Package 'normalblockr'

September 11, 2026

Type Package

Title Gaussian Graphical Models with Latent Clustering Structure

Version 0.3.0

Description Implements the Normal-Block model, a Gaussian graphical model with a latent clustering structure for the multivariate analysis of continuous data. The model clusters variables and, building on the graphical lasso, infers a network of statistical dependencies between clusters rather than between individual variables, for known or unknown clusterings, with an optional zero-inflation extension for data with an excess of exact zeros. A complementary family clusters variables by their regression response to covariates rather than by their covariance, sharing one profile per cluster. See Tous & Chiquet (2026) <doi:10.1016/j.csda.2026.108347> for the model itself and its variational expectation-maximization estimation procedure.

License GPL (>= 3)

URL https://github.com/jchiquet/normalblockr, https://jchiquet.github.io/normalblockr/

BugReports https://github.com/jchiquet/normalblockr/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports Matrix, dplyr, tidyr, tibble, purrr, R6, ggplot2, igraph, sbm, corrplot, scales, MASS, stats, Rcpp

LinkingTo Rcpp, RcppArmadillo

Suggests testthat (>= 3.0.0), aricode, covr, Metrics, rmarkdown, pheatmap, paletteer, ggthemes, knitr

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Jeanne Tous [aut],
Nestor Ngalala Manguitini [ctb],
Julien Chiquet [aut, cre] (ORCID:
<https://orcid.org/0000-0002-3629-3429>)

Maintainer Julien Chiquet <julien.chiquet@inrae.fr>

Repository CRAN

Date/Publication 2026-09-11 18:10:02 UTC

Contents

`BIC.NormalBlockBase` *Bayesian Information Criterion for a Normal-Block Model*

Description

Extracts the (variational) BIC of a fitted normal-block model, computed as ‘deviance + log(n) * nb_param‘ (lower is better).

Usage

```
## S3 method for class 'NormalBlockBase'
BIC(object, ...)
```

Arguments

`object`	An object of class NormalBlockBase.
`...`	not used, only here for S3 compatibility

Value

A scalar: the (variational) BIC.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
model <- normal_block(data, blocks = 3, control = NB_control(verbose = FALSE))
BIC(model)
```

BIC.NormalBlockCollection *Bayesian Information Criterion for a Collection of Normal-Block Models*

Description

Returns the (variational) BIC of every model in a collection of normal-block models.

Usage

```
## S3 method for class 'NormalBlockCollection'
BIC(object, ...)
```

Arguments

object	An object inheriting from NormalBlockCollection.
...	not used, only here for S3 compatibility

Value

A numeric vector of BIC values, one per model in the collection.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
models <- normal_block(data, blocks = 2:5, control = NB_control(verbose = FALSE))
BIC(models)
```

brca_rppa *Breast cancer proteomics data (TCGA, RPPA)*

Description

Reverse-phase protein array (RPPA) measurements of 163 proteins across 346 breast cancer tumor samples from The Cancer Genome Atlas (TCGA), along with the PAM50 molecular subtype of each sample. Used as the running example in Tous & Chiquet (2026).

Usage

```
brca_rppa
```

Format

A list with 3 elements:

expr a 346 x 163 numeric matrix of protein expression levels (normalized log-ratios), samples in rows (named by TCGA sample identifier), proteins in columns.

covariates a data frame with one row per sample, in the same order as the rows of ‘expr‘: ‘sampleId‘ (matches ‘rownames(expr)‘) and 11 clinical variables, factors unless noted otherwise – ‘CN_CLUSTER‘ (copy-number cluster, 5 levels), ‘CONVERTED_STAGE‘ (tumor stage), ‘ER_STATUS‘ (estrogen receptor status), ‘FRACTION_GENOME_ALTERED‘ (numeric, in \[0, 1\]), ‘HER2_STATUS‘, ‘METASTASIS_CODED‘, ‘METHYLATION_CLUSTER‘ (5 levels), ‘MUTATION_COUNT‘ (numeric), ‘PAM50_SUBTYPE‘ (5 levels: Basal-like, HER2-enriched, Luminal A, Luminal B, Normal-like), ‘RPPA_CLUSTER‘ (7 levels) and ‘AGE‘ (numeric, in years). A few of these have missing values for a handful of samples (‘FRACTION_GENOME_ALTERED‘, ‘HER2_STATUS‘, ‘METASTASIS_CODED‘).

gene_annotation a data frame with one row per protein, in the same order as the columns of ‘expr‘: ‘protein‘ (matches ‘colnames(expr)‘), ‘entrezGeneId‘, ‘hugoGeneSymbol‘, ‘type‘ (‘"protein-coding"‘ or ‘"phosphoprotein"‘) and ‘go_bp_term‘ (a Gene Ontology Biological Process classification, one term per protein, or ‘"unknown"‘ if none could be found – see Details).

Details

‘go_bp_term‘ is derived from ‘org.Hs.eg.db‘, queried by ‘entrezGeneId‘ for ‘"protein-coding"‘ rows. Phospho-site antibodies (‘type == "phosphoprotein"‘, e.g. ‘"EGFR_PY1068"‘) carry a placeholder, negative ‘entrezGeneId‘ (there is no separate gene for a specific phosphorylation site, only the underlying gene) and are instead queried by gene symbol (the part of ‘protein‘ before the first ‘"_"‘). Each protein typically has dozens of GO Biological Process terms; ‘go_bp_term‘ keeps, for each protein, the one term shared by the most *other* proteins in this dataset – giving a handful of non-trivial groups rather than one near-singleton group per protein, more useful for comparing against an inferred clustering. See ‘data-raw/brca_rppa_go_annotation.R‘ for the full extraction code.

Source

TCGA breast cancer cohort (Cancer Genome Atlas Network, 2012, doi:10.1038/nature11412), downloaded via cBioPortal (‘brca_tcga_pub‘ study, https://www.cbioportal.org). ‘go_bp_term‘ (see Details) was added afterwards from ‘org.Hs.eg.db‘.

Examples

```
expr <- brca_rppa$expr
X <- model.matrix(~ 0 + PAM50_SUBTYPE, data = brca_rppa$covariates)
nb_data <- NormalBlockData$new(expr, X)
table(brca_rppa$gene_annotation$go_bp_term)
```

coef.NormalBlockBase *Extract Model Coefficients*

Description

Extract coefficients from a normal-block model.

Usage

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

Arguments

object	An object of class NormalBlockBase.
...	not used, only here for S3 compatibility

Value

A matrix of coefficients extracted from the model.

fitted.NormalBlockBase *Extract Fitted Values*

Description

Extract fitted values from a normal-block model.

Usage

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

Arguments

object	An object of class NormalBlockBase.
...	not used, only here for S3 compatibility

Value

A matrix of fitted values extracted from the object.

`generate_normal_block_mean_data` *Generate Normal Block Mean Data*

Description

A function to draw data from the normal block model (see details). The function returns both the generated data and the corresponding model parameters, in a list.

