptive and the invariants as inferential; see `nmfkc.ar.latent.inference`.

Oscillation. A complex pair means a cycle, but a *negative real* root also oscillates — it alternates in sign each period, i.e. with period 2. `cycle.period` covers the first case and alternating the second; `non.oscillatory` is TRUE only when neither occurs. Positive real roots rule out oscillation but **not overshoot**: a non-normal G can send a coordinate up before it comes down, which is exactly what the off-diagonal impulse responses of the Canada fit do. Note also that for $Q = 2$, $D = 1$ and $G \geq 0$ the roots are real by construction (the discriminant is $(a - d)^2 + 4bc \geq 0$), so in that setting the absence of a cycle is a property of the design, not a finding.

Value

An object of class "nmfkc.ar.latent": a list with

G	List of D latent transition matrices G_d ($Q \times Q$), named lag1, ..., with dimnames = basis labels (row = effect, column = cause).
G.sum	$\sum_d G_d$.
eigenvalues	The DQ eigenvalues of the latent companion matrix. The $PD \times PD$ companion of the Ξ_d has the same non-zero eigenvalues, so these are all of the model's roots.
spectral.radius	ρ , the largest modulus among them.
spectral.radius.sum	$\rho(\sum_d G_d)$; below 1 iff stationary.
stationary	Logical; TRUE when spectral.radius < 1.
cycle.period	Period $2\pi/ \arg \lambda $ implied by the dominant complex root, or NA when no root is complex.
alternating	TRUE when some root is real and negative, i.e. the decay alternates in sign (period 2).
non.oscillatory	TRUE only when every root is real and positive. Named after the roots, not the trajectory — overshoot is still possible.
theta0	Latent intercept θ (or NULL).
mu.b, mu.y	Long-run means $(I - \sum_d G_d)^{-1}\theta$ and $X\mu_b$; NA when the fit is not stationary or has no intercept.
p.star	Composition of the long-run mean, $\mu_b/(\mathbf{1}'\mu_b)$ — equivalently the fixed point of the <i>residual-free</i> composition dynamics. It is not in general the limit of $\mathbf{b}_t/(\mathbf{1}'\mathbf{b}_t)$ for the stochastic process, nor its expectation: the ratio of expectations is not the expectation of the ratio.
colsum	Column sums c_j of $\sum_d \Xi_d = X \sum_d \Theta_d$, computed as colSums(X) %*% Lambda (length P). Being column sums of a non-negative matrix whose radius is $\rho(\sum_d G_d)$, they bracket it. The unweighted colSums(Lambda) would only be correct when $\mathbf{1}'X = \mathbf{1}'$.
colsum.max	$\max_j c_j$; below 1 it certifies stationarity.
df	Effective dimension $Q(P+m-Q)$ of the fit, where m is the number of columns of Θ ($PD+1$ with an intercept, PD without). These are the degrees of freedom to use when comparing R^2 or information criteria across Q : the naive count $Q(P+m)$ ignores the Q^2 rotational indeterminacy.
separability	Per-basis maximum of the row-normalised X ; a value near 1 means that basis owns a variable loading almost only on it. This is a <i>necessary</i> -looking but not sufficient condition for uniqueness — see the counterexample under Identification.
X, Theta	The basis and the full coefficient matrix the quantities above were computed from, carried along so that nmfkc.ar.latent.inference does not need the fitted object a second time.
X.restriction	The constraint nmfkc placed on the columns of X , which decides how much of T is pinned down.
dims	Named vector of P, Q, D.

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

[plot.nmfkc.ar.latent](#), [nmfkc.ar.latent.inference](#), [nmfkc.ar.stationarity](#), [nmfkc.ar](#), [nmfkc.ar.DOT](#)

Examples

```
d <- AirPassengers
ar_data <- nmfkc.ar(d, degree = 2)
result <- nmfkc(ar_data$Y, ar_data$A, rank = 1)
lat <- nmfkc.ar.latent(result)
lat$G$lag1      # latent transition matrix of lag 1
lat$eigenvalues # all roots of the model
lat             # full report
```

nmfkc.ar.latent.inference

Bootstrap inference for the latent VAR of an NMF-VAR model

Description

This function is **experimental**. The interface may change in future versions: argument names, defaults and the contents of the returned object are not yet stable.

`nmfkc.ar.latent.inference` attaches a wild bootstrap to the quantities reported by `nmfkc.ar.latent`: standard errors and intervals for the spectral radius, the long-run mean, the long-run composition and the entries of the latent transition matrices G_d .

Usage

```
nmfkc.ar.latent.inference(object, Y, A, ...)
```

Arguments

<code>object</code>	An object from <code>nmfkc.ar.latent</code> .
<code>Y, A</code>	The observation and covariate matrices the model was fitted on, i.e. the <code>Y</code> and <code>A</code> returned by <code>nmfkc.ar</code> .
<code>...</code>	Additional arguments:
	<code>wild.B</code> Number of bootstrap replicates. Default 500.
	<code>wild.dist</code> Multiplier distribution, "rademacher" (default) or "exp" (mean-centred).

wild.unit "column" (default, one multiplier per time point) or "element" (one per cell).

wild.level Confidence level. Default 0.95.

wild.seed Seed for the bootstrap. Default 123.

fit The fitted "nmfkc" object. Only needed when object does not carry X / Θ .

truncate Logical; clip Y^* at 0 (default TRUE).

epsilon, maxit Convergence control of the re-fits, default $1e-8$ and 20000 — tight, because a coefficient heading for the non-negativity boundary moves slowly and a loose tolerance biases anything that is compared with zero.

kkt.tol, kkt.ratio.min Thresholds of the strict complementarity diagnostic (defaults $1e-3$ and 5).

sep.tol Row-normalised loading a column of X must reach for that basis to count as anchored. Default 0.999. The zero threshold for the pure-column test on Θ is kkt.tol.

refit.args Named list of extra arguments for the bootstrap re-fits. method and X .restriction are taken from the original fit automatically, so the replicates use the same estimator; anything else the original call set (X .init, penalties, ...) cannot be recovered from the fitted object and must be passed here. The function warns when it sees such arguments in the recorded call.

cores Number of workers for the re-fits, default `getOption("mc.cores", 1L)` (sequential). The bootstrap multipliers are drawn before the loop and each re-fit is deterministic given its data, so the result does not depend on cores.

Details

What is identified, and why that decides what can be inferred. The factorization admits $(X, \Theta) \rightarrow (XT, T^{-1}\Theta)$ for invertible T , so an entry of Θ means nothing across replicates; [nmfkc.inference](#) deals with this by conditioning on the estimated X . This function instead re-estimates the basis in every replicate, propagating its uncertainty. That splits the output in two:

- **Inferential.** ρ , the eigenvalues and the observed long-run mean $\mu_y = X\mu_b$ are invariant under the whole family ($G_d \rightarrow T^{-1}G_dT$ leaves the spectrum alone), so their replicates need no alignment and their intervals mean what they say.
- **Conditional on identification.** The entries $(G_d)_{qq'}$ and the composition p^* need T to be a permutation, and three conditions together deliver that: the column scale fixed (X .restriction = "colSums"), an **anchor row** for every column of X , and a **pure column** for every row of Θ . The anchor rows force $T \geq 0$, the pure columns force $T^{-1} \geq 0$, and a non-negative matrix with a non-negative inverse is monomial. **Either one-sided condition alone is not enough:** with $X = I_2$, $\Theta = \begin{pmatrix} 2 & 3 \\ 1 & 1 \end{pmatrix}$ and $T = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$ the basis is perfectly separable and column-normalized, yet $XT \geq 0$, $T^{-1}\Theta \geq 0$, $X\Theta$ is unchanged, T is no permutation, and G differs between the two representatives — Θ has no pure column there. `identified.approx` reports the verdict and identification the two sides plus how far each is from exact; the function warns either way, naming the side that fails or the margins when it only passes approximately. **The thresholds make this a diagnostic, not a certificate:** the argument needs exact anchors and exact zeros, which multiplicative updates do not deliver. Replicates are permutation-aligned to the original basis regardless. X .restriction = "fixed" is the one clean case for $Q > 1$ — the basis is not estimated, so $T = I$.

prob.nonstationary **is not a unit-root test**. It is the bootstrap tail probability $P^*(\rho^* \geq 1)$, with replicates drawn around the *estimated* model. Nothing imposes $H_0 : \rho = 1$, so it does not have the interpretation of a p-value for that hypothesis; a proper test would have to resample under the null.

The zeros of Θ are only meaningful under misspecification. The strict-complementarity diagnostic reports the dual margin $\delta_{\text{dual}} = \min |\Lambda^*|$ over the zero coefficients and the primal margin $\delta_{\text{prim}} = \min \Theta$ over the positive ones. Both must be bounded away from zero for the support to be recovered exactly and for the active coefficients to attain the distribution that knowing the support would give. If the model is correctly specified, $\Xi_0 = X\Theta$ makes $\Lambda^* = 0$ exactly, so strict complementarity fails at *every* zero: $P(\hat{\theta}_k = 0)$ then tends to a constant in $(0, 1)$ (one half for an isolated zero) instead of to one, and no resampling scheme is consistent there. A small kkt\$ratio is therefore not a numerical defect — it says the zeros are not estimating a sign restriction. What makes them estimate something is misspecification: they recover the coordinates whose *unconstrained* population coefficient is strictly negative.

Bootstrap validity at the boundary. A naive nonparametric bootstrap is inconsistent for a parameter on the boundary. The scheme used here resamples the residuals and re-imposes the non-negativity constraint in every re-fit, which is the practical remedy; a moving-block scheme would in addition preserve the serial dependence that the fixed design discards.

Bootstrap p-values invert the centred distribution. Every replicate of a non-negative coefficient is ≥ 0 , so comparing the raw replicates with zero would return $p = 0$ for every entry. The reported G.p.value is the basic-bootstrap two-sided p obtained from $P^*(\hat{G}^* \geq 2\hat{G})$.

Resampling scheme. With $\hat{Y} = X\Theta A$ and residuals $E = Y - \hat{Y}$, each replicate forms $Y^* = \hat{Y} + E \odot W$ and re-fits nmfkc at the same rank. W carries one multiplier per time point (wild.unit = "column", the default, which preserves the contemporaneous covariance between series) or one per cell. Negative entries of Y^* are truncated at 0 to keep the response non-negative; the truncation rate is reported, and a large one is a warning sign for the whole procedure.

This is a fixed-design bootstrap: the covariate matrix A is held at the observed lags rather than rebuilt from Y^* , so the intervals are conditional on the observed past and do not propagate the randomness of the lagged design itself. They may therefore be somewhat optimistic.

Value

An object of class "nmfkc.ar.latent.inference": a list with

coefficients	Data frame of the DQ^2 entries of the G_d , with columns Lag, Basis (effect at t), Covariate (cause at $t - d$), Estimate, BSE, CI_low, CI_high, p_value. Descriptive unless identified.approx.
G, G.se, G.ci.lower, G.ci.upper, G.p.value	The same quantities as lists of $Q \times Q$ matrices, one per lag.
spectral.radius	The radius of the $DQ \times DQ$ latent companion matrix — the same quantity nmfkc.ar.latent reports, not $\rho(\sum_d G_d)$, which differs when $D > 1$. With spectral.radius.se, spectral.radius.ci.lower / .upper and spectral.radius.boot.mean.
prob.nonstationary	Bootstrap tail probability $P^*(\rho^* \geq 1)$. Not a p-value for $H_0 : \rho \geq 1$ — see Details.
complex, complex.frac	Whether the point estimate has complex roots, and the fraction of replicates that do.

<code>mu.y</code>	With <code>.ci.lower / .ci.upper</code> ; invariant, so inferential.
<code>p.star</code>	With <code>.ci.lower / .ci.upper</code> ; descriptive unless <code>identified.approx</code> .
<code>kkt</code>	Strict-complementarity diagnostic: <code>n.boundary</code> , <code>all.positive</code> , <code>ratio</code> , and the two margins <code>delta.dual</code> and <code>delta.prim</code> .
<code>identified.approx</code>	Logical, an approximate diagnostic : TRUE for $Q = 1$, for <code>X.restriction = "fixed"</code> (where X is not estimated, so $T = I$), or when the per-column scale is fixed (" <code>colSums</code> " / " <code>colSqSums</code> ") and the anchor-row and pure-column tests both pass at <code>sep.tol / kkt.tol</code> . It is not a certificate — the uniqueness argument needs <i>exact</i> anchors and <i>exact</i> zeros, and multiplicative updates leak a little. When FALSE the entry-level summaries and <code>p.star</code> are descriptive; ρ , the eigenvalues and μ_y remain inferential either way.
<code>identification</code>	The two sides separately: anchor (per column of X), pure (per row of Θ), ok, and the margins <code>anchor.margin / pure.margin</code> saying how far the fit is from an exact configuration (0 = exact).
<code>refit.args</code>	The arguments the bootstrap re-fits actually used, so the replicates can be checked against the original estimator.

Plus the bookkeeping entries `wild.B`, `wild.B.requested`, `n.fail`, `truncate.rate`, `wild.unit`, `wild.dist`, `wild.level`, `dims`.

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.ar.latent](#), [nmfkc.ar.stationarity](#), [nmfkc.ar](#), [nmfkc.inference](#)

Examples

```
set.seed(1)
Y <- matrix(abs(rnorm(4 * 40)) + 1, 4, 40)
ar <- nmfkc.ar(Y, degree = 1)
fit <- nmfkc(ar$Y, ar$A, rank = 2, verbose = FALSE)
lat <- nmfkc.ar.latent(fit)
nmfkc.ar.latent.inference(lat, ar$Y, ar$A, wild.B = 50)
```

nmfkc.ar.predict	<i>Forecast future values for NMF-VAR model</i>
------------------	---

Description

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

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

Usage

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

Arguments

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

Value

A list with components:

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

See Also

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

Examples

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

Description

Reports whether a fitted NMF-VAR is stationary, and how cheaply that could be decided. Three numbers are returned, in increasing order of sharpness: `colsum.max` = $\max_j c_j$, the largest column sum of $\sum_d \Xi_d = X \sum_d \Theta_d$, certifies stationarity when below 1 with no eigenvalue computation at all (a sufficient condition only, so a value above 1 is merely inconclusive); `spectral.radius.sum` = $\rho(\sum_d G_d)$ is below 1 exactly when the model is stationary (and is bracketed by $\min_j c_j$ and $\max_j c_j$); and `spectral.radius` is the spectral radius of the companion matrix itself.

Note that the c_j are weighted by $1'X$, i.e. `colSums(X) %*% Lambda`. Using the raw column sums of $\sum_d \Theta_d$ is correct only when $1'X = 1'$ and can certify stationarity for a non-stationary fit otherwise.

