Package 'nmfkc'

October 30, 2026

Type Package

Title Non-Negative Matrix Factorization with Kernel Covariates

Version 1.0.2

Date 2026-10-29

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 with feed-forward and feedback
structure (NMF-FFB) via nmf.ffb(), 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 (2026) <doi:10.1007/s42081-026-00349-x>;
Satoh (2025) <doi:10.48550/arXiv.2512.18250>;
Satoh (2026) <doi:10.48550/arXiv.2603.01468>;
Satoh and Tokuda (2026) <doi:10.48550/arXiv.2607.27474>;
Satoh (2026) <doi:10.1007/s42081-025-00314-0>.

License MIT + file LICENSE

Imports stats, graphics, utils, grDevices

Encoding UTF-8

Language en-US

ByteCompile true

VignetteBuilder knitr, rmarkdown

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

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

RoxygenNote 7.2.3

NeedsCompilation no

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

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2026-10-30 03:20:02 UTC

Contents

coef.nmf . . . 5
coef.nmf.gmm . . . 6
fitted.nmf . . . 6
fitted.nmf.gmm . . . 7
nmf.cluster.criteria . . . 8
nmf.cluster.flow . . . 9
nmf.ffb . . . 10
nmf.ffb.cv . . . 16
nmf.ffb.diagnostics . . . 18
nmf.ffb.DOT . . . 19
nmf.ffb.ecv . . . 21
nmf.ffb.inference . . . 23
nmf.ffb.split . . . 26
nmf.ffb.test . . . 28
nmf.gmm . . . 30
nmf.gmm.inference . . . 33
nmf.gmm.select . . . 34
nmf.gmm.twostage . . . 35
nmf.rrr . . . 36
nmf.rrr.cv . . . 39
nmf.rrr.DOT . . . 41
nmf.rrr.ecv . . . 43
nmf.rrr.heatmap . . . 44
nmf.rrr.inference . . . 45
nmf.rrr.kernel.beta.cv . . . 47
nmf.rrr.rank . . . 48
nmf.rrr.rename . . . 49
nmf.rrr.signed . . . 50
nmf.rrr.signed.ecv . . . 54
nmf.rrr.signed.heatmap . . . 55

nmf.rrr.signed.inference . . . 56
nmf.rrr.signed.rank . . . 57
nmf.rrr.signed.rename . . . 59
nmfkc . . . 60
nmfkc.ar . . . 66
nmfkc.ar.degree.cv . . . 67
nmfkc.ar.DOT . . . 69
nmfkc.ar.latent . . . 70
nmfkc.ar.latent.inference . . . 73
nmfkc.ar.predict . . . 77
nmfkc.ar.stationarity . . . 78
nmfkc.ard . . . 80
nmfkc.bicv . . . 81
nmfkc.class . . . 83
nmfkc.consensus . . . 83
nmfkc.criterion . . . 85
nmfkc.cv . . . 86
nmfkc.cv.methods . . . 88
nmfkc.denormalize . . . 89
nmfkc.DOT . . . 90
nmfkc.ecv . . . 92
nmfkc.inference . . . 93
nmfkc.kernel . . . 96
nmfkc.kernel.beta.cv . . . 97
nmfkc.kernel.beta.nearest.med . . . 98
nmfkc.kernel.gaussian . . . 100
nmfkc.kernel.gram . . . 102
nmfkc.net . . . 104
nmfkc.net.DOT . . . 107
nmfkc.net.ecv . . . 109
nmfkc.net.inference . . . 111
nmfkc.net.rank . . . 112
nmfkc.normalize . . . 113
nmfkc.rank . . . 114
nmfkc.residual.plot . . . 115
nmfkc.rff.beta.cv . . . 117
nmfkc.rff.positive . . . 119
nmfkc.rff.positive.gram . . . 120
nmfkc.signed . . . 122
nmfkc.signed.cv . . . 127
nmfkc.signed.ecv . . . 128
nmfkc.signed.rank . . . 129
nmfkc.signed.rff . . . 130
nmfkc.signed.rff.gram . . . 132
nmfre . . . 134
nmfre.ecv . . . 139
nmfre.inference . . . 140
plot.nmf.cluster.criteria . . . 142

plot.nmf.cluster.flow . . . 143
plot.nmf.gmm . . . 144
plot.nmf.gmm.select . . . 145
plot.nmf.rank . . . 146
plot.nmfae . . . 147
plot.nmfkc . . . 147
plot.nmfkc.ar.latent . . . 148
plot.nmfkc.ar.stationarity . . . 150
plot.nmfkc.ard . . . 151
plot.nmfkc.consensus . . . 151
plot.nmfkc.DOT . . . 152
plot.nmfkc.signed . . . 153
plot.nmfre . . . 154
plot.nmfre.ecv . . . 155
predict.nmf.gmm . . . 156
predict.nmfkc . . . 156
predict.nmfkc.signed . . . 158
predict.nmfre . . . 159
print.nmf . . . 160
print.nmf.cluster.criteria . . . 161
print.nmf.cluster.flow . . . 161
print.nmf.ffb.test . . . 162
print.nmf.gmm . . . 162
print.nmf.gmm.select . . . 163
print.nmf.inference . . . 164
print.nmf.rank . . . 164
print.nmfae.ecv . . . 165
print.nmfkc.ar.latent . . . 165
print.nmfkc.ar.latent.inference . . . 166
print.nmfkc.ar.stationarity . . . 166
print.nmfkc.ard . . . 167
print.nmfkc.consensus . . . 167
print.nmfkc.DOT . . . 168
print.nmfkc.gram . . . 168
print.nmfre.ecv . . . 169
print.summary.nmf.ffb . . . 169
print.summary.nmf.gmm . . . 170
print.summary.nmfkc . . . 171
print.summary.nmfkc.inference . . . 171
print.summary.nmfkc.net . . . 172
print.summary.nmfkc.net.inference . . . 173
print.summary.nmfkc.net.signed . . . 174
print.summary.nmfkc.signed . . . 174
residuals.nmf . . . 175
residuals.nmf.gmm . . . 176
summary.nmf.ffb . . . 177
summary.nmf.gmm . . . 178
summary.nmfkc . . . 178

summary.nmfkc.inference 179
summary.nmfkc.net 180
summary.nmfkc.net.inference 180
summary.nmfkc.net.signed 181
summary.nmfkc.signed 182
summary.nmfre 182

Index **184**

`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`, `nmf.rrr.inference`, `nmfre.inference`).

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

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

Usage

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

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

Arguments

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

Value

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

See Also

`nmfkc.inference`, `nmf.rrr.inference`, `nmfre.inference`, `nmf.ffb.inference`

Examples

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

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

coef.nmf.gmm *Extract the covariate-coefficient matrix from an NMF-GMM fit*

Description

Extract the covariate-coefficient matrix from an NMF-GMM fit

Usage

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

Arguments

object	An object of class "nmf.gmm".
...	Ignored.

Value

The $Q \times R$ coefficient matrix C ($= \Theta$).

Development status

Part of the **experimental** NMF-GMM family: see nmf.gmm. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm, nmf.gmm.inference, nmf.gmm.select

fitted.nmf *Extract fitted values from NMF models*

Description

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

For nmf.ffb 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 'nmfre'
fitted(object, type = c("blup", "fixed"), ...)

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

Arguments

`object`	A fitted model object of class `"nmf"`, `"nmfkc"`, `"nmfae"`, `"nmfre"`, or `"nmf.ffb"`.
`...`	For `nmf.ffb`: optionally `Y1` and `Y2`.
`type`	For `nmfre` objects: `"blup"` (default, the fit $X(\Theta A + U)$ including random effects) or `"fixed"` ($X\Theta A$, fixed effects only). Chosen so that `Y - fitted(object, type)` equals `residuals(object, Y, type)`.

Value

The fitted matrix XB.

See Also

nmfkc, nmf.rrr, nmfre, nmf.ffb, residuals.nmf

Examples

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

---

fitted.nmf.gmm *Fitted (responsibility-averaged) reconstruction of an NMF-GMM fit*

Description

Fitted (responsibility-averaged) reconstruction of an NMF-GMM fit

Usage

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

Arguments

`object`	An object of class `"nmf.gmm"`.
`...`	Ignored.

Value

The $P \times N$ fitted matrix $\hat{Y} = X(CA + \mu\gamma^\top)$.

Development status

Part of the **experimental** NMF-GMM family: see `nmf.gmm`. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

`nmf.gmm`, `nmf.gmm.inference`, `nmf.gmm.select`

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

Description

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

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

Usage

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

Arguments

`fits`	A list of fitted models, one per rank, all over the same N individuals (a single fitted model is also accepted and wrapped automatically). Supported families: `nmfkc`, `nmfkc.signed`, `nmf.rrr`, `nmfae.signed`, `nmfkc.net`, `nmfre`, and `nmf.ffb`.
`Y`	The original data matrix used to fit the models (Y_1 for `nmf.ffb`); required for the data-space distances.
`Y2`	Exogenous block, required only for `nmf.ffb`.

names	Optional character vector (length length(fits)) of x-axis tick labels. Defaults to each result's $rank.
plot	Logical; draw the diagnostics plot immediately (default TRUE); see plot.nmf.cluster.criteria.
...	When plot = TRUE, graphical arguments forwarded to plot.nmf.cluster.criteria.

Value

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

See Also

nmf.cluster.flow, nmfkc.rank

Examples

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

---

nmf.cluster.flow *Cluster-flow (alluvial) diagram across a sequence of fits*

Description

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

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

Usage

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

Arguments

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

Value

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

See Also

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

Examples

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

---

nmf.ffb *NMF-FFB: non-negative matrix factorization with latent feedback*

Description

Fits the NMF-FFB model

$$Y_1 = XB + E, \qquad B = \Theta_1 Y_1 + \Theta_2 Y_2 + U,$$

in which a non-negative basis X ($P_1 \times Q$) generates the endogenous block Y_1 from latent scores B that are driven by the exogenous block Y_2 (feed-forward, Θ_2) and fed back by Y_1 itself (latent feedback, Θ_1). The function returns the estimated basis, the 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.

Two estimators are available through `method`:

`"fiml"` **(default)** A two-stage likelihood-based estimator. **Stage 1** estimates the basis with the feed-forward fit `nmfkc(Y1, A = Y2)` (`X.init`, `X.L2.ortho`, `epsilon`, `maxit`, `seed` are forwarded), unless a basis is supplied through `X`. Columns are normalized to unit sum. **Stage 2** holds $\hat{X}$ fixed and fits the Gaussian working model $U \sim N(0, \Phi)$, $E \sim N(0, \text{diag}(\psi))$ by full-information maximum likelihood (L-BFGS-B with an analytic gradient) under the non-negativity of Θ_1, Θ_2 and an exclusion restriction on Θ_1. It fits (a) the feed-forward null $\Theta_1 = 0$ (a non-negative MIMIC factor model with correlated factors), (b) the unpenalized feedback model on the admitted entries, and (c) an L1 path over `C1.L1.path`. The penalized problem is non-convex, so every point of the path is fitted from several starting points (`starts`); every distinct support proposed by any (penalty, start) pair is re-estimated without penalty (itself from three starts, keeping the best log-likelihood) and BIC is minimized over all distinct candidates together with the null and the unpenalized model. The support with the smallest BIC is the *selected* model, reported in `C1`, `C2`, `Phi`, `psi`. `C1.L1` and `C2.L1` are not used by this estimator. The likelihood-ratio statistics against the null are returned without p-values: $\Theta_1 \geq 0$ puts the null on the boundary of the parameter space and the BIC refit is a post-selection statistic, so a chi-square reference is not valid. Their calibration by parametric bootstrap is done by `nmf.ffb.inference`.

`"mu"` The legacy estimator: joint multiplicative updates of X, Θ_1, Θ_2 for the structural-form squared error $\|Y_1 - X(\Theta_1 Y_1 + \Theta_2 Y_2)\|_F^2$ with an orthogonality penalty on X and L1 penalties `C1.L1`, `C2.L1`. Kept, bit-identical, so that analyses published with it reproduce. Because the structural and reduced forms have the same fit once X is free, this estimator cannot separate Θ_1 from Θ_2; prefer `"fiml"`.

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,
  ...,
  method = c("fiml", "mu"),
  X = NULL,
  C1.restriction = c("union", "none"),
  C1.restriction.threshold = 0.05,
  Phi.restriction = c("full", "diag"),
  C1.L1.path = NULL,
  select = c("BIC", "none")
)
```

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 $[0, 1]$. • `"spa"`: the anchor rows the successive projection algorithm finds in Y_1 (see nmfkc); a starting value only. `method = "fiml"` only. • A numeric $P_1 \times Q$ matrix supplied by the user; negative entries are projected to 0. • `NULL`: backward-compatible alias for `"nndsvd"`. To impose anchor rows on the stage-1 basis instead, pass `X.anchor` (`"spa"` or one row of Y_1 per basis) through `...`, as in nmfkc: stage 2 holds the basis fixed, so the zeros stay. It needs `rank <= nrow(Y2)`, is not available with `method = "mu"` or a supplied `X`, and the rows used are returned as `X.anchor`. 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`. **Scale note**: this function adds `C1.L1` to the multiplicative denominator, whereas nmfkc, nmf.rrr and

nmfkc.net add C.L1 / 2. The same nominal value is therefore twice as strong here. The difference is retained so that published nmf.ffb fits reproduce; halve the value to match the other models.

C2.L1 L1 sparsity penalty for C2 (i.e., Θ_2). Default: 0.1. Same scale note as C1.L1.

epsilon Relative convergence threshold for the objective function. Iterations stop when the relative change in **reconstruction loss** falls below this value. Default: 1e-6. **Note**: the test is on the unpenalized loss (objfunc), not on the penalized objective the updates actually minimize (objfunc.penalized). Every other optimizer in the package tests the penalized value, so with a large X.L2.ortho or C*.L1 this function can stop while the quantity being optimized is still moving. Both traces are returned; compare them if the penalties are strong.

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. For method = "fiml" the seed only reaches the stage-1 nmfkc fit; the FIML stage is deterministic.

... Additional hidden arguments. For method = "fiml": fiml.maxit (L-BFGS-B iteration cap, default 3000), factr (optim tolerance, default 1e3), and Q (alias of rank). For method = "mu" the following control 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.ffb 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.ffb MU iterations, and SC.map is computed. This means nmf.ffb(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.

method "fiml" (default) or "mu"; see Description.

X Optional basis for method = "fiml": a $P_1 \times Q$ non-negative matrix, or an nmfkc / nmf.ffb object whose $X is used. When supplied, stage 1 is skipped and rank is taken from ncol(X).

C1.restriction Exclusion restriction on Θ_1 for method = "fiml": "union" (default), "none", or a $Q \times P_1$ 0/1 matrix (1 = free). See the section *Exclusion restriction*.

`C1.restriction.threshold`
: Loading threshold for `C1.restriction = "union"`. Default `0.05`.

`Phi.restriction`
: Covariance of the latent disturbance U for `method = "fiml"`: `"full"` (default; positive definite via Cholesky) or `"diag"`.

`C1.L1.path`
: Numeric vector of L1 penalties on Θ_1 defining the path (`method = "fiml"`). Default `NULL`, meaning `N * c(0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5)`.

`select`
: `"BIC"` (default): the support with the smallest BIC among all candidates proposed along the path is the selected model; `"none"`: the unpenalized feedback fit is returned as the selected model and the path is skipped.

Value

An object of class `c("nmf.ffb", "nmf")`, a list with components:

`X`
: Estimated basis matrix ($P_1 \times Q$).

`C1`
: Estimated latent feedback matrix (Θ_1, $Q \times P_1$); for `"fiml"` the BIC-selected estimate.

`C2`
: Estimated exogenous loading matrix (Θ_2, $Q \times P_2$).

`XC1`
: Feedback matrix $X\Theta_1$.

`XC2`
: Direct-effect matrix $X\Theta_2$.

`XC1.radius`
: Spectral radius $\rho(X\Theta_1)$.

`XC1.norm1`
: Induced 1-norm $\|X\Theta_1\|_{1,\mathrm{op}}$.

`Leontief.inv`
: Leontief-type inverse $(I - X\Theta_1)^{-1}$.

`M.model`
: Equilibrium mapping $M_{\mathrm{model}} = (I - X\Theta_1)^{-1}X\Theta_2$.

`amplification`
: Latent amplification factor $\|M_{\mathrm{model}}\|_{1,\mathrm{op}} / \left\|X\Theta_2\right\|_{1,\mathrm{op}}$; meaningful only when `XC1.radius > 0` (it equals 1 when no feedback is selected).

`amplification.bound`
: Geometric-series upper bound $1/(1-\|X\Theta_1\|_{1,\mathrm{op}})$ if $\|X\Theta_1\|_{1,\mathrm{op}} < 1$, otherwise `Inf`.

`rank`
: 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 (`"mu"`); `NULL` for `"fiml"`.

`objfunc.penalized`
: Vector of penalized objective values per iteration (`"mu"`); `NULL` for `"fiml"`.

`iter`, `maxit`, `epsilon`, `converged`
: Convergence bookkeeping. For `"fiml"`: the number of objective evaluations of the selected fit, the L-BFGS-B cap (`fiml.maxit`), the stage-1 tolerance, and `optim()$convergence == 0`.

`method` `"fiml"` or `"mu"`.

The following are present for `method = "fiml"` only (`SC.cov` and `SC.map` are then `NULL`):

`Phi`, `psi`, `loglik`, `npar`
: Latent-disturbance covariance ($Q \times Q$), unique variances (length P_1), log-likelihood and parameter count of the selected model.

`null`
: The feed-forward null: list `C2`, `Phi`, `psi`, `loglik`, `npar`, `M.model`.

`full`
: The unpenalized feedback fit: list `C1`, `C2`, `Phi`, `psi`, `loglik`, `npar`, `XC1.radius`.

`path`
: Data frame with one row per (`C1.L1.path`, start) of the L1 path (`C1.L1.path = 0` is the unpenalized fit re-estimated on its non-zero entries, `Inf` the null): `C1.L1.path`, `start`, `support_id`, `nnz`, `rho`, `loglik`, `BIC`, `MAE` (each after re-estimation on the proposed support), `pen.value` (the penalized objective reached by that start, smaller is better) and `duplicate` (`TRUE` when the same support was already proposed by an earlier row).

`candidates`, `supports`, `support.selected`
: One row per distinct support (`support_id`, `nnz`, `rho`, `loglik`, `BIC`, `MAE`, `lambda1.first`, `start.first`, `selected`; the null and the model with every admitted entry free are always candidates, so `candidates` can hold a support that appears in no row of `path`); the supports themselves (list of logical $Q \times P_1$ matrices indexed by `support_id`); and the id of the selected one.

`C1.free`, `C1.L1.path`, `C1.L1.selected`, `support`
: The free-entry matrix used ($Q \times P_1$ 0/1), the path, the smallest penalty at which the selected support was proposed (`0` for the unpenalized model, `Inf` for the null), the selected support (logical $Q \times P_1$) and the starts used.

`LR`, `LR.df`
: Likelihood-ratio statistics `c(full = 2(l_full - l_null), selected = 2(l_sel - l_null))` and the naive degrees of freedom `c(full = sum(C1.restriction), selected = nnz)` (also stored as `attr(LR, "df")`). No p-value is attached; see `nmf.ffb.inference`.

`BIC`, `AIC`
: Named vectors `c(null, full, selected)`.

`call`
: The matched call, from which `nmf.ffb.inference` inherits the design.

Exclusion restriction

Feedback is identified only through exclusion restrictions: an outcome may not feed back into a factor on which it has more than a negligible loading, because such an entry is nearly equivalent to a change of the outcome's loading and uniqueness. With `C1.restriction = "union"` (default) entry (q, i) of Θ_1 is excluded if $q = \arg\max_{q'} X_{iq'}$ (the dominant factor) or $X_{iq} \geq$ `C1.restriction.threshold`. Both halves are needed: blocking the dominant factor alone would leave an outcome with a substantial second loading free to feed that factor, and blocking only the factors above the threshold would leave an outcome whose largest loading is below the threshold free to feed its own. `"none"` frees every entry (not recommended: the model is then identified only through the non-negativity and the covariance structure). A user-supplied $Q \times P_1$ 0/1 matrix is used as given.

The restriction is derived from the estimated basis and therefore from the same Y_1 that is subsequently tested; see the *Calibration* section of `nmf.ffb.inference` for what this implies.

Lifecycle

`method = "fiml"` became the default in version 0.9.8, as did `C1.restriction = "union"` (earlier fiml fits blocked the dominant factor only), and `nmf.ffb.inference` gained the `calibration` argument. `method = "mu"` is the legacy estimator, kept for the reproducibility of published analyses; it will be deprecated in a later release.

References

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

See Also

`nmf.ffb.inference`, `nmf.ffb.cv`, `nmf.ffb.split`, `nmf.ffb.DOT`, `summary.nmf.ffb`

Examples

```
# Simple NMF-FFB with iris data (non-negative)
Y <- t(iris[, -5])
Y1 <- Y[1:2, ]  # Sepal
Y2 <- Y[3:4, ]  # Petal
result <- nmf.ffb(Y1, Y2, rank = 2)
result$LR          # feedback vs feed-forward null (no p-value here)
result$BIC
result$path

# Legacy multiplicative-update estimator
result.mu <- nmf.ffb(Y1, Y2, rank = 2, maxit = 500, method = "mu")
result.mu$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.ffb` 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.

`method = "fiml"`**.** When `method = "fiml"` is passed (through `...`), the column-wise scheme above is not run: the equilibrium prediction $M_{\text{model}} Y_2$ is a reduced-form quantity, and the reduced form is the same with and without feedback, so predicting held-out columns cannot select Θ_1 (that is

what the BIC path and the bootstrap-calibrated LR test in `nmf.ffb` / `nmf.ffb.inference` are for). The only tunable quantity is the stage-1 rank, and the call is delegated to the element-wise CV of the feed-forward fit, `nmfkc.ecv(Y = Y1, A = Y2, rank = rank, ...)`, whose object is returned (`$objfunc` holds the CV error per rank; `rank` may be a vector, default `1:min(3, P1)`). `C1.L1` and `C2.L1` are ignored in that case.

Usage

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

Arguments

`Y1`	A non-negative numeric matrix of endogenous variables with **rows = variables (P1), columns = samples (N)**.
`Y2`	A non-negative numeric matrix of exogenous variables with **rows = variables (P2), columns = samples (N)**. Must satisfy `ncol(Y1) == ncol(Y2)`.
`rank`	Integer; rank (number of latent factors) passed to `nmf.ffb`. If `NULL`, `nmf.ffb` decides the effective rank (via `...` or `nrow(Y2)`).
`X.init`	Initialization strategy for `X`, forwarded to `nmf.ffb`. One of `"nndsvd"` (default), `"kmeans"`, `"kmeansar"`, `"runif"`, a numeric $P_1 \times Q$ matrix, or `NULL` (alias for `"nndsvd"`). See `nmf.ffb` 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.ffb`.
`maxit`	Maximum number of iterations for `nmf.ffb`.
`...`	Additional arguments passed to `nmf.ffb` (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`), `cores` (integer; number of parallel workers for the fold loop, default `getOption("mc.cores", 1L)`). Fold partitions and per-fold seeds are drawn before the loop and each `nmf.ffb` fit self-seeds, so results are identical to the sequential run for any `cores`.

Value

A numeric scalar: mean MAE across CV folds (the fits use `method = "mu"` unless `method` is given). With `method = "fiml"`, the `nmfkc.ecv` object of the stage-1 rank sweep.