Usage

```
generate_normal_block_mean_data(
  n = 100,
  p = 40,
  d = 1,
  q = 3,
  kappa = 0,
  omega_structure = "erdos-renyi",
  u_v = c(0.3, 0.1),
  SNR = 5,
  alpha = rep(1/q, q),
  range_X = c(0, 10)
)
```

Arguments

`n`	number of individuals. Default to 100.
`p`	number of variables. Default to 40.
`d`	number of covariates. Default to 1.
`q`	number of groups. Default to 3.
`kappa`	vector (or scalar) of variable-wise probability of zero inflation. Default to 0.
`omega_structure`	the structure of the graph on which the precision matrix between variables is built. Can be a symmetric matrix with p rows/columns or a character picked in "erdos-renyi", "preferential_attachment", "community" in which case a graph is drawn with sensible generation parameters. See generate_precision_matrix for details.
`u_v`	two-size vector of positive numbers v and u controlling the generation of the precision matrix Omega: v scales the off-diagonal elements of the precision matrix (magnitude of partial correlations), and u is a positive number added to the diagonal elements to ensure positive-definiteness. The default value is c(0.3, 0.1).
`SNR`	Signal to noise ratio: magnitude of the regression parameters B will be adjusted so that tr(var(XB)) and tr(Sigma) match the desired SNR.
`alpha`	the q-size vector of group proportion. Default to rep(1/q, q)

range_X A 2-size vector defining the range of the uniform distribution used to draw values in X, the regressor matrix. Default is c(0, 10)

Value

A named list with the following element - Y a matrix of responses - X a regressor/design matrix - a list of model parameters, encompassing - B: matrix of regression coefficients - C: matrix of group membership - Omega: precision matrix of the variables - Sigma: covariance matrix of the variables - kappa: vector of ZI inflation probabilities (one per variable)

---

generate_normal_block_var_data

Generate Normal Block Var Data

Description

A function to draw data from the normal block model (see details). The function returns both the generated data and the corresponding model parameters, in a list.

Usage

```
generate_normal_block_var_data(
  n = 100,
  p = 40,
  d = 1,
  q = 3,
  kappa = 0,
  omega_structure = "erdos-renyi",
  u_v = c(0.3, 0.1),
  SNR = 0.75,
  alpha = rep(1/q, q),
  range_X = c(0, 10),
  range_D = c(0.5, 1.5)
)
```

Arguments

n	number of individuals. Default to 100.
p	number of variables. Default to 40.
d	number of covariates. Default to 1.
q	number of groups. Default to 3.
kappa	vector (or scalar) of variable-wise probability of zero inflation. Default to 0.
omega_structure	the structure of the graph on which the precision matrix between groups is built. Can be a symmetric matrix with q rows/columns or a character picked in "erdos-renyi", "preferential_attachment", "community" in which case a graph is drawn with sensible generation parameters. See generate_precision_matrix for details.

u_v	two-size vector of positive numbers v and u controlling the generation of the precision matrix Omega: v scales the off-diagonal elements of the precision matrix (magnitude of partial correlations), and u is a positive number added to the diagonal elements to ensure positive-definiteness. The default value is c(0.3, 0.1).
SNR	Signal to noise ratio: magnitude of the regression parameters B will be adjusted so that tr(var(XB)) and tr(Sigma) match the desired SNR.
alpha	the q-size vector of group proportion. Default to rep(1/q, q)
range_X	A 2-size vector defining the range of the uniform distribution used to draw values in X, the regressor matrix. Default is c(0, 10)
range_D	A 2-size vector defining the range of the uniform distribution used to draw values in D, the diagonal matrix of variances of variables. Default is c(0.5, 1.5)

Value

A named list with the following element - Y a matrix of responses - X a regressor/design matrix - a list of model parameters, encompassing - B: matrix of regression coefficients - C: matrix of group membership - D: diagonal matrix of variance of the variables - Omega: precision matrix of the groups - Sigma: covariance matrix of the groups - kappa: vector of ZI inflation probabilities (one per variable)

---

get_model *Create a Normal-Block Model Object*

Description

Creates the appropriate normal-block model (or collection of models) depending on the parametrization.

Usage

```
get_model(
  data,
  blocks,
  sparsity = 0,
  zero_inflation = FALSE,
  control = NB_control(),
  model = c("var", "mean")
)
```

Arguments

data	contains the matrix of responses (Y) and the design matrix (X).
blocks	either an integer (number of blocks), a vector of integer (list of possible number of block) or a p * q matrix (for indicating block membership when its known)

sparsity	boolean to say whether the model should have a changing penalty OR float to run model with a single penalty value
zero_inflation	boolean to indicate if Y is zero-inflated and adjust fitted model as a consequence
control	a list-like structure for detailed control on parameters should be generated with NB_control() for collections of sparse models
model	which model family to fit, "var" (the default) or "mean" – see [normal_block()]

`isNB` *Check if an Object is a Normal-Block Model*

Description

Checks if a model is of class [NormalBlockBase()], i.e. any normal-block model (variance-block or mean-block family).

Usage

```
isNB(object)
```

Arguments

object	An R object.

Value

A boolean telling whether object inherits from the NormalBlockBase class.

`logLik.NormalBlockBase` *Extract Log-Likelihood of a Normal-Block Model*

Description

Returns the (variational) log-likelihood of a fitted normal-block model as a ‘"logLik"‘ object, compatible with [stats::AIC()] and [stats::BIC()].

Usage

```
## S3 method for class 'NormalBlockBase'
logLik(object, ...)
```

Arguments

object	An object of class NormalBlockBase.
...	not used, only here for S3 compatibility

Value

An object of class '"logLik"'. The numeric value is the log-likelihood or its variational lower bound (ELBO). Attributes 'df' and 'nobs' hold the number of parameters and observations.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
model <- normal_block(data, blocks = 3, control = NB_control(verbose = FALSE))
logLik(model)
```

---

logLik.NormalBlockCollection

Extract Log-Likelihood of a Collection of Normal-Block Models

---

Description

Returns the log-likelihood of every model in a collection of normal-block models (see [NormalBlockVarCollectionClusters], [NormalBlockVarCollectionSparsity], [NormalBlockVarCollectionClustersSparsity]).

Usage

```
## S3 method for class 'NormalBlockCollection'
logLik(object, ...)
```

Arguments

`object`	An object inheriting from NormalBlockCollection.
`...`	not used, only here for S3 compatibility

Value

A numeric vector of log-likelihood values, one per model in the collection.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
models <- normal_block(data, blocks = 2:5, control = NB_control(verbose = FALSE))
logLik(models)
```

NB_control *NB_control*

Description

Control the model settings and various optimization parameters

Usage

```
NB_control(
  niter = 500,
  threshold = 1e-04,
  fixed_point_niter = 5,
  sparsity_weights = NULL,
  sparsity_penalties = NULL,
  n_sparsity_penalties = 30,
  min_ratio = 0.01,
  fixed_tau = FALSE,
  clustering_init = NULL,
  verbose = TRUE,
  heuristic = FALSE,
  noise_covariance = NULL,
  refine = FALSE
)
```

Arguments

niter	number of iterations in model optimization
threshold	loglikelihood / elbo threshold under which optimization stops
fixed_point_niter	number of sweeps of the tau update for Normal-Block-Mean with unknown clusters. Each sweep visits the rows of tau sequentially and maximizes the ELBO exactly in each, so it can never decrease the ELBO.
sparsity_weights	weights with which the penalty should be applied in case sparsity is required, non-0 values on the diagonal mean diagonal shall be penalized too (default is non-penalized diagonal and 1s off-diagonal)
sparsity_penalties	list of penalties the user wants to test, other parameters are only used if penalties is not specified
n_sparsity_penalties	number of penalties to test.
min_ratio	ratio for sparsity between max penalty (0 edge penalty) and min penalty to test
fixed_tau	whether tau should be fixed at clustering_init during optimization useful for calls to fixed_q models in stability_selection