This function answers the yes/no question only. For the dynamics behind the number — the transition matrices, the roots, the impulse responses, the long-run mean — use [nmfkc.ar.latent](#), and for the standard error of ρ use [nmfkc.ar.latent.inference](#).

Usage

```
nmfkc.ar.stationarity(x, method = c("latent", "companion"))
```

Arguments

<code>x</code>	A fitted "nmfkc" object obtained from an nmfkc.ar design, or an object from nmfkc.ar.latent .
<code>method</code>	Route used for <code>spectral.radius</code> . "latent" (default) uses the $DQ \times DQ$ companion matrix of the G_d ; "companion" uses the $PD \times PD$ companion matrix of the Ξ_d . The two companion matrices share the same non-zero eigenvalues, so the results agree up to floating-point rounding, but "latent" costs $O((DQ)^3)$ instead of $O((PD)^3)$ — for a 47-variable, 7-lag, rank-4 model that is a 28×28 eigenproblem instead of a 329×329 one. Use "companion" to reproduce the value bit-for-bit as computed by earlier versions of the package.

Value

An object of class "nmfkc.ar.stationarity": a list with

<code>spectral.radius</code>	Spectral radius of the companion matrix (route selected by method). A value below 1 indicates stationarity.
<code>stationary</code>	Logical; TRUE when <code>spectral.radius</code> < 1.
<code>method</code>	The route actually used.
<code>spectral.radius.sum</code>	$\rho(\sum_d G_d)$; below 1 iff stationary.
<code>colsum</code>	Column sums c_j of $\sum_d \Xi_d$, i.e. <code>colSums(X) %*% Lambda</code> (length P).
<code>colsum.max</code>	$\max_j c_j$; below 1 it certifies stationarity.
<code>dims</code>	Named vector of P, Q, D .

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.ar.latent](#), [nmfkc.ar.latent.inference](#), [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)
st <- nmfkc.ar.stationarity(result)
st$spectral.radius
st$stationary
st # full report
```

nmfkc.ard

Automatic relevance determination for NMF rank (experimental)

Description

Prototype of Tan & Fevotte’s (2013) ARD-NMF (Euclidean / $\beta = 2$). Unlike the cross-validation ([nmfkc.ecv](#), [nmfkc.bicv](#)) and stability ([nmfkc.consensus](#)) engines that *scan* a range of ranks, ARD fits NMF **once** at an over-complete rank and prunes automatically: each component k carries a relevance weight λ_k with an inverse-gamma prior; the multiplicative updates gain a penalty $+w_{fk}/\lambda_k$ (L2 / half-normal prior) or $+1/\lambda_k$ (L1 / exponential), which drives unsupported components to zero. The number of surviving components is the estimated rank. Covariates are ignored (plain NMF).

This is a model-based point estimate: the result depends on the prior, the starting rank and the random initialization, and can vary run to run. Use it as a complement to the CV / consensus engines, not a sole criterion.

Usage

```
nmfkc.ard(Y, rank = NULL, nrun = 10, plot = FALSE, ...)
```

Arguments

Y	Observation matrix ($F \times N$), non-negative.
rank	Over-complete starting rank K (must exceed the true rank). NULL (default) uses $\min(F, N, 20)$.

nmfkc.bicv

Bi-cross-validation for NMF rank selection

Description

Owen & Perry's (2009) bi-cross-validation (BCV) for choosing the NMF rank. A lightweight CV engine in the spirit of `nmfkc.ecv`: it returns the held-out error per rank and nothing more (no plot, no table) – pass the result to `which.min(sigma)`, or build your own diagnostics.

Unlike the element-wise CV of `nmfkc.ecv` (which holds out scattered *entries* and refits with weights), BCV holds out a whole **row-block and column-block** simultaneously: the model is fit only on the retained block D , and the held-out block A is predicted by folding the held-out rows/columns onto the fixed D -factors via non-negative regression ($\hat{A} = L_I R_J$). Because the held-out rows and columns never enter the fit, there is no information leakage. Covariates are ignored (plain NMF). The recommended setting is to leave out roughly half the rows and half the columns (`nfolds = 2`).

Usage

```
nmfkc.bicv(Y, rank = 1:3, ...)
```

Arguments

<code>Y</code>	Observation matrix ($P \times N$), non-negative.
<code>rank</code>	Integer vector of ranks to evaluate.
<code>...</code>	Advanced options, rarely needed (defaults in parentheses): <code>nfolds</code> (2), the number of row and column folds (the grid is <code>nfolds</code> x <code>nfolds</code> ; 2 leaves out half the rows / columns, Owen & Perry's recommendation); <code>seed</code> (123, fold-assignment seed); and <code>nnls.maxit</code> (100, multiplicative-update iterations for the fold-in non-negative regressions). Any other arguments are passed to <code>nmfkc</code> for the per-block fits (e.g. <code>\maxit</code>). <code>cores</code> (<code>getOption("mc.cores", 1L)</code>) evaluates the ranks in parallel; results are identical to the sequential loop for any cores (PSOCK cluster on Windows, forking elsewhere).

Details

Each fold keeps about $(1 - 1/\text{nfolds})$ of the rows and columns, so the retained block D must have more than `rank` rows **and** columns. The largest testable rank is therefore about $(1 - 1/\text{nfolds}) \min(P, N) - 1$; with `nfolds = 2` this is roughly $\min(P, N)/2 - 1$. Ranks above this return NA and trigger a [warning](#) that names the limit and the `nfolds` (or `nmfkc.ecv`) that would reach the requested ranks. Raising `nfolds` lifts the limit at the cost of a smaller hold-out and more compute ($(\text{nfolds} - 1)^2$ full fits per rank).

Value

A list (cf. `nmfkc.ecv`) with:

<code>objfunc</code>	Held-out mean squared error for each rank.
<code>sigma</code>	Its square root (RMSE) for each rank.

rank	The evaluated rank vector.
nfolds	The number of folds used.

References

A. B. Owen and P. O. Perry (2009). Bi-cross-validation of the SVD and the nonnegative matrix factorization. *Ann. Appl. Stat.* 3(2):564–594. doi:[10.1214/08AOAS227](https://doi.org/10.1214/08AOAS227).

See Also

[nmfkc.rank](#), [nmfkc.ecv](#), [nmfkc.consensus](#), [nmfkc.ard](#)

Examples

```
## rank-3 non-negative data; bi-CV needs enough kept rows/cols per
## fold (> rank), so use a matrix with ample dimensions.
set.seed(1)
X <- matrix(abs(rnorm(30 * 3)), 30, 3)
B <- matrix(abs(rnorm(3 * 40)), 3, 40)
bv <- nmfkc.bicv(X %*% B, rank = 1:6) # nfolds = 2 (Owen & Perry) by default
bv$sigma # held-out RMSE per rank
bv$rank[which.min(bv$sigma)]
```

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.consensus

Consensus-clustering rank selection for NMF (Brunet 2004)

Description

The bioinformatics-standard stability approach to choosing the NMF rank. A lightweight engine like [nmfkc.ecv](#) / [nmfkc.bicv](#): it returns one stability score per rank and nothing more.

For each rank, NMF is run `nrun` times from different random initializations (`X.init = "runif"`). Each run gives a hard clustering of the samples (the column `arg max` of the coefficient matrix). Averaging the $N \times N$ connectivity matrices (1 if two samples share a cluster) over the runs yields the **consensus matrix**; its crispness measures how reproducible the clustering is. Two summaries are reported per rank:

- **cophenetic**: the cophenetic correlation coefficient (CPCC) of the consensus matrix (Brunet et al. 2004). Close to 1 = stable.
- **dispersion**: the Kim & Park (2007) dispersion $\frac{1}{N^2} \sum_{ij} 4(C_{ij} - 1/2)^2 \in [0, 1]$; 1 when every consensus entry is exactly 0 or 1 (perfectly crisp).
- **pac**: the Proportion of Ambiguous Clustering (Senbabaoglu et al. 2014) – the fraction of off-diagonal consensus entries falling in the ambiguous interval `pac.range` (default (0.1, 0.9)). **Lower is better** (less ambiguity); a more sensitive readout than **cophenetic**, which tends to saturate.

Unlike the cross-validation engines (where the rank *minimizes* the error), here a good rank **maximizes** stability, or is the largest rank before it drops.

Usage

```
nmfkc.consensus(
  Y,
  A = NULL,
  rank = 2:4,
  nrun = 30,
  keep.consensus = FALSE,
  ...
)
```

Arguments

<code>Y</code>	Observation matrix ($P \times N$), non-negative.
<code>A</code>	Optional covariate matrix passed to nmfkc .
<code>rank</code>	Integer vector of ranks to evaluate (≥ 2).

nrun	Number of random-initialization runs per rank (default 30).
keep.consensus	Logical; if TRUE also return the list of consensus matrices (one $N \times N$ matrix per rank).
...	Advanced options, rarely needed (defaults in parentheses): seed (123, base seed; run r of rank index i uses $\text{seed} + 1000 * i + r$) and pac.range (c(0.1, 0.9), the ambiguous interval (u_1, u_2) for the PAC measure). Any other arguments are passed to <code>nmfkc</code> (e.g. <code>\maxit</code>); <code>X.init</code> is forced to "runif". cores (getOption("mc.cores", 1L)) runs the nrun restarts of each rank in parallel; results are identical to the sequential loop for any cores (PSOCK cluster on Windows, forking elsewhere).

Value

An object of class "nmfkc.consensus" (a list) with:

cophenetic	Cophenetic correlation coefficient for each rank.
dispersion	Dispersion coefficient $([0, 1])$ for each rank.
pac	Proportion of Ambiguous Clustering $([0, 1])$, lower is better) for each rank.
rank	The evaluated rank vector.
nrun	Number of runs per rank.
consensus	List of consensus matrices, or NULL.

It has `print.nmfkc.consensus` and `plot.nmfkc.consensus` (type = "criteria" / "heatmap") methods.

References

Brunet, J.-P., Tamayo, P., Golub, T. R., Mesirov, J. P. (2004). Metagenes and molecular pattern discovery using matrix factorization. *PNAS* 101(12):4164–4169. doi:[10.1073/pnas.0308531101](https://doi.org/10.1073/pnas.0308531101). Kim, H., Park, H. (2007). Sparse non-negative matrix factorizations ... *Bioinformatics* 23(12):1495–1502. Senbabaoglu, Y., Michailidis, G., Li, J. Z. (2014). Critical limitations of consensus clustering in class discovery. *Sci. Rep.* 4:6207. doi:[10.1038/srep06207](https://doi.org/10.1038/srep06207).

See Also

[nmfkc.rank](#), [nmfkc.ecv](#), [nmfkc.bicv](#), [nmfkc.ard](#), [nmf.cluster.criteria](#)

Examples

```
Y <- t(as.matrix(iris[, 1:4]))
cs <- nmfkc.consensus(Y, rank = 2:5, nrun = 20, keep.consensus = TRUE)
cs                                # stability table per rank
plot(cs)                         # type = "criteria": stability curves
plot(cs, type = "heatmap")       # all ranks, n2mfrow grid
plot(cs, type = "heatmap", rank = 3) # one rank, with labels
```

nmfkc.criterion

*Compute model selection criteria for a fitted nmfkc model***Description**

nmfkc.criterion computes the effective rank (raw and as the $[0, 1]$ broken-stick index) and the sample-clustering quality measures (silhouette, CPCC, dist.cor) 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 nmfkc .
<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" skips the fit and clustering statistics.
<code>...</code>	Additional arguments: <code>Y.weights</code> (non-negative weight matrix; <code>lm()</code> -style loss $\sum W r^2$; default: all ones). See nmfkc for full details.

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: effective.rank, effective.rank.index, 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

Y	Observation matrix, or a formula (see nmfkc for Formula Mode).
A	Covariate matrix. If NULL, the identity matrix is used. Ignored when Y is a formula.
rank	Rank of the basis matrix X . Default is 2.
data	A data frame (required when Y is a formula with column names).
...	Additional arguments controlling CV and the internal nmfkc call:
Y.weights	Non-negative weight matrix or vector (lm()-style: $\text{loss} \sum W r^2$). Binary $\{0, 1\}$ masks (TRUE / FALSE also accepted) are the typical ECV usage – 0/FALSE excludes an element. See nmfkc for full details.
div	Number of folds (k); default: 5.
seed	Integer seed for reproducible partitioning; default: 123.
shuffle	Logical. If TRUE (default), randomly shuffles samples (standard CV); if FALSE, splits sequentially (block CV; recommended for time series).
Q	(Deprecated) Alias for rank.
Arguments passed to nmfkc e.g., C.L1, epsilon, maxit, method ("EU" or "KL"), X.restriction, X.init, etc.	

Value

A list with components:

`objfunc` Mean loss per valid entry over all folds (MSE for `method="EU"`).

`sigma` Residual standard error (RMSE). Available only if `method="EU"`; on the same scale as `Y`.

`objfunc.block` Loss for each fold.

`block` Vector of fold indices (1, ..., `div`) assigned to each column of `Y`.

See Also

[nmfkc](#), [nmfkc.kernel.beta.cv](#), [nmfkc.ar.degree.cv](#)

Examples

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

Print/plot methods for nmfkc cross-validation objects

Description

Compact print and (for rank sweeps) a score-vs-rank plot with a “Best (Min)” marker, for the objects returned by [nmfkc.ecv](#), [nmfkc.cv](#) and [nmfkc.bicv](#).

Usage

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

## S3 method for class 'nmfkc.ecv'
plot(x, main = "nmfkc element-wise CV", ylab = NULL, ...)
```

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

## S3 method for class 'nmfkc.bicv'
plot(
  x,
  main = "nmfkc bi-cross-validation",
  ylab = "sigma (held-out RMSE)",
  ...
)

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

## S3 method for class 'nmfkc.cv'
plot(
  x,
  main = "nmfkc CV: per-fold held-out loss",
  ylab = "objfunc (block)",
  ...
)
```

Arguments

x	A "nmfkc.ecv" / "nmfkc.cv" / "nmfkc.bicv" object.
...	Passed to plot .
main, ylab	Plot labels.

Value

x, invisibly.

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,
  C.signed = NULL,
  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. When C.signed = TRUE (signed Θ) this is an absolute-value cut, i.e. an edge is drawn when $ coef \geq$ threshold.
C.signed	Logical or NULL. Whether the coefficient matrix Θ (= C) may be signed (real-valued). When TRUE, the threshold is applied to $ coef $, edge widths are scaled by $ coef $, and negative edges are drawn as black dashed lines (positive edges remain solid) to distinguish them; the numeric edge labels keep their sign. When FALSE, the historical non-negative behaviour is used (negative coefficients fall below the threshold and are hidden). When NULL (default), the mode is auto-detected from the fit (result\$C.signed) or from the presence of negative entries in C / XC . The basis X is always non-negative, so $X \rightarrow Y$ edges are unaffected.
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.

<code>Y.title</code>	Cluster title for Y nodes.
<code>X.title</code>	Cluster title for X nodes.
<code>A.title</code>	Cluster title for A nodes.
<code>hide.isolated</code>	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](#), [nmf.rrr.DOT](#), [nmf.frb.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.

For symmetric (network) data use [nmfkc.net.ecv](#), which creates upper-triangle folds to prevent information leakage through the symmetric entries $Y_{ij} = Y_{ji}$. Passing the old `Y.symmetric` argument here is no longer supported and stops with a redirect message.

Usage

