Package: nmfkc (via r-universe)

May 13, 2026

Type Package

Title Non-Negative Matrix Factorization with Kernel Covariates

Version 0.7.3

Date 2026-05-14

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>; Satoh (2025) <doi:10.48550/arXiv.2510.10375>; Satoh (2025) <doi:10.48550/arXiv.2512.18250>; Satoh (2026) <doi:10.48550/arXiv.2603.01468>; Satoh (2026) <doi:10.1007/s42081-025-00314-0>.

License MIT + file LICENSE

Imports stats, graphics, utils, grDevices

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

ByteCompile true

VignetteBuilder knitr

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

Config/testthat/edition 3

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

Date/Publication 2026-05-13 22:54:06 UTC

RemoteUrl https://github.com/ksatohds/nmfkc

RemoteRef HEAD

RemoteSha c6803b3d31b830e78b2898b15bd7f63dd1fe9ed4

Contents

coef.nmf . . . 4
fitted.nmf . . . 5
nmf.ffb . . . 6
nmf.ffb.cv . . . 9
nmf.ffb.DOT . . . 10
nmf.ffb.inference . . . 12
nmf.ffb.split . . . 15
nmfae . . . 17
nmfae.cv . . . 19
nmfae.DOT . . . 20
nmfae.ecv . . . 22
nmfae.heatmap . . . 24
nmfae.inference . . . 25
nmfae.kernel.beta.cv . . . 26
nmfae.rename . . . 27
nmfae.signed . . . 28
nmfae.signed.ecv . . . 31
nmfae.signed.heatmap . . . 32
nmfae.signed.inference . . . 33
nmfae.signed.rename . . . 35
nmfkc . . . 36
nmfkc.ar . . . 40
nmfkc.ar.degree.cv . . . 41
nmfkc.ar.DOT . . . 42
nmfkc.ar.predict . . . 44
nmfkc.ar.stationarity . . . 45
nmfkc.class . . . 46
nmfkc.criterion . . . 46
nmfkc.cv . . . 47
nmfkc.denormalize . . . 49
nmfkc.DOT . . . 50
nmfkc.ecv . . . 52
nmfkc.inference . . . 53

nmfkc.kernel . . . 54
nmfkc.kernel.beta.cv . . . 55
nmfkc.kernel.beta.nearest.med . . . 56
nmfkc.kernel.gaussian . . . 59
nmfkc.net . . . 60
nmfkc.net.DOT . . . 62
nmfkc.net.ecv . . . 64
nmfkc.net.inference . . . 65
nmfkc.normalize . . . 66
nmfkc.rank . . . 67
nmfkc.residual.plot . . . 68
nmfkc.signed . . . 69
nmfkc.signed.cv . . . 72
nmfkc.signed.ecv . . . 73
nmfre . . . 74
nmfre.dfU.scan . . . 78
nmfre.inference . . . 80
plot.nmfae . . . 82
plot.nmfae.cv . . . 82
plot.nmfae.ecv . . . 83
plot.nmfae.kernel.beta.cv . . . 84
plot.nmfae.signed . . . 84
plot.nmfkc . . . 85
plot.nmfkc.DOT . . . 86
plot.nmfkc.signed . . . 86
plot.nmfre . . . 87
plot.predict.nmfae . . . 88
predict.nmfae . . . 89
predict.nmfae.signed . . . 90
predict.nmfkc . . . 91
predict.nmfkc.signed . . . 92
print.nmf.inference . . . 92
print.summary.nmfae . . . 93
print.summary.nmfae.inference . . . 94
print.summary.nmfae.signed . . . 94
print.summary.nmfae.signed.inference . . . 95
print.summary.nmfkc . . . 96
print.summary.nmfkc.inference . . . 96
print.summary.nmfkc.net . . . 97
print.summary.nmfkc.net.inference . . . 98
print.summary.nmfkc.net.signed . . . 98
print.summary.nmfkc.signed . . . 99
residuals.nmf . . . 100
summary.nmf.sem . . . 101
summary.nmfae . . . 101
summary.nmfae.inference . . . 102
summary.nmfae.signed . . . 103
summary.nmfae.signed.inference . . . 104

summary.nmfkc . . . 105
summary.nmfkc.inference . . . 105
summary.nmfkc.net . . . 106
summary.nmfkc.net.inference . . . 107
summary.nmfkc.net.signed . . . 107
summary.nmfkc.signed . . . 108
summary.nmfre . . . 109

Index **110**

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

Description

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

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

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

Usage

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

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

Arguments

`object`	A fitted model object of class `"nmf"`, `"nmfkc"`, `"nmfae"`, `"nmfre"`, or `"nmf.sem"`.
`...`	Not used.

Value

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

See Also

`nmfkc.inference`, `nmfae.inference`, `nmfre.inference`, `nmf.sem.inference`

Examples

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

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

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

Description

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

For `nmf.sem` objects, returns the equilibrium prediction $\hat{Y}_1 = M_{model} Y_2$ if available. Supply `Y1` and `Y2` to get the direct reconstruction $X(C_1 Y_1 + C_2 Y_2)$ instead.

Usage

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

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

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

Arguments

`object`	A fitted model object of class `"nmf"`, `"nmfkc"`, `"nmfae"`, `"nmfre"`, or `"nmf.sem"`.
`...`	For `nmf.sem`: optionally `Y1` and `Y2`.

Value

The fitted matrix XB.