`clustering_init`
: how to obtain the initial clustering of the q unknown blocks: a heuristic name ("ward2", "kmeans", "sbm" or "spectral"), an actual clustering (a vector of labels or a p x q indicator matrix, or a list of either per q for a collection), or "best_of_inits" to try several heuristics per model and keep the best-ELBO fit (see [NormalBlockVarBase]'s 'best_of_inits()'; not supported with 'sparsity = TRUE'). Default 'NULL', resolved per model family at fit time: "ward2" for variance-block models, "kmeans" for mean-block models ("ward2" was benchmarked substantially worse there – see [NormalBlockMeanBase]). See 'inst/methods_initialization_and_r for the heuristics' rationale, why no single one dominates, and how this interacts with 'refine' (below).

`verbose`
: telling if information should be printed during optimization

`heuristic`
: whether to use the heuristic approach (moment-based, no (V)EM recursion) instead of the full (V)EM. Default is FALSE. In heuristic mode, no likelihood/ELBO is computed, so 'entropy', 'loglik', 'BIC', 'ICL' and 'EBIC' are all 'NA' on the resulting model.

`noise_covariance`
: shape of the residual covariance. Variance-block models accept "diagonal" (variable-specific) or "spherical" (common); mean-block models, whose Sigma is the full p x p residual covariance, also accept "full". Default 'NULL', resolved per model family at fit time to "diagonal" – except for a mean-block model with 'sparsity > 0', which implies "full" since a penalty on a diagonal precision matrix would have nothing to act on (explicitly asking for both is an error). The mean-block "diagonal"/"spherical" variants need no matrix inversion, hence no 'n > p' requirement, and they select the number of clusters markedly better than a full Sigma once p approaches n: the p(p+1)/2 covariance parameters otherwise drown the mean structure BIC/ICL are weighing. Use "full" when the residual associations are themselves of interest.

`refine`
: for [NormalBlockVarCollectionClusters] only: whether 'optimize()' should automatically call 'refine()' afterwards. Default 'FALSE' since it adds real cost; call 'collection$refine()' directly at any point afterwards for the same effect without setting this.

Value

A named list of parameters to pass to [normal_block()]'s 'control' argument.

`NormalBlockData` *Data Container for Normal-Block Models*

Description

R6 class holding the responses and design matrix used to fit a normal-block model.

Public fields

`Y` the matrix of responses (rescaled column-wise if 'scale = TRUE')

`Y_scale` the per-column standard deviation Y was divided by (all 1's if 'scale = FALSE')

`X` the matrix of covariates

`X0` the matrix of zero-inflation covariates, if applicable

`formula` describes the relationship between Y and X, and X0 if applicable, useful if not all of X's or X0's covariates should be used, should be formatted ~ X1 + X2... | Z1 + Z2... with the Normal formula before the | and the ZI formula after the |

`n` sample size

`d` number of covariates

`d0` number of zero-inflation covariates, if applicable

`p` number of variables

`XtX` useful for inference in some cases

`XtXm1` inverse of XtX, useful for inference

`XtY` useful for inference

`npY` total number of non zeros in Y

`nY` total number of non zeros for each column/variable in Y

`zeros` where are the zero in Y

`zeros_bar` where are the non-zeros in Y

Methods

Public methods:

- `NormalBlockData$new()`
- `NormalBlockData$ols_fit()`
- `NormalBlockData$zi_ols_fit()`
- `NormalBlockData$zi_fit()`
- `NormalBlockData$clone()`

`NormalBlockData$new()`: Create a new ['NormalBlockData'] object.

Usage:

```
NormalBlockData$new(
  Y,
  X,
  X0 = NULL,
  formula = NULL,
  scale = TRUE,
  zeros = NULL
)
```

Arguments:

`Y` the matrix of responses (called Y in the model).

`X` design matrix (called X in the model).

`X0` zero-inflation design matrix, if applicable.

`formula` describes the relationship between Y and X, useful if not all of X's covariates should be used.

`scale` whether to rescale each column of Y by its own standard deviation (no centering). Default TRUE, see the class-level documentation for the rationale and its limits.

`zeros` an optional 0/1 matrix of structural zeros, overriding the default 'Y == 0'.

`NormalBlockData$ols_fit()`**:** Ordinary-least-squares fit of Y on X, with its residuals and their covariance. Computed once and memoized.

Usage:

```
NormalBlockData$ols_fit()
```

Returns: a list with 'B' (d x p), 'R' (n x p residuals) and 'Sigma' (p x p residual covariance)

`NormalBlockData$zi_ols_fit()`**:** Masked counterpart of 'ols_fit()': a per-variable weighted least-squares fit of B under the zero-inflation mask, with its inverse residual variances and residuals (see 'zi_weighted_fit()'). Computed once and memoized.

Usage:

```
NormalBlockData$zi_ols_fit()
```

Returns: a list with 'B' (d x p), 'dm1' (p) and 'R' (n x p masked residuals)

`NormalBlockData$zi_fit()`**:** Zero-inflation component: 'p' independent logistic regressions of each variable's zero pattern on 'X0', and the fixed contribution they make to the log-likelihood. The (V)EM never revisits these, so they are a property of the data rather than of a model.

Usage:

```
NormalBlockData$zi_fit()
```

Returns: a list with 'B0' (d0 x p), 'kappa' (n x p zero-inflation probabilities) and 'ZI_cond_mean' (a scalar)

`NormalBlockData$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NormalBlockData$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
c(n = data$n, p = data$p, d = data$d)
```

`NormalBlockMeanCollectionClusters` *Collection of Mean-Block Models over a Range of Cluster Counts*

Description

R6 class for a collection of mean-block models ([NormalBlockMeanBase]) with different numbers of clusters (q). Inherits its scaffolding (‘print()‘/‘summary()‘/‘plot()‘/‘optimize()‘, the ‘criteria‘ table) from [NormalBlockCollection]. Unlike [NormalBlockVarCollectionClusters], there is no SBM-path shortcut here: the shared clustering-heuristic registry’s cov()/correlation-based methods are ill-suited to the mean-block family (see [NormalBlockMeanBase]’s own default), so each q is fit independently.

Super classes

`NormalBlockCollection` -> `NormalBlockCollectionClusters` -> `NormalBlockMeanCollectionClusters`

Active bindings

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- `NormalBlockMeanCollectionClusters$new()`
- `NormalBlockMeanCollectionClusters$clone()`

`NormalBlockMeanCollectionClusters$new()`: Create a new [‘NormalBlockMeanCollectionClusters‘] object.

Usage:

```
NormalBlockMeanCollectionClusters$new(
  mydata,
  q_list,
  zero_inflation = FALSE,
  sparsity = 0,
  control = NB_control()
)
```

Arguments:

`mydata` object of NormalBlockData class, with responses and design matrix

`q_list` list of q values (number of groups) in the collection

`zero_inflation` whether Y carries structural zeros; every model in the collection is then zero-inflated (Sigma diagonal or spherical only, see [NormalBlockMeanBase])

`sparsity` sparsity penalty on the network density

`control` structured list of more specific parameters, to generate with NB_control

Returns: A new [‘NormalBlockMeanCollectionClusters‘] object

`NormalBlockMeanCollectionClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockMeanCollectionClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_mean_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
models <- normal_block(data, blocks = 2:5, model = "mean")
models$plot(c("BIC", "ICL"))
models$get_best_model()
```

`NormalBlockMeanCollectionClustersSparsity` *Collection of Mean-Block Models over Cluster Counts and Sparsity Levels*

Description

R6 class for a collection of mean-block models ([NormalBlockMeanBase]) over both a range of cluster counts (q) and a sparsity path, i.e. one [NormalBlockMeanCollectionSparsity] per q. Mirrors [NormalBlockVarCollectionClustersSparsity], minus its SBM-path shortcut for the initial clustering (ill-suited to this family, see [NormalBlockMeanCollectionClusters]).

Super classes

`NormalBlockCollection` -> `NormalBlockCollectionClustersSparsity` -> `NormalBlockMeanCollectionClustersSpa`

Active bindings

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- `NormalBlockMeanCollectionClustersSparsity$new()`
- `NormalBlockMeanCollectionClustersSparsity$clone()`

`NormalBlockMeanCollectionClustersSparsity$new()`: Create a new [‘NormalBlockMeanCollectionClustersSparsity‘] object.

Usage:

```
NormalBlockMeanCollectionClustersSparsity$new(
  mydata,
  q_list,
  control = NB_control()
)
```

Arguments:

`mydata` object of NormalBlockData class, with responses and design matrix

`q_list` list of q values (number of groups) in the collection

`control` structured list of parameters to handle sparsity control

Returns: A new [‘NormalBlockMeanCollectionClustersSparsity‘] object

`NormalBlockMeanCollectionClustersSparsity$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NormalBlockMeanCollectionClustersSparsity$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_mean_data(n = 60, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
models <- normal_block(data, blocks = 2:4, sparsity = TRUE, model = "mean",
                       control = NB_control(n_sparsity_penalties = 4))
models$plot("BIC")
models$get_best_model("BIC")
```

NormalBlockMeanCollectionSparsity

Collection of Mean-Block Models over a Sparsity Path

Description

R6 class for a collection of mean-block models ([NormalBlockMeanBase]) with a fixed clustering (or a fixed number of blocks) and different sparsity levels applied to the p x p precision matrix of the variables. Mirrors [NormalBlockVarCollectionSparsity], minus the StARS/stability selection path, which relies on ‘fixed_tau‘, not supported by the mean-block VEM.

Super classes

`NormalBlockCollection` -> `NormalBlockCollectionSparsity` -> `NormalBlockMeanCollectionSparsity`

Active bindings

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- NormalBlockMeanCollectionSparsity$new()
- NormalBlockMeanCollectionSparsity$get_best_model()
- NormalBlockMeanCollectionSparsity$clone()

`NormalBlockMeanCollectionSparsity$new()`: Create a new ['NormalBlockMeanCollectionSparsity'] object.

Usage:

```
NormalBlockMeanCollectionSparsity$new(mydata, blocks, control = NB_control())
```

Arguments:

`mydata` object of NormalBlockData class, with responses and design matrix

`blocks` either a clustering matrix (known, fixed clustering) or a single integer (number of blocks to infer)

`control` structured list of parameters to handle sparsity control

Returns: A new ['NormalBlockMeanCollectionSparsity'] object

`NormalBlockMeanCollectionSparsity$get_best_model()`: Extract best model in the collection

Usage:

```
NormalBlockMeanCollectionSparsity$get_best_model(
  crit = c("BIC", "EBIC", "ICL")
)
```

Arguments:

`crit` a character for the criterion used to perform the selection, either "BIC", "EBIC" or "ICL". Default is BIC

Returns: a ['NormalBlockMeanBase'] object

`NormalBlockMeanCollectionSparsity$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockMeanCollectionSparsity$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_mean_data(n = 60, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
models <- normal_block(data, blocks = 3, sparsity = TRUE, model = "mean",
                       control = NB_control(n_sparsity_penalties = 5))
models$plot(c("BIC", "EBIC"))
```

NormalBlockMeanKnownClusters

Mean-Block Model with Known Clustering

Description

R6 class for a Normal-Block-Mean model with a known clustering.

Super classes

NormalBlockBase -> NormalBlockMeanBase -> NormalBlockMeanKnownClusters

Active bindings

`fitted` Y values predicted by the model, in Y's original units

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- NormalBlockMeanKnownClusters$new()
- NormalBlockMeanKnownClusters$clone()

`NormalBlockMeanKnownClusters$new()`: Create a new ['NormalBlockMeanKnownClusters'] object.

Usage:

```
NormalBlockMeanKnownClusters$new(data, C, sparsity = 0, control = NB_control())
```

Arguments:

`data` object of NormalBlockData class, with responses and design matrix

`C` clustering matrix C_jk = 1 if species j belongs to cluster k

`sparsity` to apply on variance matrix when calling GLASSO

`control` structured list of more specific parameters, to generate with NB_control

Returns: A new ['NormalBlockMeanKnownClusters'] object

`NormalBlockMeanKnownClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockMeanKnownClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_mean_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
model <- normal_block(data, blocks = ex$parameters$C, model = "mean")
model$plot()
```

NormalBlockMeanUnknownClusters
Mean-Block Model with Unknown Clustering

Description

R6 class for a Normal-Block-Mean model with a fixed number of clusters (but unknown clustering), inferred by variational EM.

Super classes

NormalBlockBase -> NormalBlockMeanBase -> NormalBlockMeanUnknownClusters

Active bindings

`fitted` Y values predicted by the model, in Y's original units

`var_par` a list with the variational parameter: tau (posterior group probabilities)

`nb_param` number of parameters in the model

`entropy` Entropy of the conditional distribution The only latent variable is the clustering, so the entropy of the variational distribution reduces to -sum(tau * log(tau)).

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- NormalBlockMeanUnknownClusters$new()
- NormalBlockMeanUnknownClusters$clone()

`NormalBlockMeanUnknownClusters$new()`: Create a new ['NormalBlockMeanUnknownClusters'] object.

Usage:

```
NormalBlockMeanUnknownClusters$new(
  data,
  q,
  sparsity = 0,
  control = NB_control()
)
```

Arguments:

`data` object of NormalBlockData class, with responses and design matrix

`q` number of clusters

`sparsity` to apply on variance matrix when calling GLASSO

`control` structured list of more specific parameters, to generate with NB_control

Returns: A new ['NormalBlockMeanUnknownClusters'] object

`NormalBlockMeanUnknownClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockMeanUnknownClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_mean_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
model <- normal_block(data, blocks = 3, model = "mean")
model$clustering
```

NormalBlockVarCollectionClusters

Collection of Normal-Block Models over a Range of Cluster Counts

---

Description

R6 class for a collection of normal-block models with different number of clusters (q) and a fixed sparsity level.