```
nmfkc.ecv(Y, A = NULL, rank = 1:3, data, ...)
```

Arguments

Y	Observation matrix, or a formula (see nmfkc for Formula Mode).
A	Covariate matrix. Ignored when Y is a formula.
rank	Vector of ranks to evaluate (e.g., 1:5). For backward compatibility, Q is accepted via . . .
data	A data frame (required when Y is a formula with column names).
. . .	Additional arguments passed to nmfkc (e.g., method="EU"). Also accepts: nfolds (number of folds, default 5; div also accepted), seed (integer seed, default 123), and cores (evaluate the rank x fold task grid in parallel; default <code>getOption("mc.cores", 1L)</code>). Parallelism uses a PSOCK cluster on Windows and forking elsewhere; because each task is a deterministic self-seeded fit and results are returned in order, objfunc/sigma are identical for any cores.

Value

A list with components:

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

References

Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4), 397–405. doi:10.1080/00401706.1978.10489693

Owen, A. B., & Perry, P. O. (2009). Bi-cross-validation of the SVD and the nonnegative matrix factorization. *Ann. Appl. Stat.* 3(2), 564–594. doi:10.1214/08AOAS227 (cross-validation of the NMF rank; see also [nmfkc.bicv](#)).

See Also

[nmfkc](#), [nmfkc.cv](#); other rank-selection criteria: [nmfkc.rank](#), [nmfkc.bicv](#), [nmfkc.consensus](#), [nmfkc.ard](#).

Examples

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

nmfkc.inference

*Statistical inference for the parameter matrix C (Θ)***Description**

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

Under the working model $Y = XCA + \varepsilon$ where $\varepsilon_{pn} \stackrel{iid}{\sim} N(0, \sigma^2)$, 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:
 - method Bootstrap engine. "onestep" (default): a one-step Newton linearisation around the fit using the inverse information Hinv (fast; the sandwich SE is primary). "refit": a residual wild bootstrap that re-estimates C to convergence with the basis X held FIXED. The "refit" engine uses no information matrix, so it stays valid when the Fisher information AA' is singular (over-parameterised covariates) or C sits on the ≥ 0 boundary; the bootstrap SE/CI become primary and the p-value is a two-sided bootstrap p-value.
 - wild.dist Multiplier distribution (orthogonal to method): "rademacher" (± 1), "mammen" (Mammen 1993 two-point), or "exp" ($\text{Exp}(1) - 1$). Default "exp" for method="onestep" (backward compatible), "rademacher" for "refit".
 - wild.unit For method="refit": "element" (default, i.i.d. multiplier per matrix cell) or "column" (one multiplier per sample column, shared over rows).
 - refit.epsilon, refit.maxit For method="refit": convergence tolerance and iteration cap of the re-fits, defaulting to $1e-8$ and 100000 . These are deliberately far tighter than nmfkc's own $1e-4$: under multiplicative updates a coefficient heading for the non-negativity boundary approaches 0 slowly, so at a loose tolerance every replicate stops while that coefficient

is still spuriously positive. The replicates then pile up above the point estimate, the percentile interval can exclude it entirely, and the bootstrap SE is understated. Loosen only if the re-fits are too slow, and check that the intervals still contain the estimates.

`wild.B` Number of bootstrap replicates. Default is 500.
`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 σ^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 one row per element of C . SE is always the sandwich standard error and BSE always the bootstrap one, so the two columns differ; <code>z_value</code> and <code>p_value</code> use whichever is primary for the chosen method (bootstrap under "refit", sandwich under "onestep"). <code>on.boundary</code> flags coefficients at or below <code>boundary.tol</code> (default 1e-3): their interval and p-value behave correctly in the "do not reject" direction, but the value is not a calibrated p-value — a one-sided bootstrap p piles up near 1/2 at the boundary.
<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 nmfkc.cv. Also accepts cores (evaluate the beta candidates in parallel; default getOption("mc.cores", 1L); PSOCK cluster on Windows, forking elsewhere). Each candidate is a deterministic nmfkc.cv call (its folds are seeded) and results are returned in order, so objfunc and the selected beta are identical for any cores.

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(Y))
lines(as.vector(V),as.vector(result$XB),col=2,lwd=2)
```

nmfkc.kernel.beta.nearest.med

Estimate Gaussian/RBF kernel parameter beta from covariates (supports landmarks)

Description

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

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

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

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

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

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

Usage

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

Arguments

<code>U</code>	A numeric matrix of covariates ($K \times N$); columns are samples.
<code>Uk</code>	An optional numeric matrix of landmarks ($K \times M$); 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 $N \leq 1000$, it is automatically set to N .
<code>block.size.Uk</code>	Integer. Number of columns of <code>Uk</code> processed per block when <code>Uk</code> is not <code>NULL</code> (controls memory usage). If $M \leq 2000$, it is automatically set to M .
<code>sample.size</code>	Integer or <code>NULL</code> . If not <code>NULL</code> , randomly subsamples this many columns of <code>U</code> (without replacement) before computing distances, to reduce computational cost.
<code>...</code>	Additional arguments. Hidden option <code>candidates</code> controls the candidate grid: one of "7points" (default), "4points", or a numeric vector of t values. See Details.

Details

Candidate grid: Along with `beta`, the function returns `beta_candidates`, a logarithmic grid suitable for cross-validation. The grid is symmetric on the bandwidth scale σ around σ_0 :

$$\sigma = \sigma_0 \times 10^t,$$

and since $\beta = 1/(2\sigma^2)$, this corresponds to $\beta = \beta_0 \times 10^{-2t}$.

The grid of t values can be customized through the hidden argument `candidates` (passed via `...`):

- "7points" (default): $t \in \{-1, -2/3, -1/3, 0, 1/3, 2/3, 1\}$ (7 candidates spanning one decade, matches the grid used in the RFF-NMF research memo).
- "4points": $t \in \{-1/2, 0, 1/2, 1\}$ yielding $\beta_0 \times 10^{(1,0,-1,-2)}$ (the legacy short grid).
- A numeric vector: user-specified t values. The grid returned is $\beta_0 \times 10^{-2t}$.

Prior to version 0.6.8, the grid depended on whether `Uk` was supplied (4 candidates for `Uk = NULL`, 7 for supplied `Uk`). The current implementation unifies both branches via `candidates`.

Notes:

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

Value

A list with elements:

- `beta`: Estimated kernel parameter $\beta_0 = 1/(2\sigma_0^2)$.
- `beta_candidates`: Numeric vector of candidate β values (logarithmic grid) intended for cross-validation.
- `dist_median`: The estimated distance scale σ_0 (median of nearest-neighbor or nearest-landmark distances).

- `block.size.used`: The effective block size(s) used. Either a scalar (no `Uk`) or a named vector `c(U=..., Uk=...)` when `Uk` is provided.
- `sample.size.used`: The number of columns of `U` actually used (after subsampling).
- `uk_is_u`: Logical flag indicating whether `Uk` was detected as identical to `U` (only returned when `Uk` is provided).

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.net

Symmetric NMF for networks (tri / bi / signed)

Description

Single entry point for symmetric NMF of network data with correct multiplicative updates. Three model types are supported via type:

- **tri** (type="tri", default): $Y \approx X C X^\top$ with $X, C \geq 0$ (both non-negative; C symmetric by design). Uses Frobenius-full bilateral gradient.
- **bi** (type="bi"): $Y \approx X X^\top$ (C fixed to I_Q), cube-root damping (He et al. 2011).
- **signed** (type="signed"): $Y \approx X(C_+ - C_-)X^\top$ with $X \geq 0$ and signed $C = C_+ - C_-$. Preserves the soft-clustering interpretation of X while allowing negative off-diagonals of C (inter-cluster *repulsion*).

Non-negative adjacency matrix assumption. All three types assume $Y \geq 0$ (a non-negative adjacency/affinity matrix). The qualifier “signed” in type = “signed” refers to the middle coefficient C , not to Y itself. The underlying Ding, Li & Jordan (2010) sign-splitting updates require $Y \geq 0$ to guarantee monotone descent; supplying a signed Y triggers an error. For a signed data matrix, see [nmfkc.signed](#).

Usage

```
nmfkc.net(
  Y,
  rank = 2,
  type = c("tri", "bi", "signed"),
  epsilon = 1e-04,
  maxit = 5000,
  verbose = FALSE,
  ...
)
```

Arguments

<code>Y</code>	Symmetric (N x N) non-negative adjacency matrix. NA entries are automatically treated as masked edges (equivalent to supplying <code>Y.weights</code> with 0 at those positions); see the note on <code>Y.weights</code> below.
<code>rank</code>	Integer Q .
<code>type</code>	"tri" (default), "bi", or "signed".
<code>epsilon, maxit, verbose</code>	Standard. Note: the convergence test uses the <i>unpenalized</i> squared error, not the penalized objective the updates minimize, so with a large <code>C.L1</code> or <code>X.L2.ortho</code> the iteration can stop while the optimized quantity is still moving. The other fitters test the penalized value; this is kept as-is so existing fits reproduce.

...

Hidden options: `nstart` (default 1; see note below), `seed` (default 123), `X.restriction`, `X.init`, `C.init` (tri only) or `Cp.init/Cn.init` (signed only), `Y.weights`, `C.L1` (tri only), `X.L2.ortho`, `prefix`.

`Y.weights` is an optional non-negative $N \times N$ weight matrix (symmetric, same shape as `Y`). When supplied, the loss becomes $\sum W_{ij} (Y_{ij} - \hat{Y}_{ij})^2$ (**lm**-style, **linear** in W). Logical matrices (TRUE / FALSE) are also accepted. Typical usage by nmfkc.net is a binary mask ($W \in \{0, 1\}$) holding out test edges on the upper triangle; real-valued weights for edge-level importance weighting are also supported. If `Y.weights` is NULL (default) and `Y` contains NA, a binary mask is auto-generated (0 at NA positions, 1 elsewhere), and the NA entries in `Y` are replaced by 0 so the multiplicative updates can proceed.

`X.init` controls the initialization of the $N \times Q$ basis matrix X . Accepted values:

- "kmeans" (default): k-means on the rows of Y (equivalently columns, since Y is symmetric); the Q cluster centers become the columns of X . Each node is treated as an N -dimensional connectivity profile, so clusters correspond to nodes with similar neighborhood structure – essentially a fast proxy for spectral clustering (Kuang, Yun & Park 2015, *SymNMF*). Scales well and is the recommended default for network data.
- "kmeansar": "kmeans" followed by filling zero entries of X with $\text{Uniform}(0, \bar{Y}/100)$ to escape trivial stationary points.
- "nndsvd": Non-negative Double SVD with additive randomness (NNDSVDar). Requires a full SVD of Y , so for very large networks ($N > \text{a few thousand}$) "kmeans" is preferable.
- "runif": Uniform random entries in $[0, 1]$.
- "random": Legacy default (pre-v0.6.8), equivalent to $\text{abs}(\text{rnorm}(N * Q)) * 0.1$. Kept for backward compatibility.
- A numeric $N \times Q$ matrix supplied by the user (used as-is).

When `nstart` > 1, each restart uses a distinct seed so that k-means / runif / NNDSVDar produce different candidate initial values across the multi-start loop.

Multi-start recommendation. For `type = "signed"` the $C = C_+ - C_-$ bottleneck can take both positive and negative values, so the objective has more local minima than for "tri" or "bi". A larger `nstart` (e.g., 10-50) is recommended during exploration to reduce the chance of being trapped at a suboptimal stationary point. The default 1 is intended for fast development; raise for publication-grade runs.

Value

Object of class `c("nmfkc.net.<type>", "nmfkc.net", "nmfkc")`. For `type = "signed"` the return also carries `$Cp`, `$Cn`.

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

[nmfkc.net.ecv](#), [nmfkc.net.DOT](#)

nmfkc.net.DOT

Generate a Graphviz DOT Diagram for a Symmetric NMF Network

Description

Creates a Graphviz DOT script that visualizes the two-layer structure of a symmetric NMF model ($Y \approx X C X^T$).

The resulting diagram displays:

- outer nodes: the original network nodes (rows/columns of Y),
- inner nodes: the latent basis/group nodes (columns of X),
- directed edges from basis to node: membership weights from X,
- undirected edges between basis nodes: inter-group interactions from C matrix (for tri-symmetric models only).