See Also

`nmfkc`, `nmfae`, `nmfre`, `nmf.sem`, `residuals.nmf`

Examples

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

`nmf.ffb` *NMF-FFB Main Estimation Algorithm (formerly NMF-SEM)*

Description

Fits the NMF-FFB model

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

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

At equilibrium, the model can be written as

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

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

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

Usage

```
nmf.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,
  ...
)

nmf.sem(
  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 `ncol(Y1) == ncol(Y2)`.
`rank`	Integer; number of latent factors Q. If `NULL`, Q is taken from a hidden argument in `...` or defaults to `nrow(Y2)`.
`X.init`	Initialization strategy for the basis matrix `X` ($P_1 \times Q$). One of: • `"nndsvd"` (default): Non-negative Double SVD with additive randomness (NNDSVDar; Boutsidis & Gallopoulos 2008), computed internally via `.nndsvdar(Y1, Q)`. 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 `nmfkc`. • `"kmeansar"`: `"kmeans"` followed by filling zero entries of X with $\mathrm{Uniform}(0, \bar{Y}_1/100)$ (NNDSVDar-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 `colSums(X) = 1` before iteration. The menu mirrors `nmfkc`'s `X.init` option for consistency across the package.
`X.L2.ortho`	L2 orthogonality penalty for `X`. This controls the penalty term $\lambda_X \|X^\top X - \mathrm{diag}(X^\top X)\|_F^2$. Default: `100`.
`C1.L1`	L1 sparsity penalty for `C1` (i.e., Θ_1). Default: `1.0`.
`C2.L1`	L1 sparsity penalty for `C2` (i.e., Θ_2). Default: `0.1`.
`epsilon`	Relative convergence threshold for the objective function. Iterations stop when the relative change in reconstruction loss falls below this value. Default: `1e-6`.
`maxit`	Maximum number of iterations for the multiplicative updates. Default: `5000` (matches `nmfkc` and other MU functions in the package).
`seed`	Random seed used to initialize `X`, `C1`, and `C2`. Default: `123`.
`...`	Additional hidden arguments controlling the optional feedforward baseline (used both as an X warm-start and as the reference for `SC.map`, the input-output structural fidelity defined in Satoh (2025) §4.SC.map): `nmfkc.baseline` Controls whether a feedforward `nmfkc`(Y1, A = Y2) fit is used as baseline. Possible values: • Default (not given) — `nmf.sem` runs `nmfkc` **internally** when `X.init` is a string method (`"nndsvd"`, `"kmeans"`, `...`) or `NULL`, forwarding `X.init`, `X.L2.ortho`, `epsilon`, `maxit`, `seed`. The fitted X of the baseline is then used as warm-start for the nmf.sem MU iterations, and `SC.map` is computed. This means `nmf.sem(Y1, Y2, rank = Q)` runs end-to-end without a prior `nmfkc` call.

- TRUE — same as above, but force the internal nmfkc call even when X.init is a user-supplied matrix (the matrix is overridden).
- FALSE — opt out; no internal call, SC.map = NA (pre-v0.6.8 behavior).
- An nmfkc result (list with $X and $C) — use as the baseline directly (no internal call); also adopted as X.init when the latter is a string / NULL.

M.simple Optional $P_1 \times P_2$ pre-computed baseline mapping. Takes precedence over nmfkc.baseline for the SC.map calculation but does not affect warm-start.

Q Backward-compat alias for rank.

Value

A list with components:

X	Estimated basis matrix ($P_1 \times Q$).
C1	Estimated latent feedback matrix (Θ_1, $Q \times P_1$).
C2	Estimated exogenous loading matrix (Θ_2, $Q \times P_2$).
XC1	Feedback matrix $X\Theta_1$.
XC2	Direct-effect matrix $X\Theta_2$.
XC1.radius	Spectral radius $\rho(X\Theta_1)$.
XC1.norm1	Induced 1-norm $\|X\Theta_1\|_{1,\mathrm{op}}$.
Leontief.inv	Leontief-type inverse $(I - X\Theta_1)^{-1}$.
M.model	Equilibrium mapping $M_{\mathrm{model}} = (I - X\Theta_1)^{-1}X\Theta_2$.
amplification	Latent amplification factor $\|M_{\mathrm{model}}\|_{1,\mathrm{op}} / \|X\Theta_2\|_{1,\mathrm{op}}$.
amplification.bound	Geometric-series upper bound $1/(1-\|X\Theta_1\|_{1,\mathrm{op}})$ if $\|X\Theta_1\|_{1,\mathrm{op}} < 1$, otherwise Inf.
Q	Effective latent dimension used in the fit.
SC.cov	Correlation between sample and model-implied covariance (flattened) of Y_1. See *second-moment fidelity* in Satoh (2025).
SC.map	Correlation between the equilibrium operator M_{model} and a feedforward baseline mapping $M_{\mathrm{simple}} = X_0\Theta_0$, computed only when the baseline is supplied via M.simple or nmfkc.baseline in ...; otherwise NA. See *input-output structural fidelity* in Satoh (2025).
MAE	Mean absolute error between Y_1 and its equilibrium prediction $\hat{Y}_1 = M_{\mathrm{model}}Y_2$.
objfunc	Vector of reconstruction losses per iteration.
objfunc.full	Vector of penalized objective values per iteration.
iter	Number of iterations actually performed.

References

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

See Also

nmf.sem.inference, nmf.sem.cv, nmf.sem.split, nmf.sem.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.sem(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(...)

nmf.sem.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

`...`	Additional arguments passed to `nmf.sem` (except for `rank`, `seed`, `div`, `shuffle`, which are handled here). Also accepts: `nfolds` (number of folds, default 5; `div` also accepted), `seed` (master random seed, default `NULL`), `shuffle` (logical, default `TRUE`).
`Y1`	A non-negative numeric matrix of endogenous variables with **rows = variables (P1), columns = samples (N)**.
`Y2`	A non-negative numeric matrix of exogenous variables with **rows = variables (P2), columns = samples (N)**. Must satisfy `ncol(Y1) == ncol(Y2)`.
`rank`	Integer; rank (number of latent factors) passed to `nmf.sem`. If `NULL`, `nmf.sem` decides the effective rank (via `...` or `nrow(Y2)`).
`X.init`	Initialization strategy for `X`, forwarded to `nmf.sem`. One of `"nndsvd"` (default), `"kmeans"`, `"kmeansar"`, `"runif"`, a numeric $P_1 \times Q$ matrix, or `NULL` (alias for `"nndsvd"`). See `nmf.sem` for details.
`X.L2.ortho`	L2 orthogonality penalty for `X`.
`C1.L1`	L1 sparsity penalty for `C1` (Θ_1).
`C2.L1`	L1 sparsity penalty for `C2` (Θ_2).
`epsilon`	Convergence threshold for `nmf.sem`.
`maxit`	Maximum number of iterations for `nmf.sem`.

Value

A numeric scalar: mean MAE across CV folds.

See Also

`nmf.sem`

Examples

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

`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 ($F_1, \ldots, F_Q$),

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

Directed edges represent estimated relationships:

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

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

Usage

```
nmf.ffb.DOT(result, ...)

nmf.sem.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

`result`	A list returned by `nmf.sem`, containing matrices `X`, `C1`, and `C2`.
`...`	For backward compatibility: accepts deprecated names `weight_scale_y2f` (use `weight_scale_c2`) and `weight_scale_fy1` (use `weight_scale_x1`).
`weight_scale`	Base scaling factor for edge widths.
`weight_scale_c2`	Scaling factor for edges $Y_2 \to F_q$ (C2 matrix). Defaults to `weight_scale`.

`weight_scale_x1`
: Scaling factor for edges $F_q \to Y_1$ (X matrix). Defaults to `weight_scale`.

`weight_scale_feedback`
: Scaling factor for feedback edges $Y_1 \to F_q$ (C1 matrix). Defaults to `weight_scale`.

`threshold`
: Minimum coefficient value needed for an edge to be drawn.

`sig.level`
: Significance level for filtering structural edges (C_1 feedback and C_2 exogenous loadings) when inference results are present. If `result` contains a `coefficients` data frame from `nmf.sem.inference`, only edges with `p_value` < `sig.level` are drawn, with significance stars (`*` `**` `***`) 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 `NULL` to disable significance filtering and fall back to the `threshold` magnitude filter for both C_1 and C_2. Default is `0.1`.

`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.

Value

A character string representing a valid Graphviz DOT script.

See Also

`nmf.sem`, `nmf.sem.inference`, `plot.nmfkc.DOT`

Examples

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

`nmf.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 $b = 1, \ldots, B$, resample column indices $(i_1, \ldots, i_N)$ with replacement from $\{1, \ldots, N\}$ 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`):

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

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 $p = 1 - \sup$: * (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 $\sup = 1 - p$.