See Also

`nmf.ffb`

Examples

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

# rank selection for the likelihood-based estimator
ecv <- nmf.ffb.cv(Y1, Y2, rank = 1:2, method = "fiml", nfolds = 3)
ecv$objfunc
```

---

nmf.ffb.diagnostics *Structural Diagnostics of a Fitted NMF-FFB Feedback Matrix*

Description

Two quantities that describe *what* the selected feedback is, as opposed to whether it is there (which is the job of the calibrated likelihood-ratio test in `nmf.ffb.inference`).

Cycle structure. The equilibrium operator is $A = X\Theta_1$ on the indicators and $A_Q = \Theta_1 X$ on the factors, with the same spectral radius ρ. The strongly connected components of the directed graph of A_Q are the cycles. $\rho = 0$ with a non-empty support means A is nilpotent: the selected feedback is a cascade of directed cross-factor paths without a closed loop. Since $A \geq 0$, Perron–Frobenius gives a non-negative leading eigenvector, and its normalized entries say how much of the loop each factor carries.

Factor-level alignment. Feedback of the form $\Theta_1 = aX^\top \Psi^{-1}$ enters each factor through the factor scores of the factors and is observationally equivalent, under the Gaussian working model, to a change of Θ_2 and of the factor covariance Φ; only the departure from that family is identified, through the uniquenesses Ψ. On the free set F of the exclusion restriction the family is a subspace of dimension at most $Q(Q-1)$, and `omega` is the share of the squared norm of Θ_1 that lies in it. `omega` has no natural zero: for an isotropic direction its expectation is `omega0 = dim / n.free`, so it should be read against `omega0` and against a simulation in which the true feedback is indicator-level (there it averages about `1.3 * omega0`).

Usage

```
nmf.ffb.diagnostics(object, which = c("selected", "full"))
```

Arguments

object	A fitted nmf.ffb object from nmf.ffb with method = "fiml".
which	Which feedback matrix to describe: "selected" (default, the BIC-selected support) or "full" (the unpenalized fit).

Value

A list with cycles (A.factor, rho, cycle, n.cycles, perron) and omega (omega, omega0, dim, n.free).

See Also

nmf.ffb, nmf.ffb.inference

Examples

```
set.seed(1)
Y2 <- matrix(runif(2 * 200), 2, 200)
X <- matrix(0.02, 6, 2); X[1:3, 1] <- 1; X[4:6, 2] <- 1
X <- sweep(X, 2, colSums(X), "/")
T1 <- matrix(0, 2, 6); T1[1, 5] <- 0.6; T1[2, 2] <- 0.5
L <- solve(diag(6) - X %*% T1)
Y1 <- pmax(L %*% (X %*% (matrix(c(1, 0.5, 0.3, 1), 2, 2) %*% Y2) +
                  matrix(rnorm(6 * 200, 0, 0.05), 6, 200)), 0)
fit <- nmf.ffb(Y1, Y2, Q = 2)
d <- nmf.ffb.diagnostics(fit)
d$cycles$rho          # spectral radius
d$cycles$cycle        # which factors lie on a cycle
c(d$omega$omega, d$omega$omega0)
```

---

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.ffb. 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,
  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,
  model = c("selected", "null", "full"),
  ...
)
```

Arguments

`result`	A list returned by `nmf.ffb`, containing matrices `X`, `C1`, and `C2`.
`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.ffb.inference`, only edges with `prob.unsupported` < `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.
`model`	Which fit of a `method = "fiml"` object to draw: `"selected"` (default; the BIC-selected model, with significance stars when inference results are present), `"null"` (the feed-forward null: $\Theta_1 = 0$, Θ_2 from `result$null`) or `"full"` (the unpenalized feedback fit from `result$full`). The latter two ignore `result$coefficients`, which refer to the selected model. Ignored for `method = "mu"` objects, which carry a single fit.
`...`	For backward compatibility: accepts deprecated names `weight_scale_y2f` (use `weight_scale_c2`) and `weight_scale_fy1` (use `weight_scale_x1`).

Value

A character string representing a valid Graphviz DOT script.

See Also

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

Examples

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

---

nmf.ffb.ecv — *Choose the number of factors for NMF-FFB by element-wise cross-validation*

Description

Step 1 of the NMF-FFB workflow: select Q by element-wise cross-validation of the *feed-forward* fit. Held-out entries of Y_1 are predicted from the remaining ones, so the criterion measures how well a Q-factor non-negative basis describes the outcomes, which is what Q is for.

Usage

```
nmf.ffb.ecv(
  Y1,
  Y2,
  rank = NULL,
  X.init = "nndsvd",
  X.L2.ortho = 100,
  epsilon = 1e-06,
  maxit = 5000,
  ...
)
```

Arguments

Y1, Y2	Endogenous and exogenous matrices (variables in rows, units in columns).
rank	Candidate values of Q. Defaults to seq_len(min(3, nrow(Y1))).
X.init	Stage-1 initialization, passed to nmfkc.
X.L2.ortho	Orthogonality penalty on the basis in stage 1. Keep the value that will be used in nmf.ffb: Q and the penalty are not separable.
epsilon, maxit	Convergence tolerance and iteration cap for stage 1.
...	Passed to nmfkc.ecv: nfolds, seed, and cores (or ncores) to evaluate the rank x fold grid in parallel, defaulting to getOption("mc.cores", 1L) as elsewhere in the package.

Details

Feedback cannot be cross-validated, and that is not a limitation of the implementation. By the reduced-form equivalence, the equilibrium mapping $M = (I - X\Theta_1)^{-1}X\Theta_2$ is reproduced exactly by a feed-forward model with the exogenous matrix $L_Q\Theta_2$: the two models predict Y_1 from Y_2 identically, so no prediction criterion – column-wise CV, element-wise CV, or a test-sample error – can prefer one over the other. Feedback is identified by the *conditional covariance*, not by prediction, which is why it is addressed by nmf.ffb.test and not here.

Value

The object returned by nmfkc.ecv: the held-out error by rank, with the selected rank.

Workflow

```
ecv <- nmf.ffb.ecv(Y1, Y2, rank = 1:5)   # 1. choose Q          <- this function
fit <- nmf.ffb(Y1, Y2, rank = ecv$rank)  # 2. estimate; BIC selects the support
tst <- nmf.ffb.test(fit, Y1, Y2)         # 3. test the feed-forward null
dgn <- nmf.ffb.diagnostics(fit)       # 4. cycles, spectral radius, identifiability
inf <- nmf.ffb.inference(fit, Y1, Y2)    # 5. intervals for the retained entries
```

See Also

nmf.ffb, nmf.ffb.test, nmfkc.ecv

Examples

```
set.seed(1)
Y <- t(as.matrix(iris[, 1:4]))
Y1 <- Y[1:2, ]; Y2 <- Y[3:4, ]
ecv <- nmf.ffb.ecv(Y1, Y2, rank = 1:2, nfolds = 3)
ecv$rank
```

`nmf.ffb.inference` *Bootstrap inference for NMF-FFB (X held fixed)*

Description

`nmf.ffb.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.ffb` model, *after* the feed-forward null has been rejected by `nmf.ffb.test`. The basis $\hat{X}$ is held fixed throughout, which avoids label switching and gives a clean conditional interpretation: uncertainty of the structural coefficients given the measurement model. Which resampling scheme is run depends on `object$method`.

`method = "fiml"`**: a parametric bootstrap from the selected model.** B data sets are drawn from the selected model (with its Θ_1) and $(\Theta_1, \Theta_2, \Phi, \psi)$ are re-estimated with X and the selected support held fixed. Confidence intervals are the centred (basic) percentile intervals $[2\hat{\theta} - q^*_{1-\alpha/2},\ 2\hat{\theta} - q^*_{\alpha/2}]$; `support_rate` is the share of replicates with $|\hat{\theta}^*| >$ `threshold` (0 for entries outside the selected support), and `prob.unsupported` is $1 - \text{support_rate}$.

These are *post-selection* quantities, conditional on the support that BIC picked, and they can overstate the evidence. They answer "how large is this entry, given that the procedure kept it", not "is there feedback at all"; the calibrated statement about the presence of feedback is `nmf.ffb.test`, which is the step that belongs before this one. Each replicate seeds itself (`seed + B + b`), so the result does not depend on `cores`.

`method = "mu"` **(legacy fits, and objects without a** `method` **field).** 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.ffb` 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 $p = 1 - \text{sup}$: * (sup > 0.95), ** (sup > 0.99), *** (sup > 0.999). Cutoffs use strict greater-than so the rule mirrors the standard R convention for p-values (p < 0.05 / 0.01 / 0.001 $\rightarrow$ //), translated to support_rate via $\text{sup} = 1 - p$.

Usage

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

Arguments

`object`	A fitted object returned by `nmf.ffb`. Must contain `X`, `C1`, `C2`.
`Y1`	Endogenous variable matrix (P1 x N). Must match the data used in `nmf.ffb()`.
`Y2`	Exogenous variable matrix (P2 x N). Same.
`B`	Number of bootstrap replicates. Default `1000`, the value the published analysis used; the other inference functions in the package default their `wild.B` to 500, and this one is deliberately left at 1000 so the manuscript results reproduce out of the box. Note that *** (sup > 0.999) is only reachable when *every* replicate clears `threshold`, at any `B`: raising `B` does not add a finer grade, it makes the same grade stronger evidence (all 1000 rather than all 500). Lowering it to 500 roughly halves the running time, since each replicate is a re-fit. For `method = "fiml"` a replicate is one FIML fit on the selected support, so `B = 1000` is affordable; the expensive bootstrap is the null one, which `nmf.ffb.test` runs.
`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.
`boot.level`	Confidence level for the bootstrap CI. Default `0.95`.
`C1.L1`, `C2.L1`	L1 sparsity penalties used by the original `method = "mu"` fit. These must match the fit's hyperparameters for the bootstrap to estimate the correct model. Defaults (`1.0`, `0.1`) match `nmf.ffb`'s defaults but you should pass the actual values used. Not used for `method = "fiml"`.
`seed`	Base RNG seed for the bootstrap. For `method = "mu"` each replicate uses `seed + b` (resampling) and `seed + 1000 + b` (C_1, C_2 initialization); for `method = "fiml"` see Description. Default `123`.
`...`	Hidden options. Shared: `cores` (number of parallel workers, default `getOption("mc.cores", 1L)` for `"fiml"`, 1 for `"mu"`; `ncores` is accepted as an alias) and `print.trace`.

For `method = "fiml"`: `factr` and `fiml.maxit` (L-BFGS-B tolerance and cap, default the values recorded on `object`). The null bootstrap is no longer run here at all: see `nmf.ffb.test`. (Removed: `boot.null`, which used to skip it, and the coefficient intervals). For `method = "mu"`:

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

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

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`, `prob.unsupported` ($= 1 - \text{support_rate}$) 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 CI bounds (percentile for `"mu"`, centred percentile for `"fiml"`).

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

`rho.boot.draws` Per-replicate spectral radius $\rho(XC_1^*)$.

`AR.boot`, `iter.boot`
(`"mu"` only) per-replicate amplification ratio and inner-loop iteration count.

#'

`boot.B`, `boot.threshold`, `boot.level`
Inputs recorded for reproducibility.

`boot.n.valid`, `boot.n.invalid`
Validity counts (for `"fiml"`: of the selected-model bootstrap).

`boot.method` How the replicates were drawn.

`identified.approx`, `identification`
Whether $(X, [C_1\ C_2])$ is unique up to a permutation of the bases – the diagnostic of `nmfkc.inference` with $A = [Y_1; Y_2]$ (see the Details of `nmfkc.ar.latent.inference`). The inference is conditional on $\hat{X}$; when `TRUE` it concerns the same coefficients whichever admissible representative is taken. The exclusion restriction's zeros in C_1 count. An algebraic check on the point estimate: no re-fits.

The calibrated test of the feed-forward null – `LR.p.boot`, `prob.select.null`, `C1.restriction.change.rate` and the null bootstrap behind them – is **not** returned here. It moved to `nmf.ffb.test` in 0.9.8, so that an object carrying coefficient intervals cannot be mistaken for a test of feedback (see NEWS).

Lifecycle

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

References

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

See Also

`nmf.ffb`, `nmf.ffb.DOT`

Examples

```
Y <- t(iris[, -5])
Y1 <- Y[1:2, ]; Y2 <- Y[3:4, ]
res  <- nmf.ffb(Y1, Y2, rank = 2)
## intervals for the entries the test has already licensed reading.
inf <- nmf.ffb.inference(res, Y1, Y2, B = 20)   # use B = 1000 in practice
head(inf$coefficients)  # estimate, interval and support rate per entry

res.mu  <- nmf.ffb(Y1, Y2, rank = 2, method = "mu")
res.mu2 <- nmf.ffb.inference(res.mu, Y1, Y2, B = 200)
head(res.mu2$coefficients)
```

---

nmf.ffb.split *Heuristic Variable Splitting for NMF-FFB*

Description

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

The procedure:

- internally standardizes variables (mean 0, sd 1),
- optionally flips signs so that most variables align positively with PC1,

- infers a causal ordering by repeatedly regressing each variable on the remaining ones and selecting the variable with the largest minimum standardized coefficient,
- determines an exogenous block by scanning the ordering from upstream and stopping at the first variable whose strongest parent coefficient exceeds `threshold`.

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

Usage

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

Arguments

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

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.ffb

Examples

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

---

nmf.ffb.test *Test the feed-forward null of an NMF-FFB fit*

Description

Calibrated test of the hypothesis that the outcomes need no indicator-level feedback: the feed-forward model $Y_1 = X(\Theta_2 Y_2 + U) + \mathcal{E}$ with a full random-effect covariance against the feedback model with $\Theta_1 \neq 0$. This is the only inferential step of the procedure (see the workflow below); nmf.ffb.inference is for the uncertainty of the coefficients *after* the null has been rejected, and nmf.ffb.diagnostics for what the selected feedback looks like.

Usage

```
nmf.ffb.test(
  object,
  Y1,
  Y2,
  B = 1000L,
  calibration = c("procedure", "conditional"),
  seed = 123L,
  ...
)
```

Arguments

object	A fit from nmf.ffb with method = "fiml".
Y1, Y2	The endogenous and exogenous matrices the fit was computed from (variables in rows, units in columns).
B	Number of bootstrap replicates (default 1000). The smallest reportable p-value is $1/(1+B)$.
calibration	What is re-applied to each null replicate; see above. "conditional" is selected automatically, with a warning, when the fit used a basis or an exclusion restriction supplied by the caller, since there is then nothing to re-estimate.
seed	Base seed; replicate b uses seed + b. The caller's random stream is restored on exit.

`...` Passed to the bootstrap: `cores` (or `ncores`) to run the replicates in parallel, defaulting to `getOption("mc.cores", 1L)` as elsewhere in the package; `fiml.maxit` and `factr` to control the optimizer in the replicates; `print.trace = TRUE` for progress.

Details

The likelihood ratio does not have a χ^2 null distribution: the constraint $\Theta_1 \geq 0$ puts the null on the boundary of the parameter space, and the support of Θ_1 is chosen from the same data. It is therefore calibrated by a parametric bootstrap from the fitted feed-forward null,

$$Y_1^* = (\hat{X}(\hat{\Theta}_2 Y_2 + U^*) + \mathcal{E}^*)_+, \quad U^* \sim N(0, \hat{\Phi}),\ \mathcal{E}^* \sim N(0, \hat{\Psi}),$$

with negative draws clipped at zero, and the reported p-value is $(1 + \#\{LR^* \geq LR\})/(1 + B_{ok})$.

What is re-applied to each replicate: `calibration = "procedure"` (the default) re-applies the *whole procedure*: stage 1 is re-run on each Y_1^*, the basis is re-estimated, the exclusion restriction is re-derived from that basis, and stage 2 follows. The p-value is then the operating characteristic of the complete exploratory procedure, which is what is applied to a fresh data set. This is the calibration to use.

`"conditional"` holds $\hat{X}$ and the exclusion restriction at their fitted values and re-runs stage 2 only. It is valid only if basis and restriction are independent of the Y_1 being tested, which they are not when all three come from the same sample; it is provided for comparison with that practice, and it is anti-conservative.

Sample splitting – basis and restriction from one half of the units, test on other – also removes the dependence, and was offered here in 0.9.8. It was withdrawn in 0.9.8: measured against `"procedure"` it has the same size and lower power, because the test uses $N/2$ units, and it cannot be run at all when the halves are too small for stage 1.

Reading the result: Report `LR.p.boot` together with `prob.select.null` and, for `calibration = "procedure"`, `C1.restriction.change.rate`. The last two say whether the exclusion restriction is determined well enough for the test to mean anything: if the dominant factor of an outcome moves from one null replicate to the next, a previously blocked entry becomes free and a feedback coefficient can absorb loading structure that the feed-forward model attributes to X. A large `prob.select.null` with a large null quantile is the signature of a data set in which the procedure finds feedback whether or not there is any.

Value

An object of class `"nmf.ffb.test"`:

LR, LR.df Observed `c(full, selected)` likelihood ratios and the number of free feedback entries left by the exclusion restriction.

LR.p.boot Calibrated p-values of the two statistics.

LR.null.quantile The 0.95 quantiles of the null distributions.

LR.boot $B \times 2$ matrix of the null replicates.

prob.select.null Null false-selection rate: the probability that BIC retains at least one feedback entry when the feed-forward model is true.

C1.restriction.change.rate (`"procedure"` only) share of null replicates in which the exclusion restriction moved.

LR.boot.df Free entries per null replicate (varies under `"procedure"`).

LR.boot.n.ok, LR.boot.n.nonconv Usable replicates, and how many did not meet the optimizer tolerance.

Workflow

```
ecv <- nmf.ffb.ecv(Y1, Y2, rank = 1:5)   # 1. choose Q
fit <- nmf.ffb(Y1, Y2, rank = ecv$rank)  # 2. estimate; BIC selects the support
tst <- nmf.ffb.test(fit, Y1, Y2)      # 3. test the feed-forward null <- only inference
dgn <- nmf.ffb.diagnostics(fit)       # 4. cycles, spectral radius, identifiability
inf <- nmf.ffb.inference(fit, Y1, Y2)    # 5. intervals for the retained entries
```

See Also

`nmf.ffb`, `nmf.ffb.ecv`, `nmf.ffb.inference`, `nmf.ffb.diagnostics`

`nmf.gmm` *Fit NMF-GMM: a Gaussian-mixture latent-class extension of NMF with covariates*

Description

This function is **experimental and still under development**. The interface may change in future versions: argument names, defaults and the contents of the returned object are not yet stable. Code written against it today may need adjusting after an update. The rest of the package does not carry this caveat.

`nmf.gmm` fits the model

$$\boldsymbol{b}_n \mid (z_n = k) \sim N_Q(C\boldsymbol{a}_n + \boldsymbol{\mu}_k, \Sigma_k), \qquad \boldsymbol{y}_n = X\boldsymbol{b}_n + \boldsymbol{\varepsilon}_n, \quad X \geq 0,$$

by a generalized EM algorithm: a K-component Gaussian mixture is placed on the latent NMF scores $\boldsymbol{b}_n$, whose class-conditional mean combines a covariate regression $C\boldsymbol{a}_n$ ($C = \Theta$ in the paper, the $Q \times R$ coefficient matrix) with a class shift $\boldsymbol{\mu}_k$. The shared basis X stays non-negative and column-normalized. Clustering is model based, through the posterior responsibilities. $K = 1$ (tied covariance) reduces exactly to NMF with random effects (`nmfre`).

Like the rest of the package, `nmf.gmm` performs **optimization only**; use `nmf.gmm.inference` for standard errors and tests of C, and `nmf.gmm.select` to choose K.

Usage

```
nmf.gmm(Y, A = NULL, rank, K = 1, ...)
```

Arguments

Y	Data matrix Y (P x N).
A	Covariate matrix A (R x N). Default NULL uses an intercept-only $1 \times N$ matrix (a plain Gaussian mixture on the scores, with no covariate adjustment). A row of ones should be present for the identifiability constraint $\sum_k \xi_k \boldsymbol{\mu}_k = 0$ (see intercept).
rank	Integer rank Q of the basis.
K	Integer number of mixture components. Default 1 (= NMF-RE).
...	Additional arguments:

- cov: score-covariance structure. "tied" (default, shared diagonal $\Sigma_k = \mathrm{diag}(\tau_1^2, \ldots, \tau_Q^2)$, Q variances); "free" (per-class diagonal, $Q \times K$); or "scalar" (isotropic $\Sigma_k = \tau^2 I_Q$, a single variance — the most parsimonious variant; at $K = 1$ it is the nmfre model, and tau2 is returned as one number).
- X.init: initial basis. A $P \times Q$ non-negative matrix, or an initialization method name: "nndsvd" (default), "kmeans++", "kmeans", ... (passed to the shared initializer), or "nmf", the basis of an ordinary NMF fitted by nmfkc(Y, rank = rank, seed = seed) without covariates (requires a non-negative Y; nmfkc's own default seed is 123). The EM restarts (nstart) all start from this one basis and differ only in the seeding of the mixture, so the fit can depend on the choice of X.init; the basis actually used is returned as X0. "spa" starts from the anchor rows the successive projection algorithm finds in Y (see nmfkc; requires a non-negative Y).
- X.anchor: anchor rows of X, imposed as in nmfkc: NULL (default), "spa", or one row of Y per basis (requires a non-negative Y). The EM's multiplicative basis update keeps the zeros. There is no rank <= nrow(A) limit here: the mixture gives the scores full rank. nmf.gmm.twostage() accepts it too, finding the anchors on Y. The rows used are returned as X.anchor.
- intercept: index of the intercept row of A (default 1); the class-mean average is absorbed into that column of C.
- nstart: EM restarts (default 1 for K=1, else 8).
- cores: run the nstart restarts in parallel (default getOption("mc.cores", 1L); PSOCK cluster on Windows, forking elsewhere). Each restart is an independent self-seeded EM and results are combined in order, so the returned fit is identical for any cores. (nmf.gmm.select parallelizes over K instead and runs each inner fit with cores = 1.)
- maxit, tol: outer EM cap / tolerance (2e5, 1e-7). tol is the relative change of the marginal log-likelihood, the usual EM rule.
- ptol: a second stopping rule, on the relative change of the *reported* parameters X, C and $\boldsymbol{\mu}$ (default 1e-9; NULL disables it). The log-likelihood rule alone can fail to fire when a nuisance variance drifts to its boundary – $\tau^2 \to 0$ on a sample that carries no random effect – so the objective keeps creeping up after the partition, the basis and the covariate effect have settled. The returned stop_by says which rule fired ("loglik", "parameters" or "maxit").

- `vfloor`: lower bound on the variance parameters. The default (`NULL`) makes it scale-relative, $10^{-12}\,\mathrm{mean}(Y^2)$, so the fit is equivariant under $Y \to cY$; a fixed bound would bind on small-scale data.
- `fixX`: hold the basis at its initial value instead of updating it (default `FALSE`). Used to compare the joint fit with `nmf.gmm.twostage` on the *same* X, so that the two routes share the basis. They still fit different data: the two-stage route fits the reconstituted, shifted residuals.
- `class.effects`: covariates whose coefficients differ by class (default `NULL`: every covariate has one coefficient shared by all classes). Row indices of `A`, row names, or — when `A` is a formula — term names, so `class.effects = "diet"` frees every indicator row of a factor `diet`. The class-k mean becomes $C\boldsymbol{a}_n + \Delta_k \boldsymbol{a}_{n,S} + \boldsymbol{\mu}_k$ with $\sum_k \xi_k \Delta_k = 0$, so the columns S of `C` hold the ξ-weighted average of the class-specific coefficients, which are returned in `C.class`. The intercept row cannot be freed: $\boldsymbol{\mu}_k$ already makes it class-specific. With a continuous covariate, the class-specific coefficients are identified when the covariate varies within every class. With a categorical covariate, freeing all of its indicator rows gives every level its own class means, so the likelihood no longer ties the classes of one level to those of another: which class of one level corresponds to which class of another is then carried by the starting values rather than by the data. Not supported by `nmf.gmm.inference` or `nmf.gmm.twostage`.
- `seed`: RNG seed (default 1). `prefix`: basis-name prefix (default `"Basis"`).
- `data`: a data frame with one row per column of `Y`, required when `A` is a formula (see below).
- `standardize`: when `A` is a formula, center and scale the constructed covariate columns (default `TRUE`). Factors are expanded to treatment indicators before standardization, so a 5-level factor becomes four centered, scaled columns.