Super classes

`NormalBlockCollection` -> `NormalBlockCollectionClusters` -> `NormalBlockVarCollectionClusters`

Active bindings

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- `NormalBlockVarCollectionClusters$new()`
- `NormalBlockVarCollectionClusters$clone()`

`NormalBlockVarCollectionClusters$new()`: Create a new ['NormalBlockVarCollectionClusters'] object.

Usage:

```
NormalBlockVarCollectionClusters$new(
  mydata,
  q_list,
  zero_inflation = FALSE,
  sparsity = 0,
  control = NB_control()
)
```

Arguments:

`mydata` object of NormalBlockData class, with responses and design matrix

`q_list` list of q values (number of groups) in the collection

`zero_inflation` whether the models in the collection should be zero-inflated or not

`sparsity` sparsity penalty on the network density

`control` structured list of more specific parameters, to generate with NB_control

Returns: A new ['NormalBlockVarCollectionClusters'] object

`NormalBlockVarCollectionClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockVarCollectionClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
models <- normal_block(data, blocks = 2:4, control = NB_control(verbose = FALSE))
models$get_best_model("ICL")$q
```

`NormalBlockVarCollectionClustersSparsity` *Collection of Normal-Block Models over Cluster Counts and Sparsity Levels*

Description

R6 class for a collection of normal-block models with different number of clusters (q) and different sparsity levels.

Super classes

`NormalBlockCollection` -> `NormalBlockCollectionClustersSparsity` -> `NormalBlockVarCollectionClustersSpar`

Active bindings

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- NormalBlockVarCollectionClustersSparsity$new()
- NormalBlockVarCollectionClustersSparsity$clone()

`NormalBlockVarCollectionClustersSparsity$new()`: Create a new ['NormalBlockVarCollectionClustersSparsity'] object.

Usage:

```
NormalBlockVarCollectionClustersSparsity$new(
  mydata,
  q_list,
  zero_inflation = FALSE,
  control = NB_control()
)
```

Arguments:

`mydata` object of NormalBlockData class, with responses and design matrix

`q_list` list of q values (number of groups) in the collection

`zero_inflation` boolean to specify whether data is zero-inflated

`control` structured list of parameters to handle sparsity control

Returns: A new ['NormalBlockVarCollectionClustersSparsity'] object

`NormalBlockVarCollectionClustersSparsity$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockVarCollectionClustersSparsity$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
models <- normal_block(data, blocks = 2:3, sparsity = TRUE,
                       control = NB_control(verbose = FALSE, n_sparsity_penalties = 3))
models$get_best_model("BIC")$q
```

NormalBlockVarCollectionSparsity

Collection of Normal-Block Models over a Sparsity Path

Description

R6 class for a collection of normal-block models with a fixed clustering (blocks) and different sparsity levels.

Super classes

NormalBlockCollection -> NormalBlockCollectionSparsity -> NormalBlockVarCollectionSparsity

Active bindings

`sparsity_details` list of information about model's penalties

`criteria` a data frame with the values of some criteria ((approximated) log-likelihood, BIC) for the collection of models

`stability_path` measure of edges stability based on StARS method

`stability` mean edge stability along the sparsity penalties path

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- `NormalBlockVarCollectionSparsity$new()`
- `NormalBlockVarCollectionSparsity$get_best_model()`
- `NormalBlockVarCollectionSparsity$stability_selection()`
- `NormalBlockVarCollectionSparsity$clone()`

`NormalBlockVarCollectionSparsity$new()`: Create a new ['NormalBlockVarCollectionSparsity'] object.

Usage:

```
NormalBlockVarCollectionSparsity$new(
  mydata,
  blocks,
  zero_inflation = FALSE,
  control = NB_control()
)
```

Arguments:

`mydata` object of NormalBlockData class, with responses and design matrix

`blocks` either a clustering matrix (known, fixed clustering) or a single integer (number of blocks to infer)

`zero_inflation` boolean to specify whether data is zero-inflated

`control` structured list of parameters to handle sparsity control

Returns: A new ['NormalBlockVarCollectionSparsity'] object

`NormalBlockVarCollectionSparsity$get_best_model()`: Extract best model in the collection

Usage:

```
NormalBlockVarCollectionSparsity$get_best_model(
  crit = c("BIC", "EBIC", "ICL", "StARS"),
  stability = 0.9
)
```

Arguments:

`crit` a character for the criterion used to performed the selection.

`stability` if criterion = "StARS" gives level of stability required. Either "BIC", "EBIC", "ICL" or "StARS". Default is BIC

Returns: a [‘NormalBlockVarUnknownClusters‘] object

`NormalBlockVarCollectionSparsity$stability_selection()`: Compute the stability path by stability selection

Usage:

```
NormalBlockVarCollectionSparsity$stability_selection(
  subsamples = NULL,
  n_subsamples = 10
)
```

Arguments:

`subsamples` a list of vectors describing the subsamples. The number of vectors (or list length) determines the number of subsamples used in the stability selection. Automatically set to 20 subsamples with size ‘10*sqrt(n)‘ if ‘n >= 144‘ and ‘0.8*n‘ otherwise following Liu et al. (2010) recommendations.

`n_subsamples` number of subsamples to create if the subsamples are not given

`NormalBlockVarCollectionSparsity$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockVarCollectionSparsity$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
models <- normal_block(data, blocks = ex$parameters$C, sparsity = TRUE,
                       control = NB_control(verbose = FALSE, n_sparsity_penalties = 5))
models$get_best_model("BIC")$sparsity
```

NormalBlockVarKnownClusters

Normal-Block Model with Known Clustering

Description

R6 class for a normal-block model with known clustering.

Super classes

NormalBlockBase -> NormalBlockVarBase -> NormalBlockVarKnownClusters

Active bindings

`posterior_par` a list with the parameters of posterior distribution W | Y

`entropy` Entropy of the conditional distribution

`fitted` Y values predicted by the model, in Y's original units

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- `NormalBlockVarKnownClusters$new()`
- `NormalBlockVarKnownClusters$clone()`

`NormalBlockVarKnownClusters$new()`: Create a new ['NormalBlockVarKnownClusters'] object.

Usage:

```
NormalBlockVarKnownClusters$new(data, C, sparsity = 0, control = NB_control())
```

Arguments:

`data` object of NormalBlockVarData class, with responses and design matrix

`C` clustering matrix C_jk = 1 if species j belongs to cluster k

`sparsity` to apply on variance matrix when calling GLASSO

`control` structured list of more specific parameters, to generate with NB_control

Returns: A new ['NormalBlockVarKnownClusters'] object

`NormalBlockVarKnownClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockVarKnownClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
model <- normal_block(data, blocks = ex$parameters$C, control = NB_control(verbose = FALSE))
model$clustering
```

NormalBlockVarUnknownClusters

Normal-Block Model with Unknown Clustering

Description

R6 class for a normal-block model with a fixed number of clusters (but unknown clustering).