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,
  ...
)

nmf.sem.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 `nmf.sem`. Must contain `X`, `C1`, `C2`.
`Y1`	Endogenous variable matrix (P1 x N). Must match the data used in `nmf.sem()`.
`Y2`	Exogenous variable matrix (P2 x N). Same.
`B`	Number of bootstrap replicates. Default `1000`; required for the `***` threshold (sup > 0.999). Reduce to 500 for exploratory speed (only `*` / `**` stay reliable).
`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 `nmf.sem` fit. These must match the fit's hyperparameters for the bootstrap to estimate the correct model. Defaults (`1.0`, `0.1`) match `nmf.sem`'s defaults but you should pass the actual values used.
`seed`	Base RNG seed for the bootstrap. Each replicate uses `seed + b` (resampling) and `seed + 1000 + b` (C_1, C_2 initialization). Default `123`.
`...`	Hidden options: `epsilon` Convergence tolerance for the inner fixed-X MU loop. Default `1e-6`. `maxit` Maximum iterations for the inner MU loop. Default `5000`. `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` ($= 1 - \text{support\_rate}$, 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.sem`, `nmf.sem.DOT`

Examples

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

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

Arguments

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

Value

A list with:

endogenous.variables
: Character vector of variables selected as endogenous (Y_1).

exogenous.variables
: Character vector of variables selected as exogenous (Y_2).

ordered.variables
: Variables in inferred causal order (from exogenous to endogenous).

is.flipped
: Logical vector indicating which variables were sign-flipped during processing.

n.exogenous
: Integer giving the number of exogenous variables.

See Also

nmf.sem

Examples

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

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

Description

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

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

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

Usage

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

Arguments

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

`verbose` Logical. If `TRUE`, prints progress messages during fitting. Default is `FALSE`.

`...` Additional arguments:

`Y1.weights` Optional non-negative weight matrix (P1 x 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.

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

Value

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

`X1` Decoder basis matrix (P1 x Q), column sum 1.

`C` Parameter matrix (Q x R).

`X2` Encoder basis matrix (R x P2), row sum 1.

`Y1hat` Fitted values $X_1 \Theta X_2 Y_2$ (P1 x N).

`rank` Named integer vector `c(Q, R)`.

`objfunc` Final objective value.

`objfunc.iter` Objective values by iteration.

`r.squared` Coefficient of determination R^2.

`niter` Number of iterations performed.

`runtime` Elapsed time as a `difftime` object.

`n.missing` Number of missing (or zero-weighted) elements in Y_1.

`n.total` Total number of elements in Y_1 (P1 x N).

Lifecycle

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

Source

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

References

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

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

See Also

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

Examples

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

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

---

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

Description

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

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

Usage

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

Arguments

Y1	Output matrix Y_1 (P1 x N). Non-negative.
Y2	Input matrix Y_2 (P2 x N), or a kernel matrix (N x N). Default is Y1 (autoencoder).
rank	Integer. Rank of the decoder basis. Default is 2.
rank.encoder	Integer. Rank of the encoder basis. Default is rank.

`...`	Additional arguments passed to `nmfae` (e.g., `epsilon`, `maxit`, `Y1.weights`). Also accepts: `nfolds` (number of folds, default 5; `div` also accepted), `seed` (integer seed, default 123), `shuffle` (logical, default `TRUE`). For backward compatibility, `Q`, `R` are accepted as aliases for `rank`, `rank.encoder`.

Value

A list with components:

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

Lifecycle

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

See Also

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

Examples

```
Y <- t(iris[1:30, 1:4])
res <- nmfae.cv(Y, rank = 2, rank.encoder = 2, nfolds = 5, maxit = 500)
res$sigma
```

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

Description

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

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

Usage

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

Arguments

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

`Y1.title`	Character. Title for output node group. Default is `"Output (Y1)"`.
`X1.title`	Character. Title for decoder node group. Default is `"Decoder (X1)"`.
`X2.title`	Character. Title for encoder node group. Default is `"Encoder (X2)"`.
`Y2.title`	Character. Title for input node group. Default is `"Input (Y2)"`.
`hide.isolated`	Logical. If `TRUE` (default), Y1 and Y2 nodes that have no edges at or above `threshold` are excluded from the graph. Only applies when `type = "YXCXY"`.

Value

A character string containing the DOT graph specification.

Lifecycle

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

See Also

`nmfae`

Examples

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

---

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

Description

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

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

Usage

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

Arguments

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

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

nmfae, nmfkc.ecv

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

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

Lifecycle

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

See Also

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

Examples

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

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

Description

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

Usage

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

Arguments

`object`	An object of class `"nmfae"` returned by `nmfae`.
`Y1`	Output matrix Y_1 (P1 x N). Must match the data used in `nmfae()`.
`Y2`	Input matrix Y_2 (P2 x N). Default is `Y1` (autoencoder).
`wild.bootstrap`	Logical. If `TRUE` (default), performs wild bootstrap for bootstrap SE and confidence intervals. If `FALSE`, only sandwich SE and z-test p-values are computed (faster).
`...`	Additional arguments: `wild.B` Number of bootstrap replicates. Default is 1000. `wild.seed` Seed for bootstrap. Default is 42. `wild.level` Confidence level for bootstrap CI. Default is 0.95. `sandwich` Logical. Use sandwich covariance. Default is `TRUE`. `C.p.side` P-value type: `"one.sided"` (default) or `"two.sided"`. `cov.ridge` Ridge stabilization for information matrix inversion. Default is 1e-8. `print.trace` Logical. If `TRUE`, prints progress. Default is `FALSE`.

Value

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

`sigma2.used`	Estimated σ^2 used for inference.
`C.se`	Sandwich standard errors for Θ (Q x R matrix).

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

Lifecycle

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

See Also

`nmfae`, `summary.nmfae.inference`

Examples

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

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

Description

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

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

Usage

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

Arguments

Y1	Output matrix Y_1 (P1 x N). Non-negative.
rank	Integer. Rank of the decoder basis. Default is 2.
rank.encoder	Integer. Rank of the encoder basis. Default is rank.
U	Covariate matrix U (K x M). Rows are features, columns are samples (or knot points for non-symmetric kernels).
V	Covariate matrix V (K x N). If NULL (default), V = U and a symmetric kernel is used.
beta	Numeric vector of candidate beta values. If NULL, automatically determined via 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. For backward compatibility, Q and R are accepted as aliases for rank and rank.encoder.

Value

A list with components:

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

See Also

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

Examples

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

---

nmfae.rename *Rename decoder and encoder bases*

Description

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

Usage

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

Arguments

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

Value

A modified copy of x with updated names.

See Also

nmfae

Examples

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

---

nmfae.signed — *Signed-Bottleneck NMF-AE: Three-Layer NMF-AE with Signed Bottleneck*

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 <-> low Abbe number trade-off).

Usage

```
nmfae.signed(
  Y1,
  Y2 = Y1,
  rank = 2,
  rank.encoder = rank,
  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).
rank	Integer. Decoder rank Q. Default 2. Alias Q accepted via ....
rank.encoder	Integer. Encoder rank R. Default rank. Alias R accepted via ....
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 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.

Y1.weights Optional non-negative weight matrix (P1 x 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 decoder/encoder factors. Default "Dec", "Enc".

Value

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

X1	Decoder basis (P1 x Q), column sum 1.
Cp, Cn	Non-negative parts of Θ (each Q x R).
C	Signed bottleneck $\Theta = C_+ - C_-$ (Q x R).
X2	Encoder basis (R x P2), row sum 1.
Y1hat	Fitted values $X_1(C_+ - C_-)X_2Y_2$.
H	Encoding $(C_+ - C_-)X_2Y_2$ (Q x N, signed).
rank	c(Q = Q, R = R).
dims	c(P1, P2, N).
objfunc, objfunc.iter	Final and per-iteration objective values.
r.squared, sigma, mae	Goodness of fit statistics.
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

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

nmfae, predict.nmfae.signed, summary.nmfae.signed, nmfae.signed.rename

Examples