`A` may also be a one-sided **formula** (e.g. `~ size` or `~ diet`), evaluated in `data`: the design matrix is built with `model.matrix`, its intercept column is replaced by the package's own intercept row, and the remaining columns are standardized by default. The constructed numeric A is returned in the fit (fields `A`, `A.formula`, `A.center`, `A.scale`), so `nmf.gmm.inference` works unchanged.

Value

An object of class `"nmf.gmm"`: a list with `X` (basis), `X0` (the column-normalized initial basis the EM started from), `C` (= Θ, Q x R), `mu` (class means, Q x K), `tau2`, `sigma2`, `xi` (mixing proportions), `gamma` (responsibilities, N x K), `cluster` (hard labels), `loglik`, `BIC`, `ICL`, `n.params`, `entropy`, `Yhat`, `objfunc(.iter)`, `iter`, `converged`, `stop_by`, `K`, `rank`, `cov`, `dims` and `runtime`. With `class.effects`, also `class.effects` (the freed rows of `A`, by name), `class.rows` (their indices) and `C.class` (a Q x |S| x K array of class-specific coefficients; the class-specific effect in observation space is `X %*% C.class[, , k]`).

See Also

`nmf.gmm.inference`, `nmf.gmm.select`, `nmfre` (the $K = 1$ special case).

Examples

```
set.seed(1)
Y <- matrix(abs(rnorm(20 * 60)) + 1, 20, 60)
A <- rbind(1, rnorm(60))                # intercept + one covariate
fit <- nmf.gmm(Y, A, rank = 2, K = 2)
fit$BIC
table(fit$cluster)
```

`nmf.gmm.inference` *Statistical inference for an NMF-GMM fit (given basis)*

Description

This function is **experimental and still under development**. The interface may change in future versions: argument names, defaults and the contents of the returned object are not yet stable. Code written against it today may need adjusting after an update. The rest of the package does not carry this caveat.

`nmf.gmm.inference` adds Wald inference on the covariate-coefficient matrix C ($= \Theta$) of a fitted `nmf.gmm` object, conditional on the estimated basis $\hat{X}$ and mixture. It reports the analytic outer-product (mixture-information) standard error — the coverage-calibrated choice at $K > 1$ — together with a **wild-bootstrap** standard error and percentile confidence interval, mirroring `nmfre.inference` / `nmfkc.inference`. Supported for the tied-covariance variant.

Usage

```
nmf.gmm.inference(object, Y, A = object$A, ...)
```

Arguments

`object`	A fitted `"nmf.gmm"` object (with `cov = "tied"`).
`Y`	The data matrix used in the fit.
`A`	The covariate matrix used in the fit. Default `object$A`.
`...`	Additional arguments, named as in the other wild-bootstrap inference functions of the package: `wild.B` (bootstrap replicates, default 500), `wild.level` (confidence level, default 0.95) and `seed` (default 123).

Value

`object` with class `c("nmf.gmm.inference", "nmf.gmm")` and added components: `coefficients` (data frame with `Basis`, `Covariate`, `Estimate`, `SE`, `BSE`, `z_value`, `p_value`, `CI_low`, `CI_high`), `C.se.outer`, `C.se.sandwich`, `C.se.boot` (Q x R matrices) and `Xi` ($= XC$).

See Also

`nmf.gmm`, `nmf.gmm.select`

Examples

```
set.seed(1)
Y <- matrix(abs(rnorm(20 * 60)) + 1, 20, 60); A <- rbind(1, rnorm(60))
fit <- nmf.gmm(Y, A, rank = 2, K = 2)
fit <- nmf.gmm.inference(fit, Y, A, wild.B = 300)
head(fit$coefficients)
```

`nmf.gmm.select` *Choose the number of mixture components K for NMF-GMM*

Description

This function is **experimental and still under development**. The interface may change in future versions: argument names, defaults and the contents of the returned object are not yet stable. Code written against it today may need adjusting after an update. The rest of the package does not carry this caveat.

`nmf.gmm.select` fits `nmf.gmm` over a vector of K values and reports the log-likelihood, BIC and ICL for each, selecting `K.best` by BIC (`K.best.icl` by ICL). If a vector of known labels is supplied via `truth`, the adjusted Rand index of the hard clustering is added.

Usage

```
nmf.gmm.select(Y, A = NULL, rank, K = 1:5, ...)
```

Arguments

`Y`, `A`, `rank`	As in `nmf.gmm`.
`K`	Integer vector of candidate component counts. Default `1:5`.
`...`	Additional arguments passed to `nmf.gmm` (e.g. `cov`, `X.init`, `nstart`, `maxit`, `seed`). Also accepts `truth` (a length-N vector of known class labels for the ARI column), `verbose` (logical, print the table; default `TRUE`), and `cores` (evaluate the `K` candidates in parallel; default `getOption("mc.cores", 1L)`). Parallelism uses a PSOCK cluster on Windows and forking elsewhere; because each `K` is an independent self-seeded fit and results are returned in order, the table and the selected `K` are identical for any `cores`.

Value

An object of class `"nmf.gmm.select"`: a list with the `table` (data frame over `K`), `K.best` (min BIC), `K.best.icl` (min ICL) and `fits` (the fitted objects).

See Also

`nmf.gmm`

Examples

```
set.seed(1)
Y <- matrix(abs(rnorm(20 * 80)) + 1, 20, 80); A <- rbind(1, rnorm(80))
sel <- nmf.gmm.select(Y, A, rank = 2, K = 1:4, verbose = FALSE)
sel$K.best
```

`nmf.gmm.twostage` *Two-stage (adjust-then-cluster) baseline for NMF-GMM*

Description

This function is **experimental and still under development**. The interface may change in future versions: argument names, defaults and the contents of the returned object are not yet stable. Code written against it today may need adjusting after an update. The rest of the package does not carry this caveat.

`nmf.gmm.twostage` runs the *two-stage* route that `nmf.gmm` is designed to improve on, as a matched baseline: (1) estimate least-squares scores on the initial basis, (2) regress the scores on the covariates *blind to the class* and keep the residuals, (3) reconstitute the residuals in observation space, shift them to non-negativity, and (4) refit an intercept-only `nmf.gmm` from the *same* basis initialization. Both routes start from the same basis and use the same mixture family and settings. They differ in the order of adjustment and clustering and, unless `fixX = TRUE`, in the basis each refits to its own data: the two-stage route refits it to the reconstituted, shifted residuals. When the covariate is associated with the class, the class-blind regression of stage (2) displaces the class means (the oracle proposition of Satoh 2026); when the covariate is (near-)mean-independent of the class, that displacement vanishes in the population, so both routes target the same class means, although the fitted partitions need not coincide in a finite sample.

Usage

```
nmf.gmm.twostage(Y, A = NULL, rank, K = 1, ...)
```

Arguments

`Y`	Data matrix Y (P x N).
`A`	Covariate matrix A (R x N) including an intercept row, or a one-sided formula evaluated in `data` (as in `nmf.gmm`). An intercept-only `A` is an error: there is nothing to adjust for.
`rank`	Integer rank Q of the basis.
`K`	Integer number of mixture components.
`...`	Additional arguments as in `nmf.gmm` (`cov`, `X.init`, `nstart`, `maxit`, `seed`, `data`, `standardize`, ...); they are applied to both stages.

Value

An object of class `c("nmf.gmm.twostage", "nmf.gmm")`: the stage-2 fit (all `nmf.gmm` fields and S3 methods apply), plus a `twostage` list with the non-negativity `shift`, the covariate matrix `A` that was removed, and the shared basis initialization `X0`.

See Also

`nmf.gmm` (the joint route this baselines).

Examples

```
set.seed(1)
Y <- matrix(abs(rnorm(12 * 40)) + 1, 12, 40)
A <- rbind(1, rnorm(40))
ts <- nmf.gmm.twostage(Y, A, rank = 2, K = 2, nstart = 2, maxit = 100)
table(ts$cluster)
```

`nmf.rrr` *Three-Layer Non-negative Matrix Factorization (NMF-AE)*

Description

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

When `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

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

Arguments

Y1	Output matrix Y_1 (P1 x N). Non-negative. May contain NAs (handled via Y1.weights).
Y2	Input matrix Y_2 (P2 x N). Non-negative. Default is Y1 (autoencoder).
rank1	Integer. Rank of the response basis X_1 (P1 x Q). Default is 2.
rank2	Integer. Rank of the covariate basis X_2 (R x P2). Default (NULL) sets rank2 = rank1.
epsilon	Positive convergence tolerance. Default is 1e-4.
maxit	Maximum number of multiplicative update iterations. Default is 5000.
verbose	Logical. If TRUE, prints progress messages during fitting. Default is FALSE.
...	Additional arguments:

method Objective function: Euclidean distance "EU" (default) or Kullback-Leibler divergence "KL". Both use Lee-Seung multiplicative updates for the three factors X_1, Θ, X_2 of $Y_1 \approx X_1 \Theta X_2 Y_2$; for "KL" the residual SE sigma is NA (not on the data scale).

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

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

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

X.init Initialization method for the two bases, passed to the internal nmfkc steps: "kmeans" (default), "nndsvd", "spa", ...

X.anchor Anchor rows of the response basis X_1, imposed as in nmfkc: NULL (default), "spa", or one row of Y1 per basis. The zeros are kept for the whole fit. The scores of X_1 are $\Theta X_2 Y_2$, so rank1 <= min(rank2, nrow(Y2)) is required. The rows used are returned as X.anchor.

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

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).
`B1`	Decoder-side scores $B_1 = C X_2 Y_2$ (Q x N), so that $\widehat{Y}_1 = X_1 B_1$. Analogue of `B` in `nmfkc`.
`H`	**Deprecated**; identical to `B1`. Kept so that code written against earlier releases keeps working. Use `B1`.
`B2`	Encoder-side scores $B_2 = X_2 Y_2$ (R x N), i.e. B_1 before the C map.
`B1.prob`, `B2.prob`	Column-normalized B_1 / B_2: each SAMPLE's soft membership over the Q response groups and over the R covariate groups respectively.
`B1.cluster`, `B2.cluster`	Hard label per sample (argmax over the columns of the corresponding `.prob`).
`X1.prob`	Row-normalized X_1 (P1 x Q): each RESPONSE VARIABLE's soft membership over the Q response groups.
`X1.cluster`	Hard response-group label per response variable (argmax over `X1.prob` rows).
`X2.prob`	Column-normalized X_2 (R x P2): each COVARIATE VARIABLE's soft membership over the R covariate groups.
`X2.cluster`	Hard covariate-group label per covariate variable (argmax over `X2.prob` columns).
`rank`	Named integer vector `c(Q, R)`.
`method`	Objective used (`"EU"` or `"KL"`).
`objfunc`	Final objective value.
`objfunc.iter`	Objective values by iteration.
`r.squared`	$\mathrm{cor}(Y, \widehat{Y})^2$ (Pearson; in $[0, 1]$).
`r.squared.uncentered`	Uncentered $R^2 = 1 - \|Y - \widehat{Y}\|_F^2 / \|Y\|_F^2$ (baseline = zero matrix).
`r.squared.centered`	Row-mean centered $1 - \|Y - \widehat{Y}\|_F^2 / \|Y - \bar{Y}_{p\cdot}\|_F^2$.
`niter`	Number of iterations performed.
`runtime`	Elapsed time as a `difftime` object.
`n.missing`	Number of missing (or zero-weighted) elements in Y_1.
`n.total`	Total number of elements in Y_1 (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

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

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

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

See Also

`nmf.rrr.inference`, `predict.nmfae`, `nmf.rrr.ecv`, `nmf.rrr.DOT`, `nmfkc`

Examples

```
# Autoencoder example
Y <- matrix(c(1,0,1,0, 0,1,0,1, 1,1,0,0), nrow=3, byrow=TRUE)
res <- nmf.rrr(Y, rank1=2, rank2=2)
res$r.squared

# Heteroencoder example
Y1 <- matrix(c(1,0,0,1), nrow=2)
Y2 <- matrix(runif(8), nrow=4)
res2 <- nmf.rrr(Y1, Y2, rank1=2, rank2=2)
```

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

Description

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

When `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

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

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).
rank1	Integer. Rank of the response basis. Default is 2.
rank2	Integer. Rank of the covariate basis. Default (NULL) = rank1.
...	Additional arguments passed to nmf.rrr (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), and cores (fit the folds in parallel; default getOption("mc.cores", 1L), PSOCK on Windows / forking elsewhere). Each fold is an independent self-seeded fit and the per-fold errors are summed in order, so the result is identical for any cores. For backward compatibility, Q, R are accepted as aliases for rank, rank.encoder. Rank aliases accepted here for backward compatibility: Q for rank1, R for rank2.

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

nmf.rrr, nmf.rrr.ecv, nmf.rrr.kernel.beta.cv, nmfkc.cv

Examples

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

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

Description

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

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

Usage

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

Arguments

`result`	An object of class `"nmfae"` returned by `nmf.rrr`.
`type`	Character. Graph type: `"XCX"` (default) or `"YXCXY"`.
`threshold`	Numeric. Edges with values below this are omitted. Default is 0.01.
`sig.level`	Numeric or `NULL`. Significance level for filtering C edges when inference results are available. Only edges with p-value below `sig.level` are shown, with significance stars. Set to `NULL` to disable. Default is 0.1.
`rankdir`	Character. Graph direction for DOT layout. Default is `"LR"` (left to right).
`fill`	Logical. If `TRUE`, nodes are filled with color. Default is `TRUE`.

`weight_scale` Numeric. Base scale factor for edge widths. Default is 5.

`weight_scale_x1`
Numeric. Scale factor for X_1 edges.

`weight_scale_theta`
Numeric. Scale factor for Θ edges.

`weight_scale_x2`
Numeric. Scale factor for X_2 edges.

`Y1.label` Character vector of output variable labels.

`X1.label` Character vector of decoder basis labels.

`X2.label` Character vector of encoder basis labels.

`Y2.label` Character vector of input variable labels.

`Y1.title` Character. Title for output node group. Default is `"Output (Y1)"`.

`X1.title` Character. Title for the response-basis node group. Default is `"Response (X1)"`.

`X2.title` Character. Title for the covariate-basis node group. Default is `"Covariate (X2)"`.

`Y2.title` Character. Title for input node group. Default is `"Input (Y2)"`.

`hide.isolated` Logical. If `TRUE` (default), Y1 and Y2 nodes that have no edges at or above `threshold` are excluded from the graph. Only applies when `type = "YXCXY"`.

Value

A character string containing the DOT graph specification.

Lifecycle

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

See Also

`nmf.rrr`

Examples

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

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

Description

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

This method (also known as Wold's CV) is suitable for determining the optimal rank pair (Q, R) in three-layer NMF. Both `rank1` and `rank2` accept vector inputs. When `rank2 = NULL` (default), `rank2` is set equal to `rank1` and pairs are evaluated element-wise (i.e., $(Q_1, R_1), (Q_2, R_2), \dots$). When `rank.encoder` is explicitly specified, all combinations of `rank` and `rank.encoder` are evaluated via `expand.grid`.

Usage

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

Arguments

`Y1`	Output matrix Y_1 (P1 x N).
`Y2`	Input matrix Y_2 (P2 x N). Default is `Y1`.
`rank1`	Integer vector of response-basis ranks to evaluate. Default is `1:2`.
`rank2`	Integer vector of covariate-basis ranks to evaluate. Default is `NULL`, which sets `rank2 = rank1` and evaluates element-wise pairs.
`...`	Additional arguments passed to `nmf.rrr` (e.g., `epsilon`, `maxit`). Also accepts: `nfolds` (number of folds, default 5; `div` also accepted), `seed` (integer seed, default 123), and `cores` (evaluate the (Q, R)-pair $\times$ fold grid in parallel; default `getOption("mc.cores", 1L)`, PSOCK on Windows / forking elsewhere). Each task is an independent self-seeded fit and results are aggregated in order, so the returned object is identical for any `cores`. For backward compatibility, `Q` and `R` are accepted as aliases for `rank` and `rank.encoder`. `Y1.weights` (a matrix, one weight per column, or a scalar) is kept in every fit; cells of weight 0 are never held out, and the held-out loss is the weighted mean. (Before 1.0.0 a `Y1.weights` given here was silently replaced by the fold mask.) Rank aliases accepted here for backward compatibility: `Q` for `rank1`, `R` for `rank2`.

Value

A list with components:

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

Lifecycle

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

See Also

nmf.rrr, nmfkc.ecv

Examples

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

---

nmf.rrr.heatmap *Heatmap visualization of nmfae factor matrices*

Description

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

Usage

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

Arguments

x	An object of class "nmfae" returned by nmf.rrr.
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

`nmf.rrr`, `plot.nmfae`, `nmf.rrr.DOT`

Examples

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

`nmf.rrr.inference` *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

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

Arguments

`object`	An object of class `"nmfae"` returned by `nmf.rrr`.
`Y1`	Output matrix Y_1 (P1 x N). Must match the data used in `nmf.rrr()`.
`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 500.

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

`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.

`identified.approx` Whether the factorization is unique up to permutations of both latent spaces, i.e. under $X_1 \to X_1 T_1$, $\Theta \to T_1^{-1}\Theta T_2^{-1}$, $X_2 \to T_2 X_2$. A sufficient condition: the diagnostic of `nmfkc.inference` passes on $(X_1, \Theta X_2)$ (pinning T_1) and on $(X_1\Theta, X_2)$ (pinning T_2), and Y_2 has full row rank. Zeros are judged relative only (`pin.tol`, `sep.tol` through `...`). An algebraic check on the point estimate: no re-fits.

`identification` The two checks, `left` and `right`.

Lifecycle

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

See Also

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

Examples

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

nmf.rrr.kernel.beta.cv

Optimize kernel beta for nmfae by cross-validation

Description

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

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

Usage

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

Arguments

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

Value

A list with components:

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

See Also

nmf.rrr.cv, nmfkc.kernel, nmfkc.kernel.beta.cv

Examples

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

---

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

Description

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

Usage

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

Arguments

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

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

Value

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

References

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

See Also

nmf.rrr, nmf.rrr.ecv, nmfkc.rank

---

nmf.rrr.rename *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

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

Arguments

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

Value

A modified copy of x with updated names.

See Also

nmf.rrr

Examples

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

---

nmf.rrr.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 nmf.rrr: 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

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

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

Arguments

`Y1`	Output matrix Y_1 (P1 x N). Negative entries allowed (`Y.signed = TRUE` is auto-detected).
`Y2`	Input matrix Y_2 (P2 x N). Must be non-negative. Default `Y1` (autoencoder).
`rank1`	Integer. Response-basis rank Q. Default 2.
`rank2`	Integer. Covariate-basis rank R. Default (`NULL`) = `rank1`.
`epsilon`	Relative convergence tolerance on the objective. Default `1e-4`.
`maxit`	Maximum iterations. Default 5000.
`verbose`	Logical. Print progress. Default `FALSE`.
`...`	Additional arguments:

`warm.start` One of `TRUE` (default, **hybrid**: warm-start X_1, X_2 from `nmf.rrr` 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.

`cores` Run the `nstart` restarts in parallel (only when `nstart > 1`). Default `getOption("mc.cores", 1L)` (PSOCK cluster on Windows, forking elsewhere). Each restart is fully self-seeded and the best run is selected by first-minimum objective, so the returned fit is identical for any `cores`.

`X1.L2.ortho`, `X2.L2.ortho` Non-negative L2 orthogonality penalties (default 0) on the **columns** of X_1 and the **rows** of X_2, penalizing $(\lambda/2)\|\text{offdiag}(X_1^\top X_1)\|^2$ and $(\lambda/2)\|\text{offdiag}(X_2 X_2^\top)\|^2$ respectively. Same convention as `nmf.rrr`; encourage more distinct (less overlapping) response / covariate bases.

`C.L2` Non-negative ridge penalty (default 0) on the signed bottleneck $C = C_+ - C_-$, adding $\lambda\|C_+ - C_-\|^2$. Shrinks Θ toward zero (with zero gradient on the unidentified common mode $C_+ + C_-$), injected into both the unweighted and weighted C_+/C_- updates.

`Y1.weights` Optional non-negative weight matrix (P1 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 `nmf.rrr.signed.ecv` to hold out test elements via a binary mask $W \in \{0, 1\}$; real-valued weights for importance weighting are also supported. Default: if `Y1` has `NA`, a binary mask is auto-generated (0 for `NA`, 1 elsewhere).

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

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

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

`seed` RNG seed. Default 123.

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

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

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

Value

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

`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 $X_1(C_+ - C_-)X_2Y_2$.

`B1` Decoder-side scores $B_1 = (C_+ - C_-)X_2Y_2$ (Q x N), so that $\widehat{Y}_1 = X_1B_1$. **Signed**, since C is.

`B2` Encoder-side scores $B_2 = X_2Y_2$ (R x N), i.e. B_1 before the C map. Non-negative.

`H` **Deprecated**; identical to `B1`. Kept so that code written against earlier releases keeps working. Use `B1`.

`B.prob`, `B.cluster` Sample-level soft/hard clustering from a non-negative clip of the (possibly signed) B_1; use with care since C may be signed. There are deliberately no `B1.prob` / `B2.prob` counterparts: a column-normalized membership is not a distribution when the scores can be negative.

`X1.prob` Row-normalized X_1 (P1 x Q): each RESPONSE VARIABLE's soft membership over the Q response groups. Well defined ($X_1 \geq 0$).

`X1.cluster` Hard response-group label per response variable.

`X2.prob` Column-normalized X_2 (R x P2): each COVARIATE VARIABLE's soft membership over the R covariate groups. Well defined ($X_2 \geq 0$).

`X2.cluster` Hard covariate-group label per covariate variable.

`rank` `c(Q = Q, R = R)`.

`dims` `c(P1, P2, N)`.

`objfunc`, `objfunc.iter`
Final and per-iteration objective values.

`r.squared` $\mathrm{cor}(Y_1, \widehat{Y}_1)^2$ (Pearson; in $[0, 1]$).

`r.squared.uncentered`
Uncentered $R^2 = 1 - \|Y_1 - \widehat{Y}_1\|_F^2 / \|Y_1\|_F^2$ (baseline = zero matrix).

`r.squared.centered`
Row-mean centered $1 - \|Y_1 - \widehat{Y}_1\|_F^2 / \|Y_1 - \bar{Y}_{1\cdot}\|_F^2$.

`sigma`, `mae` Residual SE and mean absolute error.

`niter`, `runtime` Iterations and elapsed seconds.

`Y.signed` Logical; whether Y_1 contained negative entries.

`call` Matched call.

Lifecycle

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

References

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

Ding, C.H.Q., Li, T., and Jordan, M.I. (2010). Convex and Semi-Nonnegative Matrix Factorizations. *IEEE TPAMI*, 32(1), 45-55.

Satoh, K. (2026). Signed-Bottleneck NMF-AE: Signed-Bottleneck 3-Layer NMF (research memo, 2026-04-18).