Usage

```
nmfkc.net.DOT(
  result,
  threshold = 0.01,
  sig.level = 0.1,
  weight_scale = 5,
  weight_scale_xy = 1,
  weight_scale_xx = weight_scale,
  rankdir = "TB",
  fill = TRUE,
  hide.isolated = TRUE,
  Y.label = NULL,
  X.label = NULL,
  Y.title = "Nodes (Y)",
  X.title = "Basis (X)",
  show.theta = NULL,
  signed = inherits(result, "nmfkc.net.signed"),
  cluster.box = c("none", "normal", "faint", "invisible"),
  layout = c("neato", "fdp", "twopi", "circo", "dot"),
  X.color = NULL,
  Y.cluster = c("soft", "hard")
)
```

Arguments

result	A list returned by <code>nmfkc()</code> with <code>Y.symmetric = "bi" or "tri"</code> , or the newer <code>nmfkc.net</code> with <code>type = "bi" or "tri"</code> . If inference results are present (from <code>nmfkc.net.inference</code>), C edges are decorated with significance stars.
threshold	Minimum coefficient value to display an edge. It is read on the membership scale for X edges (<code>X.prob</code>, rows summing to one) and on <i>C</i> for the basis-to-basis edges, and it does two jobs at once, switching over at $1/Q$: • At or below $1/Q$ it only thins edges. No node can be hidden, because a row of <code>X.prob</code> sums to one and so has a maximum of at least $1/Q$; every node genuinely keeps an edge. • Above $1/Q$ it may also start hiding nodes (when <code>hide.isolated = TRUE</code>). Not immediately: nodes go only once the threshold passes the smallest row maximum of <code>X.prob</code>, which is data-dependent and can be well above $1/Q$. Inspect <code>apply(fit\$X.prob, 1, max)</code> to see where that is. The default <code>0.01</code> therefore draws every node — guaranteed for any $Q \leq 100$, since only then is $0.01 \leq 1/Q$ — and most edges. For a readable community graph, <code>0.2–0.4</code> usually thins the edges enough while keeping all nodes.
sig.level	Significance level for filtering C edges (if inference results are present). Set to <code>NULL</code> to show all edges above threshold.
weight_scale	Base scaling factor for edge widths.
weight_scale_xy	Scaling factor for X edges (basis -> node).
weight_scale_xx	Scaling factor for C edges (basis <-> basis).
rankdir	Graphviz rank direction (" <code>TB</code> " or " <code>LR</code> ").
fill	Logical; whether nodes are filled with color.
hide.isolated	Logical; if <code>TRUE</code> , omit outer nodes with no X edge above threshold. The test is on the membership scale (<code>X.prob</code>), the same quantity the X edges are drawn from, so it does not shift with <code>type</code> or <code>X.restriction</code> . See <code>threshold</code> for when this actually removes anything — at the default it never does for any $Q \leq 100$, which is correct rather than a limitation.
Y.label	Character vector of labels for outer nodes.
X.label	Character vector of labels for basis nodes.
Y.title	Cluster title for outer nodes.
X.title	Cluster title for basis nodes.
show.theta	Logical or <code>NULL</code> . Whether to draw C edges between basis nodes. <code>NULL</code> = auto-detect (<code>TRUE</code> for <code>tri</code> , <code>FALSE</code> for <code>bi</code>).
signed	Logical. If <code>TRUE</code> , <i>C</i> is treated as a signed matrix: positive entries rendered as solid edges and negative as dashed, with edge visibility threshold on $ C $. Default is <code>inherits(result, "nmfkc.net.signed")</code> so that results from <code>nmfkc.net</code> are auto-detected.
cluster.box	Style of cluster box: " <code>none</code> ", " <code>normal</code> ", " <code>faint</code> ", " <code>invisible</code> ".

layout	Graphviz layout engine, in recommended order: "neato" (default; spring model, clearest for small/medium community graphs), "fdp" (force-directed, scales to larger graphs), "twopi" (radial), "circo" (circular), "dot" (hierarchical). For community networks, "neato" or "fdp" with a raised threshold (e.g.\0.2–0.3) separate the groups best.
X.color	Color palette for basis nodes (length Q).
Y.cluster	Coloring mode for outer nodes: "soft" (weighted mix) or "hard" (most probable basis color).

Value

A character string of class `c("nmfkc.net.DOT", "nmfkc.DOT")` representing a valid Graphviz DOT script. Use `plot()` to render (requires DiagrammeR).

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

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

Examples

```
library(nmfkc)
Y <- matrix(c(0,1,1,0,0,0,
              1,0,1,0,0,0,
              1,1,0,1,0,0,
              0,0,1,0,1,1,
              0,0,0,1,0,1,
              0,0,0,1,1,0), 6, 6)
res <- nmfkc.net(Y, rank = 2, type = "tri", nstart = 20)
dot <- nmfkc.net.DOT(res)
plot(dot)
```

nmfkc.net.ecv

Element-wise cross-validation for nmfkc.net (upper-triangle folds)

Description

k-fold CV with folds taken over the upper triangle of the symmetric Y (mirrored to the lower triangle) to prevent information leakage through symmetry. A single entry point covers all three symmetric NMF variants; type selects the fitting function:

- "tri" (default): [nmfkc.net](#) with $C \geq 0$
- "bi": [nmfkc.net](#) with $C = I_Q$
- "signed": [nmfkc.net](#) with signed $C = C_+ - C_-$

Usage

```
nmfkc.net.ecv(Y, rank = 1:3, type = c("tri", "bi", "signed"), ...)
```

Arguments

Y	Symmetric N x N non-negative matrix.
rank	Integer vector of ranks to evaluate. Default 1:3.
type	Model type: "tri" (default), "bi", or "signed".
...	Passed to the underlying fitter; also accepts nfolds (default 5; div alias), seed (default 123), and cores (getOption("mc.cores", 1L)) to evaluate the rank x fold grid in parallel; results are identical to the sequential run for any cores (PSOCK cluster on Windows, forking elsewhere).

Value

A list with objfunc, sigma, objfunc.fold, folds, Q.grid, type.

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

[nmfkc.net](#)

nmfkc.net.inference *Statistical Inference for Symmetric NMF Parameters*

Description

Performs statistical inference on the parameter matrix C of a symmetric NMF model ($Y \approx X C X^T$). This is a wrapper around [nmfkc.inference](#) that automatically sets the covariate matrix to $A = X^T$, which is the defining property of symmetric NMF.

Usage

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

Arguments

object	A fitted model returned by nmfkc() with Y.symmetric = "bi" or "tri", or the newer nmfkc.net with type = "bi" or "tri".
Y	The original symmetric matrix (P x P).
wild.bootstrap	Logical; if TRUE (default), perform wild bootstrap inference for the C matrix.
...	Additional arguments passed to nmfkc.inference() (e.g., wild.B, wild.seed, C.p.side).

Value

The object augmented with inference fields (same as `nmfkc.inference`), with class `c("nmfkc.net.inference", "nmfkc.inference", "nmf.inference", "nmfkc", "nmf")`.

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

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

Examples

```
library(nmfkc)
Y <- matrix(c(0,1,1,0,0,0,
              1,0,1,0,0,0,
              1,1,0,1,0,0,
              0,0,1,0,1,1,
              0,0,0,1,0,1,
              0,0,0,1,1,0), 6, 6)
res <- nmfkc.net(Y, rank = 2, type = "tri", nstart = 20)
res_inf <- nmfkc.net.inference(res, Y)
summary(res_inf)
```

nmfkc.net.rank

Rank selection for nmfkc.net (concise diagnostics)

Description

Fits [nmfkc.net](#) across a range of ranks and reports the three rank-selection criteria – `r.squared`, the effective rank (utilization), and the element-wise CV error `sigma.ecv` – with the same concise diagnostics plot as [nmfkc.rank](#).

Usage

```
nmfkc.net.rank(
  Y,
  rank = 1:5,
  type = c("tri", "bi", "signed"),
  detail = c("full", "fast"),
  plot = TRUE,
  ...
)
```

Arguments

<code>Y</code>	Symmetric (network) observation matrix.
<code>rank</code>	Integer vector of ranks to evaluate.
<code>type</code>	One of "tri", "bi", "signed".
<code>detail</code>	"full" (default) also runs element-wise CV (<code>sigma.ecv</code>); "fast" skips it (plots <code>r.squared</code> and <code>eff.rank</code> only, and recommends the R-squared elbow).
<code>plot</code>	Logical; draw the diagnostics plot (default TRUE).
<code>...</code>	Passed on to <code>nmfkc.net</code> and <code>nmfkc.net.ecv</code> (e.g. <code>nstart</code> , <code>maxit</code> , <code>nfolds</code> , <code>seed</code>). Also accepts <code>cores</code> (<code>getOption("mc.cores", 1L)</code>) to fit the per-rank models (and the ECV grid) in parallel; results are identical for any cores (PSOCK cluster on Windows, forking elsewhere).

Value

A list with `rank.best` (ECV minimum, or the R-squared elbow under `detail = "fast"`) and `criteria` (data frame: `rank`, `effective.rank`, `effective.rank.ratio`, `r.squared`, `sigma.ecv`).

References

Roy, O., & Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. *Proc. EUSIPCO*, 606–610. (`effective.rank`) Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4), 397–405. (`sigma.ecv`)

See Also

[nmfkc.net](#), [nmfkc.net.ecv](#), [nmfkc.rank](#)

Examples

```
Y <- matrix(c(0,1,1,0,0,0, 1,0,1,0,0,0, 1,1,0,1,0,0,
              0,0,1,0,1,1, 0,0,0,1,0,1, 0,0,0,1,1,0), 6, 6)
nmfkc.net.rank(Y, rank = 1:3, type = "tri", nstart = 5, nfolds = 3)
```

<code>nmfkc.normalize</code>	<i>Normalize a matrix to the range $[0, 1]$</i>
------------------------------	--

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

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

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. Three rank-selection measures are computed (R-squared, the effective rank, and the element-wise CV error), and results can be visualized in a plot. Sample-clustering quality (silhouette / CPCC / dist.cor) is no longer part of rank selection; use [nmf.cluster.criteria](#) on a fitted model for those.

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 nmfkc for Formula Mode).
A	Covariate matrix. If NULL, the identity matrix is used. Ignored when Y is a formula.
rank	A vector of candidate ranks to be evaluated.
detail	"full" (default) also runs the element-wise CV (sigma.ecv); "fast" skips it (the plot then shows only <code>r.squared</code> and <code>eff.rank</code> , and the recommended rank falls back to the R-squared elbow).
plot	Logical. If TRUE (default), draws a plot of the diagnostic criteria.
data	A data frame (required when Y is a formula with column names).
...	Additional arguments passed to nmfkc and nmfkc.ecv . • Q: (Deprecated) Alias for rank. • save.time: (Deprecated) TRUE maps to detail = "fast". • cores: evaluate the per-rank fits in parallel (default <code>getOption("mc.cores", 1L)</code>; PSOCK cluster on Windows, forking elsewhere). Each rank is a deterministic self-seeded fit and results are returned in order, so the criteria table is identical for any cores. (Also forwarded to nmfkc.ecv.)

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. The <code>effective.rank</code> column gives the effective rank ($\exp$ of the Shannon entropy of the explained-variance distribution $p_k = \text{var}(B_{k.}) / \sum_j \text{var}(B_{j.})$, in $[1, Q]$); when it plateaus well below the nominal rank, the extra factors are not carrying additional coefficient variance, which suggests an over-specified rank. The <code>effective.rank.ratio</code> column is the raw utilization fraction <code>effective.rank / rank</code> in $[0, 1]$ (kept in the table, not plotted). What <code>plot = TRUE</code> draws as the green <code>eff.rank</code> line is the broken-stick-corrected <code>effective.rank.index</code> column, $(\text{effective.rank} - E) / (\text{rank} - E)$ clamped to $[0, 1]$ with $E = \exp(H_Q - 1)$ the expected effective rank under a random (uniform-Dirichlet) variance split (H_Q the Q -th harmonic number, stored as <code>effective.rank.expected</code>); anchoring 0 at this null removes the small-rank inflation of the raw ratio, so a peak of the index marks the rank at which the latent factors carry the most evenly distributed variance relative to chance.

References

Roy, O., & Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. *Proc. 15th European Signal Processing Conf. (EUSIPCO)*, 606–610. (effective.rank) Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4), 397–405. doi:10.1080/00401706.1978.10489693 ([sigma.ecv](#))

See Also

[nmfkc](#), [nmfkc.ecv](#) (element-wise CV, used internally), [nmfkc.bicv](#) (block bi-cross-validation), [nmfkc.consensus](#) (stability) and [nmfkc.ard](#) (Bayesian ARD) for alternative rank criteria.

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	<i>Plot Diagnostics: Original, Fitted, and Residual Matrices as Heatmaps</i>
---------------------	--