```
set.seed(1)
Y1 <- matrix(abs(rnorm(12)), 3, 4)
Y2 <- matrix(abs(rnorm(20)), 5, 4)
res <- nmfae.signed(Y1, Y2, rank = 2, rank.encoder = 2, maxit = 500)
summary(res)
```

---

nmfae.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

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

Arguments

Y1	Output matrix (P1 x N).
Y2	Input matrix (P2 x N). Default Y1.
rank	Integer vector of candidate Q values. Default 1:2.
rank.encoder	Integer vector of candidate R values, or NULL (default: pair R = Q, diagonal grid).
...	Additional arguments:
	nfolds / div Number of folds. Default 5.
	seed RNG seed for fold assignment. Default 123.
	nstart Number of random restarts per fit. Default 1. Signed models have more local minima (the bottleneck can carry both signs), so nstart >= 10 is recommended for reproducible rank selection.
	Other args epsilon, maxit, warm.start, etc.\ are passed to nmfae.signed.

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

`nmfae.signed`, `nmfae.ecv`

`nmfae.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

```
nmfae.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

nmfae.signed, nmfae.heatmap

Examples

```
set.seed(1)
Y1 <- matrix(abs(rnorm(12)), 3, 4)
Y2 <- matrix(abs(rnorm(20)), 5, 4)
res <- nmfae.signed(Y1, Y2, rank = 2, rank.encoder = 2, maxit = 200)
nmfae.signed.heatmap(res)
```

---

nmfae.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

```
nmfae.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 x 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

`nmfae.signed`, `nmfae.inference`

`nmfae.signed.rename` *Rename Dec/Enc labels on nmfae.signed objects*

Description

Replaces the default `"Dec1"`, `"Dec2"`, `...` (decoder / X1 columns and Cp/Cn/C rows) and `"Enc1"`, `"Enc2"`, `...` (encoder / X2 rows and Cp/Cn/C columns) with user-supplied labels. Propagates to coefficients tables if present (e.g., from `nmfae.inference`).

Usage

```
nmfae.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

`nmfae.signed`

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

Description

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

This function supports two major input modes:

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

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

Usage

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

Arguments

Y	Observation matrix (P x N), OR a formula object for Formula Mode. In Formula Mode, use `Y1 + Y2 ~ A1 + A2` with `data`, or `Y_matrix ~ A_matrix` for direct matrix evaluation. Supports dot notation (`. ~ A1 + A2`) when `data` is supplied.
A	Covariate matrix. Default is `NULL` (no covariates). Ignored when `Y` is a formula.
rank	Integer. The rank of the basis matrix X (Q). Preferred over `Q`.
data	Optional. A data frame from which variables in the formula should be taken.
epsilon	Positive convergence tolerance.
maxit	Maximum number of iterations.
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 x 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 $\|X^\top X - \mathrm{diag}(X^\top X)\|_F^2$). (Formerly `lambda.ortho`).
- `B.L1`: Nonnegative penalty parameter for L1 regularization on $B = CA$ (default: 0). Promotes **sparsity in the coefficients**. (Formerly `gamma`).
- `C.L1`: Nonnegative penalty parameter for L1 regularization on C (default: 0). Promotes **sparsity in the parameter matrix**. (Formerly `lambda`).
- `Q`: Backward-compatible name for the rank of the basis matrix (Q).
- `method`: Objective function: Euclidean distance `"EU"` (default) or Kullback–Leibler divergence `"KL"`.
- `X.restriction`: Constraint for columns of X. Options: `"colSums"` (default), `"colSqSums"`, `"totalSum"`, `"none"`, or `"fixed"`. `"none"` applies no normalization to X after each update, allowing it to absorb the scale freely. This is automatically set when `Y.symmetric` = `"bi"` or `"tri"`, because column normalization would prevent $XX^\top$ (or $XCX^\top$) from approximating Y at the correct scale.
- `X.init`: Method for initializing the basis matrix X. Options: `"kmeans"` (default), `"kmeansar"`, `"runif"`, `"nndsvd"`, or a user-specified matrix. `"kmeansar"` applies k-means initialization and then fills zero entries with `Uniform(0, mean(Y)/100)`, analogous to NNDSVDar.
- `nstart`: Number of random starts for initialization of X (default: 1). Used by `kmeans` (when `X.init` = `"kmeans"` or `"kmeansar"`) and by the multi-start evaluation (when `X.init` = `"runif"`). For symmetric NMF (`Y.symmetric` = `"tri"` or `"bi"`), results are sensitive to initial values; `nstart` = `20` or higher is recommended.
- `seed`: Integer seed for reproducibility (default: 123).
- `C.init`: Optional numeric matrix giving the initial value of the parameter matrix C (i.e., Θ). If `A` is `NULL`, `C` has dimension $Q \times N$ (equivalently B); otherwise, `C` has dimension $Q \times K$ where $K = nrow(A)$. Default initializes all entries to 1.
- `Y.symmetric`: Character string specifying the type of symmetric NMF. `"none"` (default): standard NMF ($Y \approx XB$). `"bi"`: 2-factor symmetric NMF ($Y \approx XX^\top$). Internally implemented as the `"tri"` model with $C = I_Q$ (identity matrix) held fixed, so that X is updated freely without column normalization. The multiplicative update for X uses cube-root damping ($X \leftarrow X \circ (numerator/denominator)^{1/3}$) to prevent oscillation, since X appears in both factors of the decomposition (He et al., 2011,

Proposition 1). "tri": 3-factor symmetric NMF ($Y \approx XCX^\top$) where C is a $Q \times Q$ matrix representing cluster interactions (Ding et al., 2006). Both "bi" and "tri" require Y to be square and cannot be used with covariate matrix A. When Y.symmetric = "bi", X.restriction is automatically set to "none" (no column normalization), because $C = I_Q$ is fixed and cannot absorb the scale. For "tri", column normalization is retained (default "colSums") because the free parameter matrix C absorbs the scale.

- prefix: Prefix for column names of X and row names of B (default: "Basis").
- print.trace: Logical. If TRUE, prints progress every 10 iterations (default: FALSE).
- print.dims: Deprecated. Use verbose instead.
- detail: Level of post-fit criterion computation. "full" computes all criteria including silhouette, CPCC, dist.cor; "fast" skips expensive distance-based criteria; "minimal" returns only information criteria. Default is "full". For backward compatibility, save.time = TRUE maps to "fast" and save.memory = TRUE maps to "minimal".

Value

A list with components:

call	The matched call, as captured by match.call().
dims	A character string summarizing the matrix dimensions of the model.
runtime	A character string summarizing the computation time.
X	Basis matrix. Column normalization depends on X.restriction.
B	Coefficient matrix $B = CA$.
XB	Fitted values for Y.
C	Parameter matrix.
B.prob	Soft-clustering probabilities derived from columns of B.
B.cluster	Hard-clustering labels (argmax over $B.prob$ for each column).
X.prob	Row-wise soft-clustering probabilities derived from X.
X.cluster	Hard-clustering labels (argmax over $X.prob$ for each row).
A.attr	List of attributes of the input covariate matrix A, containing metadata like lag order and intercept status if created by nmfkc.ar or nmfkc.kernel.
formula.meta	If fitted via Formula Mode, a list with formula, Y_cols, and A_cols; otherwise NULL.
objfunc	Final objective value.
objfunc.iter	Objective values by iteration.
r.squared	Coefficient of determination R^2 between Y and XB.
method	Character string indicating the optimization method used ("EU" or "KL").
n.missing	Number of missing (or zero-weighted) elements in Y.
n.total	Total number of elements in Y.