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

See Also

`nmf.rrr`, `predict.nmfae.signed`, `summary.nmfae.signed`, `nmf.rrr.signed.rename`

Examples

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

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

Description

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

Usage

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

Arguments

`Y1`	Output matrix (P1 x N).
`Y2`	Input matrix (P2 x N). Default `Y1`.
`rank1`	Integer vector of candidate response-basis ranks. Default `1:2`.
`rank2`	Integer vector of candidate covariate-basis ranks, or `NULL` (default: pair rank2 = rank1, diagonal grid).
`...`	Additional arguments:
	`nfolds` / `div` Number of folds. Default 5.
	`seed` RNG seed for fold assignment. Default 123.
	`Y1.weights` A matrix, one weight per column, or a scalar; kept in every fit. Cells of weight 0 are never held out, and the held-out loss is the weighted mean. (Before 1.0.0 it was silently replaced by the fold mask.)
	`cores` Evaluate the (Q, R)-pair $\times$ fold grid in parallel. Default `getOption("mc.cores", 1L)` (PSOCK on Windows, forking elsewhere). Each task is an independent self-seeded fit and results are aggregated in order, so the returned object is identical for any `cores`.
	`nstart` Number of random restarts per fit. Default 1. Signed models have more local minima (the bottleneck can carry both signs), so `nstart >= 10` is recommended for reproducible rank selection.
	Other args `epsilon`, `maxit`, `warm.start`, etc.\ are passed to `nmf.rrr.signed`.
	Rank aliases accepted here for backward compatibility: `Q` for `rank1`, `R` for `rank2`.

Value

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

Lifecycle

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

References

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

See Also

`nmf.rrr.signed`, `nmf.rrr.ecv`

`nmf.rrr.signed.heatmap`

Heatmap visualization of nmfae.signed factor matrices

Description

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

Usage

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

Arguments

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

Value

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

Lifecycle

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

References

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

See Also

`nmf.rrr.signed`, `nmf.rrr.heatmap`

Examples

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

---

nmf.rrr.signed.inference

Statistical Inference for Signed-Bottleneck NMF-AE Signed Bottleneck

---

Description

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

Usage

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

Arguments

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

`wild.B` Bootstrap replicates. Default 500.

`wild.seed` RNG seed. Default 123.

`wild.level` CI confidence level. Default 0.95.

`sandwich` Use sandwich covariance. Default `TRUE`.

`C.p.side` P-value type: `"two.sided"` (default for Signed-Bottleneck NMF-AE) or `"one.sided"`.

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

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

Value

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

`sigma2.used`	Estimated σ^2.
`C.se`, `C.se.boot`	Sandwich / bootstrap SEs for Θ (Q 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

`nmf.rrr.signed`, `nmf.rrr.inference`

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

Description

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

Usage

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

Arguments

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

Value

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

References

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

See Also

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

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

Description

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

Usage

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

Arguments

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

Value

The renamed object (same class).

Lifecycle

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

References

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

See Also

`nmf.rrr.signed`

nmfkc *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). For large N, `A` may instead be a Gram object of class `"nmfkc.gram"` from `nmfkc.kernel.gram` (blockwise Nystr\"om kernel covariates), which holds only $S = AA^\top$ ($D \times D$) and $G_0 = YA^\top$ ($P \times D$) so that the $D \times N$ matrix never has to exist in memory. The Euclidean multiplicative updates are unchanged (they only use those two products) and the fit equals the explicit-matrix fit up to floating-point summation order. Gram input is restricted to `method = "EU"` without `Y.weights` or `NA` in `Y`; `B`, `XB` and the fit statistics are rebuilt block by block after the fit; and the resampling helpers (`nmfkc.cv`, `nmfkc.ecv`, `nmfkc.rank`) do not accept it. Gram objects built from signed features (`nmfkc.signed.rff.gram`) are refused here; they belong to `nmfkc.signed`. 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. From the 10th iteration the fit stops when $|f_i - f_{i-1}|/\max(|f_i|, 1) \leq$ `epsilon`, where f_i is the objective at iteration i. The rule is relative while the objective exceeds 1 and *absolute* below it, so on data of small magnitude (an objective well below 1) the default `1e-4` can stop early, far from the minimum. Rescale `Y` or lower `epsilon` (e.g.\ `1e-8`) in that case. Coefficients that sit on the non-negativity boundary also need a tight `epsilon`; see `C.init` under `...`.

`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`). The penalty $(\lambda/2)\sum_{q\neq r}(x_q^\top x_r)^2$ is added to the objective as it stands, so how strongly a given λ acts depends on the scale of `Y`; with column-sum-normalized X each term is also of order $1/P^2$, so small values such as $\lambda \leq 0.1$ usually do nothing. To state λ relative to the fit, pass `X.L2.ortho = lambda0 * sum(Y^2)`.
- `X.L2.smooth`: Nonnegative penalty parameter for row-smoothness of X (default: 0). Adds $\lambda\,\mathrm{tr}(X^\top L X)$ with L the path-graph Laplacian over the P rows, i.e.\ it penalizes squared differences between adjacent rows $\sum_q \sum_{j\geq 2}(x_{jq} - x_{j-1,q})^2$, yielding gently-varying (smooth) bases. Integrates with the multiplicative updates (non-negativity and monotone descent preserved). Useful when the rows of Y have a natural order (e.g.\ time points).
- `C.L1`: Nonnegative penalty parameter for L1 regularization on C (default: 0). Promotes **sparsity in the parameter matrix** Θ (variable selection over basis-covariate links). (Formerly `lambda`).
- `Q`: Backward-compatible name for the rank of the basis matrix (Q).
- `method`: Objective function: Euclidean distance `"EU"` (default) or Kullback–Leibler divergence `"KL"`.
- `X.restriction`: Constraint for columns of X. Options: `"colSums"` (default), `"colSqSums"`, `"totalSum"`, `"none"`, or `"fixed"`. `"none"` applies no normalization to X after each update, allowing it to absorb the scale freely.

- X.init: Method for initializing the basis matrix X. Options: "kmeans" (default), "kmeansar", "kmeans++", "runif", "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. "kmeans++" seeds the k-means centres by D^2 weighting (Arthur & Vassilvitskii, 2007) before Lloyd refinement, giving a more careful, stable initialization (nstart is not used in this case). If the initialization produces a zero basis column – typically "kmeans" or "kmeans++" on data with two or more all-zero observation columns, which k-means may gather into a cluster whose centre is the zero vector – and X.restriction cannot normalize it, the initialization is rerun on the non-zero columns of Y and a message says so. The fit itself still uses every column. A user-supplied matrix with a zero column stops with an error instead. "spa" starts from the anchor rows that the successive projection algorithm (SPA; Gillis & Vavasis, 2014) finds in Y: B is started at those rows of Y and X at the non-negative least-squares fit to them. The anchor rows' zeros are filled with small positive values, so this is only a starting value and does not make the factorization unique; X.anchor does. Choosing the rows and the least-squares fit involve no randomness; the filled values are drawn as for "kmeansar", under seed.
- X.anchor: Anchor rows of X, imposed as constraints (default NULL: none). An anchor row carries one basis alone, $X_{j_q r} = 0$ for $r \neq q$. The zeros are kept for the whole fit, and they make the factorization unique up to the column scale that X.restriction fixes: another solution XR with the same zeros needs a diagonal R. With covariates this identifies C (Θ) too, provided rank <= nrow(A) and A has full row rank; a larger rank is refused, because $B = CA$ then has rank below rank. "spa" chooses the rows by SPA, which recovers them when Y is (nearly) separable – each basis has a row of Y that it alone loads – and B has full rank. SPA returns rank rows even when Y has no such rows: compare the objective with a fit from X.init = "spa", which starts from the same rows without the constraint, to see what the assumption costs. Alternatively give the rows, one per basis, as indices or row names; basis q is then the one anchored at the q-th row given, and the bases are not reordered after the fit. The start is built from the anchors, so X.init is not used (a message says so if it was given). The rows used are returned as X.anchor.
- nstart: Number of random starts for initialization of X (default: 1). Used by kmeans (when X.init = "kmeans" or "kmeansar") and by the multi-start evaluation (when X.init = "runif").
- seed: Integer seed for reproducibility (default: 123).
- C.init: Optional numeric matrix giving the initial value of the parameter matrix C (i.e., Θ). If A is NULL, C has dimension $Q \times N$ (equivalently B); otherwise, C has dimension $Q \times K$ where $K = nrow(A)$. Default initializes all entries to 1. The multiplicative updates keep an exact zero at zero, so a zero in C.init (or in a user-supplied X.init) is a structural zero that holds throughout the fit. A coefficient whose optimum lies on the boundary, by contrast, is approached only slowly and may remain visibly positive at the default epsilon. To impose a zero, put it in the initial value rather than relying on the updates to reach it.

- `Y.symmetric`: **Removed.** Symmetric NMF ($Y \approx XX^\top$ or $XCX^\top$) has moved to the dedicated `nmfkc.net` function (types `"tri"`, `"bi"`, `"signed"`), which uses the correct Frobenius bilateral-gradient updates. Passing `Y.symmetric` to `nmfkc()` now stops with a message pointing to `nmfkc.net()`.
- `prefix`: Prefix for column names of X and row names of B (default: "Basis").
- `print.trace`: Logical. If `TRUE`, prints progress every 10 iterations (default: `FALSE`).
- `print.dims`: Deprecated. Use `verbose` instead.
- `detail`: Level of post-fit criterion computation. `"fast"` (**default**) computes everything except the $O(N^2)$ sample-clustering criteria; `"full"` adds `silhouette`, `CPCC` and `dist.cor`, which need two $N \times N$ distance matrices and a cophenetic correlation — at $N = 2000$ that was 26 times the cost of the rest of the call; `"minimal"` returns only information criteria and skips XB. The default changed from `"full"` in 0.8.9: nothing consumed those three (see the `criterion` entry under Value), so every call was paying for them. For backward compatibility, `save.time = TRUE` maps to `"fast"` and `save.memory = TRUE` maps to `"minimal"`.

Value

A list with components:

`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).
`X.anchor`	Present only when `X.anchor` was used: the anchor row of each basis, as a named integer vector.
`X.restriction`	The constraint that was applied to the columns of X. It decides how much of the $(X, C) \to (XT, T^{-1}C)$ freedom is pinned down, which inference on the latent parameters needs to know.
`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. The trace is trimmed to `[10:end]` so that the plot is not dominated by the first few steps, so its length is 9 short of `iter` whenever the fit ran at least ten iterations; use `iter`, not `length(objfunc.iter)`, for the iteration count.

`iter` Number of iterations performed.

`maxit` The iteration cap that was in force.

`epsilon` The convergence tolerance that was in force.

`epsilon.iter` Relative change of the objective at the last step — the quantity the stopping rule compares with `epsilon`. Measured as $|f_i - f_{i-1}|/\max(|f_i|, 1)$; the floor of 1 means that for an objective below 1 the test is an absolute one, which keeps a near-exact fit from iterating forever but makes the number depend on the scale of Y. `nmfkc.signed` divides by $|f_{i-1}|$ instead, so the two are comparable only while the objective exceeds 1.

`objfunc.increases` Number of steps at which the objective rose, counted on the recorded (trimmed) trace. The multiplicative update is monotone by itself, so a positive count means something outside it — a restriction imposed by projection, or a numerical problem — is pushing the iterate back, in which case the fit can oscillate and exhaust `maxit` instead of converging.

`converged` Logical; whether `epsilon.iter` met `epsilon` before `maxit`. A run that merely exhausted `maxit` is `FALSE`.

`r.squared` $R^2 = \mathrm{cor}(Y, XB)^2$ (Pearson; scale-invariant; $[0, 1]$).

`r.squared.uncentered` Uncentered $R^2 = 1 - \|Y - XB\|_F^2 / \|Y\|_F^2$ (baseline = zero matrix; natural for non-negative factorizations without an intercept).

`r.squared.centered` Row-mean centered $R^2 = 1 - \|Y - XB\|_F^2 / \|Y - \bar{Y}_{p\cdot}\|_F^2$, the multivariate regression R^2.

`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 with `effective.rank`, `effective.rank.index`, and the separability of the basis: `separability` (one value; 1 = every basis has a row of its own), `anchor.purity` (per basis) and `anchor.row` (where each basis is purest). See the Details of `summary.nmfkc` for their definition and reading.

`effective.rank.index` is `effective.rank` rescaled onto $[0, 1]$ by the broken-stick correction $(\hat{r}_Q - E_Q)/(Q - E_Q)$ with $E_Q = \exp(H_Q - 1)$, the effective rank a random split of the variance would produce; it is the quantity `nmfkc.rank` plots, and `summary` prints it in place of the crispness. Read it as **how evenly the across-sample coefficient variance is shared**, which is not the same as how

useful the factors are: a factor whose coefficient is large but constant carries no variance and pulls the index down (correctly — the raw value can go negative and is clamped at 0, meaning the variance is spread over fewer factors than chance would give), while two duplicated factors split the variance evenly and score near 1. `NA` at $Q = 1$, where $E_1 = Q$.

With `detail = "full"` the list additionally carries the **sample-clustering** criteria `silhouette` (mean silhouette width of the hard clustering, computed in the original data space `dist(t(Y))` with the per-sample labels), `CPCC` (cophenetic correlation of a hierarchical clustering of the coefficient distances `dist(t(B))`) and `dist.cor` (correlation between original-data and coefficient distances). They are **absent by default** because they cost $O(N^2)$ and nothing consumes them: they are no longer part of rank selection, `summary.nmfkc` does not print them, and `nmf.cluster.criteria` — the right entry point when several ranks are to be compared — recomputes them from the fits it is given.

`effective.rank` is the **effective rank**: $\exp$ of the Shannon entropy of the explained-variance distribution $p_k = \mathrm{var}(B_{k\cdot}) / \sum_j \mathrm{var}(B_{j\cdot})$. By the trace identity $\sum_k \mathrm{var}(B_{k\cdot}) = \mathrm{tr}(\mathrm{Cov}(B))$, p_k is the exact fraction of the total coefficient variance carried by factor k, so the entropy measures how that variance is spread across factors. It ranges in $[1, Q]$ (1 when one factor carries all the variance, Q when all contribute equally) and counts the number of latent factors that actively shape across-sample variation. This is the PCA-style explained-variance / effective-dimensionality measure and reuses the $\exp(\mathrm{entropy})$ functional form of Roy & Vetterli (2007).

References

Satoh, K. (2026). Applying Non-negative Matrix Factorization with Covariates to the Longitudinal Data as Growth Curve Model. *American Journal of Mathematical and Management Sciences*. In press. 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. (2026). Applying non-negative matrix factorization with covariates to label matrix for classification. *Japanese Journal of Statistics and Data Science*. doi:10.1007/s4208102600349x

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

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

Gillis, N., & Vavasis, S. A. (2014). Fast and robust recursive algorithms for separable nonnegative matrix factorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4), 698–714. doi:10.1109/TPAMI.2013.226

See Also

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

Examples

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

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

# Example 3. Anchors: in Example 1, rows P1 and P2 of X carry one basis
# each.  X.anchor imposes such rows (here found by SPA), which makes the
# factorization unique; X.init = "spa" only starts from them.
res_a <- nmfkc(Y, rank = 2, X.anchor = "spa", epsilon = 1e-6)
res_a$X.anchor

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

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

Description

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

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

Usage

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

Arguments

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

Value

A list containing:

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

References

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

See Also

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

Examples

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

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

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

Description

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

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

Usage

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

Arguments

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

Value

A list with components:

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

See Also

nmfkc.ar, nmfkc.cv

Examples

```
# Example using ts object directly
d <- AirPassengers

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

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

Description

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

The diagram displays three types of directed relationships:

- Lagged predictors: $T_{t-k} \to X$,
- 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.latent *Latent transition matrices of an NMF-VAR model*

Description

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

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

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

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

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

Usage

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

Arguments

x	A fitted "nmfkc" object obtained from an nmfkc.ar design (nmfkc(arY, arA, rank = Q)).

Details

The latent process is VARMA, not VAR. The display above drops the observation residual. With $\boldsymbol{y}_t = X\boldsymbol{b}_t + \boldsymbol{e}_t$ the exact identity is

$$\boldsymbol{b}_t = \sum_{d=1}^{D} G_d\, \boldsymbol{b}_{t-d} + \boldsymbol{\theta} + \sum_{d=1}^{D} \Theta_d\, \boldsymbol{e}_{t-d},$$

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

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

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

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

Value

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

G	List of D latent transition matrices G_d ($Q \times Q$), named lag1, ..., with dimnames = basis labels (row = effect, column = cause).
G.sum	$\sum_d G_d$.
eigenvalues	The DQ eigenvalues of the latent companion matrix. The $PD \times PD$ companion of the Ξ_d has the same non-zero eigenvalues, so these are all of the model's roots.
spectral.radius	ρ, the largest modulus among them.
spectral.radius.sum	$\rho(\sum_d G_d)$; below 1 iff stationary.

`stationary` Logical; `TRUE` when `spectral.radius` < 1.

`cycle.period` Period $2\pi/|\arg\lambda|$ implied by the dominant complex root, or `NA` when no root is complex.

`alternating` `TRUE` when some root is real and negative, i.e. the decay alternates in sign (period 2).

`non.oscillatory` `TRUE` only when every root is real and positive. Named after the roots, not the trajectory — overshoot is still possible.

`theta0` Latent intercept $\boldsymbol{\theta}$ (or `NULL`).

`mu.b`, `mu.y` Long-run means $(I-\sum_d G_d)^{-1}\boldsymbol{\theta}$ and $X\boldsymbol{\mu}_b$; `NA` when the fit is not stationary or has no intercept.

`p.star` Composition of the long-run mean, $\boldsymbol{\mu}_b/(\mathbf{1}'\boldsymbol{\mu}_b)$ — equivalently the fixed point of the *residual-free* composition dynamics. It is **not** in general the limit of $\boldsymbol{b}_t/(\mathbf{1}'\boldsymbol{b}_t)$ for the stochastic process, nor its expectation: the ratio of expectations is not the expectation of the ratio.

`colsum` Column sums c_j of $\sum_d \Xi_d = X\sum_d \Theta_d$, computed as `colSums(X) %*% Lambda` (length P). Being column sums of a non-negative matrix whose radius is $\rho(\sum_d G_d)$, they bracket it. The unweighted `colSums(Lambda)` would only be correct when $\mathbf{1}'X = \mathbf{1}'$.

`colsum.max` $\max_j c_j$; below 1 it certifies stationarity.

`df` Effective dimension $Q(P+m-Q)$ of the fit, where m is the number of columns of Θ ($PD+1$ with an intercept, PD without). These are the degrees of freedom to use when comparing R^2 or information criteria across Q: the naive count $Q(P+m)$ ignores the Q^2 rotational indeterminacy.

`separability` Per-basis maximum of the row-normalised X; a value near 1 means that basis owns a variable loading almost only on it. This is a *necessary*-looking but not sufficient condition for uniqueness — see the counterexample under Identification.

`X`, `Theta` The basis and the full coefficient matrix the quantities above were computed from, carried along so that `nmfkc.ar.latent.inference` does not need the fitted object a second time.

`X.restriction` The constraint `nmfkc` placed on the columns of X, which decides how much of T is pinned down.

`dims` Named vector of `P`, `Q`, `D`.

References

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

See Also

`plot.nmfkc.ar.latent`, `nmfkc.ar.latent.inference`, `nmfkc.ar.stationarity`, `nmfkc.ar`, `nmfkc.ar.DOT`

Examples

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

`nmfkc.ar.latent.inference`

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

Description

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

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

Usage

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

Arguments

`object`	An object from `nmfkc.ar.latent`.
`Y`, `A`	The observation and covariate matrices the model was fitted on, i.e. the `Y` and `A` returned by `nmfkc.ar`.
`...`	Additional arguments: `wild.B` Number of bootstrap replicates. Default 500. `wild.dist` Multiplier distribution, `"rademacher"` (default) or `"exp"` (mean-centred). `wild.unit` `"column"` (default, one multiplier per time point), `"block"` (one per block of `max(1, round(n^(1/3)))` consecutive time points, `n` the number of time points) or `"element"` (one per cell). `wild.level` Confidence level. Default 0.95. `wild.seed` Seed for the bootstrap. Default 123. `fit` The fitted `"nmfkc"` object. Only needed when `object` does not carry `X` / `Theta`. `truncate` Logical; clip Y^* at 0 (default `TRUE`). `epsilon`, `maxit` Convergence control of the re-fits, default `1e-8` and `20000` — tight, because a coefficient heading for the non-negativity boundary moves slowly and a loose tolerance biases anything that is compared with zero.

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

`sep.tol` Row-normalised loading a column of X must reach for that basis to count as anchored. Default `0.999`.

`pin.tol` A coefficient of Θ at most `pin.tol` times the coefficient of the same covariate for the basis being pinned counts as zero when the rows of Θ are pinned; so does one at most `kkt.tol`. Default `0.01`: a mixing of the bases of up to 1\ actually reached is reported – `print()` states it, and `identification$pin.margin`, `$pure.margin` and `$X.pin.margin` hold the largest leaks that were needed.

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

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

Details

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

- **Inferential.** ρ, the eigenvalues and the observed long-run mean $\boldsymbol{\mu}_y = X\boldsymbol{\mu}_b$ are invariant under the whole family ($G_d \to T^{-1}G_dT$ leaves the spectrum alone), so their replicates need no alignment and their intervals mean what they say.
- **Conditional on identification.** The entries $(G_d)_{qq'}$ and the composition $\boldsymbol{p}^*$ need T to be a permutation, and three conditions together deliver that: the column scale fixed (`X.restriction = "colSums"`), an **anchor row** for every column of X, and every row of Θ **pinned**. The anchor rows force $T \geq 0$. The rows are then pinned one at a time: row k joins the pinned set S when some covariate (column of Θ) is non-zero only for bases in $S \cup \{k\}$, including k, and, for every $q \in S$, such a covariate has a zero coefficient for q. The first row pinned needs a **pure column** (a covariate that loads on that basis alone); later rows only need zeros that separate them from the rows already pinned, possibly in different columns. Each pinned row makes e_k a column of T, so pinning every row makes T a permutation. (This used to require a pure column for every row, which is stronger: every fit that passed before passes now.) **Either condition alone is not enough**: with $X = I_2$, $\Theta = \left(\begin{smallmatrix} 2 & 3 \\ 1 & 1 \end{smallmatrix}\right)$ and $T = \left(\begin{smallmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{smallmatrix}\right)$ the basis is perfectly separable and column-normalized, yet $XT \geq 0$, $T^{-1}\Theta \geq 0$, $X\Theta$ is unchanged, T is no permutation, and G differs between the two representatives — no row of Θ can be pinned there. Nor are zeros that merely separate every pair of rows enough: with anchors and the covariate columns $(0,1,1)$, $(1,0,1)$, $(1,1,0)$, $T = (J-I)/2$ is admissible. A coefficient counts as zero when it is at most `kkt.tol` or at most `pin.tol` times the coefficient of the same covariate for the basis being pinned (that ratio is the mixing the covariate still allows), and a covariate whose coefficients are all numerical noise is ignored. **The mirror route** swaps the two factors and is checked as well, with nothing to choose: a pure covariate

for every basis (forcing $T^{-1} \geq 0$), and the bases pinned one at a time through the rows of X – the first by an anchor row, the later ones by variables whose zero loadings separate them from the bases already pinned (an entry of X counts as zero when it is at most `1 - sep.tol` times the loading of the basis being pinned). Either route suffices; `identification$route` names the one that passed. `identified.approx` reports the verdict and `identification` its parts. **What a pass means, to the precision of the fit**: multiplicative updates leave small positive values where the argument wants exact zeros, so the check treats values within the tolerances as zero and reports how exact they are. The margin of the route that passed is the largest such value relative to the entry it was compared with, i.e. the rotation slack: every equivalent fit $(XT, T^{-1}\Theta)$ mixes one basis into another by at most about that much, so the entries of G_d are determined to about that relative precision. `print()` states it (e.g. "the zeros it rests on hold to 0.0024, so an equivalent fit mixes the bases by at most about 0.24\ part that fails. Two things lie outside the check: whether the estimated zeros are zeros in the population (a question for their intervals), and other fits that are only nearly as good (different starting values can reach one). Replicates are permutation-aligned to the original basis regardless. `X.restriction = "fixed"` is the one clean case for $Q > 1$ — the basis is not estimated, so $T = I$.

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

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

Bootstrap validity at the boundary. A naive nonparametric bootstrap is inconsistent for a parameter on the boundary. The scheme used here resamples the residuals and re-imposes the non-negativity constraint in every re-fit, which is the practical remedy. `wild.unit = "block"` in addition keeps the serial dependence of the residuals within a block; the lagged design itself stays fixed (see below).

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

Resampling scheme. With $\hat{Y} = X\Theta A$ and residuals $E = Y - \hat{Y}$, each replicate forms $Y^* = \hat{Y} + E \odot W$ and re-fits `nmfkc` at the same rank. W carries one multiplier per time point (`wild.unit = "column"`, the default, which preserves the contemporaneous covariance between series), one per block of consecutive time points (`wild.unit = "block"`, below), or one per cell. With i.i.d. multipliers over time the intervals ignore residual autocorrelation and may be too narrow when it is present. Negative entries of Y^* are truncated at 0 to keep the response non-negative; the truncation rate is reported, and a large one is a warning sign for the whole procedure.

Block wild bootstrap (`wild.unit = "block"`). One multiplier is drawn per block of $L = \max(1, \mathrm{round}(n^{1/3}))$ consecutive time points and shared by all series, where n is the number of time points, `ncol(Y)` (after the lags are removed): $L = 3$ for $n = 27$, 5 for 100, 8 for 500, 10 for 1000, and $L = 1$ (the

same draws as "column") for $n \leq 3$. The block grid starts at a random offset in $0, \ldots, L-1$, drawn anew in each replicate, so the first and last blocks may be shorter and no time point is always at a block edge. Residuals within a block keep their serial dependence, which matters when the residuals are autocorrelated (for example when the lag order is too small); for a correctly specified VAR the default "column" suffices. $n^{1/3}$ is the usual rate for the block length of block resampling. The length is fixed by n; it cannot be set and is not stored in the result.

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

Value

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

`coefficients` Data frame of the DQ^2 entries of the G_d, with columns `Lag`, `Basis` (effect at t), `Covariate` (cause at $t-d$), `Estimate`, `BSE`, `CI_low`, `CI_high`, `p_value`. Descriptive unless `identified.approx`.

`G`, `G.se`, `G.ci.lower`, `G.ci.upper`, `G.p.value`
The same quantities as lists of $Q \times Q$ matrices, one per lag.

`spectral.radius`
The radius of the $DQ \times DQ$ latent companion matrix — the same quantity `nmfkc.ar.latent` reports, not $\rho(\sum_d G_d)$, which differs when $D > 1$. With `spectral.radius.se`, `spectral.radius.ci.lower` / `.upper` and `spectral.radius.boot.mean`.

`prob.nonstationary`
Bootstrap tail probability $P^*(\rho^* \geq 1)$. **Not** a p-value for $H_0 : \rho \geq 1$ — see Details.

`complex`, `complex.frac`
Whether the point estimate has complex roots, and the fraction of replicates that do.

`mu.y` With `.ci.lower` / `.ci.upper`; invariant, so inferential.

`p.star` With `.ci.lower` / `.ci.upper`; descriptive unless `identified.approx`.

`kkt` Strict-complementarity diagnostic: `n.boundary`, `all.positive`, `ratio`, and the two margins `delta.dual` and `delta.prim`.

`identified.approx`
Logical: is the factorization unique up to a permutation of the bases, to the precision of the fit? `TRUE` for $Q = 1$, for `X.restriction = "fixed"` (where X is not estimated, so $T = I$), or when the per-column scale is fixed (`"colSums"` / `"colSqSums"`) and either every column of X has an anchor row at `sep.tol` and every row of Θ is pinned at `pin.tol` / `kkt.tol`, or the mirror condition holds (see Details). The precision is the margin of the route that passed: an equivalent fit mixes the bases by at most about that much, and `print()` states it. When `FALSE` the entry-level summaries and `p.star` are descriptive; ρ, the eigenvalues and $\boldsymbol{\mu}_y$ remain inferential either way.

`identification` The parts of the diagnostic: for the first route `anchor` (per column of X), `pinned` (per row of Θ) and `pin.order`; for the mirror route `pure` (a pure covariate per row of Θ), `X.pinned` (per basis) and `X.pin.order`; `ok` and `route` (the route(s) that passed); and the margins `anchor.margin`, `pin.margin`, `pure.margin` and

	`X.pin.margin` saying how far the fit is from an exact configuration (0 = exact; the pin margins are the largest relative entry the pinning had to treat as zero).
`refit.args`	The arguments the bootstrap re-fits actually used, so the replicates can be checked against the original estimator.

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

References

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

See Also

`nmfkc.ar.latent`, `nmfkc.ar.stationarity`, `nmfkc.ar`, `nmfkc.inference`

Examples

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

`nmfkc.ar.predict` *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` *Stationarity of an NMF-VAR model*

Description

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

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

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

Usage

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

Arguments

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

Value

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

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

References

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

See Also

nmfkc.ar.latent, nmfkc.ar.latent.inference, nmfkc.ar

Examples

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

`nmfkc.ard` *Automatic relevance determination for NMF rank (experimental)*

Description

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

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

Usage

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

Arguments

`Y`	Observation matrix ($F \times N$), non-negative.
`rank`	Over-complete starting rank K (must exceed the true rank). `NULL` (default) uses `min(F, N, 20)`.
`nrun`	Number of random-initialization restarts (default `10`). ARD is a sensitive point estimate, so several restarts are advisable; the reported rank is the **mode** of the per-run estimates (`rank.runs`), with a representative modal fit kept for `plot`/`W`/`H`.
`plot`	Logical; draw the relevance bar plot.
`...`	Advanced options, rarely needed (defaults in parentheses): `prior` (`"L2"`: half-normal / squared-energy group shrinkage, `"L1"`: exponential / sparser); `seed` (`123`, random-initialization seed); `a` (`1`) and `b` (`(F + N)/K * mean(Y)`), the inverse-gamma prior (a smaller `b` over-prunes, a larger one prunes nothing); `maxit` (`3000`) and `epsilon` (`1e-6`) for optimisation control; `tol` (`1e-3`), the relevance threshold below which a component is counted as pruned; `cores` (`getOption("mc.cores", 1L)`) runs the `nrun` restarts in parallel (results identical to the sequential loop for any `cores`; PSOCK cluster on Windows, forking elsewhere).

Details

Relation to Tan & Fevotte (2013). The update equations reproduce the paper's ℓ_2 / ℓ_1 ARD-NMF for the **Euclidean** ($\beta = 2$) case exactly: the multiplicative penalties, the closed-form λ_k update $(f(w_k) + f(h_k) + b)/c$ and the constant c (their Eq. 33). Two deliberate simplifications keep it practical: only $\beta = 2$ is implemented (the paper covers the general β-divergence), and the default `b` is an *empirical* per-component energy scale $(F+N)/K \cdot \bar{Y}$ that reliably avoids the winner-take-all collapse, rather than the paper's method-of-moments value (their Eq. 38).

Value

An object of class "nmfkc.ard": a list with rank (estimated = mode over restarts), rank.runs (the per-run estimates), relevance (representative run, descending), lambda, W, H (ordered by relevance), rank.init, prior, nrun, objfunc (final objective value of the representative run) and objfunc.iter (its per-iteration trajectory).

References

V. Y. F. Tan and C. Fevotte (2013). Automatic relevance determination in nonnegative matrix factorization with the beta-divergence. *IEEE TPAMI* 35(7):1592–1605. doi:10.1109/TPAMI.2012.240.

See Also

nmfkc.ecv, nmfkc.bicv, nmfkc.consensus, nmfkc.rank

Examples

```
set.seed(1)
X <- matrix(abs(rnorm(40 * 3)), 40, 3)
B <- matrix(abs(rnorm(3 * 60)), 3, 60)
ar <- nmfkc.ard(X %*% B, rank = 10)   # over-complete start
ar$rank                                # ~ 3 surviving components
plot(ar)
```

---

nmfkc.bicv *Bi-cross-validation for NMF rank selection*

Description

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

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

Usage

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

Arguments

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

Details

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

Value

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

objfunc	Held-out mean squared error for each rank.
sigma	Its square root (RMSE) for each rank.
rank	The evaluated rank vector.
nfolds	The number of folds used.

References

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

See Also

nmfkc.rank, nmfkc.ecv, nmfkc.consensus, nmfkc.ard

Examples

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

```
bv$rank[which.min(bv$sigma)]
```

`nmfkc.class` *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.consensus` *Consensus-clustering rank selection for NMF (Brunet 2004)*

Description

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

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

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

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

Usage

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

Arguments

`Y`	Observation matrix ($P \times N$), non-negative.
`A`	Optional covariate matrix passed to `nmfkc`.
`rank`	Integer vector of ranks to evaluate (≥ 2).
`nrun`	Number of random-initialization runs per rank (default 30).
`keep.consensus`	Logical; if `TRUE` also return the list of consensus matrices (one $N \times N$ matrix per rank).
`...`	Advanced options, rarely needed (defaults in parentheses): `seed` (`123`, base seed; run r of rank index i uses `seed + 1000 * i + r`) and `pac.range` (`c(0.1, 0.9)`, the ambiguous interval (u_1, u_2) for the PAC measure). Any other arguments are passed to `nmfkc` (e.g.\ `maxit`); `X.init` is forced to `"runif"`. `cores` (`getOption("mc.cores", 1L)`) runs the `nrun` restarts of each rank in parallel; results are identical to the sequential loop for any `cores` (PSOCK cluster on Windows, forking elsewhere).

Value

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

`cophenetic`	Cophenetic correlation coefficient for each rank.
`dispersion`	Dispersion coefficient ($[0,1]$) for each rank.
`pac`	Proportion of Ambiguous Clustering ($[0,1]$, lower is better) for each rank.
`rank`	The evaluated rank vector.

nrun	Number of runs per rank.
consensus	List of consensus matrices, or NULL.

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

References

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

See Also

nmfkc.rank, nmfkc.ecv, nmfkc.bicv, nmfkc.ard, nmf.cluster.criteria

Examples

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

---

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

Description

nmfkc.criterion computes the effective rank (raw and as the $[0, 1]$ broken-stick index) and the sample-clustering quality measures (silhouette, CPCC, dist.cor) from a fitted nmfkc model.

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

Usage

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

Arguments

`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"` skips the fit and clustering statistics.
`...`	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: effective.rank, effective.rank.index, silhouette, CPCC, dist.cor.

See Also

`nmfkc`, `nmfkc.rank`

Examples

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

---

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

Description

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

- $Y(P, N)$ is the observation matrix,
- $A(R, N)$ is the covariate (or kernel) matrix,

- $X(P, Q)$ is the basis matrix,
- $C(Q, R)$ is the parameter matrix, and
- $B(Q, N)$ is the coefficient matrix ($B = CA$).

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

Usage

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

Arguments

`Y`	Observation matrix, or a formula (see `nmfkc` for Formula Mode).
`A`	Covariate matrix. If `NULL`, the identity matrix is used. Ignored when `Y` is a formula.
`rank`	Rank of the basis matrix X. Default is 2.
`data`	A data frame (required when `Y` is a formula with column names).
`...`	Additional arguments controlling CV and the internal `nmfkc` call:

`Y.weights` Non-negative weight matrix or vector (`lm()`-style: loss $\sum W r^2$). Binary `{0,1}` masks (`TRUE` / `FALSE` also accepted) are the typical ECV usage – 0/`FALSE` excludes an element. See `nmfkc` for full details.

`div` Number of folds (k); default: `5`.

`seed` Integer seed for reproducible partitioning; default: `123`.

`shuffle` Logical. If `TRUE` (default), randomly shuffles samples (standard CV); if `FALSE`, splits sequentially (block CV; recommended for time series).

`Q` (Deprecated) Alias for `rank`.

Arguments passed to `nmfkc` e.g., `C.L1`, `epsilon`, `maxit`, `method` (`"EU"` or `"KL"`), `X.restriction`, `X.init`, etc.

Value

A list with components:

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

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

`objfunc.block` Loss for each fold.

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

See Also

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

Examples

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

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

---

nmfkc.cv.methods *Print/plot methods for nmfkc cross-validation objects*

Description

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

Usage

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

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

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

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

## S3 method for class 'nmfkc.cv'
```

```
print(x, ...)

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

Arguments

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

Value

x, invisibly.

---

nmfkc.denormalize *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,
  C.signed = NULL,
  sig.level = 0.1,
  rankdir = "LR",
  fill = TRUE,
  weight_scale = 5,
  weight_scale_ax = weight_scale,
  weight_scale_xy = weight_scale,
  weight_scale_ay = weight_scale,
  Y.label = NULL,
  X.label = NULL,
  A.label = NULL,
  Y.title = "Observation (Y)",
  X.title = "Basis (X)",
  A.title = "Covariates (A)",
  hide.isolated = TRUE
)
```

Arguments

result	The return value from nmfkc, containing matrices X, B, and optionally C.
type	Character string specifying the visualization style: one of "YX", "YA", "YXA".
threshold	Minimum coefficient magnitude to display an edge. When C.signed = TRUE (signed Θ) this is an absolute-value cut, i.e. an edge is drawn when $\|coef\| \geq$ threshold.
C.signed	Logical or NULL. Whether the coefficient matrix Θ (= C) may be signed (real-valued). When TRUE, the threshold is applied to $\|coef\|$, edge widths are scaled by $\|coef\|$, and negative edges are drawn as black **dashed** lines (positive edges remain solid) to distinguish them; the numeric edge labels keep their sign. When FALSE, the historical non-negative behaviour is used (negative coefficients fall below the threshold and are hidden). When NULL (default), the mode is auto-detected from the fit (result$C.signed) or from the presence of negative entries in C / XC. The basis X is always non-negative, so $X \to Y$ edges are unaffected.
sig.level	Significance level for filtering C edges when inference results are available (i.e., x is of class "nmfkc.inference"). Only edges with p-value below sig.level are shown, decorated with significance stars (*, **, ***). Set to NULL to disable filtering and show all edges above threshold. Default is 0.1.
rankdir	Graphviz rank direction (e.g., "LR", "TB").
fill	Logical; whether nodes should be drawn with filled shapes.
weight_scale	Base scaling factor for edge widths.
weight_scale_ax	Scaling factor for edges $A \to X$ (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, nmf.rrr.DOT, nmf.ffb.DOT, nmfkc.ar.DOT, plot.nmfkc.DOT

Examples

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

---

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

Description

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

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

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

Usage

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

Arguments

Y	Observation matrix, or a formula (see nmfkc for Formula Mode).
A	Covariate matrix. Ignored when Y is a formula.
rank	Vector of ranks to evaluate (e.g., 1:5). For backward compatibility, Q is accepted via ....
data	A data frame (required when Y is a formula with column names).
...	Additional arguments passed to nmfkc (e.g., method="EU"). Also accepts: nfolds (number of folds, default 5; div also accepted), seed (integer seed, default 123), and cores (evaluate the rank x fold task grid in parallel; default getOption("mc.cores", 1L)). Parallelism uses a PSOCK cluster on Windows and forking elsewhere; because each task is a deterministic self-seeded fit and results are returned in order, objfunc/sigma are identical for any cores. Y.weights (as in nmfkc: a P x N matrix, one weight per column, or a scalar) is kept in every fit; cells of weight 0 are never held out, and the held-out loss is the weighted mean. (Before 1.0.0 a Y.weights given here was silently replaced by the fold mask.)

Value

A list with components:

objfunc	Numeric vector containing the Mean Squared Error (MSE) for each Q, named "Q=2", "Q=3", ... The names travel with the values: unname() them before combining results into a data frame or with c() if they are not wanted.
sigma	Numeric vector containing the Residual Standard Error (RMSE) for each Q. Only available if method="EU".
objfunc.fold	List of length equal to Q vector. Each element contains the MSE values for the k folds.
folds	A list of length div, containing the linear indices of held-out elements for each fold (shared across all Q).

References

Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4), 397–405. doi:10.1080/00401706.1978.10489693
Owen, A. B., & Perry, P. O. (2009). Bi-cross-validation of the SVD and the nonnegative matrix factorization. *Ann. Appl. Stat.* 3(2), 564–594. doi:10.1214/08AOAS227 (cross-validation of the NMF rank; see also nmfkc.bicv).

See Also

nmfkc, nmfkc.cv; other rank-selection criteria: nmfkc.rank, nmfkc.bicv, nmfkc.consensus, nmfkc.ard.

Examples

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

---

nmfkc.inference *Statistical inference for the parameter matrix C (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:

`method` Bootstrap engine. `"onestep"` (default): a one-step Newton linearisation around the fit using the inverse information `Hinv` (fast; the sandwich SE is primary). `"refit"`: a residual wild bootstrap that re-estimates C to convergence with the basis X held FIXED. The "refit" engine uses no information matrix, so it stays valid when the Fisher information AA' is singular (over-parameterised covariates) or C sits on the ≥ 0 boundary; the bootstrap SE/CI become primary and the p-value is a two-sided bootstrap p-value.

`wild.dist` Multiplier distribution (orthogonal to `method`): `"rademacher"` (± 1), `"mammen"` (Mammen 1993 two-point), or `"exp"` ($\mathrm{Exp}(1) - 1$). Default `"exp"` for `method="onestep"` (backward compatible), `"rademacher"` for `"refit"`.

`wild.unit` For `method="refit"`: `"column"` (default since 1.0.2; one multiplier per sample column, shared over rows), `"element"` (an i.i.d. multiplier per matrix cell; the default up to 1.0.1) or `"block"` (one multiplier per block of `L = max(1, round(N^(1/3)))` consecutive columns, shared over rows; `N = ncol(Y)`). Use `"block"` when the columns are time points with serially dependent residuals, e.g. the design of `nmfkc.ar`: residuals at nearby time points then share a multiplier. `L` is 3 for `N = 27`, 5 for 100, 8 for 500 and 1 (the same draws as `"column"`) for `N <= 3`. The block grid starts at a random offset in `0, ..., L-1`, drawn anew in each replicate, so the first and last blocks may be shorter. `L` is fixed by `N`; it cannot be set and is not stored in the result. `"column"` treats the columns of `Y` as the independent units and allows any dependence among the `P` rows of a column, as the sandwich SE and the `"onestep"` engine already do (both are built from one score per column). `"element"` also treats the rows of a column as independent; with `P = 1` the two draw the same multipliers and agree exactly.

`refit.epsilon`, `refit.maxit` For `method="refit"`: convergence tolerance and iteration cap of the re-fits, defaulting to `1e-8` and `100000`. These are deliberately far tighter than `nmfkc`'s own `1e-4`: under multiplicative updates a coefficient heading for the non-negativity boundary approaches 0 slowly, so at a loose tolerance every replicate stops while that coefficient is still spuriously positive. The replicates then pile up above the point estimate, the percentile interval can exclude it entirely, and the bootstrap SE is understated. Loosen only if the re-fits are too slow, and check that the intervals still contain the estimates.

`wild.B` Number of bootstrap replicates. Default is 500.

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

`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`.

`sep.tol`, `pin.tol` Tolerances of the identification diagnostic (defaults `0.999` and `0.01`); see `identified.approx` below and the Details of `nmfkc.ar.latent.inference`. Coefficients at or below `boundary.tol` also count as zero there.

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 one row per element of C. `SE` is always the sandwich standard error and `BSE` always the bootstrap one, so the two columns differ; `z_value` and `p_value` use whichever is primary for the chosen `method` (bootstrap under `"refit"`, sandwich under `"onestep"`). `on.boundary` flags coefficients at or below `boundary.tol` (default `1e-3`): their interval and p-value behave correctly in the "do not reject" direction, but the value is not a calibrated p-value — a one-sided bootstrap p piles up near $1/2$ at the boundary.

`C.p.side` P-value type used.

`wild.unit` Multiplier unit of the `method = "refit"` bootstrap (`"column"`, `"element"` or `"block"`); `NA` otherwise.

`identified.approx` Logical, an approximate diagnostic of whether the factorization is unique up to a permutation of the bases. The intervals are conditional on $\hat{X}$; when this is `TRUE` they concern the same C whichever admissible representative $(XT, T^{-1}C)$ is taken, and when `FALSE` the entries of C describe this representative. `TRUE` needs A of full row rank and either $Q = 1$, `X.restriction = "fixed"`, or a fixed per-column scale (`"colSums"` / `"colSqSums"`) with one of two zero patterns: an anchor row of X for every basis and the rows of C pinned one at a time, or the mirror condition. The argument is in the Details of `nmfkc.ar.latent.inference`. An algebraic check on the point estimate: no re-fits.

`identification` Its parts: `anchor`, `pinned`, `pin.order`, `pure`, `X.pinned`, `X.pin.order`, `ok`, `route`, `A.full.rank` and the margins `anchor.margin`, `pin.margin`, `pure.margin`, `X.pin.margin` (0 = exact).

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. (2026). Applying Non-negative Matrix Factorization with Covariates to the Longitudinal Data as Growth Curve Model. *American Journal of Mathematical and Management Sciences*. In press. 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`. Also accepts `cores` (evaluate the `beta` candidates in parallel; default `getOption("mc.cores", 1L)`; PSOCK cluster on Windows, forking elsewhere). Each candidate is a deterministic `nmfkc.cv` call (its folds are seeded) and results are returned in order, so `objfunc` and the selected `beta` are identical for any `cores`.

Value

A list with components:

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

See Also

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

Examples

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

---

nmfkc.kernel.beta.nearest.med

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

Description

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

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

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

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

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

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

Usage

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

Arguments

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.kernel.gram` *Block-wise Gram accumulation of Nystroem kernel covariates for large N*

Description

Builds the two summary matrices that `nmfkc` needs from the Nystr\"om kernel covariate matrix $A = C^\top$ ($M \times N$, $C_{nm} = k(\boldsymbol{u}_n, \boldsymbol{v}_m)$) without ever forming A itself:

$$S = AA^\top \quad (M \times M), \qquad G_0 = YA^\top \quad (P \times M).$$

The input `U` is processed in column blocks; for each block the kernel values $k(V, U_b)$ are computed with `nmfkc.kernel`, added into S and G_0, and discarded. Peak memory is $O(M^2 + PM + M \cdot$ `block.size`$)$ instead of $O(MN)$.

The returned object is passed to `nmfkc` as its `A` argument. The multiplicative updates are unchanged, since they only ever use S and G_0; the fit equals the one obtained from the explicit matrix `nmfkc.kernel(V, U, beta = beta)` up to floating-point summation order. This is the large-N form of the NMF-LAB Kernel design with Nystr\"om approximation (Satoh 2026).