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

<code>Y</code>	The original observation matrix ($P \times N$).
<code>result</code>	The result object returned by the <code>nmfkc</code> function.
<code>fitted.palette</code>	A vector of colors for Y and XB heatmaps. Defaults to white-orange-red. For backward compatibility, <code>Y_XB_palette</code> is accepted via
<code>residual.palette</code>	A vector of colors for the residuals heatmap. Defaults to blue-white-red. For backward compatibility, <code>E_palette</code> is accepted via
<code>...</code>	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)
```

nmfkc.signed

NMF-KC with signed covariate matrix

Description

Solves

$$Y \approx X \Theta A, \quad X \geq 0, \Theta \in \mathbb{R}^{Q \times D}, A \in \mathbb{R}^{D \times N},$$

where the covariate matrix A and the coefficient matrix Θ may be **signed**. Internally $A = A_+ - A_-$ and $\Theta = C_+ - C_-$ with $A_{\pm}, C_{\pm} \geq 0$ (sign-splitting trick, Ding et al. 2010), and the problem is solved by a Direct Multiplicative Update algorithm whose iteration cost is $O(QD^2)$, independent of N .

Only X is structurally constrained to be non-negative (Semi-NMF sense of Ding, Li, & Jordan 2010). In particular, Y **may contain negative entries**, in which case the response is fit in the least-squares sense without any non-negativity requirement on Y .

When $A \geq 0$ (so $A_- = 0$), the result reduces to `nmfkc`(Y, A, rank) with Euclidean loss, up to reordering.

Usage

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

Arguments

- | | |
|---------|---|
| Y | Real-valued $Q_{\text{obs}} \times N$ response matrix. Unlike <code>nmfkc</code> , negative entries are allowed. |
| A | Real-valued $D \times N$ covariate matrix (signed). A single matrix is passed; its positive and negative parts $A_+ = \max(A, 0)$ and $A_- = \max(-A, 0)$ are computed internally. When using Random Fourier Features (Rahimi & Recht 2007) as A , supply the RFF parameters via the hidden pars argument so that <code>predict()</code> can regenerate features for new data (see pars entry in . . . below). |
| rank | Integer. Number of latent components Q in X . |
| epsilon | Relative convergence tolerance on the objective (default 1e-4). |
| maxit | Maximum number of iterations (default 5000). |
| verbose | Logical. Print dimensions at start (default TRUE). |
| ... | Additional arguments: <ul style="list-style-type: none"> • Q: alias for rank. • X.restriction: constraint applied to columns of X after every update, with the scale absorbed into C_+, C_-. One of "colSums" (default, <code>colSums(X) = 1</code>), "colSqSums", "totalSum", "none", "fixed". • X.L2.ortho: non-negative L2 orthogonality penalty on the columns of X (default 0), penalizing $(\lambda/2)\ \text{offdiag}(X^\top X)\ ^2$. Same convention as <code>nmfkc</code>; skipped when <code>X.restriction = "fixed"</code>. • X.L2.smooth: non-negative L2 row-smoothness penalty on X (default 0), penalizing $(\lambda/2)\text{tr}(X^\top LX)$ with L the path-graph Laplacian over rows (adjacent-row differences). Useful for ordered rows (e.g. time / space); skipped when <code>X.restriction = "fixed"</code>. • C.L2: non-negative ridge penalty on the signed coefficient matrix $C = C_+ - C_-$ (default 0), adding $\lambda\ C_+ - C_-\ ^2$. Shrinks Θ toward zero (with zero gradient on the unidentified common mode $C_+ + C_-$), injected into both the fast unweighted and the weighted C_+/C_- updates. • X.init: initialization strategy for the basis matrix X ($Q_{\text{obs}} \times Q$). Accepts the same menu as <code>nmfkc</code>: "kmeans" (default), "kmeansar", "nndsvd", "runif", or a user-supplied $Q_{\text{obs}} \times Q$ non-negative numeric matrix. String methods delegate to the shared internal helper <code>.init_X_method()</code> (see <code>nmfkc</code> for the definitions of each method). For signed Y, "kmeans" cluster centers may contain negative entries; they are clipped to zero to satisfy $X \geq 0$, and any column that collapses to all-zeros is re-filled with small <code>Uniform(0, 0.1)</code> noise. • C.init: explicit initial $Q \times D$ coefficient matrix Θ (signed). Split internally. • warm.start: logical (default TRUE). If TRUE and $Y \geq 0$, runs <code>nmfkc(Y, A = rbind(A_+, A_-), rank = Q)</code> internally to seed X, C_+, C_-. The user's <code>X.init</code>, <code>seed</code>, <code>nstart</code>, and <code>X.restriction</code> are forwarded to the internal <code>nmfkc</code> call so that initialization choices propagate consistently between the warm-start and the signed MU loop. Ignored when Y has negative entries (warm-start is disabled; <code>X.init</code> is used directly by the signed branch instead). |

- seed: RNG seed for random initialization (default 123).
- prefix: name prefix for rows of C and columns of X (default "Basis").
- pars: optional list list(omega, b, D, beta) of Random Fourier Feature parameters (Rahimi & Recht 2007; omega: frequency matrix, b: phase offset, D: feature dimension, beta: bandwidth). When supplied, it is stored in the returned object so that summary() can report β and downstream predict() calls can regenerate RFF features for new data. If A is not RFF features, leave this NULL.
- Y.weights: Optional non-negative weight matrix ($Q_{\text{obs}} \times N$) or vector (length N), analogous to the weights argument of `lm`. Loss becomes $\sum W_{ij} (Y_{ij} - (XCA)_{ij})^2$ (`lm`-style, **linear** in W). Logical matrices (TRUE / FALSE) are also accepted. Typical usage by `nmfkc.signed.cv` / `nmfkc.signed.ecv` passes a binary mask $W \in \{0, 1\}$ to hold out test elements; real-valued weights for observation-level importance weighting are also supported. Default NULL: if Y has NA, a binary mask is auto-constructed (0 for NA, 1 elsewhere); otherwise no weighting.
- nstart: number of random restarts. **Signed models have more local minima than non-negative ones** because $\Theta = C_+ - C_-$ can take both positive and negative values. Since `nmfkc.signed()` itself does not loop over restarts (callers control it), set the outer-loop size via e.g. running the function several times with different seed and keeping the fit with the smallest \$objfunc. A restart budget of 10-50 is recommended for publication-grade runs on signed data.

Value

An object of class `c("nmfkc.signed", "nmfkc")` with

- X: $Q_{\text{obs}} \times Q$ basis matrix (non-negative, column-normalized according to X.restriction).
- Cp, Cn: $Q \times D$ non-negative parts of Θ , so that $\Theta = C_+ - C_-$.
- C: $C_+ - C_-$ ($= \Theta$), signed.
- B: CA , $Q \times N$ (signed).
- objfunc.iter: objective values per iteration.
- objfunc: final objective.
- r.squared: $\text{cor}(Y, \hat{Y})^2$ (Pearson; in $[0, 1]$).
- r.squared.uncentered: uncentered $R^2 = 1 - \|Y - \hat{Y}\|_F^2 / \|Y\|_F^2$ (baseline = zero matrix).
- r.squared.centered: row-mean centered $1 - \|Y - \hat{Y}\|_F^2 / \|Y - \bar{Y}_p\|_F^2$.
- mae: mean absolute error.
- iter: number of iterations performed.
- runtime: elapsed seconds.
- Y.signed: logical; whether Y contained negative entries during fitting.
- pars: RFF generating parameters, if supplied.
- call: the matched call.

Lifecycle

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

References

- Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE TPAMI*, 32(1), 45–55.
- Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in NIPS*, 20.

See Also

[nmfkc](#), [predict.nmfkc.signed](#)

Examples

```
set.seed(1)
## Example 1: signed A (e.g., hand-built RFF features), non-negative Y
## Build simple signed features  $Z = \sqrt{2/D} * \cos(\omega^T U + b)$ :
U    <- matrix(stats::rnorm(5 * 40), 5, 40)      # raw input
D    <- 20                                       # feature dim
omega <- matrix(stats::rnorm(5 * D), 5, D)       # random freqs
b    <- stats::runif(D, 0, 2 * pi)             # phase
Z    <- sqrt(2 / D) *
      cos(t(omega) %*% U + matrix(b, D, 40))    # D x 40, signed
Y    <- matrix(abs(stats::rnorm(8 * 40)), 8, 40)
res1 <- nmfkc.signed(Y, A = Z, rank = 3, maxit = 200)

## Example 2: signed Y (regression)
Y2   <- matrix(stats::rnorm(8 * 40), 8, 40)      # signed response
res2 <- nmfkc.signed(Y2, A = Z, rank = 3, maxit = 200,
                    warm.start = FALSE)
```

nmfkc.signed.cv

Column-wise k-fold cross-validation for nmfkc.signed

Description

Column-wise k-fold CV by held-out samples: for each fold, the model is fit on the training columns and evaluated on the held-out columns by solving for new-sample coefficients via a weighted refit with X fixed.

Usage

```
nmfkc.signed.cv(Y, A, rank = 2, ...)
```

Arguments

Y	Real-valued $Q_{\text{obs}} \times N$ response matrix (signed entries allowed).
A	Real-valued $D \times N$ covariate matrix (signed).
rank	Integer Q .
...	Passed to nmfkc.signed ; also accepts nfolds (default 5; div alias), seed (default 123), shuffle (default TRUE).

Value

A list with objfunc (mean squared prediction error), sigma (RMSE), objfunc.block (per-fold MSE vector), block (integer fold assignment of length N). Field names match [nmfkc.cv](#).

Lifecycle

This function is **experimental**.

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

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

nmfkc.signed.ecv	<i>Element-wise cross-validation for nmfkc.signed</i>
------------------	---

Description

Element-wise k-fold CV (Wold's CV): held-out elements are masked via `Y.weights = 0` during fitting, and the RMSE on those elements is reported. Loops over candidate rank values.

Usage

```
nmfkc.signed.ecv(Y, A, rank = 1:3, ...)
```

Arguments

Y	Real-valued $Q_{\text{obs}} \times N$ response matrix (signed entries allowed).
A	Real-valued $D \times N$ covariate matrix (signed).
rank	Integer vector of candidate ranks (default 1:3).
...	Passed to nmfkc.signed ; also accepts nfolds (default 5; div alias), seed (default 123).

Value

A list with objfunc (MSE per rank), sigma (RMSE), objfunc.fold (per-fold per-rank), folds, Q.grid.

Lifecycle

This function is **experimental**.

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

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

nmfkc.signed.rank	<i>Rank selection for nmfkc.signed (concise diagnostics)</i>
-------------------	--

Description

Fits [nmfkc.signed](#) across a range of ranks and reports r.squared, the effective rank, and the element-wise CV error sigma.ecv, with the same concise plot as [nmfkc.rank](#).

Usage

```
nmfkc.signed.rank(
  Y,
  A,
  rank = 1:5,
  detail = c("full", "fast"),
  plot = TRUE,
  ...
)
```

Arguments

Y	Observation matrix (may contain negative entries).
A	Covariate matrix (may be signed).
rank	Integer vector of ranks to evaluate.
detail	"full" (default) also runs element-wise CV (sigma.ecv); "fast" skips it (plots r.squared and eff.rank only, and recommends the R-squared elbow).
plot	Logical; draw the diagnostics plot (default TRUE).
...	Passed on to nmfkc.signed and nmfkc.signed.ecv (e.g.\maxit, nfolds, seed).

Value

A list with `rank.best` and `criteria` (`rank`, `effective.rank`, `effective.rank.ratio`, `r.squared`, `sigma.ecv`).

References

Roy, O., & Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. *Proc. EUSIPCO*, 606–610. (`effective.rank`) Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4), 397–405. (`sigma.ecv`)

See Also

[nmfkc.signed](#), [nmfkc.rank](#)

nmfkc.signed.rff	<i>Random Fourier Features for nmfkc.signed()</i>
------------------	---

Description

Generates RFF random parameters $\omega_d \sim \mathcal{N}(0, 2\beta I_p)$, $b_d \sim \text{Uniform}(0, 2\pi)$ (Rahimi & Recht, 2007) and applies the RFF transform

$$z_d(u) = \sqrt{2/D} \cos(\omega_d^\top u + b_d)$$

to each column of U , yielding a sign-unrestricted $D \times N$ feature matrix Z such that $Z^\top Z \approx K$, the Gaussian kernel matrix with bandwidth `beta`.

The return value is a list with the feature matrix Z and the generating parameters `pars = list(omega, b, D, beta)` so that the same random map can be re-applied to new data (by passing `pars` back) and [nmfkc.signed](#) can record the parameters for downstream summary.

Usage

```
nmfkc.signed.rff(U, beta = NULL, D = ceiling(ncol(U)/2), seed = NULL, ...)
```

Arguments

<code>U</code>	A $p \times N$ numeric matrix; columns are data points.
<code>beta</code>	Positive scalar. Gaussian kernel bandwidth parameter. Can be obtained via nmfkc.kernel.beta.nearest.med . May be <code>NULL</code> only when <code>pars</code> is supplied through <code>...</code>
<code>D</code>	Integer. Number of random features. Defaults to <code>ceiling(ncol(U) / 2)</code> . This default is intended for the training-time fresh generation only; for test data, always supply <code>pars</code> via <code>...</code> to inherit the training D together with ω, b . For very large N the default may be excessive (direct MU cost is $O(QD^2)$); choose a smaller D manually. For very small N , RFF is not recommended; use a full kernel matrix with nmfkc instead.

seed Optional integer passed to `set.seed()` before generating ω, b , for reproducibility. Ignored when `pars` is supplied.

... Hidden option pars: a list `list(omega, b, D, beta)` obtained from a previous call (`Ztrain$pars`). When supplied, ω, b are reused and the `beta, D, seed` arguments are ignored. Use this to apply the same random map to test data.

Value

A list with two elements:

Z A $D \times N$ sign-unrestricted numeric matrix. Pass this to `nmfkc.signed` as its `A` argument.

pars A list `list(omega, b, D, beta)`. Pass this to `nmfkc.signed` via its `pars` argument (for summary display) and to subsequent `nmfkc.signed.rff()` calls (to reuse the same random map on new data).

Lifecycle

This function is **experimental**. The interface may change in future versions; details are to be described in an upcoming paper.

See Also

`nmfkc.signed`, `nmfkc.kernel.beta.nearest.med`

Examples

```
## Iris 3-class classification with RFF + direct MU (Ding-Li-Jordan)
data(iris)
set.seed(1)
idx <- sample(nrow(iris), 100) # 100 training, 50 test

## Scale features using TRAINING mean/sd; transpose to p x N layout
mn <- colMeans(iris[idx, 1:4])
sc <- apply(iris[idx, 1:4], 2, sd)
U.train <- t(scale(iris[idx, 1:4], center = mn, scale = sc)) # 4 x 100
U.test <- t(scale(iris[-idx, 1:4], center = mn, scale = sc)) # 4 x 50

## One-hot encode training labels as a Q_obs x N target matrix
levs <- levels(iris$Species)
Y.train <- sapply(iris$Species[idx], function(s) as.integer(levs == s))
rownames(Y.train) <- levs # 3 x 100
lab.train <- iris$Species[idx]
lab.test <- iris$Species[-idx]

## Beta candidates from nearest-neighbour median heuristic
beta_info <- nmfkc.kernel.beta.nearest.med(U.train)
betas <- beta_info$beta_candidates

## CV over beta candidates: for each beta, generate RFF, fit, and
## evaluate column-wise CV-MSE on training data
cv_mse <- numeric(length(betas))
```

```

for (i in seq_along(betas)) {
  rff_i <- nmfkc.signed.rff(U.train, beta = betas[i], D = 50, seed = 1)
  cv_i <- nmfkc.signed.cv(Y.train, A = rff_i$Z, rank = 3, seed = 123)
  cv_mse[i] <- cv_i$objfunc
}
beta_best <- betas[which.min(cv_mse)]

## Generate signed RFF features with the best beta
rff.train <- nmfkc.signed.rff(U.train, beta = beta_best, D = 50, seed = 1)
rff.test <- nmfkc.signed.rff(U.test, pars = rff.train$pars)

## Fit on training data only
res <- nmfkc.signed(Y.train, A = rff.train$Z, rank = 3,
  pars = rff.train$pars, verbose = FALSE)

## Predict on training and test separately
pred.train <- predict(res, newA = rff.train$Z, type = "class")
pred.test <- predict(res, newA = rff.test$Z, type = "class")
mean(pred.train == as.character(lab.train))
mean(pred.test == as.character(lab.test))

```

nmfre

Non-negative Matrix Factorization with Random Effects

Description

Estimates the NMF-RE model

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

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

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

Estimation is an outer-inner ECM: an inner loop (fixed λ) runs a ridge/BLUP update for U , a complete-EM semi-NMF update for X , and a fixed-effect update for Θ ; an outer loop runs the EM M-steps for (σ^2, τ^2) . The variance components are estimated from the data; the effective degrees of freedom df_U are reported only as a diagnostic.

`nmfre()` performs **optimization only**. Hypothesis tests and standard errors for Θ are obtained separately with `nmfre.inference` (sandwich SE + wild bootstrap), mirroring the `nmfkc` / `nmfkc.inference` split.

Usage