`rank`	The rank Q used in the factorization.
`sigma`	The residual standard error, representing the typical deviation of the observed values Y from the fitted values XB.
`mae`	Mean Absolute Error between Y and XB.
`criterion`	A list of selection criteria, including `ICp` (`ICp1`, `ICp2`, `ICp3`), `CPCC`, `silhouette`, `AIC`, `BIC`, `dist.cor`, `B.prob.sd.min`, `B.prob.max.mean`, and `B.prob.entropy.mean`.

References

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

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

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

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

See Also

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

Examples

```
# Example 1. Matrix Mode (Existing)
X <- cbind(c(1,0,1),c(0,1,0))
B <- cbind(c(1,0),c(0,1),c(1,1))
Y <- X %*% B
rownames(Y) <- paste0("P",1:nrow(Y))
colnames(Y) <- paste0("N",1:ncol(Y))
print(X); print(B); print(Y)
res <- nmfkc(Y,rank=2,epsilon=1e-6)
res$X
res$B

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

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

# Example 4. Symmetric NMF (tri: Y ~ X C X^T)
```

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

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

Description

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

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

Usage

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

Arguments

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

Value

A list containing:

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

References

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

See Also

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

Examples

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

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

---

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

Description

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

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

Usage

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

Arguments

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

Value

A list with components:

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

See Also

`nmfkc.ar`, `nmfkc.cv`

Examples

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

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

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

Usage

```
nmfkc.ar.DOT(
  result,
  degree = 1,
  intercept = any(colnames(result$C) == "(Intercept)"),
  threshold = 0.1,
```

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

Arguments

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

Value

A character string representing a Graphviz DOT file.

See Also

nmfkc.ar, nmfkc, plot.nmfkc.DOT

Examples

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

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

Description

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

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

Usage

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

Arguments

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

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

Description

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

Usage

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

Arguments

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

Value

A list with components:

`spectral.radius`
: Numeric. The spectral radius of the companion matrix. A value less than 1 indicates stationarity.

`stationary`
: Logical. `TRUE` if the spectral radius is less than 1 (i.e., the system is stationary), `FALSE` otherwise.

See Also

`nmfkc`, `nmfkc.ar`

Examples

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

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

Description

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

Usage

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

Arguments

x	A categorical vector or a factor.

Value

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

See Also

nmfkc

Examples

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

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

Description

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

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

Usage

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

Arguments

`object`	An object of class `"nmfkc"` returned by `nmfkc`.
`Y`	The original observation matrix (P x N) used for fitting.
`detail`	Character string controlling the level of computation: `"full"` (default) computes all criteria including silhouette, CPCC and dist.cor; `"fast"` skips the expensive distance-based criteria; `"minimal"` returns only information criteria.
`...`	Additional arguments: `Y.weights` (non-negative weight matrix; `lm()`-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: ICp, ICp1, ICp2, ICp3, AIC, BIC, B.prob.sd.min, B.prob.max.mean, B.prob.entropy.mean, silhouette, CPCC, dist.cor.

See Also

`nmfkc`, `nmfkc.rank`

Examples

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

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

Description

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

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

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

Usage

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

Arguments

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: 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., gamma (B.L1), epsilon, maxit, method ("EU" or "KL"), X.restriction, X.init, etc.

Value

A list with components:

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

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

objfunc.block Loss for each fold.

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

See Also

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

Examples

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

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

---

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc.normalize

Examples

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

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

Description

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

Supported visualization types:

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

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

Usage

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

Arguments

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

Value

A character string representing a Graphviz DOT script.

See Also

nmfkc, nmfae.DOT, nmf.sem.DOT, nmfkc.ar.DOT, plot.nmfkc.DOT

Examples

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

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

Description

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

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

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

Usage

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

Arguments

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).

Value

A list with components:

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

See Also

nmfkc, nmfkc.cv

Examples

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

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

Description

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

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

Usage

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

Arguments

`object`	An object of class `"nmfkc"` returned by `nmfkc`.
`Y`	Observation matrix (P x N). Must match the data used in `nmfkc()`.
`A`	Covariate matrix (K x N). Default is `NULL` (same as identity; in this case $B = C$ and inference is on B directly).
`wild.bootstrap`	Logical. If `TRUE` (default), performs wild bootstrap for confidence intervals and bootstrap standard errors. Set to `FALSE` to skip bootstrap (faster, only sandwich SE is computed).
`...`	Additional arguments: `wild.B` Number of bootstrap replicates. Default is 1000. `wild.seed` Seed for bootstrap. Default is 42. `wild.level` Confidence level for bootstrap CI. Default is 0.95. `sandwich` Logical. Use sandwich covariance. Default is `TRUE`. `C.p.side` P-value type: `"one.sided"` (default) or `"two.sided"`. `cov.ridge` Ridge stabilization for information matrix inversion. Default is 1e-8. `print.trace` Logical. If `TRUE`, prints progress. Default is `FALSE`.

Value

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

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

References

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

See Also

`nmfkc`, `summary.nmfkc.inference`

Examples

```
Y <- matrix(cars$dist, nrow = 1)
A <- rbind(intercept = 1, speed = cars$speed)
result <- nmfkc(Y, A, 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 $U(K,N) = (u_1,\ldots,u_N)$. Each row may be normalized in advance.
V	Covariate matrix $V(K,M) = (v_1,\ldots,v_M)$, typically used for prediction. If NULL, the default is U.
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 $U(K, N) = (u_1, \ldots, u_N)$. Each row may be normalized in advance.
V	Covariate matrix $V(K, M) = (v_1, \ldots, v_M)$, typically used for prediction. If NULL, the default is U.
beta	A numeric vector of candidate kernel parameters to evaluate via cross-validation.
plot	Logical. If TRUE (default), plots the objective function values for each candidate beta.
...	Additional arguments passed to nmfkc.cv.

Value

A list with components:

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

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

`U`	A numeric matrix of covariates (`K x N`); columns are samples.
`Uk`	An optional numeric matrix of landmarks (`K x M`); columns are landmark points. If provided, distances are computed from samples in `U` to landmarks in `Uk`.
`block.size`	Integer. Number of columns of `U` processed per block when computing distances (controls memory usage). If `N <= 1000`, it is automatically set to `N`.
`block.size.Uk`	Integer. Number of columns of `Uk` processed per block when `Uk` is not `NULL` (controls memory usage). If `M <= 2000`, it is automatically set to `M`.
`sample.size`	Integer or `NULL`. If not `NULL`, randomly subsamples this many columns of `U` (without replacement) before computing distances, to reduce computational cost.
`...`	Additional arguments. Hidden option `candidates` 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 $U(K, N) = (u_1, \ldots, u_N)$. Each row may be normalized in advance.
`V`	Covariate matrix $V(K, M) = (v_1, \ldots, v_M)$, typically used for prediction. If `NULL`, the default is `U`. May contain `NA` values.
`beta`	Bandwidth parameter for the Gaussian kernel. Default is `0.5`.
`na.method`	Method for handling `NA` values in `V`. Either `"pds"` or `"egk"`. Ignored if `V` has no `NA`.
`...`	Additional arguments for EGK method:
	`gmm.G` Number of GMM components for EGK. Default is `3` (Mesquita et al., 2019).

Value

Kernel matrix $A(N, M)$.

Source

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

See Also

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

Examples

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

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

---

nmfkc.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): $Y \approx XCX^\top$ with $X, C \geq 0$ (both non-negative; C symmetric by design). Uses Frobenius-full bilateral gradient.
- **bi** (`type="bi"`): $Y \approx XX^\top$ (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

`Y`	Symmetric (N x N) **non-negative** adjacency matrix. `NA` entries are automatically treated as masked edges (equivalent to supplying `Y.weights` with 0 at those positions); see the note on `Y.weights` below.
`rank`	Integer Q.
`type`	`"tri"` (default), `"bi"`, or `"signed"`.
`epsilon`, `maxit`, `verbose`	Standard.
`...`	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 x 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.ecv` 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 x 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 $\mathrm{Uniform}(0, \bar{Y}/100)$ 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 > a few thousand) `"kmeans"` is preferable.
- `"runif"`: Uniform random entries in $[0, 1]$.
- `"random"`: Legacy default (pre-v0.6.8), equivalent to `abs(rnorm(N * Q)) * 0.1`. Kept for backward compatibility.

- A numeric N x 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 ($Y \approx XCX^\top$).

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("fdp", "dot", "neato", "circo", "twopi"),
  X.color = NULL,
  Y.cluster = c("soft", "hard")
)
```