Usage

```
nmfkc.kernel.gram(Y, U, V, beta = NULL, block.size = NULL, ...)
```

Arguments

`Y`	Non-negative $P \times N$ response matrix (e.g. a one-hot label matrix). `NA` is not allowed.
`U`	A $p \times N$ numeric matrix of inputs; columns are samples.
`V`	Landmarks. Either a $p \times M$ numeric matrix of landmark points, or a single integer M, in which case M landmarks are chosen from a random subsample of the columns of `U` by the method in `landmarks` (default k-means++ seeding followed by k-means refinement).
`beta`	Positive scalar. Gaussian kernel bandwidth $k(u, v) = \exp(-\beta\|u-v\|^2)$. If `NULL` (default), the nearest-landmark median heuristic `nmfkc.kernel.beta.nearest.med(U_sub, Uk = V)` on the subsample is used, and the value is reported in a message. For bandwidth selection build one Gram object per candidate in `$beta_candidates` of that helper (see Details).
`block.size`	Integer. Number of columns of `U` per block. Default: as many columns as keep one kernel block near 256 MB (`2^25 / M` columns), capped at `N`.
`...`	Hidden options: • `landmarks`: how to choose M landmarks when `V` is an integer: `"kmeans++"` (default; Arthur & Vassilvitskii 2007 seeding then Lloyd refinement), `"kmeans"` (`stats::kmeans` with `nstart = 5`), or `"random"` (uniform subsample). • `sample.size`: size of the random subsample of columns used for landmark selection and the bandwidth heuristic (default `min(N, 10000)`).

- `seed`: integer seed for the subsample and the landmark selection (default 123; the caller's random stream is restored).
- `kernel`: kernel name passed to `nmfkc.kernel` (default `"Gaussian"`); further kernel parameters are passed through as well.

Details

With Gram input the fold-based `nmfkc.cv` is not available (it must split the columns of A). Select β as in the NMF-LAB paper: fix the landmarks, build one Gram object per candidate β on the training columns (pass the landmark matrix `g$landmarks` as `V` so all candidates share it), fit with `nmfkc()`, and score validation columns with `predict(fit, newA = nmfkc.kernel(g$landmarks, U.val, beta = beta))`.

Value

An object of class `"nmfkc.gram"`: a list with

`S` $M \times M$ matrix $AA^\top$ (non-negative, positive semi-definite).

`G0` $P \times M$ matrix $YA^\top$.

`N, D` Number of samples and of covariates (`D = M`, the number of landmarks).

`type, signed` `"nystrom"` and `FALSE` (non-negative features; accepted by `nmfkc` and `nmfkc.signed`).

`landmarks` The $p \times M$ landmark matrix V. Pass it to `nmfkc.kernel` as its first argument to build covariates for new data.

`beta, kernel` Bandwidth and kernel used.

`landmarks.method, sample.size` How the landmarks were chosen (`"supplied"` when `V` was a matrix).

`block.size, n.blocks` Blocking actually used.

`A.block` A function `A.block(idx)` returning the $M \times |\texttt{idx}|$ kernel block $k(V, U_{idx})$. `nmfkc` uses it to rebuild $B = CA$ block by block after the fit. It closes over `U` and `V`, so the object keeps `U` alive (no copy is made).

`rownames` Row names of A (`colnames(V)` if any, else `NULL`).

Lifecycle

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

References

Williams, C. K. I., & Seeger, M. (2001). Using the Nystr\"om method to speed up kernel machines. *Advances in NIPS*, 13.

Zhang, K., Tsang, I. W., & Kwok, J. T. (2008). Improved Nystr\"om low-rank approximation and error analysis. *ICML*.

Arthur, D., & Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding. *SODA*.

Satoh, K. (2026). Applying non-negative matrix factorization with covariates to label matrix for classification. *Japanese Journal of Statistics and Data Science*. doi:10.1007/s4208102600349x

See Also

nmfkc, nmfkc.kernel, nmfkc.kernel.beta.nearest.med, nmfkc.signed.rff.gram (the signed, Random-Fourier counterpart for nmfkc.signed)

Examples

```
## Iris, 3 classes: Nystroem covariates with 12 k-means++ landmarks.
## The Gram route gives the same fit as the explicit M x N matrix.
data(iris)
set.seed(1)
idx <- sample(nrow(iris), 100)
mn <- colMeans(iris[idx, 1:4]); sc <- apply(iris[idx, 1:4], 2, sd)
U.train <- t(scale(iris[idx,  1:4], center = mn, scale = sc))   # 4 x 100
U.test  <- t(scale(iris[-idx, 1:4], center = mn, scale = sc))   # 4 x  50
levs    <- levels(iris$Species)
Y.train <- sapply(iris$Species[idx], function(s) as.integer(levs == s))
rownames(Y.train) <- levs

## 12 landmarks by k-means++, bandwidth by the nearest-landmark median
## heuristic, S and G0 accumulated over blocks of 25 columns
g <- nmfkc.kernel.gram(Y.train, U.train, V = 12, block.size = 25, seed = 1)
g
res.g <- nmfkc(Y.train, A = g, rank = 3, verbose = FALSE)

## Same landmarks and beta, explicit 12 x 100 matrix: same fit
A <- nmfkc.kernel(g$landmarks, U.train, beta = g$beta)
res.m <- nmfkc(Y.train, A = A, rank = 3, verbose = FALSE)
all.equal(res.g$C, res.m$C)

## Test-set prediction: kernel values against the stored landmarks
A.test <- nmfkc.kernel(g$landmarks, U.test, beta = g$beta)
pred   <- predict(res.g, newA = A.test, type = "class")
mean(pred == as.character(iris$Species[-idx]))
```

---

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. **Note**: the convergence test uses the *unpenalized* squared error, not the penalized objective the updates minimize, so with a large `C.L1` or `X.L2.ortho` the iteration can stop while the optimized quantity is still moving. The other fitters test the penalized value; this is kept as-is so existing fits reproduce.
`...`	Hidden options: `nstart` (default 1; see note below), `seed` (default 123), `X.restriction`, `X.init`, `C.init` (tri only) or `Cp.init`/`Cn.init` (signed only), `Y.weights`, `diag.exclude`, `C.L1` (tri only), `X.L2.ortho`, `prefix`. `diag.exclude` (default `FALSE`): when `TRUE` the diagonal of Y (self-loops) is neither fitted nor counted in the fit statistics – for a network without self-loops, whose zero diagonal is not an observation. It is the same as a `Y.weights` with a zero diagonal (and is combined with a `Y.weights` or an `NA` mask when one is given); the fit then carries `diag.exclude = TRUE`. `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).

The anchor options of nmfkc (X.init = "spa", X.anchor) are not available for the symmetric model; passing them is an error rather than being ignored. 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("neato", "fdp", "twopi", "circo", "dot"),
  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. It is read on the membership scale for X edges (`X.prob`, rows summing to one) and on C for the basis-to-basis edges, and it does two jobs at once, switching over at $1/Q$:

- **At or below** $1/Q$ it only thins edges. No node can be hidden, because a row of `X.prob` sums to one and so has a maximum of at least $1/Q$; every node genuinely keeps an edge.
- **Above** $1/Q$ it may also start hiding nodes (when `hide.isolated = TRUE`). Not immediately: nodes go only once the threshold passes the smallest row maximum of `X.prob`, which is data-dependent and can be well above $1/Q$. Inspect `apply(fit$X.prob, 1, max)` to see where that is.

The default `0.01` therefore draws every node — guaranteed for any $Q \leq 100$, since only then is $0.01 \leq 1/Q$ — and most edges. For a readable community graph, 0.2–0.4 usually thins the edges enough while keeping all nodes.

`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`. The test is on the membership scale (`X.prob`), the same quantity the X edges are drawn from, so it does not shift with `type` or `X.restriction`. See `threshold` for when this actually removes anything — at the default it never does for any $Q \leq 100$, which is correct rather than a limitation.

`Y.label` Character vector of labels for outer nodes.

`X.label` Character vector of labels for basis nodes.

`Y.title` Cluster title for outer nodes.

`X.title` Cluster title for basis nodes.

`show.theta` Logical or 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, in recommended order: `"neato"` (default; spring model, clearest for small/medium community graphs), `"fdp"` (force-directed, scales to larger graphs), `"twopi"` (radial), `"circo"` (circular), `"dot"` (hierarchical). For community networks, `"neato"` or `"fdp"` with a raised `threshold` (e.g.\ 0.2–0.3) separate the groups best.

X.color	Color palette for basis nodes (length Q).
Y.cluster	Coloring mode for outer nodes: "soft" (weighted mix) or "hard" (most probable basis color).

Value

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

Lifecycle

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

See Also

nmfkc.net.inference, nmfkc.DOT, plot.nmfkc.DOT

Examples

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

---

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

Description

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

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

Usage

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

Arguments

Y	Symmetric N x N non-negative matrix.
rank	Integer vector of ranks to evaluate. Default 1:3.
type	Model type: "tri" (default), "bi", or "signed".
...	Passed to the underlying fitter; also accepts:

- nfolds (default 5; div alias) and seed (default 123).
- diag.exclude (default FALSE): leave the diagonal out of the folds and out of every fit (see nmfkc.net), for a network without self-loops. Otherwise its zeros would be scored as edges to predict.
- folds: the folds to use, as a list of linear indices into Y – e.g.\ the $folds of an earlier run, so that two models are compared on the same split. A lower-triangle index is read as its upper-triangle mirror. An entry in two folds, or one that cannot be held out (NA, zero weight, or the diagonal under diag.exclude), is an error. nfolds is then length(folds). (Within this function the folds depend only on N, nfolds, seed and the entries that can be held out, so runs that share these already share their folds.)
- seeds: a vector of seeds (overriding seed), to repeat the CV over different splits. Each split refits with its seed, so the results for seed s are those of seed = s. objfunc, sigma and r.squared.cv are then averaged over the seeds, with their spread in objfunc.sd and r.squared.cv.sd. Not with folds.
- pred (default FALSE): also return the held-out predictions (an $N \times N$ array per rank; kept optional because it is N^2 per rank).
- Y.weights: as in nmfkc.net. Entries of weight 0 are never held out, every fit keeps the weights, and the held-out loss is the weighted mean. (Before 1.0.0 a Y.weights given here was silently replaced by the fold mask.)
- cores (getOption("mc.cores", 1L)) to evaluate the rank x fold grid in parallel; results are identical to the sequential run for any cores (PSOCK cluster on Windows, forking elsewhere).

NA entries of Y are never held out and stay masked in every fit (before 1.0.0 they were fitted as zeros and made objfunc NA).

Value

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

r.squared.cv Held-out R^2 per rank, pooled over the folds: $1 - \sum(y-\hat{y})^2 / \sum(y-\bar{y})^2$ over all held-out upper-triangle entries (weighted when Y.weights is given). Unlike the r.squared of a fit (a squared correlation), it can be negative.

pred With pred = TRUE: $N \times N \times$ (ranks) array of held-out predictions, mirrored to the lower triangle; NA where an entry was never held out.

diag.exclude TRUE, present only when used.

seeds, objfunc.rep, objfunc.sd, r.squared.cv.rep, r.squared.cv.sd, folds.rep Present only with two or more seeds: the per-seed values (ranks x seeds), their standard deviations, and each seed's folds. objfunc.fold, folds and pred are those of the first seed.

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.net.rank` *Rank selection for nmfkc.net (concise diagnostics)*

Description

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

Usage

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

Arguments

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

Value

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

References

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

See Also

`nmfkc.net`, `nmfkc.net.ecv`, `nmfkc.rank`

Examples

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

---

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

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

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

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

Usage

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

Arguments

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

Value

A list containing:

`rank.best` The estimated optimal rank. Prioritizes ECV minimum if available, otherwise R-squared Elbow.

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

References

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

See Also

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

Examples

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

`nmfkc.residual.plot` *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.rff.beta.cv` *Select the bandwidth of random-feature covariates by cross-validation*

Description

Chooses the Gaussian-kernel bandwidth β (and optionally the number of random features D) for Random Fourier Feature covariates by cross-validation, in the same way that `nmfkc.kernel.beta.cv` does for exact kernel covariates. For every candidate the random features are regenerated on the columns of `U` and the model is cross-validated column-wise:

- `type = "signed"`: cosine features from `nmfkc.signed.rff` (signed), fitted with `nmfkc.signed` through `nmfkc.signed.cv`;
- `type = "positive"`: positive random features from `nmfkc.rff.positive` (non-negative), fitted with `nmfkc` through `nmfkc.cv`.

The candidate with the smallest cross-validated prediction error is returned. Since the same random map cannot be shared across folds without leaking (the map is data-independent, so it can: only the fit is cross-validated), one draw per candidate is used and its `seed` is fixed so the comparison across β is paired.

For large N set `sample.size`: the selection then runs on a random subsample of the columns (default: all columns), after which the chosen β is used with `nmfkc.signed.rff.gram` / `nmfkc.rff.positive.gram` on the full data.

Usage

```
nmfkc.rff.beta.cv(
  Y,
  rank = 2,
  U,
  beta = NULL,
  D = 500L,
  type = c("signed", "positive"),
  plot = TRUE,
  ...
)
```

Arguments

`Y`	Response matrix $P \times N$ (non-negative for `type = "positive"`).
`rank`	Number of bases Q.
`U`	Input matrix $p \times N$; columns are samples.
`beta`	Numeric vector of candidate bandwidths. Default `NULL`: the seven-point grid of `nmfkc.kernel.beta.nearest.med` around the median nearest-neighbour heuristic (computed on the subsample when `sample.size` is set).
`D`	Integer vector of candidate numbers of random features. A single value (default `500`) selects β only; several values select over the (β, D) grid.

type	"signed" (cosine RFF, default) or "positive".
plot	Logical. If TRUE (default), plot the CV objective against beta (one line per D).
...	Passed to the CV function (nmfkc.signed.cv or nmfkc.cv: e.g. nfolds, epsilon, maxit). Also accepts: • sample.size: number of columns used for the selection (default all). • seed: integer seed for the subsample and the random maps (default 123; the caller's stream is restored). • cores: evaluate the candidates in parallel with .nmfkc.parlapply (default getOption("mc.cores", 1L)). Each candidate is deterministic given its index, so the result is identical for any cores. • hyperbolic: passed to nmfkc.rff.positive.

Value

A list with

beta, D The selected bandwidth and feature count.

objfunc Matrix of CV objective values, length(beta) rows by length(D) columns.

beta.candidates, D.candidates The grids.

type, sample.size, n.used Settings used.

Lifecycle

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

See Also

nmfkc.kernel.beta.cv (exact kernel; also works for Nystr\"om covariates with U = landmarks, V = data), nmfkc.signed.rff.gram, nmfkc.rff.positive.gram

Examples

```
data(iris)
set.seed(1)
idx <- sample(nrow(iris), 100)
U <- t(scale(iris[idx, 1:4]))
levs <- levels(iris$Species)
Y <- sapply(iris$Species[idx], function(s) as.integer(levs == s)); rownames(Y) <- levs

## signed RFF: choose beta over the median-heuristic grid at D = 50
sel <- nmfkc.rff.beta.cv(Y, rank = 3, U, D = 50, plot = FALSE)
sel$beta
g   <- nmfkc.signed.rff.gram(Y, U, beta = sel$beta, D = 50, seed = 1)
fit <- nmfkc.signed(Y, A = g, rank = 3, verbose = FALSE)

## positive RFF: select over a (beta, D) grid
sel2 <- nmfkc.rff.beta.cv(Y, rank = 3, U, D = c(100, 400), type = "positive",
                          plot = FALSE, verbose = FALSE)
sel2$objfunc
```

`nmfkc.rff.positive` *Positive (non-negative) random features for the Gaussian kernel*

Description

Generates $\omega_d \sim \mathcal{N}(0, I_p)$ and applies the non-negative feature map

$$\phi_d(u) = \frac{1}{\sqrt{D}} \exp\big(\sqrt{2\beta}\,\omega_d^\top u - 2\beta\|u\|^2 + \kappa\big) \;\geq 0,$$

for which $\mathbb{E}[\phi(u)^\top \phi(u')] = e^{2\kappa}\exp(-\beta\|u-u'\|^2)$: the Gaussian kernel with bandwidth `beta`, times a constant. The $D \times N$ matrix Φ therefore satisfies $\Phi^\top\Phi \approx e^{2\kappa}K$ *and is non-negative*, so it can be used as the covariate matrix of `nmfkc` directly – unlike the cosine Random Fourier Features of `nmfkc.signed.rff`, which take both signs and need `nmfkc.signed`. This is the "positive random features" construction of Choromanski et al. (2021), stated there for the softmax kernel, transported to the Gaussian kernel by rescaling.

The constant κ is minus the largest exponent $\sqrt{2\beta}\,\omega_d^\top u_n - 2\beta\|u_n\|^2$ observed on the training `U` (stored in `pars`), so every training feature is at most $1/\sqrt{D}$ and `exp()` cannot overflow (points far from the others underflow to zero at large β, which only means that bandwidth fits poorly). It scales the kernel estimate by $e^{2\kappa}$, which Θ absorbs, so the fitted model and its predictions do not depend on it. Note that κ depends on the realized ω: the stabilized features are an unbiased estimator of the randomly rescaled kernel $e^{2\kappa}k$, not of k itself; the unbiasedness statement above is for the unstabilized map. Inputs should nevertheless be centred and scaled: the estimator's variance grows with $\beta\|u-u'\|^2$, so positive features need a larger `D` than cosine features for the same accuracy.

Usage

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

Arguments

`U`	A $p \times N$ numeric matrix; columns are data points.
`beta`	Positive scalar. Gaussian kernel bandwidth parameter $k(u,u') = \exp(-\beta\|u-u'\|^2)$. Can be obtained via `nmfkc.kernel.beta.nearest.med`. Ignored when `pars` is supplied.
`D`	Integer. Number of random features (default `ceiling(ncol(U) / 2)`; choose explicitly for large N). With `hyperbolic = TRUE` the D features are $D/2$ antithetic pairs. Ignored when `pars` is supplied.
`seed`	Optional integer passed to `set.seed()` before drawing ω; the caller's random stream is restored.
`...`	Hidden options: • `pars`: a list from a previous call; when supplied the same random map (and the same κ) is reused and `beta`, `D`, `seed` are ignored. Use this for test data. • `hyperbolic`: logical (default `FALSE`). If `TRUE`, each ω_d is paired with $-\omega_d$ (the "hyperbolic" / antithetic variant of Choromanski et al.).

Value

A list with `Z`, the non-negative $D \times N$ feature matrix, and `pars = list(omega, D, beta, kappa, hyperbolic, type = "positive")`. Pass `Z` to `nmfkc` (or `nmfkc.signed`) as `A`, and `pars` back to this function to build features for new data.

Lifecycle

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

References

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., & Weller, A. (2021). Rethinking attention with Performers. *ICLR*. arXiv:2009.14794.

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in NIPS*, 20.

See Also

`nmfkc.rff.positive.gram` (large N), `nmfkc.signed.rff` (signed cosine features), `nmfkc.kernel`, `nmfkc`

Examples

```
set.seed(1)
U <- scale(matrix(stats::rnorm(3 * 40), 3, 40))        # centred inputs
pr <- nmfkc.rff.positive(U, beta = 0.5, D = 20000, seed = 1)
all(pr$Z >= 0)
## Phi' Phi / exp(2 kappa) approximates the Gaussian kernel
K  <- nmfkc.kernel(U, beta = 0.5)
Kp <- crossprod(pr$Z) / exp(2 * pr$pars$kappa)
max(abs(Kp - K))

## The same random map on new data
U.new <- matrix(stats::rnorm(3 * 5), 3, 5)
dim(nmfkc.rff.positive(U.new, pars = pr$pars)$Z)      # 20000 x 5
```

`nmfkc.rff.positive.gram` *Block-wise Gram accumulation of positive random features for large N*

Description

The nmfkc.rff.positive counterpart of nmfkc.signed.rff.gram: walks the columns of U in blocks, generates the non-negative feature block Φ_b with the same random map, accumulates $S = \Phi\Phi^\top$ ($D \times D$) and $G_0 = Y\Phi^\top$ ($P \times D$), and discards the block, so the $D \times N$ feature matrix never exists in memory. Because the features are non-negative, the returned "nmfkc.gram" object is accepted by nmfkc (as well as by nmfkc.signed), and the NMF-LAB membership interpretation of $B = \Theta\Phi$ is retained.

Usage

```
nmfkc.rff.positive.gram(
  Y,
  U,
  beta = NULL,
  D = NULL,
  seed = NULL,
  block.size = NULL,
  ...
)
```

Arguments

Y	Non-negative $P \times N$ response matrix (e.g. one-hot labels). NA is not allowed.
U	A $p \times N$ numeric matrix of inputs; columns are samples.
beta	Positive scalar. Gaussian kernel bandwidth. Ignored when pars is supplied.
D	Integer. Number of random features. Required (no $N/2$ default here). Ignored when pars is supplied.
seed	Optional integer seed for ω; the caller's stream is restored. Ignored when pars is supplied.
block.size	Integer. Columns of U per block. Default: about 256 MB of features per block (2^25 / D columns), capped at N.
...	Hidden options pars (reuse a random map from nmfkc.rff.positive or a previous call) and hyperbolic (see nmfkc.rff.positive).

Value

An object of class "nmfkc.gram" with S, G0, N, D, type = "prf", signed = FALSE, pars, beta, block.size, n.blocks, A.block (block generator) and rownames = NULL. Features for new data: nmfkc.rff.positive(U.new, pars = g$pars)$Z.

Lifecycle

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

References

Choromanski, K. et al. (2021). Rethinking attention with Performers. *ICLR*. arXiv:2009.14794.

See Also

nmfkc.rff.positive, nmfkc, nmfkc.kernel.gram (Nystr\"om), nmfkc.signed.rff.gram

Examples

```
data(iris)
set.seed(1)
idx <- sample(nrow(iris), 100)
mn <- colMeans(iris[idx, 1:4]); sc <- apply(iris[idx, 1:4], 2, sd)
U.train <- t(scale(iris[idx,  1:4], center = mn, scale = sc))
U.test  <- t(scale(iris[-idx, 1:4], center = mn, scale = sc))
levs    <- levels(iris$Species)
Y.train <- sapply(iris$Species[idx], function(s) as.integer(levs == s))
rownames(Y.train) <- levs
beta <- nmfkc.kernel.beta.nearest.med(U.train)$beta

g <- nmfkc.rff.positive.gram(Y.train, U.train, beta = beta, D = 200,
                             seed = 1, block.size = 25)
g
res <- nmfkc(Y.train, A = g, rank = 3, verbose = FALSE)   # standard NMF-LAB
Z.test <- nmfkc.rff.positive(U.test, pars = g$pars)$Z
pred   <- predict(res, newA = Z.test, type = "class")
mean(pred == as.character(iris$Species[-idx]))
```

---

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). **Large N:** A may instead be a Gram object of class "nmfkc.gram" returned by nmfkc.signed.rff.gram, which holds only $S = AA^\top$ ($D \times D$) and $G_0 = YA^\top$ ($Q_{\mathrm{obs}} \times D$) accumulated over column blocks, so the $D \times N$ feature matrix never has to exist in memory. The multiplicative updates are unchanged (they only ever use S and G_0), and the fit is the same as with the matrix up to floating-point summation order. Three things differ with Gram input: warm.start is unavailable (the posneg warm-start needs the $2D \times N$ split matrix) and the direct initialization is used with a message; Y.weights and NA in Y are errors (the weighted loss needs A on every iteration); and the fold-based helpers nmfkc.signed.cv / nmfkc.signed.ecv / nmfkc.signed.rank do not accept it – select tuning parameters on a held-out validation set, scoring with predict.nmfkc.signed on features regenerated by nmfkc.signed.rff with the stored pars. B, XB and the residual statistics are rebuilt block by block from the object's feature generator.
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: normalization applied to X after every update. One of "colSums" (default, $\mathrm{colSums}(X) = 1$), "colSqSums", "totalSum", "none", "fixed". All of them are gauge fixes: the scale is divided out of X and multiplied into C_+, C_-, so XC and hence the fit is unchanged and only the parametrization is pinned down. "rowSums" was removed in 0.9.8 and is now refused: scaling the rows of X changes XCA, so it was a restriction

acting rather than a reparametrization, and the objective oscillated instead of descending.