```
nmfre(
  Y,
  A = NULL,
  rank = 2,
  C.signed = TRUE,
  epsilon = 1e-05,
  maxit = 5000,
  ...
)
```

Arguments

<code>Y</code>	Observation matrix (P x N), non-negative.
<code>A</code>	Covariate matrix (K x N). Default is a row of ones (intercept only).
<code>rank</code>	Integer. Rank of the basis matrix X . Default is 2. For backward compatibility, <code>Q</code> is accepted via <code>...</code>
<code>C.signed</code>	Logical. Whether the fixed-effect coefficients C ($= \Theta$ in the paper) are sign-free; the single switch that selects the whole estimation scheme. <code>TRUE</code> (default, recommended, matches the paper) treats C as real-valued and updates it by exact least squares, with the basis X estimated by the complete-EM semi-NMF step and a two-sided test (interior null). <code>FALSE</code> constrains $C \geq 0$ via a multiplicative update, with X estimated by the positive-part multiplicative update and a one-sided test (boundary null). The basis X is always non-negative. A character value ("signed" / "nonneg") is also accepted for backward compatibility.
<code>epsilon</code>	Convergence tolerance for relative change in objective (default 1e-5).
<code>maxit</code>	Maximum number of iterations. Default 5000 (matches <code>nmfkc</code> and the other MU functions in the package). When the cap is hit without meeting the relative-tolerance criterion, a "maximum iterations (...) reached..." warning is emitted so users notice unconverged fits.
<code>...</code>	Additional arguments for initialization, variance control, dfU control, optimization, and inference settings. <code>X.init</code>: Initial basis matrix (P x Q), or <code>NULL</code>. When <code>NULL</code>, <code>nmfkc</code> is called internally to generate initial values. <code>C.init</code>: Initial coefficient matrix (Q x K), or <code>NULL</code>. When <code>NULL</code>, <code>nmfkc</code> is called internally to generate initial values. <code>U.init</code>: Initial random effects matrix (Q x N), or <code>NULL</code> (all zeros). <code>prefix</code>: Prefix for basis names (default "Basis"). <code>sigma2</code>: Initial residual variance (default 1). <code>sigma2.update</code>: Logical. Update σ^2 during iterations (default <code>TRUE</code>). <code>tau2</code>: Initial random effect variance (default 1). <code>tau2.update</code>: Logical. Update τ^2 by moment matching (default <code>TRUE</code>). <code>x.postvar</code>: Logical. Include the posterior-variance term in the semi-NMF X-step (default <code>TRUE</code>; advanced). Applies only to the sign-free / semi-NMF path (<code>C.signed = TRUE</code>).

- `X.L2.smooth`: Non-negative row-smoothness penalty on the basis X (default 0), adding $(\lambda/2) \text{tr}(X^\top LX)$ with L the path-graph Laplacian over rows (squared adjacent-row differences). Well suited to ordered rows (e.g. longitudinal / spatial bases). Injected into the X -step for both `C.signed = TRUE/FALSE`; leaves the random-effect variance updates unchanged.
- `X.L2.ortho`: Non-negative column-orthogonality penalty on X (default 0), penalizing $(\lambda/2) \|\text{offdiag}(X^\top X)\|^2$.
- `C.L2`: Non-negative ridge penalty on $\Theta = C$ (default 0), adding $\lambda \|C\|^2$. A Gaussian prior on the fixed effects; for `C.signed = TRUE` the C -step stays a closed-form Sylvester ridge-LS solve, for `C.signed = FALSE` it is added to the multiplicative-update denominator.
- `dfU.control`: Deprecated and inert. The algorithm imposes no cap on df_U ("off", the only behaviour); df_U is reported as a diagnostic only.
- `print.trace`: Logical. If TRUE, print progress every 100 iterations (default FALSE).
- `seed`: Integer seed for reproducibility (default 1).
- `nstart`: Number of random restarts for the `nmfkc()` initialisation step (passed to the k-means initialiser). Default 1 (single start; historical behaviour). A larger value (e.g. 10-20) gives a more stable initialisation.
- `inner.maxit`, `outer.maxit`: Maximum inner (fixed- λ block-coordinate) and outer (EM variance) iterations (defaults 10000 and 500).
- `epsilon.outer`: Convergence tolerance for the outer EM loop on λ (default $1e-6$).

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.

`X.prob` Row-wise soft-clustering probabilities from the non-negative X (each row normalized to sum to 1), as in `nmfkc`.

`X.cluster` Hard-clustering label for each row of X (argmax over `X.prob`).

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

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

Variance components

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

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

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

Convergence diagnostics

`converged` Logical. Whether the algorithm converged.

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

`iter` Number of iterations performed.
`maxit` Maximum iterations setting used.
`epsilon` Convergence tolerance used.
`objfunc` Final objective function value $\|Y - X(\Theta A + U)\|^2 + \lambda\|U\|^2$.
`rel.change.final` Final relative change in objective.
`objfunc.iter` Numeric vector of the fixed- λ penalized objective $\|Y - X(\Theta A + U)\|^2 + \lambda\|U\|^2$ per iteration. Monotone within an inner loop but *not* across outer iterations (the penalty jumps when λ is updated).
`rss.trace` Numeric vector of $\|Y - X(\Theta A + U)\|^2$ per iteration.
`nll.trace` Numeric vector of the marginal negative log-likelihood $\ell(X, \Theta, \sigma^2, \tau^2)$ per iteration (random effects integrated out). This is the ECM-monotone quantity and is what `plot.nmfre` displays.

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.
`lambda.enforced` Final λ enforced to satisfy the cap.
`dfU.hit.cap` Logical. Whether the cap was binding.
`dfU.hit.iter` Iteration at which the cap first bound.
`dfU.frac` $df_U / (NQ)$, fraction of maximum df.
`dfU.cap.frac` $df_U^{\text{cap}} / (NQ)$.

Fitted matrices

`B` Fixed-effect scores ΘA ($Q \times N$).
`B.prob` Column-normalized probabilities from $\max(\Theta A, 0)$.
`B.blup` BLUP scores $\Theta A + U$ ($Q \times N$).
`B.blup.pos` Non-negative BLUP scores $\max(\Theta A + U, 0)$ ($Q \times N$).
`B.blup.prob` Column-normalized probabilities from $\max(\Theta A + U, 0)$.
`XB` Fitted values from fixed effects $X\Theta A$ ($P \times N$).
`XB.blup` Fitted values including random effects $X(\Theta A + U)$ ($P \times N$).

Fit statistics

`r.squared` Pearson $\text{cor}(Y, X(\Theta A + U))^2$ (BLUP prediction).
`r.squared.uncentered` Uncentered $1 - \|Y - X(\Theta A + U)\|_F^2 / \|Y\|_F^2$ (BLUP; baseline = zero matrix).
`r.squared.centered` Row-mean centered $1 - \|Y - X(\Theta A + U)\|_F^2 / \|Y - \bar{Y}_p\|_F^2$ (BLUP; baseline = per-row mean).
`r.squared.fixed` Pearson $\text{cor}(Y, X\Theta A)^2$ (fixed-only prediction).

`r.squared.fixed.uncentered`, `r.squared.fixed.centered` Uncentered and centered R^2 for the fixed-only prediction.

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.

Sign convention

`C.signed` Logical. Whether C was sign-free (TRUE) or non-negative (FALSE).

Standard errors, z-values, p-values, and confidence intervals for Θ are **not** computed here; obtain them by passing the fit to `nmfre.inference`.

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`, `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)

  # Fit (sign-free Theta by default; variances estimated by EM/ECM)
  res <- nmfre(Y, A, rank = 1)
  summary(res)
}
```

nmfre.ecv

NMF-RE element-wise cross-validation for rank selection

Description

Selects the basis rank Q for `nmfre` by Wold-style element-wise (entry-holdout) cross-validation that **exercises the random effects**. For each fold the held-out entries are hidden and filled by iterative imputation: fit the NMF-RE model on the current matrix, replace the held-out entries with the BLUP prediction $X(\Theta A + U)$, and repeat. The score is the held-out prediction RMSE (sigma); the selected rank minimizes it.

Because the held-out entries of a column are predicted using the random effect u_n fitted from that column's *retained* entries, this evaluates the full NMF-RE model (including U) — unlike `nmfkc.ecv`, which masks entries by zero weight and predicts from the fixed-effect fit $X\Theta A$ only. The two scores are therefore *not* directly comparable.

Usage

```
nmfre.ecv(Y, A = NULL, rank = 1:3, C.signed = TRUE, ...)
```

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 vector of ranks Q to evaluate (default 1:3).
C.signed	Logical. Sign convention for Θ passed to each <code>nmfre</code> fit (TRUE = sign-free, default; FALSE = non-negative). The basis update rule follows this choice automatically.
...	Additional arguments: • <code>nfolds</code>: Number of folds (default 5; legacy <code>nfold</code> also accepted). • <code>rounds</code>: Iterative-imputation rounds per fold (default 4). • <code>seed</code>: RNG seed for the fold assignment (default 1). • <code>cores</code>: Integer; number of parallel workers for the rank sweep (default <code>getOption("mc.cores", 1L)</code>). Because the fold matrix is fixed before the sweep and each <code>nmfre</code> fit self-seeds, results are identical to the sequential run for any cores. • <code>print.trace</code>: Logical; print per-rank scores (default FALSE). • Convergence controls forwarded to <code>nmfre</code>: <code>epsilon</code> (default 1e-5), <code>epsilon.outer</code> (default 1e-3), <code>inner.maxit</code> (default 1500), <code>outer.maxit</code> (default 80), <code>maxit</code> (default 40000). CV does not need the tight tolerances of a final fit, so these are loosened by default.

Value

A list of class "nmfre.ecv" with components:

rank The ranks evaluated.

sigma Named numeric vector of held-out RMSE per rank (same field name as [nmfkc.ecv](#)).

best The rank minimizing sigma.

nfolds, rounds, C.signed Settings used.

See Also

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

Examples

```
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)
  cv <- nmfre.ecv(Y, A, rank = 1:3)
  cv$best
  plot(cv)
}
```

nmfre.inference

Statistical inference for the coefficient matrix C from NMF-RE

Description

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

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

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

Usage

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

Arguments

<code>object</code>	An object of class "nmfre" returned by <code>nmfre</code> .
<code>Y</code>	Observation matrix (P x N). Must match the data used in <code>nmfre()</code> .
<code>A</code>	Covariate matrix (K x N). Default is NULL (intercept only).
<code>wild.bootstrap</code>	Logical. If TRUE (default), performs wild bootstrap for confidence intervals and bootstrap standard errors.
<code>...</code>	Additional arguments:
	<code>wild.B</code> Number of bootstrap replicates. Default is 500.
	<code>wild.seed</code> Seed for bootstrap. Default is 123.
	<code>wild.level</code> Confidence level for bootstrap CI. Default is 0.95.
	<code>C.p.side</code> P-value type: "one.sided" (default) or "two.sided".
	<code>cov.ridge</code> Ridge stabilization. Default is 1e-8.
	<code>print.trace</code> Logical. Default is FALSE.

Value

The input object with additional inference components:

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

`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.nmf.cluster.criteria`*Plot clustering-quality criteria across a sequence of fits*

Description

Line plot of silhouette, CPCC, and `dist.cor` against the result index, for an object from `nmf.cluster.criteria`. X-axis ticks default to each result's `$rank` (overridable via the `names` argument of `nmf.cluster.criteria`).

Usage

```
## S3 method for class 'nmf.cluster.criteria'
plot(
  x,
  main = "Clustering quality across rank",
  xlab = "rank (Q)",
  ylab = "criterion",
  lwd = 2,
  ...
)
```

Arguments

<code>x</code>	An object of class <code>"nmf.cluster.criteria"</code> .
<code>main</code>	Plot title.
<code>xlab, ylab</code>	Axis labels.
<code>lwd</code>	Line width.
<code>...</code>	Further arguments passed to the initial <code>plot</code> .

Value

`x`, invisibly.

See Also

`nmf.cluster.criteria`

plot.nmf.cluster.flow *Plot a cluster-flow (alluvial) diagram*

Description

Draws the alluvial / Sankey-style cluster-flow diagram for an object created by `nmf.cluster.flow`.

Usage

```
## S3 method for class 'nmf.cluster.flow'
plot(
  x,
  col = NULL,
  lwd = 1,
  xlab = "rank (Q)",
  ylab = "individuals",
  main = "Cluster flow across rank",
  ...
)
```

Arguments

x	An object of class "nmf.cluster.flow".
col	Optional colour vector indexed by reference cluster id (col[k] colours every individual whose reference cluster is k); recycled if shorter than the number of reference clusters. Defaults to the object's palette.
lwd	Line width of the flow segments.
xlab, ylab	Axis labels.
main	Plot title.
...	Further arguments passed to the initial <code>plot</code> call.

Value

x, invisibly.

See Also

`nmf.cluster.flow`

plot.nmf.rank	<i>Plot a rank-selection (nmf.rank) object</i>
---------------	--

Description

Draws the concise three-criterion rank-selection plot for an object returned by [nmfkc.rank](#) (or `nmfkc.net.rank`, `nmfkc.signed.rank`, `nmfae.rank`, `nmfae.signed.rank`): `r.squared` (red) and the broken-stick-corrected effective-rank index `effective.rank.index` (green, legend label `eff.rank`; **not** the raw `effective.rank.ratio`) on the left $[0, 1]$ axis, and the cross-validation error `sigma.ecv` (blue) on the right axis, each with points, rank-number labels and a highlighted best marker.

Usage

```
## S3 method for class 'nmf.rank'
plot(
  x,
  main = NULL,
  xlab = "Rank (Q)",
  ylab = "R-squared / eff.rank (0-1)",
  lwd = 3,
  ...
)
```

Arguments

<code>x</code>	An object of class <code>"nmf.rank"</code> .
<code>main</code>	Plot title (defaults to the title stored in <code>x</code>).
<code>xlab</code> , <code>ylab</code>	Axis labels.
<code>lwd</code>	Line width of the criterion curves.
<code>...</code>	Further arguments passed to the initial plot .

Value

`x`, invisibly.

See Also

[nmfkc.rank](#)

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 nmfae .
...	Additional graphical parameters passed to plot.

Value

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

See Also

[nmf.rrr](#), [nmf.rrr.heatmap](#)

Examples

```
set.seed(1)
Y <- matrix(runif(20), nrow = 4)
res <- nmf.rrr(Y, rank1 = 2)
plot(res)
```

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.ar.latent` *Plot the latent dynamics of an NMF-VAR model*

Description

Draws the model's own dynamics — what the fit *does* — rather than the realized series. All of it lives in the $Q \times Q$ latent transition matrices $G_d = \Theta_d X$ returned by `nmfkc.ar.latent`, so every panel stays readable at a size the observed space cannot: a 47-region, 7-lag, rank-4 fit has $47^2 = 2209$ observed impulse responses but only $4^2 = 16$ latent ones.

Usage

```
## S3 method for class 'nmfkc.ar.latent'
plot(
  x,
  type = c("roots", "graph", "irf", "lag", "phase"),
  lag = NULL,
  horizon = 16L,
  ...
)
```

Arguments

`x` An object from `nmfkc.ar.latent`.
`type` Which panel to draw; see Details.
`lag` Which lag to use for "graph" and "phase". Default NULL means $\sum_d G_d$ for "graph" (the total one-step influence) and lag 1 for "phase".

horizon	Number of periods for "irf". Default 16.
...	Passed to the underlying plot call. "phase" also accepts start, a matrix of initial b_0 values (one per row).

Details

The available panels are

"roots" (default) The DQ eigenvalues of the latent companion matrix on the unit circle. Complex roots are drawn in a different colour and their implied period is reported: a fit can sit near the boundary either because it cycles or because it is seasonal, and the spectral radius alone does not say which.