Arguments

result	A list returned by nmfkc() with Y.symmetric = "bi" or "tri", or the newer nmfkc.net with type = "bi" or "tri". If inference results are present (from nmfkc.net.inference), C edges are decorated with significance stars.
threshold	Minimum coefficient value to display an edge.
sig.level	Significance level for filtering C edges (if inference results are present). Set to NULL 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 ("TB" or "LR").
fill	Logical; whether nodes are filled with color.
hide.isolated	Logical; if TRUE, omit outer nodes with no X edge above threshold.
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 NULL. Whether to draw C edges between basis nodes. NULL = auto-detect (TRUE for tri, FALSE for bi).
signed	Logical. If TRUE, C is treated as a signed matrix: positive entries rendered as solid edges and negative as dashed, with edge visibility threshold on $\|C\|$. Default is inherits(result, "nmfkc.net.signed") so that results from nmfkc.net are auto-detected.
cluster.box	Style of cluster box: "none", "normal", "faint", "invisible".

layout	Graphviz layout engine: "fdp", "dot", "neato", "circo", "twopi".
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).

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 ($Y \approx XCX^\top$).

This is a wrapper around nmfkc.inference that automatically sets the covariate matrix to $A = X^\top$, 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.normalize` *Normalize a matrix to the range* $[0, 1]$

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc.denormalize

Examples

```
# Example.
x <- nmfkc.normalize(iris[,-5])
apply(x,2,range)
```

---

nmfkc.rank *Rank selection diagnostics with graphical output*

Description

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

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

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

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

Usage

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

Arguments

Y	Observation matrix, or a formula (see nmfkc for Formula Mode).
A	Covariate matrix. If NULL, the identity matrix is used. Ignored when Y is a formula.
rank	A vector of candidate ranks to be evaluated.
detail	Level of criterion computation: "full" (default) computes all criteria including ECV; "fast" skips ECV and distance-based criteria.
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".

Value

A list containing:

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

References

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

See Also

nmfkc, nmfkc.ecv

Examples

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

---

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

Description

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

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

Usage

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

Arguments

`Y`	The original observation matrix (P x N).
`result`	The result object returned by the nmfkc function.
`fitted.palette`	A vector of colors for Y and XB heatmaps. Defaults to white-orange-red. For backward compatibility, `Y_XB_palette` is accepted via `....`
`residual.palette`	A vector of colors for the residuals heatmap. Defaults to blue-white-red. For backward compatibility, `E_palette` is accepted via `....`
`...`	Additional graphical parameters passed to the internal image calls.

Value

NULL. The function generates a plot.

See Also

`nmfkc`, `residuals.nmf`

Examples

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

---

nmfkc.signed *NMF-KC with signed covariate matrix*

Description

Solves

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

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 $Q_{\mathrm{obs}} \times N$ response matrix. Unlike nmfkc, 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 predict() 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: • 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, $\mathrm{colSums}(X) = 1$), "colSqSums", "totalSum", "none", "fixed". • X.init: initialization strategy for the basis matrix X ($Q_{\mathrm{obs}} \times Q$). Accepts the same menu as nmfkc: "kmeans" (default), "kmeansar", "nndsvd", "runif", or a user-supplied $Q_{\mathrm{obs}} \times Q$ non-negative numeric matrix. String methods delegate to the shared internal helper .init_X_method() (see nmfkc 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 $\mathrm{Uniform}(0, 0.1)$ 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 nmfkc(Y, A = rbind(A_+, A_-), rank = Q) internally to seed X, C_+, C_-. The user's X.init, seed, nstart, and X.restriction are forwarded to the internal nmfkc 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; X.init 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 ($Q_{\mathrm{obs}} \times N$) 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`: $Q_{\mathrm{obs}} \times Q$ 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`: $C_+ - C_-$ (= Θ), signed.
- `B`: $C\,A$, $Q \times N$ (signed).
- `objfunc.iter`: objective values per iteration.
- `objfunc`: final objective.
- `r.squared`: $\mathrm{cor}(Y, \widehat{Y})^2$.
- `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 $Q_{\mathrm{obs}} \times N$ 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 *Element-wise cross-validation for nmfkc.signed*

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 $Q_{\mathrm{obs}} \times N$ 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`

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

Description

Estimates the NMF-RE model

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

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

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

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

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

Usage

```
nmfre(
  Y,
  A = NULL,
  rank = 2,
  df.rate = NULL,
  wild.bootstrap = TRUE,
  epsilon = 1e-05,
  maxit = 5000,
  ...
)
```

Arguments

Y	Observation matrix (P x N), non-negative.
A	Covariate matrix (K x N). Default is a row of ones (intercept only).
rank	Integer. Rank of the basis matrix X. Default is 2. For backward compatibility, Q is accepted via ....
df.rate	Rate for computing the dfU cap (cap = rate * N * Q). For backward compatibility, dfU.cap.rate is accepted via .... If NULL (default), runs nmfre.dfU.scan internally and selects the minimum rate where the cap is not binding. Use nmfre.dfU.scan beforehand to examine dfU behavior across rates and choose an appropriate value.
wild.bootstrap	Logical. If TRUE (default), perform wild bootstrap inference on Θ.
epsilon	Convergence tolerance for relative change in objective (default 1e-5).
maxit	Maximum number of iterations. Default 5000 (matches nmfkc 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.
...	Additional arguments for initialization, variance control, dfU control, optimization, and inference settings. • X.init: Initial basis matrix (P x Q), or NULL. When NULL, nmfkc is called internally to generate initial values. • C.init: Initial coefficient matrix (Q x K), or NULL. When NULL, nmfkc is called internally to generate initial values. • U.init: Initial random effects matrix (Q x N), or NULL (all zeros). • prefix: Prefix for basis names (default "Basis"). • sigma2: Initial residual variance (default 1). • sigma2.update: Logical. Update σ^2 during iterations (default TRUE). • tau2: Initial random effect variance (default 1). • tau2.update: Logical. Update τ^2 by moment matching (default TRUE). Disabled when dfU cap is active. • dfU.control: Either "cap" (default) to enforce a cap on dfU, or "off" for no cap.