- `X.L2.ortho`: non-negative L2 orthogonality penalty on the columns of X (default 0), penalizing $(\lambda/2)\|\mathrm{offdiag}(X^\top X)\|^2$. Same convention as `nmfkc`; skipped when `X.restriction = "fixed"`.
- `X.L2.smooth`: non-negative L2 row-smoothness penalty on X (default 0), penalizing $(\lambda/2)\,\mathrm{tr}(X^\top L X)$ with L the path-graph Laplacian over rows (adjacent-row differences). Useful for ordered rows (e.g. time / space); skipped when `X.restriction = "fixed"`.
- `C.L2`: non-negative ridge penalty on the signed coefficient matrix $C = C_+ - C_-$ (default 0), adding $\lambda\|C_+ - C_-\|^2$. Shrinks Θ toward zero (with zero gradient on the unidentified common mode $C_+ + C_-$), injected into both the fast unweighted and the weighted C_+/C_- updates.
- `C.L1`: non-negative lasso penalty on the signed coefficient matrix (default 0), adding $\lambda\|C\|_1$. Under the split $C = C_+ - C_-$ this is $\lambda\sum(C_+ + C_-)$: at a minimizer C_+ and C_- have disjoint supports, so the sum equals the absolute value, and the constant gradient enters only the denominators of both updates (same $\lambda/2$ convention as `nmfkc`). Multiplicative updates approach zero geometrically rather than reaching it, so the effect is shrinkage; threshold the result if a sparse support is wanted.
- `update.power`: exponent applied to the multiplicative ratio in the unweighted sweep (default 1). With 1 the updates are of the Lee–Seung form and coincide with those of `nmfkc` when $A \geq 0$; with 0.5 they are the square-root form of Ding, Li and Jordan (2010), which is the form their monotonicity proof for the nonnegative factor under a signed Gram matrix covers. Both have the same fixed points; the root takes smaller steps and needs more iterations. Ignored when `Y.weights` is supplied (the weighted sweep has its own damping with backtracking).
- `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. `"spa"` (see `nmfkc`) requires a non-negative Y.
- `X.anchor`: anchor rows of X, imposed as in `nmfkc`: `NULL` (default), `"spa"`, or one row of Y per basis. With a signed Y the rows must be given: SPA compares rows by their shares, which a negative entry leaves undefined, and the start then uses the semi-NMF basis step. As in `nmfkc()`, $B = CA$, so `rank <= nrow(A)` is required. The rows used are returned as `X.anchor`.
- `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.signed`: number of restarts of the signed fit (default 1). **Signed models have more local minima than non-negative ones** because $\Theta = C_+ - C_-$ can take both positive and negative values. With `nstart.signed > 1` the whole fit is repeated from that many consecutive seeds (`seed`, `seed + 1`, ...) and the fit with the smallest `$objfunc` is returned; the returned object then also carries `$restarts`, a data frame of the seed, objective, iteration count and convergence flag of every start. Note that this is a different quantity from `nstart`, which – here as in `nmfkc` – only governs the initialization of X and is forwarded to the non-negative warm-start fit. A budget of 10-50 restarts is recommended for publication-grade runs, especially when the number of classes is large. Restarts are cheap for Gram input, where S and G_0 are already accumulated.
- `cores`: number of workers used for the restarts when `nstart.signed > 1` (default 1, i.e. sequential).

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`: $\mathrm{cor}(Y, \widehat{Y})^2$ (Pearson; in $[0, 1]$).
- `r.squared.uncentered`: uncentered $R^2 = 1 - \|Y - \widehat{Y}\|_F^2 / \|Y\|_F^2$ (baseline = zero matrix).

- `r.squared.centered`: row-mean centered $1 - \|Y - \widehat{Y}\|_F^2 / \|Y - \bar{Y}_{p\cdot}\|_F^2$.
- `mae`: mean absolute error.
- `iter`: number of iterations performed.
- `converged`: logical; whether the relative change of the objective fell below `epsilon` before `maxit`.
- `epsilon.iter`: relative change of the objective at the last step (the quantity compared with `epsilon`).
- `objfunc.increases`: number of steps at which the objective rose. Every restriction on offer is a gauge fix and the unweighted multiplicative update is monotone, so this should be zero; a positive count means something outside the multiplicative form is pushing the iterate back, and such a fit can oscillate and exhaust `maxit`. It is what identified the two offenders removed in 0.9.8.
- `runtime`: elapsed seconds.
- `Y.signed`: logical; whether Y contained negative entries during fitting.
- `pars`: RFF generating parameters, if supplied.
- `restarts`: present only when `nstart.signed` > 1; a data frame with one row per start (`seed`, `objfunc`, `iter`, `converged`, `epsilon.iter`, `objfunc.increases`). A start whose objective is larger than the best one has reached a different local minimum; such a start typically also stops after far fewer iterations.
- `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). `Y.weights` (a matrix, one weight per column, or a scalar) is kept in every fit; cells of weight 0 are never held out, and the held-out loss is the weighted mean. (Before 1.0.0 a `Y.weights` given here was dropped.)

Value

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

Lifecycle

This function is **experimental**.

References

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

See Also

`nmfkc.signed`, `nmfkc.signed.cv`

nmfkc.signed.rank *Rank selection for nmfkc.signed (concise diagnostics)*

Description

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

Usage

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

Arguments

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

Value

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

References

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

See Also

nmfkc.signed, nmfkc.rank

`nmfkc.signed.rff` *Random Fourier Features for nmfkc.signed()*

Description

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

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

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

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

Usage

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

Arguments

`U`	A $p \times N$ numeric matrix; columns are data points.
`beta`	Positive scalar. Gaussian kernel bandwidth parameter. Can be obtained via `nmfkc.kernel.beta.nearest.med`. May be `NULL` only when `pars` is supplied through `....`
`D`	Integer. Number of random features. Defaults to `ceiling(ncol(U) / 2)`. This default is intended for the **training-time fresh generation** only; for test data, always supply `pars` via `...` to inherit the training D together with ω, b. For very large N the default may be excessive (direct MU cost is $O(QD^2)$); choose a smaller D manually. For very small N, RFF is not recommended; use a full kernel matrix with `nmfkc` instead.
`seed`	Optional integer passed to `set.seed()` before generating ω, b, for reproducibility. Ignored when `pars` is supplied.
`...`	Hidden option `pars`: a list `list(omega, b, D, beta)` obtained from a previous call (`Ztrain$pars`). When supplied, ω, b are reused and the `beta`, `D`, `seed` arguments are ignored. Use this to apply the same random map to test data.

Value

A list with two elements:

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

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

Lifecycle

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

See Also

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

Examples

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

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

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

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

## CV over beta candidates: for each beta, generate RFF, fit, and
## evaluate column-wise CV-MSE on training data
cv_mse <- numeric(length(betas))
for (i in seq_along(betas)) {
  rff_i   <- nmfkc.signed.rff(U.train, beta = betas[i], D = 50, seed = 1)
  cv_i    <- nmfkc.signed.cv(Y.train, A = rff_i$Z, rank = 3, seed = 123)
  cv_mse[i] <- cv_i$objfunc
}
beta_best <- betas[which.min(cv_mse)]

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

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

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

```
pred.test  <- predict(res, newA = rff.test$Z,  type = "class")
mean(pred.train == as.character(lab.train))
mean(pred.test  == as.character(lab.test))
```

`nmfkc.signed.rff.gram` *Block-wise Gram accumulation of Random Fourier Features for large N*

Description

Builds the two summary matrices that `nmfkc.signed` needs from a Random Fourier Feature (RFF) covariate matrix Z ($D \times N$) without ever forming Z itself:

$$S = ZZ^\top \quad (D \times D), \qquad G_0 = YZ^\top \quad (Q_{\mathrm{obs}} \times D).$$

The input `U` is processed in column blocks; for each block the features $Z_b = \sqrt{2/D}\cos(\omega U_b + b)$ are generated with `nmfkc.signed.rff`, added into S and G_0, and discarded. Peak memory is therefore $O(D^2 + Q_{\mathrm{obs}}D + D \cdot \texttt{block.size})$ instead of $O(DN)$, which is what makes the RFF route usable when N is in the hundreds of thousands and D exceeds the input dimension p by a large factor (the $D \times N$ matrix is D/p times the size of the data itself).

The returned object is passed to `nmfkc.signed` as its `A` argument. The multiplicative updates are unchanged, since they only ever use S and G_0; the fit equals the one obtained from the explicit matrix `nmfkc.signed.rff(U, ...)$Z` with `warm.start = FALSE` up to floating-point summation order.

Usage

```
nmfkc.signed.rff.gram(
  Y,
  U,
  beta = NULL,
  D = NULL,
  seed = NULL,
  block.size = NULL,
  ...
)
```

Arguments

`Y`	Real-valued $Q_{\mathrm{obs}} \times N$ response matrix (e.g. a one-hot label matrix). `NA` is not allowed.
`U`	A $p \times N$ numeric matrix of inputs; columns are samples.
`beta`	Positive scalar. Gaussian kernel bandwidth $k(u, u') = \exp(-\beta\|u - u'\|^2)$. Can be obtained with `nmfkc.kernel.beta.nearest.med`, which itself works on a random subsample. Ignored when `pars` is supplied.

`D`	Integer. Number of random features. Required (there is no `N/2` default here: on the large-N problems this function is for, that default would be enormous, and the per-iteration cost of `nmfkc.signed()` is $O(QD^2)$). Ignored when `pars` is supplied.
`seed`	Optional integer passed to `set.seed()` before generating ω, b; the caller's random stream is restored afterwards. Ignored when `pars` is supplied.
`block.size`	Integer. Number of columns of `U` per block. Default: as many columns as keep one block of features near 256 MB (`2^25 / D` columns), capped at `N`.
`...`	Hidden option `pars`: a list `list(omega, b, D, beta)` from a previous `nmfkc.signed.rff` or `nmfkc.signed.rff.gram` call. When supplied, the same random map is reused and `beta`, `D`, `seed` are ignored.

Value

An object of class `"nmfkc.gram"`: a list with

`S` $D \times D$ matrix $ZZ^\top$ (signed, positive semi-definite).

`G0` $Q_{\mathrm{obs}} \times D$ matrix $YZ^\top$ (signed).

`N`, `D` Number of samples and of features.

`pars` The RFF parameters `list(omega, b, D, beta)`. Pass them to `nmfkc.signed.rff` to generate features for new data, e.g. for `predict.nmfkc.signed`.

`block.size`, `n.blocks` Blocking actually used.

`A.block` A function `A.block(idx)` returning the $D \times |\texttt{idx}|$ feature block for the columns `idx` of `U`. `nmfkc.signed` uses it to rebuild $B = CZ$ block by block after the fit. It closes over `U` and `pars`, so the object keeps `U` alive (no copy is made).

`rownames` Always `NULL` (RFF features are unnamed); present so that `nmfkc.signed()` can treat the object like a matrix.

Choosing the bandwidth

With Gram input the fold-based `nmfkc.signed.cv` is not available (it must split the columns of A). Select β as in the NMF-LAB paper: build one Gram object per candidate on the training columns, fit, and score the validation columns with `predict()` on features from `nmfkc.signed.rff(U.val,pars = g$pars)`.

Lifecycle

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

References

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in NIPS*, 20.

Satoh, K. (2026). Applying non-negative matrix factorization with covariates to label matrix for classification. *Japanese Journal of Statistics and Data Science*. doi:10.1007/s4208102600349x

See Also

nmfkc.signed, nmfkc.signed.rff, predict.nmfkc.signed, nmfkc.kernel.beta.nearest.med

Examples

```
## Iris, 3 classes: the Gram route gives the same fit as the explicit
## feature matrix, without ever holding the D x N matrix.
data(iris)
set.seed(1)
idx <- sample(nrow(iris), 100)
mn <- colMeans(iris[idx, 1:4]); sc <- apply(iris[idx, 1:4], 2, sd)
U.train <- t(scale(iris[idx,  1:4], center = mn, scale = sc))   # 4 x 100
U.test  <- t(scale(iris[-idx, 1:4], center = mn, scale = sc))   # 4 x  50
levs    <- levels(iris$Species)
Y.train <- sapply(iris$Species[idx], function(s) as.integer(levs == s))
rownames(Y.train) <- levs
beta <- nmfkc.kernel.beta.nearest.med(U.train)$beta

## Accumulate S, G0 over blocks of 25 columns (4 blocks here)
g <- nmfkc.signed.rff.gram(Y.train, U.train, beta = beta, D = 50,
                           seed = 1, block.size = 25)
g
res.g <- nmfkc.signed(Y.train, A = g, rank = 3, verbose = FALSE)

## Same random map, explicit matrix, no warm-start: same fit
Z <- nmfkc.signed.rff(U.train, pars = g$pars)$Z              # 50 x 100
res.m <- nmfkc.signed(Y.train, A = Z, rank = 3, warm.start = FALSE,
                      verbose = FALSE)
all.equal(res.g$C, res.m$C)

## Test-set prediction: regenerate features with the stored pars
Z.test <- nmfkc.signed.rff(U.test, pars = g$pars)$Z
pred   <- predict(res.g, newA = Z.test, type = "class")
mean(pred == as.character(iris$Species[-idx]))
```

---

nmfre *Non-negative Matrix Factorization with Random Effects*

Description

Estimates the NMF-RE model

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

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

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

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

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

Usage

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

Arguments

`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 `...`.
`C.signed`	Logical. Whether the fixed-effect coefficients C ($= \Theta$ in the paper) are sign-free; the single switch that selects the whole estimation scheme. `TRUE` (default, recommended, matches the paper) treats C as real-valued and updates it by exact least squares, with the basis X estimated by the complete-EM semi-NMF step and a two-sided test (interior null). `FALSE` constrains $C \geq 0$ via a multiplicative update, with X estimated by the positive-part multiplicative update and a one-sided test (boundary null). The basis X is always non-negative. A character value (`"signed"` / `"nonneg"`) is also accepted for backward compatibility.
`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. A method name (`"kmeans"`, `"nndsvd"`, `"spa"`, ...) is forwarded to that `nmfkc()` initialization.
- `X.anchor`: anchor rows of X, imposed as in `nmfkc`: `NULL` (default), `"spa"`, or one row of `Y` per basis. The zeros are kept for the whole fit, including through the floor this function applies to X. There is no `rank <= nrow(A)` limit here: the random effects give the scores full rank. The rows used are returned as `X.anchor`.
- `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`).
- `x.postvar`: Logical. Include the posterior-variance term in the semi-NMF X-step (default `TRUE`; advanced). Applies only to the sign-free / semi-NMF path (`C.signed = TRUE`).
- `X.L2.smooth`: Non-negative row-smoothness penalty on the basis X (default 0), adding $(\lambda/2)\,\mathrm{tr}(X^\top L X)$ with L the path-graph Laplacian over rows (squared adjacent-row differences). Well suited to ordered rows (e.g.\ longitudinal / spatial bases). Injected into the X-step for both `C.signed = TRUE/FALSE`; leaves the random-effect variance updates unchanged.
- `X.L2.ortho`: Non-negative column-orthogonality penalty on X (default 0), penalizing $(\lambda/2)\|\mathrm{offdiag}(X^\top X)\|^2$.
- `C.L2`: Non-negative ridge penalty on $\Theta = C$ (default 0), adding $\lambda\|C\|^2$. A Gaussian prior on the fixed effects; for `C.signed = TRUE` the C-step stays a closed-form Sylvester ridge-LS solve, for `C.signed = FALSE` it is added to the multiplicative-update denominator.
- `dfU.control`: Deprecated and inert. The algorithm imposes no cap on df_U (`"off"`, the only behaviour); df_U is reported as a diagnostic only.
- `print.trace`: Logical. If `TRUE`, print progress every 100 iterations (default `FALSE`).
- `seed`: Integer seed for reproducibility (default 1).
- `nstart`: Number of random restarts for the `nmfkc()` initialisation step (passed to the k-means initialiser). Default `1` (single start; historical behaviour). A larger value (e.g.\ 10-20) gives a more stable initialisation.
- `inner.maxit`, `outer.maxit`: Maximum inner (fixed-λ block-coordinate) and outer (EM variance) iterations (defaults `10000` and `500`).
- `epsilon.outer`: Convergence tolerance for the outer EM loop on λ (default `1e-6`).

Value

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

Core matrices

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

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

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

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

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

Variance components

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

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

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

Convergence diagnostics

`converged` Logical. Whether the algorithm converged.

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

`iter` Number of iterations performed.

`maxit` Maximum iterations setting used.

`epsilon` Convergence tolerance used.

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

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

`objfunc.iter` Numeric vector of the fixed-λ penalized objective $\|Y - X(\Theta A + U)\|^2 + \lambda\|U\|^2$ per iteration. Monotone within an inner loop but *not* across outer iterations (the penalty jumps when λ is updated).

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

`nll.trace` Numeric vector of the marginal negative log-likelihood $\ell(X, \Theta, \sigma^2, \tau^2)$ per iteration (random effects integrated out). This is the ECM-monotone quantity and is what `plot.nmfre` displays.

Effective degrees of freedom (dfU) diagnostics

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

`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` Pearson $\mathrm{cor}(Y, X(\Theta A + U))^2$ (BLUP prediction).

`r.squared.uncentered` Uncentered $1 - \|Y - X(\Theta A + U)\|_F^2 / \|Y\|_F^2$ (BLUP; baseline = zero matrix).

`r.squared.centered` Row-mean centered $1 - \|Y - X(\Theta A + U)\|_F^2 / \|Y - \bar{Y}_{p\cdot}\|_F^2$ (BLUP; baseline = per-row mean).

`r.squared.fixed` Pearson $\mathrm{cor}(Y, X\Theta A)^2$ (fixed-only prediction).

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

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

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

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

Sign convention

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

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

References

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

See Also

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

Examples

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


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

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

`nmfre.ecv` *NMF-RE element-wise cross-validation for rank selection*

Description

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

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

Usage

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

Arguments

`Y`	Observation matrix (P x N), non-negative.
`A`	Covariate matrix (K x N). Default is a row of ones (intercept only).
`rank`	Integer vector of ranks Q to evaluate (default `1:3`).
`C.signed`	Logical. Sign convention for Θ passed to each `nmfre` fit (`TRUE` = sign-free, default; `FALSE` = non-negative). The basis update rule follows this choice automatically.

... Additional arguments:

- nfolds: Number of folds (default 5; legacy nfold also accepted).
- rounds: Iterative-imputation rounds per fold (default 4).
- seed: RNG seed for the fold assignment (default 1).
- cores: Integer; number of parallel workers for the rank sweep (default getOption("mc.cores", 1L)). Because the fold matrix is fixed before the sweep and each nmfre fit self-seeds, results are identical to the sequential run for any cores.
- print.trace: Logical; print per-rank scores (default FALSE).
- Convergence controls forwarded to nmfre: epsilon (default 1e-5), epsilon.outer (default 1e-3), inner.maxit (default 1500), outer.maxit (default 80), maxit (default 40000). CV does not need the tight tolerances of a final fit, so these are loosened by default.

Value

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

rank The ranks evaluated.

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

best The rank minimizing sigma.

nfolds, rounds, C.signed Settings used.

See Also

nmfre, nmfre.inference, nmfkc.ecv

Examples

```
if (requireNamespace("nlme", quietly = TRUE)) {
  Y <- matrix(nlme::Orthodont$distance, 4, 27)
  male <- ifelse(nlme::Orthodont$Sex[seq(1, 108, 4)] == "Male", 1, 0)
  A <- rbind(intercept = 1, male = male)
  cv <- nmfre.ecv(Y, A, rank = 1:3)
  cv$best
  plot(cv)
}
```

---

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

Description

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

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

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

Usage

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

Arguments

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.

`identified.approx`
: Whether (X, C) is unique up to a permutation of the bases, by the data. With `C.signed = FALSE` it is the diagnostic of `nmfkc.inference` on (X, C). With a signed C (the default) non-negativity cannot pin the rotation, so it is `NA`: only the working model's isotropic random effects $U \sim N(0, \tau^2 I)$ do, by allowing orthogonal T alone, which with the column sums of X fixed is a permutation when $Q = 2$ or every basis has an anchor row. That is identification by assumption. An algebraic check on the point estimate: no re-fits.

`identification`
: The parts of the diagnostic, plus `by.assumption` (the isotropic argument above holds, with $\tau^2 > 0$) and, for a signed C, `note`.

References

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

See Also

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

Examples

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

---

plot.nmf.cluster.criteria

Plot clustering-quality criteria across a sequence of fits

Description

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

Usage

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

Arguments

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

Value

x, invisibly.

See Also

nmf.cluster.criteria

---

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

Description

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

Usage

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

Arguments

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

Value

x, invisibly.

See Also

nmf.cluster.flow

---

plot.nmf.gmm *Plot method for nmf.gmm objects*

Description

type = "convergence" (default) plots the EM objective ($-\log L$) over iterations. type = "adjusted.scores" draws the cluster scatterplot, coloured by the hard cluster assignment (or by group).

Usage

```
## S3 method for class 'nmf.gmm'
plot(
  x,
  type = c("convergence", "adjusted.scores", "scores"),
  group = NULL,
  ...
)
```

Arguments

x	An object of class "nmf.gmm".
type	"convergence" or "adjusted.scores" ("scores" is an alias for the latter).
group	Optional length-N factor/vector to colour the scatterplot by (e.g. known labels). Default NULL colours by x$cluster.
...	Additional graphical parameters passed to plot.

Details

The scatterplot shows the adjusted scores, not the scores. What is drawn is $\boldsymbol{b}_n - C\boldsymbol{a}_n$, the least-squares scores with the covariate effect removed — which is the space the mixture actually acts in, since the model puts the covariate effect in the component means. The distinction matters: on the crabs example the *raw* scores separate the four true groups no better with a size covariate than without (0.635 versus 0.648, between/within on the first two principal components), while the adjusted space separates them at 4.628. The visible separation is produced by the adjustment, so a panel labelled "scores" would claim something the picture does not show. type = "scores" is accepted as an alias.

The projection depends on the rank: a rank-2 fit is drawn in its own two coordinates (rotating it through prcomp would trade two interpretable part axes for arbitrary ones), a higher rank is projected onto its first two principal components, and a rank-1 fit becomes a strip plot with deterministic stacking.

Value

Invisible NULL.

Development status

Part of the **experimental** NMF-GMM family: see nmf.gmm. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm, nmf.gmm.inference, nmf.gmm.select

Examples

```
set.seed(1)
Y <- matrix(abs(rnorm(20 * 80)) + 1, 20, 80); A <- rbind(1, rnorm(80))
fit <- nmf.gmm(Y, A, rank = 2, K = 3)
plot(fit, type = "adjusted.scores")
```

---

plot.nmf.gmm.select *Plot method for nmf.gmm.select objects*

Description

Draws BIC and ICL against K, with the selected K marked. Every other selector in the package (nmfkc.rank, nmfkc.ard, nmfkc.consensus, nmfkc.bicv) has a plot method; without this one plot(nmf.gmm.select(...)) fell through to plot.default and errored with "'x' is a list, but does not have components 'x' and 'y'".

Usage

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

Arguments

x	An object of class "nmf.gmm.select".
...	Passed to the underlying plot call.

Value

x, invisibly.

Development status

Part of the **experimental** NMF-GMM family: see nmf.gmm. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm.select, nmf.gmm

---

plot.nmf.rank *Plot a rank-selection (nmf.rank) object*

Description

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

Usage

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

Arguments

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

Value

x, invisibly.