"graph" The transition graph of G_d : self-loops are persistence, arrows $q' \rightarrow q$ are spillover, widths are proportional to the entries. Unlike a $y \rightarrow b \rightarrow y$ bipartite picture this graph is closed, so an asymmetric feedback loop is visible as such.

"irf" Latent impulse responses Ψ_h , a $Q \times Q$ panel. Since $G_d \geq 0$ the responses never change sign, so the shape is a decay or a hump and its half-life can be read off.

"lag" $(G_d)_{qq'}$ as a function of d : how many periods a shock takes to arrive. With $Q = 1$ this is a single bar plot of the lag profile.

"phase" Phase portrait of $b_t \mapsto Gb_{t-1} + \theta$ with the fixed point μ_b and the eigen-directions. Requires $D = 1$ and $Q = 2$. This is where a monotone relaxation and a true cycle look different; two coefficient series plotted against time do not distinguish them.

Value

invisible(NULL); called for the plot. "irf" and "lag" set up their own multi-panel layout and restore the previous one on exit.

See Also

[nmfkc.ar.latent](#), [nmfkc.ar.latent.inference](#), [plot.nmfkc.ar.stationarity](#), [nmfkc.ar.DOT](#)

Examples

```
set.seed(1)
Y <- matrix(abs(rnorm(4 * 60)) + 1, 4, 60)
ar <- nmfkc.ar(Y, degree = 3)
fit <- nmfkc(ar$Y, ar$A, rank = 2, verbose = FALSE)
lat <- nmfkc.ar.latent(fit)
plot(lat) # roots on the unit circle
plot(lat, type = "graph") # latent transition graph
plot(lat, type = "irf") # latent impulse responses
```

`plot.nmfkc.ar.stationarity`*Plot the stationarity bracket of an NMF-VAR model*

Description

Draws the Perron-Frobenius bracket $\min_j c_j \leq \rho(\sum_d G_d) \leq \max_j c_j$ against the unit boundary, where c_j are the column sums of $\sum_d \Theta_d$. It shows at a glance whether the cheap sufficient condition settles stationarity or an eigenvalue computation is needed. For the model's dynamics see [plot.nmfkc.ar.latent](#).

Usage

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

Arguments

<code>x</code>	An object from nmfkc.ar.stationarity .
<code>...</code>	Passed to the underlying plot call.

Value

`invisible(NULL)`; called for the plot.

See Also

[nmfkc.ar.stationarity](#), [plot.nmfkc.ar.latent](#)

Examples

```
set.seed(1)  
Y <- matrix(abs(rnorm(4 * 60)) + 1, 4, 60)  
ar <- nmfkc.ar(Y, degree = 2)  
fit <- nmfkc(ar$Y, ar$A, rank = 2, verbose = FALSE)  
plot(nmfkc.ar.stationarity(fit))
```

plot.nmfkc.ard	<i>Plot method for nmfkc.ard objects</i>
----------------	--

Description

Bar plot of the per-component relevance (descending), with a line at the pruning threshold; bars above it (the estimated rank) are highlighted.

Usage

```
## S3 method for class 'nmfkc.ard'
plot(x, main = NULL, ...)
```

Arguments

x	An object of class "nmfkc.ard".
main	Plot title.
...	Passed to barplot .

Value

x, invisibly.

plot.nmfkc.consensus	<i>Plot a consensus rank-selection (nmfkc.consensus) object</i>
----------------------	---

Description

Two views of a [nmfkc.consensus](#) result:

- type = "criteria" (default): the stability curves cophenetic (blue) and dispersion (red) against rank. No "best" marker is drawn – a stable rank tends to be a coarse one, so the curves are shown for inspection rather than as an optimum.
- type = "heatmap": the consensus matrix of each requested rank, reordered by average-linkage hierarchical clustering of $1 - \bar{C}$ (blue = 0 / different cluster, red = 1 / same cluster). Requires keep.consensus = TRUE at compute time.

Usage

```
## S3 method for class 'nmfkc.consensus'
plot(
  x,
  type = c("criteria", "heatmap"),
  rank = NULL,
  mfrow = NULL,
  mar = NULL,
  col = grDevices::hcl.colors(50, "Blue-Red"),
  main = NULL,
  ...
)
```

Arguments

x	An object of class "nmfkc.consensus".
type	"criteria" or "heatmap".
rank	For type = "heatmap", the rank(s) to display. NULL (default) shows every rank stored in x.
mfrow	Panel layout for multiple heatmaps, as c(nrow, ncol). NULL (default) uses n2mfrow for a near-square grid.
mar	Per-panel margins c(b, l, t, r) for the heatmap view. NULL (default) uses tight margins for a grid and wider margins (for sample labels) when a single rank is shown.
col	Heatmap colour palette (length-50 blue-to-red by default).
main	Plot title for the "criteria" view (heatmap panels are titled per rank).
...	Further arguments passed to the underlying plot / image .

Value

x, invisibly.

See Also

[nmfkc.consensus](#)

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](#), [nmf.rrr.DOT](#), [nmf.ffb.DOT](#), [nmfkc.ar.DOT](#), [print.nmfkc.DOT](#)

plot.nmfkc.signed	<i>Plot method for nmfkc.signed (convergence)</i>
-------------------	---

Description

Plot method for nmfkc.signed (convergence)

Usage

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

Arguments

`x` An nmfkc.signed object.
`...` Passed to plot().

Value

Invisible `x`.

Lifecycle

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

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

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

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, ..., which = c("full", "reconstruction", "both"))
```

Arguments

<code>x</code>	A fitted model object.
<code>...</code>	Additional graphical arguments passed to plot .
<code>which</code>	For <code>plot.nmf.sem</code> : which objective to plot. One of "full" (default; loss + penalties, the actual monotonically-decreasing quantity that the multiplicative updates minimize), "reconstruction" (Frobenius distance only, $\ Y_1 - XB\ _F^2$), or "both" (overlay both with a legend). "both" is useful for diagnosing whether regularization is actively shaping the solution: if the two curves diverge, the penalties are pulling the optimizer away from the pure least-squares minimum.

Value

Invisible NULL.

See Also

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

Examples

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

plot.nmfre.ecv	<i>Plot method for nmfre.ecv objects</i>
----------------	--

Description

Draws the element-wise CV score (held-out RMSE) against the rank, marking the minimizing rank as “Best (Min)”.

Usage

```
## S3 method for class 'nmfre.ecv'
plot(
  x,
  main = "NMF-RE element-wise CV",
  xlab = "Rank (Q)",
  ylab = "sigma.ecv (held-out RMSE)",
  ...
)
```

Arguments

x	An object of class "nmfre.ecv".
main	Plot title.
xlab, ylab	Axis labels.
...	Passed to plot .

Value

The input object, invisibly.

See Also

[nmfre.ecv](#)

predict.nmfkc	<i>Prediction method for objects of class nmfkc</i>
---------------	---

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 = c("response", "prob", "class"),
  ...
)
```

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)
```

predict.nmfkc.signed *Predict method for nmfkc.signed*

Description

Computes $\hat{Y} = X C A_{\text{new}}$ ($= X(C_+ - C_-)(A_+^{\text{new}} - A_-^{\text{new}})$). For type = "response" the raw prediction is returned (possibly signed). For type = "prob" and "class", negative entries of $\hat{Y}$ are clipped to zero before column normalization, since probabilities must be non-negative.

Usage

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

Arguments

object	A fitted "nmfkc.signed" object.
newA	Real-valued $D \times N_{\text{new}}$ covariate matrix.
type	Output: "response" (raw signed), "prob", or "class".
...	Unused.

Value

A numeric matrix ("response" or "prob") or a character vector ("class").

Lifecycle

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

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

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

predict.nmfre	<i>Predict method for nmfre objects</i>
---------------	---

Description

Predicts the response from a fitted NMF-RE model. With newA supplied, returns the **fixed-effect** prediction $X \Theta A_{new}$ (new units carry no estimated random effect U , so only the population mean is predicted). Without newA, returns the in-sample BLUP fit $X(\Theta A + U)$ (object\$XB.blup), matching fitted().

Usage

```
## S3 method for class 'nmfre'
predict(object, newA = NULL, ...)
```

Arguments

object	An object of class "nmfre" from nmfre .
newA	Optional covariate matrix (K x M) for new units. If NULL (default), the in-sample BLUP fit is returned.
...	Not used.

Value

A matrix of predicted (P x M) values.

See Also

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

print.nmf	<i>Print method for fitted NMF models</i>
-----------	---

Description

Prints the call, the model shape and whether the fit converged, in the spirit of print.lm: enough to see what was fitted and whether to trust it, with everything else left to summary(). Registered on the shared "nmf" class, so it covers every optimizer that returns one ([nmfkc](#), [nmfae](#) / nmf.rrr, [nmfre](#), [nmf.ffb](#) and the signed variants);

Without it, printing a fit dumps the whole list — over 350 lines for a 4×82 problem, and unusable for a large one.

Unlike lm, the coefficient matrix is **not** shown: it is $Q \times K$ with K the number of covariates — the number of samples when there is no covariate matrix — so it is routinely far too large for a print method. Use [coef](#) for it.

Usage

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

Arguments

<code>x</code>	A fitted model object inheriting class "nmf".
<code>digits</code>	Number of significant digits.
<code>...</code>	Additional arguments (currently unused).

Value

`x`, invisibly.

See Also

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

Examples

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

```
print.nmf.cluster.criteria
```

Print method for nmf.cluster.criteria objects

Description

Print method for `nmf.cluster.criteria` objects

Usage

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

Arguments

<code>x</code>	An object of class "nmf.cluster.criteria".
<code>...</code>	Passed to the criteria table's print.

Value

`x`, invisibly.

```
print.nmf.cluster.flow
```

Print method for nmf.cluster.flow objects

Description

Prints a one-line header, the adjacent-result ARI, and the $N \times R$ cluster table (rows = individuals, columns = results, entries = cluster number). Use [plot](#) for the diagram.

Usage

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

Arguments

x	An object of class "nmf.cluster.flow".
...	Passed to the table's print.

Value

x, invisibly.

```
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](#), [nmf.rrr.inference](#)

print.nmf.rank	<i>Print method for rank-selection (nmf.rank) objects</i>
----------------	---

Description

Print method for rank-selection (nmf.rank) objects

Usage

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

Arguments

x	An object of class "nmf.rank".
...	Passed to the criteria table's print.

Value

x, invisibly.

print.nmfæ.ecv	<i>Print method for nmf.rrr element-wise CV results</i>
----------------	---

Description

A two-line summary, matching print.nmfkc.ecv. Without it the object printed as a raw list (33 lines of \$objfunc / \$folds).

Usage

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

Arguments

x	An object of class "nmfæ.ecv" (or "nmfæ.signed.ecv").
...	Ignored.

Value

x, invisibly.

See Also

[nmf.rrr.ecv](#), [plot.nmfæ.ecv](#)

print.nmfkc.ar.latent *Print method for nmfkc.ar.latent objects*

Description

Print method for nmfkc.ar.latent objects

Usage

```
## S3 method for class 'nmfkc.ar.latent'  
print(x, digits = 3, ...)
```

Arguments

x	An object of class "nmfkc.ar.latent".
digits	Number of digits used when printing the matrices.
...	Ignored.

Value

x, invisibly.

print.nmfkc.ar.latent.inference
Print method for nmfkc.ar.latent.inference objects

Description

Print method for nmfkc.ar.latent.inference objects

Usage

```
## S3 method for class 'nmfkc.ar.latent.inference'  
print(x, digits = 4, ...)
```

Arguments

x	An object of class "nmfkc.ar.latent.inference".
digits	Number of digits used when printing.
...	Ignored.

Value

x, invisibly.

<code>print.nmfkc.ar.stationarity</code>	<i>Print method for nmfkc.ar.stationarity objects</i>
--	---

Description

Print method for nmfkc.ar.stationarity objects

Usage

```
## S3 method for class 'nmfkc.ar.stationarity'  
print(x, digits = 3, ...)
```

Arguments

<code>x</code>	An object of class "nmfkc.ar.stationarity".
<code>digits</code>	Ignored; kept for compatibility with the print generic.
<code>...</code>	Ignored.

Value

`x`, invisibly.

<code>print.nmfkc.ard</code>	<i>Print method for nmfkc.ard objects</i>
------------------------------	---

Description

Print method for nmfkc.ard objects

Usage

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

Arguments

<code>x</code>	An object of class "nmfkc.ard".
<code>...</code>	Unused.

Value

`x`, invisibly.

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

Description

Print method for nmfkc.consensus objects

Usage

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

Arguments

x	An object of class "nmfkc.consensus".
...	Unused.

Value

x, invisibly.

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

Description

Writes the Graphviz source out as text. Without it, printing a DOT object fell to print.default, which showed the whole graph as one escaped string ([1] "digraph NMF \{\n ...") — unreadable. Shared by every *.DOT function through the common "nmfkc.DOT" class.

Usage

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

Arguments

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

Value

x, invisibly.

See Also

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

print.nmfre.ecv	<i>Print method for nmfre.ecv objects</i>
-----------------	---

Description

Print method for nmfre.ecv objects

Usage

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

Arguments

x	An object of class "nmfre.ecv".
...	Not used.

Value

The input object, invisibly.

print.summary.nmf.sem	<i>Print method for summary.nmf.sem objects</i>
-----------------------	---

Description

Prints the NMF-FFB model summary (dimensions, convergence, stability diagnostics, fit statistics, and inference results if available).

Usage

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

Arguments

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

Value

Invisible x.

See Also

[summary.nmf.sem](#)

```
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

<code>x</code>	An object of class <code>summary.nmfkc</code> .
<code>digits</code>	Minimum number of significant digits to be used.
<code>...</code>	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,
  by = c("covariate", "basis"),
  ...
)
```

Arguments

<code>x</code>	An object of class "summary.nmfkc.inference".
<code>digits</code>	Minimum number of significant digits.
<code>max.coef</code>	Maximum coefficient rows to display. Default is 20.
<code>by</code>	Character; grouping order of the coefficients table. "covariate" (default) lists all bases within each covariate (1-1, 1-2, ...); "basis" lists all covariates within each basis (1-1, 2-1, ...).
<code>...</code>	Additional arguments (currently unused).

Value

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

See Also

[summary.nmfkc.inference](#)

```
print.summary.nmfkc.net
```

Print method for summary.nmfkc.net objects

Description

Print method for summary.nmfkc.net objects

Usage

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

Arguments

<code>x</code>	An object of class "summary.nmfkc.net".
<code>digits</code>	Minimum number of significant digits.
<code>...</code>	Unused.

Value

Invisible `x`.

See Also

[summary.nmfkc.net](#)

```
print.summary.nmfkc.net.inference
```

Print method for summary.nmfkc.net.inference objects

Description

Prints a formatted summary of a symmetric NMF model with inference results. The coefficients table uses `Basis.row` and `Basis.col` labels reflecting the symmetric structure C_{qr} .