Super classes

NormalBlockBase -> NormalBlockVarBase -> NormalBlockVarUnknownClusters

Public fields

`fixed_tau` whether tau should be fixed at clustering_init during optimization, useful for stability selection

Active bindings

`model_par` a list with the matrices of the model parameters: B (covariates), dm1 (species variance), Omega (groups precision matrix))

`nb_param` number of parameters in the model

`var_par` a list with the matrices of the variational parameters: M (means), S (variances), tau (posterior group probabilities)

`entropy` Entropy of the conditional distribution

`fitted` Y values predicted by the model, in Y's original units

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- NormalBlockVarUnknownClusters$new()
- NormalBlockVarUnknownClusters$clone()

`NormalBlockVarUnknownClusters$new()`: Create a new ['NormalBlockVarUnknownClusters'] object.

Usage:

```
NormalBlockVarUnknownClusters$new(
  data,
  q,
  sparsity = 0,
  control = NB_control()
)
```

Arguments:

`data` contains the matrix of responses (Y) and the design matrix (X).

`q` required number of groups

`sparsity` sparsity penalty to add on blocks precision matrix for sparsity

`control` structured list for specific parameters

Returns: A new [‘NormalBlockVarUnknownClusters‘] object

`NormalBlockVarUnknownClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
NormalBlockVarUnknownClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
model <- normal_block(data, blocks = 3, control = NB_control(verbose = FALSE))
model$clustering
```

`normal_block` *Normal-block model*

Description

Fit a normal-block model with a variational or heuristic algorithm

Usage

```
normal_block(
  data,
  blocks,
  sparsity = 0,
  zero_inflation = FALSE,
  control = NB_control(),
  model = c("var", "mean")
)
```

Arguments

`data`	NormalBlockData object, contains the matrix of responses (Y, n x p) and the design matrix (X, n x d), must be created with NormalBlockData$new.
`blocks`	either a integer (number of blocks), a vector of integer (list of possible number of block) or a p * q matrix (for indicating block membership when its known)
`sparsity`	either TRUE to run the optimization for different sparsity penalty values OR float to run model with a single sparsity penalty value

zero_inflation	boolean to indicate if Y is zero-inflated and adjust fitted model as a consequence
control	a list-like structure for detailed control on parameters should be generated with NB_control().
model	which model family to fit: "var" (the default) structures the clustering in the latent covariance (see [NormalBlockVarBase]), "mean" structures it in the mean (mu_i = C B' X_i, see [NormalBlockMeanBase]). The mean-block family covers the same collections as the variance-block one ([NormalBlockMeanCollectionClusters], [NormalBlockMeanCollectionSparsity] and their crossing, [NormalBlockMeanCollectionClustersSparsity]) – each q simply fit independently, without the variance-block family's SBM-path shortcut. Zero-inflation is supported ([ZINormalBlockMeanKnownClusters], [ZINormalBlockMeanUnknownClusters], and collections of those over a range of q), with a diagonal or spherical Sigma only, hence not along a sparsity path – see [NormalBlockMeanBase].

Value

an R6 object with one of the model classes (or a collection of model objects).

Examples

```
## Normal Data
ex_data <- generate_normal_block_var_data(n=100, p=30, d=1, q=3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
my_normal_block <- normal_block(data, blocks = 1:6)
my_normal_block$plot(c("deviance", "BIC", "ICL"))
Y_hat <- my_normal_block$get_best_model()$fitted
plot(data$Y, Y_hat, log = "xy"); abline(0,1)
## Normal Data with Zero Inflation
ex_data_zi <- generate_normal_block_var_data(n=50, p=50, d=1, q=3, kappa = rep(0.5,50))
zidata <- NormalBlockData$new(ex_data_zi$Y, ex_data_zi$X)
my_normal_block <- normal_block(zidata, blocks = 1:6, zero_inflation = TRUE)
## Mean-Block model (clustering in the mean rather than the covariance)
ex_mean <- generate_normal_block_mean_data(n=50, p=20, d=1, q=3)
mean_data <- NormalBlockData$new(ex_mean$Y, ex_mean$X)
my_mean_block <- normal_block(mean_data, blocks = 3, model = "mean")
```

---

normal_block_sequential

Cluster variables in the mean, then in the residual covariance

---

Description

Fits the two model families one after the other: a mean-block model ([NormalBlockMeanBase]) groups the variables by how they respond to the covariates, then a variance-block model ([NormalBlockVarBase]) groups the residuals of that fit by how they co-vary. The two answer different questions and generally return unrelated partitions, so running both is often more informative than choosing one.

Usage

```
normal_block_sequential(
  data,
  blocks_mean,
  blocks_var,
  crit = c("ICL", "BIC"),
  zero_inflation = FALSE,
  control_mean = NB_control(verbose = FALSE),
  control_var = NB_control(verbose = FALSE)
)
```

Arguments

data	a [NormalBlockData] object
blocks_mean	number of clusters for the mean-block stage: an integer, a vector of integers to explore, or a p x q indicator matrix
blocks_var	idem for the variance-block stage, run on the residuals
crit	criterion used to pick a model when a range is explored, "ICL" (the default) or "BIC"
zero_inflation	whether Y carries structural zeros. Both stages are then zero-inflated, sharing the same mask: the second one would otherwise re-derive it from residuals, which are never exactly zero.
control_mean	control list for the mean-block stage, see [NB_control()]
control_var	control list for the variance-block stage

Details

The second stage uses an intercept-only design on purpose: the covariate effects have already been removed by the first stage.

This is a heuristic two-stage estimator, not a joint model. On simulated data carrying two genuinely distinct structures it recovers both exactly (see 'inst/mean_block_analyses/sequential_mean_then_variance.R').

Value

an object of class 'normal_block_sequential', a list with the fitted 'mean' and 'var' models and the residual matrix 'residuals' handed from one stage to the other.

Examples

```
ex   <- generate_normal_block_mean_data(n = 80, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex$Y, ex$X)
fit  <- normal_block_sequential(data, blocks_mean = 3, blocks_var = 2)
fit
```

onema *French stream fish community data (ONEMA / OFB electrofishing surveys)*

Description

Fish biomass per species, aggregated by sampling station, together with environmental covariates for each station. Derived from the fish community monitoring of stream sections across metropolitan France (1995-2018) by the French Office for Biodiversity (formerly ONEMA, "Office National de l'Eau et des Milieux Aquatiques"), using electrofishing.

Usage

```
onema
```

Format

A list with 2 elements:

biomass a 399 x 46 numeric matrix of total biomass (grams) per species (3-letter species code, columns) and station (row names).

covariates a data frame with one row per station, in the same order as the rows of 'biomass': 'station' (matches 'rownames(biomass)') and 14 environmental variables – 'slope', 'alt' (altitude), 'd_source' (distance to source), 'strahler' (Strahler stream order), 'width_river_mean'/'width_river_cv', 'avg_depth_station_mean'/ 'avg_depth_station_cv', 'DBO_med'/'DBO_cv' (biological oxygen demand), 'flow_med'/'flow_cv' and 'temperature_med'/'temperature_cv' ('_med' = median, '_cv' = coefficient of variation, over the monitoring period).

Details

The source data is organized around fishing *operations* ('opcod'): several electrofishing operations can be carried out at the same *station* over time. 'onema' aggregates this to one row per station: every operation is first mapped to its station ('fishing_protocol.csv'), then biomass (in grams) is summed over every operation recorded at a given station, separately for each species (i.e. 'biomass[s,]' is the total biomass of each species ever caught at station 's', not a single operation's catch). Stations without environmental data are dropped. Species observed at fewer than 10 stations are also dropped: at that level of rarity, a zero-inflated fit's iterative initialization can fit a species' handful of non-zero observations exactly, driving its estimated noise precision to infinity (these species also carry essentially no information for clustering anyway).

Source

Danet, A., Mouchet, M., Bonnaffe, W., Thebault, E., Fontaine, C. (2021) "Species richness and food-web structure jointly drive community biomass and its temporal stability in fish communities", data set, Zenodo, doi:10.5281/zenodo.5095656.

Examples