- `print.trace`: Logical. If `TRUE`, print progress every 100 iterations (default `FALSE`).
- `seed`: Integer seed for reproducibility (default 1).
- `C.p.side`: P-value sidedness: `"one.sided"` (default, for boundary null H0: C=0 vs H1: C>0) or `"two.sided"`.
- `wild.B`: Number of wild bootstrap replicates (default 500).
- `wild.seed`: Seed for wild bootstrap (default 123).

Value

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

Core matrices

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

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

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

Variance components

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

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

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

Convergence diagnostics

`converged` Logical. Whether the algorithm converged.

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

`iter` Number of iterations performed.

`maxit` Maximum iterations setting used.

`epsilon` Convergence tolerance used.

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

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

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

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

Effective degrees of freedom (dfU) diagnostics

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

`dfU.cap` Upper bound imposed on df_U.

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

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

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

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

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

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

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

Fitted matrices

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

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

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

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

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

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

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

Fit statistics

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

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

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

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

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

Inference on Θ (wild bootstrap)

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

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

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

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

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

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

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

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

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

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

References

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

See Also

nmfre.inference, nmfre.dfU.scan, nmfkc.DOT, summary.nmfre

Examples

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


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

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

  # Fit with chosen rate
  res <- nmfre(Y, A, rank = 1, df.rate = 0.2)
  summary(res)
}
```

---

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

Description

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

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

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

Usage

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

```
  rank = NULL,
  X.init = NULL,
  C.init = NULL,
  U.init = NULL,
  print.trace = FALSE,
  ...
)
```

Arguments

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

Value

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

table A data frame with the following columns:

- rate Cap rate used. The dfU cap is rate * N * Q.
- dfU.cap The dfU cap value (upper bound on effective degrees of freedom).
- dfU Final effective degrees of freedom for U at convergence.
- safeguard Logical. TRUE if the dfU cap is functioning as a safeguard (dfU / dfU.cap < 0.99): the cap prevents random-effects saturation without over-constraining U. FALSE if dfU is at or near the cap, indicating the cap is binding and the rate may be too small.
- hit Logical. TRUE if the cap was reached at least once during iteration, even if dfU later decreased below the cap.
- converged Logical. TRUE if the algorithm converged within the maximum number of iterations.
- tau2 Final random effect variance $\hat{\tau}^2$.
- sigma2 Final residual variance $\hat{\sigma}^2$.
- ICC Trace-based Intraclass Correlation Coefficient $\tau^2 \operatorname{tr}(X^\top X)/(\tau^2 \operatorname{tr}(X^\top X) + \sigma^2 P)$. See nmfre for details.

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

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

See Also

nmfre

Examples

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


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

---

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

Description

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

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

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

Usage

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

Arguments

object	An object of class "nmfre" returned by nmfre.
Y	Observation matrix (P x N). Must match the data used in nmfre().
A	Covariate matrix (K x N). Default is NULL (intercept only).
wild.bootstrap	Logical. If TRUE (default), performs wild bootstrap for confidence intervals and bootstrap standard errors.
...	Additional arguments: wild.B Number of bootstrap replicates. Default is 500.

`wild.seed` Seed for bootstrap. Default is 123.

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

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

`cov.ridge` Ridge stabilization. Default is 1e-8.

`print.trace` Logical. Default is `FALSE`.

Value

The input `object` with additional inference components:

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

References

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

See Also

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

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

`nmfae`, `nmfae.heatmap`

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfae.cv

---

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfae.ecv

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

`nmfae.kernel.beta.cv`

plot.nmfae.signed *Plot method for nmfae.signed (convergence)*

Description

Displays the convergence trajectory of the objective function.

Usage

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

Arguments

`x`	An `nmfae.signed` object.
`...`	Additional graphical parameters.

Value

Invisible NULL.

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

nmfae.signed

---

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc, summary.nmfkc

Examples

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

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

Description

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

This method handles all DOT objects produced by the nmfkc package: `nmfkc.DOT`, `nmfae.DOT`, `nmf.sem.DOT`, and `nmfkc.ar.DOT`.

Usage

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

Arguments

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

Value

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

See Also

`nmfkc.DOT`, `nmfae.DOT`, `nmf.sem.DOT`, `nmfkc.ar.DOT`

plot.nmfkc.signed *Plot method for nmfkc.signed (convergence)*

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.

`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

`x`	A fitted model object.
`...`	Additional graphical arguments passed to `plot`.
`which`	For `plot.nmf.sem`: 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.sem`

Examples

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

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

Description

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

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

Usage

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

Arguments

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

Value

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

See Also

`predict.nmfae`

Examples

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

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

Description

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

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

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

Usage

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

Arguments

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

Value

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

See Also

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

Examples

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

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

Description

Computes $\hat{Y}_1 = X_1(C_+ - C_-)X_2 Y_2^{\text{new}}$. Since $\Theta = C_+ - C_-$ is signed, predictions may contain negative entries even when $Y_1 \geq 0$ in training.

Usage

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

Arguments

`object`	A fitted `"nmfae.signed"` object.
`newY2`	New input matrix (P2 x N_new). If `NULL`, returns the training fitted values.
`Y1`	Optional reference Y1 for scatter / confusion plot.
`type`	Output: `"response"` (raw signed) or `"class"`.
`...`	Unused.

Value

A numeric matrix (`"response"`) or factor (`"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

`nmfae.signed`

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

Description

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

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc, nmfkc.cv

Examples

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

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

Description

Computes $\widehat{Y} = X\,C\,A_{\mathrm{new}}$ $(= X(C_+ - C_-)(A_+^{\mathrm{new}} - A_-^{\mathrm{new}}))$. For `type = "response"` the raw prediction is returned (possibly signed). For `type = "prob"` and `"class"`, negative entries of $\widehat{Y}$ 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 $D \times N_{\mathrm{new}}$ 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.

print.nmf.inference *Print method for NMF inference objects*

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc.inference, nmfae.inference

---

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

summary.nmfae

print.summary.nmfae.inference

Print method for summary.nmfae.inference objects

Description

Prints a formatted summary including the coefficients table.

Usage

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

Arguments

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

Value

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

See Also

summary.nmfae.inference

print.summary.nmfae.signed

Print method for summary.nmfae.signed

Description

Print method for summary.nmfae.signed

Usage

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

Arguments

x	A "summary.nmfae.signed" object.
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.

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

Description

Prints the Signed-Bottleneck NMF-AE summary followed by the coefficients table of Theta.

Usage

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

Arguments

`x`	An object of class `"summary.nmfae.signed.inference"`.
`digits`	Minimum number of significant digits.
`...`	Additional arguments (currently unused).

Value

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

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

`summary.nmfae.signed.inference`

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

Description

Prints a formatted summary of an nmfkc model fit.

Usage

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

Arguments

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

Value

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

Examples

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

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

Description

Prints a formatted summary including the coefficients table.

Usage

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

Arguments

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

Value

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

See Also

summary.nmfkc.inference

---

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

x	An object of class "summary.nmfkc.net".
digits	Minimum number of significant digits.
...	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), ...)
```

Arguments

`x`	An object of class `"summary.nmfkc.net.inference"`.
`digits`	Minimum number of significant digits.
`...`	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.

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

Description

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

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

Usage

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

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

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

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

Arguments

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

Value

The residual matrix.