Usage

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

Arguments

<code>x</code>	An object of class "summary.nmfkc.net.inference".
<code>digits</code>	Minimum number of significant digits.
<code>max.coef</code>	Largest number of coefficient rows to print (default 20). The table has one row per basis pair, so it grows as Q^2 ; beyond the cap the significant rows are preferred and the omission is reported.
<code>by</code>	Character; grouping order of the coefficients table. Because the symmetric model sets $A = X^\top$, the column factor <code>Basis.col</code> plays the covariate role: "covariate" (default) lists all <code>Basis.row</code> within each <code>Basis.col</code> (1-1, 2-1, ...), while "basis" lists all <code>Basis.col</code> within each <code>Basis.row</code> (1-1, 1-2, ...).
<code>...</code>	Additional arguments (currently unused).

Value

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

See Also

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

```
print.summary.nmfkc.net.signed
```

Print method for summary.nmfkc.net.signed objects

Description

Print method for summary.nmfkc.net.signed objects

Usage

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

Arguments

x	An object of class "summary.nmfkc.net.signed".
digits	Minimum number of significant digits.
...	Unused.

Value

Invisible x.

See Also

[summary.nmfkc.net.signed](#)

```
print.summary.nmfkc.signed
```

Print method for summary.nmfkc.signed

Description

Print method for summary.nmfkc.signed

Usage

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

Arguments

x	Object of class "summary.nmfkc.signed".
digits	Number of significant digits.
...	Unused.

Value

Invisible x.

Lifecycle

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

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

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

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

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

Value

The residual matrix.

See Also

[nmfkc](#), [nmf.rrr](#), [nmfre](#), [nmf.ffb](#), [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-FFB 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.ffb" (or legacy "nmf.sem") returned by nmf.ffb / nmf.sem .
...	Not used.

Value

An object of class "summary.nmf.sem" (the fitted model tagged for printing); printed by [print.summary.nmf.sem](#).

See Also

[nmf.ffb](#), [nmf.ffb.inference](#)

Examples

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

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

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	<i>Summary method for nmfkc.inference objects</i>
-------------------------	---

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

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

Value

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

See Also

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

summary.nmfkc.net	<i>Summary method for nmfkc.net objects</i>
-------------------	---

Description

Summary method for nmfkc.net objects

Usage

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

Arguments

object	An nmfkc.net object.
...	Unused.

Value

An object of class "summary.nmfkc.net".

See Also

[nmfkc.net](#)

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

Description

Produces a summary of a symmetric NMF model with inference results, including the coefficients table for C .

Usage

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

Arguments

<code>object</code>	An object of class "nmfkc.net.inference".
<code>...</code>	Additional arguments passed to <code>summary.nmfkc</code> .

Value

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

See Also

[nmfkc.net.inference](#)

`summary.nmfkc.net.signed`*Summary method for nmfkc.net.signed objects*

Description

Summary method for nmfkc.net.signed objects

Usage

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

Arguments

<code>object</code>	An nmfkc.net.signed object.
<code>...</code>	Unused.

Value

An object of class "summary.nmfkc.net.signed".

See Also

[nmfkc.net](#)

summary.nmfkc.signed	<i>Summary method for nmfkc.signed</i>
----------------------	--

Description

Summary method for nmfkc.signed

Usage

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

Arguments

object	An nmfkc.signed object.
...	Unused.

Value

An object of class "summary.nmfkc.signed".

Lifecycle

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

References

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1), 45–55.

See Also

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

summary.nmfre

*Summary method for objects of class nmfre***Description**

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

Usage

```
## S3 method for class 'nmfre'
summary(
  object,
  ci.show = FALSE,
  max.coef = 20,
  by = c("covariate", "basis"),
  ...
)

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

Arguments

<code>object</code>	An object of class <code>nmfre</code> , returned by nmfre .
<code>ci.show</code>	Logical. If TRUE, show confidence interval columns (default FALSE). Named object-first to match the package style (<code>C.signed</code> , <code>X.init</code> , ...). Legacy <code>show_ci</code> is accepted via ...
<code>max.coef</code>	Largest number of coefficient rows to print (default 20). The table has one row per (basis, covariate) pair, so it grows as $Q \times K$; beyond the cap the significant rows are preferred and the omission is reported.
<code>by</code>	Grouping order of the coefficient rows: "covariate" (default; list all bases for each covariate) or "basis" (list all covariates for each basis).
<code>...</code>	Additional arguments (currently unused).
<code>x</code>	An object of class "summary.nmfre".

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

barplot, [116](#)

coef, [123](#)

coef.nmf, [4](#)

fitted.nmf, [5](#), [123](#), [136](#)

fitted.nmfae(fitted.nmf), [5](#)

fitted.nmfre(fitted.nmf), [5](#)

image, [117](#)

kmeans, [34](#)

lm, [21](#), [35](#), [44](#), [83](#), [95](#)

n2mfrow, [117](#)

nmf.cluster.criteria, [6](#), [8](#), [46](#), [65](#), [90](#), [109](#)

nmf.cluster.flow, [6](#), [7](#), [7](#), [110](#)

nmf.ffb, [6](#), [8](#), [13](#), [15](#), [18](#), [19](#), [119](#), [123](#), [136](#)

nmf.ffb.cv, [11](#), [12](#)

nmf.ffb.DOT, [11](#), [13](#), [18](#), [72](#), [118](#)

nmf.ffb.inference, [5](#), [11](#), [15](#), [15](#), [136](#)

nmf.ffb.split, [11](#), [18](#), [19](#)

nmf.rrr, [6](#), [20](#), [24](#), [25](#), [27–29](#), [32](#), [33](#), [36](#), [112](#), [136](#)

nmf.rrr.cv, [23](#), [31](#)

nmf.rrr.DOT, [22](#), [24](#), [28](#), [72](#), [118](#)

nmf.rrr.ecv, [22](#), [24](#), [26](#), [32](#), [38](#), [126](#)

nmf.rrr.heatmap, [27](#), [39](#), [112](#)

nmf.rrr.inference, [5](#), [22](#), [28](#), [40](#), [125](#)

nmf.rrr.kernel.beta.cv, [24](#), [30](#)

nmf.rrr.rank, [31](#)

nmf.rrr.rename, [32](#)

nmf.rrr.signed, [33](#), [38–42](#)

nmf.rrr.signed.ecv, [37](#), [41](#)

nmf.rrr.signed.heatmap, [38](#)

nmf.rrr.signed.inference, [39](#)

nmf.rrr.signed.rank, [40](#)

nmf.rrr.signed.rename, [36](#), [42](#)

nmf.sem, [6](#), [13](#), [15](#), [16](#), [136](#)

nmf.sem.DOT, [17](#), [117](#)

nmf.sem.inference, [14](#)

nmfae, [6](#), [10](#), [23](#), [25](#), [26](#), [28](#), [29](#), [31–34](#), [112](#), [123](#)

nmfae.cv, [30](#)

nmfae.DOT, [117](#)

nmfae.ecv, [31](#), [32](#), [37](#)

nmfae.inference, [4](#), [39](#)

nmfae.signed, [37](#), [38](#), [40](#), [41](#)

nmfae.signed.ecv, [35](#), [40](#), [41](#)

nmfkc, [6](#), [9](#), [10](#), [20–22](#), [43](#), [48](#), [51](#), [53](#), [56](#), [58](#), [62–68](#), [72–75](#), [91–94](#), [96](#), [99](#), [101–103](#), [113](#), [119](#), [121–124](#), [135–137](#), [140](#)

nmfkc.ar, [47](#), [47](#), [49](#), [51](#), [52](#), [54](#), [57–60](#)

nmfkc.ar.degree.cv, [48](#), [49](#), [68](#)

nmfkc.ar.DOT, [48](#), [50](#), [54](#), [72](#), [114](#), [117](#), [118](#)

nmfkc.ar.latent, [51](#), [54](#), [56](#), [57](#), [59](#), [60](#), [113](#), [114](#)

nmfkc.ar.latent.inference, [52–54](#), [54](#), [59](#), [60](#), [114](#)

nmfkc.ar.predict, [58](#)

nmfkc.ar.stationarity, [48](#), [52](#), [54](#), [57](#), [59](#), [115](#)

nmfkc.ard, [60](#), [63](#), [65](#), [73](#), [92](#)

nmfkc.bicv, [60](#), [61](#), [62](#), [64](#), [65](#), [68](#), [73](#), [92](#)

nmfkc.class, [63](#)

nmfkc.consensus, [60](#), [61](#), [63](#), [64](#), [73](#), [92](#), [116](#), [117](#)

nmfkc.criterion, [66](#)

nmfkc.cv, [24](#), [47](#), [49](#), [67](#), [68](#), [73](#), [76](#), [97](#), [121](#)

nmfkc.cv.methods, [68](#)

nmfkc.denormalize, [69](#), [90](#)

nmfkc.DOT, [47](#), [70](#), [86](#), [105](#), [107](#), [108](#), [117](#), [118](#), [129](#)

nmfkc.ecv, [27](#), [47](#), [60–65](#), [68](#), [72](#), [90–92](#), [106](#), [107](#)

nmfkc.inference, [4](#), [5](#), [47](#), [55](#), [57](#), [74](#), [87](#), [88](#), [125](#), [137](#), [138](#)

nmfkc.kernel, [23](#), [30](#), [31](#), [47](#), [76](#), [78](#), [81](#)

- nmfkc.kernel.beta.cv, [31](#), [68](#), [77](#), [80](#), [81](#)
- nmfkc.kernel.beta.nearest.med, [30](#), [78](#), [78](#), [81](#), [99](#), [100](#)
- nmfkc.kernel.gaussian, [76](#), [78](#), [80](#), [80](#)
- nmfkc.net, [6](#), [10](#), [44](#), [82](#), [85–89](#), [138](#), [140](#)
- nmfkc.net.DOT, [84](#), [84](#), [88](#)
- nmfkc.net.ecv, [72](#), [83](#), [84](#), [86](#), [89](#)
- nmfkc.net.inference, [85](#), [86](#), [87](#), [139](#)
- nmfkc.net.rank, [88](#)
- nmfkc.normalize, [70](#), [89](#)
- nmfkc.rank, [7](#), [8](#), [31](#), [32](#), [40](#), [41](#), [46](#), [47](#), [61](#), [63](#), [65](#), [66](#), [73](#), [88](#), [89](#), [90](#), [98](#), [99](#), [111](#)
- nmfkc.residual.plot, [92](#)
- nmfkc.signed, [6](#), [82](#), [93](#), [97–100](#), [119](#), [122](#), [135](#), [140](#)
- nmfkc.signed.cv, [95](#), [96](#), [98](#), [119](#), [122](#), [135](#), [140](#)
- nmfkc.signed.ecv, [95](#), [97](#), [97](#)
- nmfkc.signed.rank, [98](#)
- nmfkc.signed.rff, [99](#)
- nmfre, [6](#), [101](#), [106–108](#), [119](#), [123](#), [136](#), [141](#)
- nmfre.ecv, [106](#), [120](#)
- nmfre.inference, [4](#), [5](#), [101](#), [105](#), [107](#), [107](#), [141](#)
- plot, [8](#), [69](#), [109–111](#), [113](#), [117](#), [119](#), [120](#), [125](#)
- plot.nmf.cluster.criteria, [7](#), [109](#)
- plot.nmf.cluster.flow, [8](#), [110](#)
- plot.nmf.rank, [111](#)
- plot.nmf.sem (plot.nmfre), [119](#)
- plot.nmfae, [28](#), [112](#)
- plot.nmfae.ecv, [126](#)
- plot.nmfkc, [47](#), [112](#), [137](#)
- plot.nmfkc.ar.latent, [52](#), [54](#), [113](#), [115](#)
- plot.nmfkc.ar.stationarity, [114](#), [115](#)
- plot.nmfkc.ard, [116](#)
- plot.nmfkc.bicv (nmfkc.cv.methods), [68](#)
- plot.nmfkc.consensus, [65](#), [116](#)
- plot.nmfkc.cv (nmfkc.cv.methods), [68](#)
- plot.nmfkc.DOT, [15](#), [51](#), [72](#), [86](#), [117](#), [129](#)
- plot.nmfkc.ecv (nmfkc.cv.methods), [68](#)
- plot.nmfkc.signed, [118](#)
- plot.nmfre, [104](#), [119](#)
- plot.nmfre.ecv, [120](#)
- predict.nmfae, [22](#)
- predict.nmfae.signed, [36](#)
- predict.nmfkc, [47](#), [120](#)
- predict.nmfkc.signed, [96](#), [122](#)
- predict.nmfre, [123](#)
- print.nmf, [123](#)
- print.nmf.cluster.criteria, [124](#)
- print.nmf.cluster.flow, [125](#)
- print.nmf.inference, [125](#)
- print.nmf.rank, [126](#)
- print.nmfae.ecv, [126](#)
- print.nmfkc.ar.latent, [127](#)
- print.nmfkc.ar.latent.inference, [127](#)
- print.nmfkc.ar.stationarity, [128](#)
- print.nmfkc.ard, [128](#)
- print.nmfkc.bicv (nmfkc.cv.methods), [68](#)
- print.nmfkc.consensus, [65](#), [129](#)
- print.nmfkc.cv (nmfkc.cv.methods), [68](#)
- print.nmfkc.DOT, [118](#), [129](#)
- print.nmfkc.ecv (nmfkc.cv.methods), [68](#)
- print.nmfre.ecv, [130](#)
- print.summary.nmf.sem, [130](#), [136](#)
- print.summary.nmfkc, [131](#)
- print.summary.nmfkc.inference, [131](#)
- print.summary.nmfkc.net, [132](#)
- print.summary.nmfkc.net.inference, [133](#)
- print.summary.nmfkc.net.signed, [134](#)
- print.summary.nmfkc.signed, [134](#)
- print.summary.nmfre (summary.nmfre), [141](#)
- residuals.nmf, [6](#), [93](#), [135](#)
- residuals.nmfae (residuals.nmf), [135](#)
- residuals.nmfre (residuals.nmf), [135](#)
- summary.nmf.sem, [11](#), [130](#), [136](#)
- summary.nmfae.inference, [29](#)
- summary.nmfae.signed, [36](#)
- summary.nmfkc, [46](#), [47](#), [113](#), [124](#), [137](#), [138](#)
- summary.nmfkc.inference, [75](#), [132](#), [137](#)
- summary.nmfkc.net, [132](#), [138](#)
- summary.nmfkc.net.inference, [88](#), [133](#), [139](#)
- summary.nmfkc.net.signed, [134](#), [139](#)
- summary.nmfkc.signed, [140](#)
- summary.nmfre, [105](#), [108](#), [141](#)
- warning, [62](#)