```
Y <- log(1 + onema$biomass)
X <- model.matrix(~ 1, data = onema$covariates)
nb_data <- NormalBlockData$new(Y, X)

out <- normal_block(nb_data, 2:15, control = NB_control(clustering_init = "ward2"))
```

`plot.NormalBlockBase` *Plot a Normal-Block Model*

Description

Plots the evolution of the objective (log-likelihood or ELBO) across the (V)EM iterations of the last call to 'optimize()', see '$plot_loglik()'.

Usage

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

Arguments

`x`	An object of class NormalBlockBase.
`...`	not used, only here for S3 compatibility

Value

Invisibly returns the [ggplot2::ggplot] object; called for its side effect of plotting.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
model <- normal_block(data, blocks = 3, control = NB_control(verbose = FALSE))
plot(model)
```

predict.NormalBlockBase *Predict Method for Variance-Block Models*

Description

Predicts observations Y for new covariates X. Specific to the variance-block family: the mean-block models have their own formula (mu = C B' X).

Usage

```
## S3 method for class 'NormalBlockBase'
predict(object, new_X, ...)
```

Arguments

object	An object of class NormalBlockVarBase.
new_X	New set of covariates.
...	not used, only here for S3 compatibility

Value

A n*p prediction matrix for new observations

print.NormalBlockBase *Print a Normal-Block Model*

Description

Print a short summary of a fitted normal-block model: model type, goodness-of-fit criteria, and the useful fields/methods to explore it further.

Usage

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

Arguments

x	An object of class NormalBlockBase.
...	not used, only here for S3 compatibility

Value

Invisibly returns 'x'.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
model <- normal_block(data, blocks = 3, control = NB_control(verbose = FALSE))
print(model)
```

---

print.NormalBlockCollection

Print a Collection of Normal-Block Models

Description

Print a short summary of a collection of normal-block models: model type and the range of q/sparsity explored. See [summary.NormalBlockCollection()] for the full criteria table.

Usage

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

Arguments

x	An object inheriting from NormalBlockCollection.
...	not used, only here for S3 compatibility

Value

Invisibly returns ‘x‘.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
models <- normal_block(data, blocks = 2:5, control = NB_control(verbose = FALSE))
print(models)
```

`print.normal_block_sequential` *Print a Sequential Mean-then-Variance Fit*

Description

Reports both stages and, when **aricode** is available, how related the two partitions are.

Usage

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

Arguments

`x`	an object of class ‘normal_block_sequential‘
`...`	not used, only here for S3 compatibility

Value

Invisibly returns ‘x‘.

`print.summary.NormalBlockBase` *Print a Normal-Block Model Summary*

Description

Print method for objects returned by [summary.NormalBlockBase()].

Usage

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

Arguments

`x`	An object of class ‘summary.NormalBlockBase‘.
`...`	not used, only here for S3 compatibility

Value

Invisibly returns ‘x‘.

print.summary.NormalBlockCollection *Print a Collection Summary*

Description

Print method for objects returned by [summary.NormalBlockCollection()].

Usage

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

Arguments

x	An object of class ‘summary.NormalBlockCollection‘.
...	not used, only here for S3 compatibility

Value

Invisibly returns ‘x‘.

sigma.NormalBlockBase *Extract the Covariance Matrix*

Description

Extract the covariance matrix ‘Omega^-1‘: between latent blocks (q x q) for the variance-block models, between variables (p x p) for the mean-block models.

Usage

```
## S3 method for class 'NormalBlockBase'
sigma(object, ...)
```

Arguments

object	An object of class NormalBlockBase.
...	not used, only here for S3 compatibility

Value

The covariance matrix, of size q x q or p x p depending on the model.

summary.NormalBlockBase

Summarize a Normal-Block Model

Description

Summarizes a fitted normal-block model: model type, goodness-of-fit criteria, cluster sizes, and the density of the inferred network between blocks.

Usage

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

Arguments

object	An object of class NormalBlockBase.
...	not used, only here for S3 compatibility

Value

An object of class ‘summary.NormalBlockBase‘ (a list with the model’s ‘who_am_I‘, ‘criteria‘, ‘cluster_sizes‘ and network ‘density‘), printed with a dedicated [print.summary.NormalBlockBase()] method.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
model <- normal_block(data, blocks = 3, control = NB_control(verbose = FALSE))
summary(model)
```

summary.NormalBlockCollection

Summarize a Collection of Normal-Block Models

Description

Summarizes a collection of normal-block models: model type, the full criteria table, and the range of q/sparsity explored.

Usage

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

Arguments

`object`	An object inheriting from NormalBlockCollection.
`...`	not used, only here for S3 compatibility

Value

An object of class ‘summary.NormalBlockCollection‘ (a list with the collection’s ‘who_am_I‘, ‘criteria‘, ‘q_range‘ and ‘sparsity_range‘), printed with a dedicated [print.summary.NormalBlockCollection()] method.

Examples

```
ex_data <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3)
data <- NormalBlockData$new(ex_data$Y, ex_data$X)
models <- normal_block(data, blocks = 2:5, control = NB_control(verbose = FALSE))
summary(models)
```

`university` *University webpages text data (CMU "4 Universities" / WebKB)*

Description

Term-frequency data derived from the student personal webpages of the CMU "World Wide Knowledge Base" (WebKB) project’s "4 Universities" dataset (Cornell, Texas, Washington, Wisconsin computer science departments, collected January 1997). Used as one of the running examples in Tous & Chiquet (2026).

Usage

```
university
```

Format

A list with 3 elements:

frequencies a 504 x 1867 numeric matrix of term frequencies (row sums to 1): for each document (row, named by its original file path) and term (column), the fraction of that document’s (post-preprocessing) word count made up of that term.

entropies a named numeric vector of length 1867 (one value per column of ‘frequencies‘, same names/order), the normalized Shannon entropy of each term’s distribution across documents.

terms a character vector of the 100 column names of ‘frequencies‘ with the highest ‘entropies‘, ordered by decreasing entropy.

Details

Each of the 504 pages (rows) is a document; each of the 1867 columns is a term retained after lower-casing, stripping URLs/HTML tags/control characters/punctuation/numbers, removing English stopwords and dropping terms occurring in fewer than 2 documents. ‘entropies‘ ranks every term by how evenly it is spread across documents (Shannon entropy of each term’s normalized document distribution, in \[0, 1\], 1 = perfectly uniform); ‘terms‘ is the 100 highest-entropy terms, i.e. the terms that are the most informative for distinguishing documents from one another rather than just reflecting a few documents’ idiosyncratic vocabulary – the transformation used by Tan et al. (2015).

Source

CMU Text Learning Group, "World Wide Knowledge Base (Web->KB) project", http://www.cs.cmu.edu/~webkb/; "4 Universities" subset, https://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/. Tan, P.-N., Steinbach, M., Kumar, V. (2015) "Introduction to Data Mining" (transform used to derive ‘entropies‘/‘terms‘).

Examples