See Also

nmfkc.rank

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmf.rrr, nmf.rrr.heatmap

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc, summary.nmfkc

Examples

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

---

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

Description

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

Usage

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

Arguments

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

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

Details

The available panels are

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

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

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

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

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

Value

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

See Also

`nmfkc.ar.latent`, `nmfkc.ar.latent.inference`, `plot.nmfkc.ar.stationarity`, `nmfkc.ar.DOT`

Examples

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

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

Description

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

Usage

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

Arguments

`x`	An object from `nmfkc.ar.stationarity`.
`...`	Passed to the underlying plot call.

Value

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

See Also

`nmfkc.ar.stationarity`, `plot.nmfkc.ar.latent`

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

x, invisibly.

plot.nmfkc.consensus *Plot a consensus rank-selection (nmfkc.consensus) object*

Description

Two views of a nmfkc.consensus result:

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

Usage

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

Arguments

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

Value

x, invisibly.

See Also

nmfkc.consensus

---

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

Description

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

This method handles all DOT objects produced by the nmfkc package: nmfkc.DOT, nmf.rrr.DOT, nmf.ffb.DOT, and nmfkc.ar.DOT.

Usage

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

Arguments

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

Value

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

See Also

nmfkc.DOT, nmf.rrr.DOT, nmf.ffb.DOT, nmfkc.ar.DOT, print.nmfkc.DOT

---

plot.nmfkc.signed *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.

See Also

nmfkc.signed, nmfkc, nmfkc.signed.cv

---

plot.nmfre *Plot convergence diagnostics for NMF models*

Description

Plots the objective function value over iterations for nmfre and nmf.ffb 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.ffb'
plot(x, ..., which = c("penalized", "reconstruction", "both"))
```

Arguments

x	A fitted model object.
...	Additional graphical arguments passed to plot.
which	For plot.nmf.ffb: 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. Ignored for nmf.ffb(method = "fiml") objects, which carry no iteration trace: for them the BIC of the L1 path (x$path) is drawn against log(lambda1): the line follows the smallest BIC at each penalty (over the starts), every individual proposal is a small grey point, the number of selected paths is printed at each point and the BIC-selected penalty is circled.

Value

Invisible NULL.

See Also

nmfre, nmf.ffb

Examples

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

`plot.nmfre.ecv` *Plot method for nmfre.ecv objects*

Description

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

Usage

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

Arguments

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

Value

The input object, invisibly.

See Also

`nmfre.ecv`

predict.nmf.gmm *Class assignments / responsibilities from an NMF-GMM fit*

Description

Class assignments / responsibilities from an NMF-GMM fit

Usage

```
## S3 method for class 'nmf.gmm'
predict(object, type = c("class", "responsibility"), ...)
```

Arguments

object	An object of class "nmf.gmm".
type	"class" (default) returns the hard cluster labels; "responsibility" returns the N x K posterior-probability matrix.
...	Ignored.

Value

A vector of labels or the responsibility matrix.

Development status

Part of the **experimental** NMF-GMM family: see nmf.gmm. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm, nmf.gmm.inference, nmf.gmm.select

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

Arguments

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

Value

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

See Also

nmfkc, nmfkc.cv

Examples

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

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

Description

Computes $\widehat{Y} = X\,C\,A_{\mathrm{new}}$ $(= X(C_+ - C_-)(A_+^{\mathrm{new}} - A_-^{\mathrm{new}}))$. For `type = "response"` the raw prediction is returned (possibly signed).

With signed covariates the scores $\boldsymbol{b}_n = C\boldsymbol{a}_n$ can be negative, so the NMF-LAB recipe of normalizing $\boldsymbol{b}_n$ to a membership vector does not apply. Since X is column-stochastic, $\widehat{\boldsymbol{y}}_n = X\boldsymbol{b}_n$ still sums to $\sum_q b_{qn}$ (close to one for one-hot targets) and is the least-squares estimate of the class indicator; for `type = "prob"` it is mapped to the probability simplex by the **Euclidean projection**

$$\widehat{\boldsymbol{p}}_n = \underset{\boldsymbol{p}\geq 0,\ \mathbf{1}^\top\boldsymbol{p}=1}{\arg\min} \|\boldsymbol{p} - \widehat{\boldsymbol{y}}_n\|^2 = \max(\widehat{\boldsymbol{y}}_n - \tau_n\mathbf{1}, 0),$$

the closest probability vector in the same Frobenius geometry the fit minimizes (Duchi et al. 2008; Wang & Carreira-Perpinan 2013; $O(P\log P)$ per column by sorting). The former rule – clip negatives to zero and renormalize – is kept as `prob.method = "clip"`. `type = "class"` is $\arg\max_p \widehat{y}_{pn}$, which both rules preserve.

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"`.
`...`	Hidden option `prob.method`: how `type = "prob"` maps $\widehat{\boldsymbol{y}}_n$ to the simplex, `"simplex"` (default; Euclidean projection) or `"clip"` (clip negatives, renormalize).

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.

Duchi, J., Shalev-Shwartz, S., Singer, Y., & Chandra, T. (2008). Efficient projections onto the l1-ball for learning in high dimensions. *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 272–279.

Wang, W., & Carreira-Perpinan, M. A. (2013). Projection onto the probability simplex: an efficient algorithm with a simple proof, and an application. *arXiv:1309.1541*.

See Also

nmfkc.signed, nmfkc, nmfkc.signed.cv

---

predict.nmfre *Predict method for nmfre objects*

Description

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

Usage

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

Arguments

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

Value

A matrix of predicted (P x M) values.

See Also

nmfre, fitted.nmf

`print.nmf` *Print method for fitted NMF models*

Description

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

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

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

Usage

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

Arguments

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

Value

`x`, invisibly.

See Also

`summary.nmfkc`, `nmfkc`

Examples

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

print.nmf.cluster.criteria

Print method for nmf.cluster.criteria objects

Description

Print method for nmf.cluster.criteria objects

Usage

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

Arguments

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

Value

x, invisibly.

print.nmf.cluster.flow

Print method for nmf.cluster.flow objects

Description

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

Usage

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

Arguments

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

Value

x, invisibly.

print.nmf.ffb.test *Print a calibrated test of the feed-forward null*

Description

Reports the observed likelihood ratio, its bootstrap p-value, and the two quantities that say whether the test means anything for these data: the null false-selection rate and, for the default calibration, how often the exclusion restriction moved.

Usage

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

Arguments

x	An object from nmf.ffb.test.
...	Ignored.

Value

x, invisibly.

See Also

nmf.ffb.test

print.nmf.gmm *Print an NMF-GMM fit*

Description

Print an NMF-GMM fit

Usage

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

Arguments

x	An object of class "nmf.gmm".
...	Ignored.

Value

x, invisibly.

Development status

Part of the **experimental** NMF-GMM family: see nmf.gmm. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm, nmf.gmm.inference, nmf.gmm.select

---

print.nmf.gmm.select *Print method for nmf.gmm.select objects*

Description

Print method for nmf.gmm.select objects

Usage

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

Arguments

x	An object of class "nmf.gmm.select".
...	Ignored.

Value

x, invisibly.

Development status

Part of the **experimental** NMF-GMM family: see nmf.gmm. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm, nmf.gmm.inference, nmf.gmm.select

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

Description

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

Usage

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

Arguments

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

Value

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

See Also

nmfkc.inference, nmf.rrr.inference

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

Description

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

Usage

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

Arguments

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

Value

x, invisibly.

print.nmfae.ecv *Print method for nmf.rrr element-wise CV results*

Description

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

Usage

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

Arguments

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

Value

`x`, invisibly.

See Also

`nmf.rrr.ecv`, `plot.nmfae.ecv`

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

Description

Print method for nmfkc.ar.latent objects

Usage

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

Arguments

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

Value

`x`, invisibly.

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

Description

Print method for nmfkc.ar.latent.inference objects

Usage

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

Arguments

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

Value

`x`, invisibly.

`print.nmfkc.ar.stationarity` *Print method for nmfkc.ar.stationarity objects*

Description

Print method for nmfkc.ar.stationarity objects

Usage

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

Arguments

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

Value

`x`, invisibly.

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

Description

Print method for nmfkc.ard objects

Usage

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

Arguments

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

Value

x, invisibly.

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

Description

Print method for nmfkc.consensus objects

Usage

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

Arguments

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

Value

x, invisibly.

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

Description

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

Usage

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

Arguments

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

Value

x, invisibly.

See Also

`plot.nmfkc.DOT`, `nmfkc.DOT`

print.nmfkc.gram *Print method for nmfkc.gram*

Description

Print method for nmfkc.gram

Usage

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

Arguments

x	Object of class `"nmfkc.gram"` (from `nmfkc.kernel.gram` or `nmfkc.signed.rff.gram`).
...	Unused.

Value

Invisible x.

Lifecycle

This function is **experimental**.

See Also

nmfkc.kernel.gram, nmfkc.signed.rff.gram

---

print.nmfre.ecv *Print method for nmfre.ecv objects*

Description

Print method for nmfre.ecv objects

Usage

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

Arguments

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

Value

The input object, invisibly.

---

print.summary.nmf.ffb *Print method for summary.nmf.ffb objects*

Description

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

Usage

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

Arguments

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

Value

Invisible x.

See Also

summary.nmf.ffb

---

print.summary.nmf.gmm *Print method for summary.nmf.gmm objects*

Description

Print method for summary.nmf.gmm objects

Usage

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

Arguments

x	A "summary.nmf.gmm" object.
digits	Minimum significant digits.
...	Ignored.

Value

x, invisibly.

Development status

Part of the **experimental** NMF-GMM family: see nmf.gmm. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm, nmf.gmm.inference, nmf.gmm.select

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,
  by = c("covariate", "basis"),
  ...
)
```

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.
by	Character; grouping order of the coefficients table. "covariate" (default) lists all bases within each covariate (1-1, 1-2, ...); "basis" lists all covariates within each basis (1-1, 2-1, ...).
...	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),
  max.coef = 20,
  by = c("covariate", "basis"),
  ...
)
```

Arguments

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

Value

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

See Also

`summary.nmfkc.net.inference`

`print.summary.nmfkc.net.signed` *Print method for summary.nmfkc.net.signed objects*

Description

Print method for summary.nmfkc.net.signed objects

Usage

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

Arguments

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

Value

Invisible `x`.

See Also

`summary.nmfkc.net.signed`

`print.summary.nmfkc.signed` *Print method for summary.nmfkc.signed*

Description

Print method for summary.nmfkc.signed

Usage

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

Arguments

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

Value

Invisible `x`.

Lifecycle

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

References

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

See Also

`nmfkc.signed`, `nmfkc`, `nmfkc.signed.cv`

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

Description

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

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

Usage

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

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

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

## S3 method for class 'nmf.ffb'
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, nmf.rrr, nmfre, nmf.ffb, fitted.nmf

Examples

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

`residuals.nmf.gmm` *Residuals from an NMF-GMM fit*

Description

$Y - \hat{Y}$, matching `residuals.nmf`. Without this method `residuals()` fell through to `stats::residuals.default` and returned `NULL`, even though `fitted()` worked.

Usage

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

Arguments

`object`	An object of class `"nmf.gmm"`.
`Y`	The observation matrix the model was fitted on.
`...`	Ignored.

Value

The residual matrix.

Development status

Part of the **experimental** NMF-GMM family: see `nmf.gmm`. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm, fitted.nmf.gmm

`summary.nmf.ffb` *Summary method for nmf.ffb 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). For `nmf.ffb(method = "fiml")` objects the fit statistics are the log-likelihoods, parameter counts and BIC of the feed-forward null, the unpenalized feedback fit and the BIC-selected model, the two likelihood-ratio statistics, and – after `nmf.ffb.inference` – their bootstrap p-values and the false-selection rate under the null.

Usage

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

Arguments

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

Value

An object of class `"summary.nmf.ffb"` (the fitted model tagged for printing); printed by `print.summary.nmf.ffb`.

See Also

`nmf.ffb`, `nmf.ffb.inference`

Examples

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

summary.nmf.gmm *Summary of an NMF-GMM fit*

Description

Summary of an NMF-GMM fit

Usage

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

Arguments

object	An object of class "nmf.gmm".
...	Ignored.

Value

An object of class "summary.nmf.gmm".

Development status

Part of the **experimental** NMF-GMM family: see nmf.gmm. The interface may change in future versions — argument names, defaults and the contents of the returned object are not yet stable.

See Also

nmf.gmm, nmf.gmm.inference, nmf.gmm.select

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).

Details

Separability measures how close the basis is to having an anchor row for every basis – a row of X that carries that basis alone. For each basis q take the largest share it has in any row, $\max_j \tilde{x}_{jq} / \sum_r \tilde{x}_{jr}$, with $\tilde{X}$ the basis rescaled so that its columns sum to one (so the value does not depend on `X.restriction`); these are `anchor.purity`, attained at rows `anchor.row`. The separability is their minimum: 1 when every basis has a row of its own, $1/Q$ when every row mixes all bases equally, and `NA` for `rank = 1`. The same three values are stored on the fit, in `fit$criterion`.

Anchors for every basis, together with $\mathrm{rank}(B) = Q$, make the factorization unique up to scale, so a value near 1 says the solution is nearly unique. A fit with `X.anchor` scores exactly 1. The measure looks at X only, however: zeros in B or in Θ can pin the solution as well, so a low value does not show that it is not unique.

Value

An object of class `summary.nmfkc`, containing summary statistics. Among them, `separability`, `anchor.purity` and `anchor.row`; see Details.

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.

See Also

nmfkc.signed, nmfkc, nmfkc.signed.cv

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

Description

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

Usage

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

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

Arguments

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

Value

The input object, invisibly.

See Also

nmfre, nmfre.inference

Examples

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

Index

barplot, *151*

coef, *160*
coef.nmf, 5
coef.nmf.gmm, 6

fitted.nmf, 6, *159*, *176*
fitted.nmf.gmm, 7, *176*
fitted.nmfae *(*fitted.nmf*)*, 6
fitted.nmfre *(*fitted.nmf*)*, 6

image, *152*

kmeans, *51*

lm, *37*, *52*, *61*, *105*, *125*

model.matrix, *32*

n2mfrow, *152*
nmf.cluster.criteria, 8, *10*, *65*, *85*, *114*, *142, 143*
nmf.cluster.flow, *8, 9*, 9, *143, 144*
nmf.ffb, *7, 8*, 10, *17–19*, *21–24*, *26*, *28*, *30*, *154*, *160*, *176, 177*
nmf.ffb.cv, *16*, 16
nmf.ffb.diagnostics, 18, *28*, *30*
nmf.ffb.DOT, *16*, 19, *26*, *91*, *152, 153*
nmf.ffb.ecv, 21, *30*
nmf.ffb.inference, *5*, *11*, *15–21*, 23, *28*, *30*, *177*
nmf.ffb.split, *16*, 26, *27*
nmf.ffb.test, *22–26*, 28, *162*
nmf.gmm, *6*, *8*, 30, *33–36*, *145, 146*, *156*, *163*, *170*, *176*, *178*
nmf.gmm.inference, *6*, *8*, *30*, *32*, 33, *145*, *156*, *163*, *170*, *178*
nmf.gmm.select, *6*, *8*, *30–33*, 34, *145, 146*, *156*, *163*, *170*, *178*
nmf.gmm.twostage, *32*, 35
nmf.rrr, *7, 8*, *12*, 36, *40–46*, *48–51*, *53*, *147*, *160*, *176*
nmf.rrr.cv, 39, *47, 48*
nmf.rrr.DOT, *39*, 41, *45*, *91*, *152, 153*
nmf.rrr.ecv, *39, 40*, 43, *48, 49*, *54, 55*, *165*
nmf.rrr.heatmap, 44, *56*, *147*
nmf.rrr.inference, *5*, *39*, 45, *56, 57*, *164*
nmf.rrr.kernel.beta.cv, *40*, 47
nmf.rrr.rank, 48
nmf.rrr.rename, 49
nmf.rrr.signed, 50, *54–59*
nmf.rrr.signed.ecv, *52*, 54, *57, 58*
nmf.rrr.signed.heatmap, 55
nmf.rrr.signed.inference, 56
nmf.rrr.signed.rank, 57
nmf.rrr.signed.rename, *53*, 59
nmfkc, *7, 8*, *11–13*, *22*, *31*, *36–39*, 60, *67*, *70*, *72*, *75*, *78*, *82–84*, *86, 87*, *91–94*, *96*, *102–104*, *106*, *114–117*, *119–126*, *130*, *135–137*, *148*, *154*, *157*, *159, 160*, *175, 176*, *179*, *182*
nmfkc.ar, *65*, 66, *68*, *70–73*, *77–79*, *94*
nmfkc.ar.degree.cv, *67*, 67, *87*
nmfkc.ar.DOT, *67*, 69, *72*, *91*, *149*, *152, 153*
nmfkc.ar.latent, 70, *73*, *76–79*, *148, 149*
nmfkc.ar.latent.inference, *25*, *70–72*, 73, *78, 79*, *95*, *149*
nmfkc.ar.predict, 77
nmfkc.ar.stationarity, *67*, *70*, *72*, *77*, 78, *150*
nmfkc.ard, 80, *82*, *85*, *93*, *115*, *145*
nmfkc.bicv, *80, 81*, 81, *83*, *85*, *88*, *93*, *115*, *145*
nmfkc.class, 83
nmfkc.consensus, *80–82*, 83, *93*, *115*, *145*, *151, 152*
nmfkc.criterion, 85
nmfkc.cv, *40*, *60*, *65*, *68*, 86, *88*, *93*, *97*, *103*, *117, 118*, *127*, *157*

nmfkc.cv.methods, 88
nmfkc.denormalize, 89, *113*
nmfkc.DOT, *65*, 90, *109*, *138*, *141, 142*, *152, 153*, *168*
nmfkc.ecv, *17, 18*, *22*, *44*, *60*, *65*, *80–83*, *85*, *88*, 92, *114, 115*, *139, 140*
nmfkc.inference, *5*, *25*, *33*, *46*, *65*, *74*, *77*, 93, *111*, *142*, *164*, *179, 180*
nmfkc.kernel, *39*, *47, 48*, *65*, 96, *98*, *101–104*, *120*
nmfkc.kernel.beta.cv, *48*, *87*, 97, *100, 101*, *117, 118*
nmfkc.kernel.beta.nearest.med, *47*, *98*, 98, *101, 102*, *104*, *117*, *119*, *130–132*, *134*
nmfkc.kernel.gaussian, *97, 98*, *100*, 100
nmfkc.kernel.gram, *60*, 102, *122*, *168, 169*
nmfkc.net, *8*, *13*, *63*, 104, *107–113*, *180, 181*
nmfkc.net.DOT, *106*, 107, *111*
nmfkc.net.ecv, *92*, *105, 106*, 109, *112, 113*
nmfkc.net.inference, *107*, *109*, 111, *181*
nmfkc.net.rank, 112
nmfkc.normalize, *89*, 113
nmfkc.rank, *9, 10*, *48, 49*, *57, 58*, *60*, *64, 65*, *81, 82*, *85, 86*, *93*, *112, 113*, 114, *129*, *145, 146*
nmfkc.residual.plot, 115
nmfkc.rff.beta.cv, 117
nmfkc.rff.positive, *117, 118*, 119, *121, 122*
nmfkc.rff.positive.gram, *117, 118*, *120*, 120
nmfkc.signed, *8*, *60*, *64*, *103–105*, *117*, *119–121*, 122, *127–134*, *154*, *159*, *175*, *182*
nmfkc.signed.cv, *117, 118*, *123*, *125*, 127, *128*, *133*, *154*, *159*, *175*, *182*
nmfkc.signed.ecv, *123*, *125*, *127*, 128
nmfkc.signed.rank, *123*, 129
nmfkc.signed.rff, *117*, *119, 120*, *123*, 130, *132–134*
nmfkc.signed.rff.gram, *60*, *104*, *117, 118*, *121–123*, 132, *168, 169*
nmfre, *7, 8*, *30–32*, 134, *139–142*, *154*, *159, 160*, *176*, *183*
nmfre.ecv, 139, *155*
nmfre.inference, *5*, *33*, *135*, *138*, *140*, 140, *183*
plot, *10*, *89*, *143*, *146*, *148*, *152*, *154, 155*, *161*
plot.nmf.cluster.criteria, *9*, 142
plot.nmf.cluster.flow, *10*, 143
plot.nmf.ffb *(*plot.nmfre*)*, 154
plot.nmf.gmm, 144
plot.nmf.gmm.select, 145
plot.nmf.rank, 146
plot.nmfae, *45*, 147
plot.nmfae.ecv, *165*
plot.nmfkc, *65*, 147, *179*
plot.nmfkc.ar.latent, *70*, *72*, 148, *150*
plot.nmfkc.ar.stationarity, *149*, 150
plot.nmfkc.ard, 151
plot.nmfkc.bicv *(*nmfkc.cv.methods*)*, 88
plot.nmfkc.consensus, *85*, 151
plot.nmfkc.cv *(*nmfkc.cv.methods*)*, 88
plot.nmfkc.DOT, *21*, *70*, *91*, *109*, 152, *168*
plot.nmfkc.ecv *(*nmfkc.cv.methods*)*, 88
plot.nmfkc.signed, 153
plot.nmfre, *137*, 154
plot.nmfre.ecv, 155
predict.nmf.gmm, 156
predict.nmfae, *39*
predict.nmfae.signed, *53*
predict.nmfkc, *65*, 156
predict.nmfkc.signed, *123*, *126*, *133, 134*, 158
predict.nmfre, 159
print.nmf, 160
print.nmf.cluster.criteria, 161
print.nmf.cluster.flow, 161
print.nmf.ffb.test, 162
print.nmf.gmm, 162
print.nmf.gmm.select, 163
print.nmf.inference, 164
print.nmf.rank, 164
print.nmfae.ecv, 165
print.nmfkc.ar.latent, 165
print.nmfkc.ar.latent.inference, 166
print.nmfkc.ar.stationarity, 166
print.nmfkc.ard, 167
print.nmfkc.bicv *(*nmfkc.cv.methods*)*, 88
print.nmfkc.consensus, *85*, 167
print.nmfkc.cv *(*nmfkc.cv.methods*)*, 88
print.nmfkc.DOT, *153*, 168
print.nmfkc.ecv *(*nmfkc.cv.methods*)*, 88
print.nmfkc.gram, 168
print.nmfre.ecv, 169

print.summary.nmf.ffb, 169, *177*
print.summary.nmf.gmm, 170
print.summary.nmfkc, 171
print.summary.nmfkc.inference, 171
print.summary.nmfkc.net, 172
print.summary.nmfkc.net.inference, 173
print.summary.nmfkc.net.signed, 174
print.summary.nmfkc.signed, 174
print.summary.nmfre *(*summary.nmfre*)*, 182

residuals.nmf, *7*, *116*, 175, *176*
residuals.nmf.gmm, 176
residuals.nmfae *(*residuals.nmf*)*, 175
residuals.nmfre *(*residuals.nmf*)*, 175

summary.nmf.ffb, *16*, *170*, 177
summary.nmf.gmm, 178
summary.nmfae.inference, *46*
summary.nmfae.signed, *53*
summary.nmfkc, *64, 65*, *148*, *160*, 178, *180*
summary.nmfkc.inference, *96*, *172*, 179
summary.nmfkc.net, *172*, 180
summary.nmfkc.net.inference, *111*, *173*, 180
summary.nmfkc.net.signed, *174*, 181
summary.nmfkc.signed, 182
summary.nmfre, *138*, *142*, 182

warning, *82*