See Also

`nmfkc`, `nmfae`, `nmfre`, `nmf.sem`, `fitted.nmf`

Examples

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

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

Description

Produces a formatted summary of a fitted NMF-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.sem" returned by nmf.sem.
...	Not used.

Value

Invisible object.

See Also

nmf.sem, nmf.sem.inference

Examples

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

summary.nmfae *Summary method for nmfae objects*

Description

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

Usage

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

Arguments

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

Value

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

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

See Also

`nmfae`, `print.summary.nmfae`

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

`nmfae.inference`, `summary.nmfae`

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

Description

Produces a summary with dimensions, convergence, fit statistics, and structure diagnostics (sparsity and negative-mass ratio).

Usage

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

Arguments

`object`	An `nmfae.signed` object.
`...`	Unused.

Value

An object of class `"summary.nmfae.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

`nmfae.signed`

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

Description

Produces a summary of a fitted Signed-Bottleneck NMF-AE model with inference results. Extends `summary.nmfae.signed` by attaching the `coefficients` table and p-value side from `nmfae.signed.inference`.

Usage

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

Arguments

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

Value

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

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

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

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc, nmfkc.inference, plot.nmfkc

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc.inference, summary.nmfkc

---

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

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

object	An object of class "nmfkc.net.inference".
...	Additional arguments passed to summary.nmfkc.

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

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

Value

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

See Also

nmfkc.net

---

summary.nmfkc.signed *Summary method for nmfkc.signed*

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.

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

Description

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

Usage

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

Arguments

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

Value

The input object, invisibly.

See Also

nmfre, nmfre.inference

Examples

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

Index

coef.nmf, 4

fitted.nmf, 5, *100*
fitted.nmfae *(*fitted.nmf*)*, 5

kmeans, *29*

lm, *18*, *29*, *37*, *61*, *71*

nmf.ffb, 6
nmf.ffb.cv, 9
nmf.ffb.DOT, 10
nmf.ffb.inference, 12
nmf.ffb.split, 15, *16*
nmf.sem, *5*, *10*, *12–16*, *87*, *100*, *101*
nmf.sem *(*nmf.ffb*)*, 6
nmf.sem.cv, *9*
nmf.sem.cv *(*nmf.ffb.cv*)*, 9
nmf.sem.DOT, *9*, *14*, *15*, *51*, *86*
nmf.sem.DOT *(*nmf.ffb.DOT*)*, 10
nmf.sem.inference, *4*, *9*, *12*, *101*
nmf.sem.inference *(*nmf.ffb.inference*)*, 12
nmf.sem.split, *9*
nmf.sem.split *(*nmf.ffb.split*)*, 15
nmfae, *5*, 17, *20–26*, *28*, *29*, *31*, *82*, *89*, *100*, *102*
nmfae.cv, 19, *26*, *27*, *82*, *83*
nmfae.DOT, *19*, 20, *24*, *51*, *86*
nmfae.ecv, *19*, *20*, 22, *31*, *32*, *83*
nmfae.heatmap, 24, *33*, *82*
nmfae.inference, *4*, *19*, 25, *33*, *34*, *93*, *103*
nmfae.kernel.beta.cv, *20*, 26, *84*
nmfae.rename, 27
nmfae.signed, 28, *31–35*, *85*, *90*, *103*
nmfae.signed.ecv, *30*, 31
nmfae.signed.heatmap, 32
nmfae.signed.inference, 33, *104*
nmfae.signed.rename, *31*, 35
nmfkc, *5*, *7*, *8*, *17*, *19*, 36, *40*, *43–49*, *51–54*, *67–70*, *72*, *75*, *85*, *91*, *100*, *105*
nmfkc.ar, *39*, 40, *42–45*
nmfkc.ar.degree.cv, *40*, 41, *49*
nmfkc.ar.DOT, *40*, 42, *51*, *86*
nmfkc.ar.predict, 44
nmfkc.ar.stationarity, *40*, 45
nmfkc.class, 46, *89*
nmfkc.criterion, 46
nmfkc.cv, *20*, *39*, *42*, 47, *52*, *55*, *73*, *91*
nmfkc.denormalize, 49, *67*
nmfkc.DOT, 50, *64*, *78*, *80*, *81*, *86*
nmfkc.ecv, *23*, 52, *67*, *68*
nmfkc.inference, *4*, 53, *65*, *66*, *93*, *105*, *106*
nmfkc.kernel, *19*, *27*, *39*, 54, *56*, *60*
nmfkc.kernel.beta.cv, *27*, *49*, 55, *58*, *60*
nmfkc.kernel.beta.nearest.med, *26*, *27*, 56, *56*, *60*
nmfkc.kernel.gaussian, *55*, *56*, *58*, 59
nmfkc.net, 60, *63–65*, *106*, *108*
nmfkc.net.DOT, 62, *62*, *66*
nmfkc.net.ecv, *61*, *62*, 64
nmfkc.net.inference, *63*, *64*, 65, *107*
nmfkc.normalize, *49*, 66
nmfkc.rank, *39*, *47*, 67
nmfkc.residual.plot, 68
nmfkc.signed, *60*, 69, *73*, *74*
nmfkc.signed.cv, *71*, 72, *74*
nmfkc.signed.ecv, *71*, 73, *73*
nmfre, *5*, 74, *78–81*, *87*, *100*, *109*
nmfre.dfU.scan, *75*, *76*, 78, *78*
nmfre.inference, *4*, *78*, 80, *109*

plot, *85*, *87*
plot.nmf.sem *(*plot.nmfre*)*, 87
plot.nmfae, *24*, 82
plot.nmfae.cv, 82
plot.nmfae.ecv, 83
plot.nmfae.kernel.beta.cv, 84
plot.nmfae.signed, 84
plot.nmfkc, 85, *105*
plot.nmfkc.DOT, *12*, *43*, *51*, *64*, 86

plot.nmfkc.signed, 86
plot.nmfre, 87
plot.predict.nmfae, 88, *89*
predict.nmfae, *19*, *88*, 89
predict.nmfae.signed, *31*, 90
predict.nmfkc, *39*, 91
predict.nmfkc.signed, *72*, 92
print.nmf.inference, 92
print.summary.nmfae, 93, *102*
print.summary.nmfae.inference, 94
print.summary.nmfae.signed, 94
print.summary.nmfae.signed.inference, 95
print.summary.nmfkc, 96
print.summary.nmfkc.inference, 96
print.summary.nmfkc.net, 97
print.summary.nmfkc.net.inference, 98
print.summary.nmfkc.net.signed, 98
print.summary.nmfkc.signed, 99

residuals.nmf, *5*, *69*, 100
residuals.nmfae *(*residuals.nmf*)*, 100
residuals.nmfre *(*residuals.nmf*)*, 100

summary.nmf.sem, *9*, 101
summary.nmfae, *93*, 101, *103*
summary.nmfae.inference, *26*, *94*, 102
summary.nmfae.signed, *31*, 103, *104*
summary.nmfae.signed.inference, *95*, 104
summary.nmfkc, *85*, 105, *106*
summary.nmfkc.inference, *54*, *97*, 105
summary.nmfkc.net, *97*, 106
summary.nmfkc.net.inference, *66*, *98*, 107
summary.nmfkc.net.signed, *99*, 107
summary.nmfkc.signed, 108
summary.nmfre, *78*, *81*, 109