```
Y <- log(1 + university$frequencies[, university$terms])
nb_data <- NormalBlockData$new(Y, X = matrix(1, nrow(Y), 1))

out <- normal_block(nb_data, 2:15)
```

ZINormalBlockMeanKnownClusters

Zero-Inflated Mean-Block Model with Known Clustering

Description

R6 class for a zero-inflated Normal-Block-Mean model with a known clustering. Sigma is diagonal or spherical here. See [NormalBlockMeanBase] for why a full one is out of reach under a mask.

Super classes

NormalBlockBase -> NormalBlockMeanBase -> ZINormalBlockMeanKnownClusters

Active bindings

`fitted` Y values predicted by the model, in Y’s original units

`model_par` a list with model parameters: B, Omega and kappa (zero-inflation probabilities)

`nb_param` number of parameters in the model

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- ZINormalBlockMeanKnownClusters$new()
- ZINormalBlockMeanKnownClusters$clone()

`ZINormalBlockMeanKnownClusters$new()`: Create a new ['ZINormalBlockMeanKnownClusters'] object.

Usage:

```
ZINormalBlockMeanKnownClusters$new(
  data,
  C,
  sparsity = 0,
  control = NB_control()
)
```

Arguments:

`data` object of NormalBlockData class, with responses and design matrix

`C` clustering matrix C_jk = 1 if species j belongs to cluster k

`sparsity` unused here, kept for signature symmetry (must be 0)

`control` structured list of more specific parameters, to generate with NB_control

Returns: A new ['ZINormalBlockMeanKnownClusters'] object

`ZINormalBlockMeanKnownClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ZINormalBlockMeanKnownClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_mean_data(n = 50, p = 20, d = 1, q = 3)
Y <- ex$Y; Y[runif(length(Y)) < 0.2] <- 0
data <- NormalBlockData$new(Y, ex$X)
model <- normal_block(data, blocks = ex$parameters$C, model = "mean",
                      zero_inflation = TRUE)
model$clustering
```

`ZINormalBlockMeanUnknownClusters` *Zero-Inflated Mean-Block Model with Unknown Clustering*

Description

R6 class for a zero-inflated Normal-Block-Mean model with a fixed number of clusters (but unknown clustering), inferred by variational EM. Sigma is diagonal or spherical here. See [NormalBlockMeanBase] for why a full one is out of reach under a mask.

Super classes

`NormalBlockBase` -> `NormalBlockMeanBase` -> `ZINormalBlockMeanUnknownClusters`

Active bindings

`fitted` Y values predicted by the model, in Y's original units

`var_par` a list with the variational parameter: tau (posterior group probabilities)

`model_par` a list with model parameters: B, Omega and kappa (zero-inflation probabilities)

`nb_param` number of parameters in the model

`entropy` Entropy of the conditional distribution

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- `ZINormalBlockMeanUnknownClusters$new()`
- `ZINormalBlockMeanUnknownClusters$clone()`

`ZINormalBlockMeanUnknownClusters$new()`: Create a new ['ZINormalBlockMeanUnknownClusters'] object.

Usage:

```
ZINormalBlockMeanUnknownClusters$new(
  data,
  q,
  sparsity = 0,
  control = NB_control()
)
```

Arguments:

`data` object of NormalBlockData class, with responses and design matrix

`q` number of clusters

`sparsity` unused here, kept for signature symmetry (must be 0)

`control` structured list of more specific parameters, to generate with NB_control

Returns: A new [‘ZINormalBlockMeanUnknownClusters‘] object

`ZINormalBlockMeanUnknownClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ZINormalBlockMeanUnknownClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_mean_data(n = 50, p = 20, d = 1, q = 3)
Y <- ex$Y; Y[runif(length(Y)) < 0.2] <- 0
data <- NormalBlockData$new(Y, ex$X)
model <- normal_block(data, blocks = 3, model = "mean", zero_inflation = TRUE)
model$clustering
```

ZINormalBlockVarKnownClusters

Zero-Inflated Normal-Block Model with Known Clustering

Description

R6 class for a zero-inflated normal-block model with a known clustering.

Super classes

`NormalBlockBase` -> `NormalBlockVarBase` -> `ZINormalBlockVarKnownClusters`

Active bindings

`posterior_par` a list with the parameters of posterior distribution W | Y

`entropy` Entropy of the conditional distribution

`nb_param` number of parameters in the model

`model_par` a list with model parameters: B (covariates), dm1 (species variance), Omega (groups precision matrix), kappa (zero-inflation probabilities)

`fitted` Y values predicted by the model, in Y's original units

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- ZINormalBlockVarKnownClusters$new()
- ZINormalBlockVarKnownClusters$clone()

`ZINormalBlockVarKnownClusters$new()`: Create a new ['ZINormalBlockVarKnownClusters'] object.

Usage:

```
ZINormalBlockVarKnownClusters$new(
  data,
  C,
  sparsity = 0,
  control = NB_control()
)
```

Arguments:

`data` object of NormalBlockData class, with responses and design matrix

`C` clustering matrix C_jk = 1 if species j belongs to cluster k

`sparsity` to apply on variance matrix when calling GLASSO

`control` structured list of more specific parameters, to generate with NB_control

Returns: A new ['ZINormalBlockVarKnownClusters'] object

`ZINormalBlockVarKnownClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ZINormalBlockVarKnownClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3, kappa = rep(0.3, 20))
data <- NormalBlockData$new(ex$Y, ex$X)
model <- normal_block(data, blocks = ex$parameters$C, zero_inflation = TRUE,
                      control = NB_control(verbose = FALSE))
model$clustering
```

ZINormalBlockVarUnknownClusters

Zero-Inflated Normal-Block Model with Unknown Clustering

Description

R6 class for a zero-inflated normal-block model with a fixed number of clusters (but unknown clustering).

Super classes

NormalBlockBase -> NormalBlockVarBase -> ZINormalBlockVarUnknownClusters

Public fields

`fixed_tau` whether tau should be fixed at clustering_init during optimization, useful for stability selection

Active bindings

`nb_param` number of parameters in the model

`var_par` a list with variational parameters

`model_par` a list with model parameters: B (covariates), dm1 (species variance), Omega (blocks precision matrix), kappa (zero-inflation probabilities)

`entropy` Entropy of the conditional distribution

`fitted` Y values predicted by the model, in Y's original units

`who_am_I` a method to print what model is being fitted

Methods

Public methods:

- `ZINormalBlockVarUnknownClusters$new()`
- `ZINormalBlockVarUnknownClusters$clone()`

`ZINormalBlockVarUnknownClusters$new()`: Create a new ['ZINormalBlockVarUnknownClusters'] object.

Usage:

```
ZINormalBlockVarUnknownClusters$new(
  data,
  q,
  sparsity = 0,
  control = NB_control()
)
```

Arguments:

`data` object of NormalBlockVarData class, with responses and design matrix

`q` required number of groups

`sparsity` to apply on variance matrix when calling GLASSO

`control` structured list of more specific parameters

Returns: A new ['ZINormalBlockVarUnknownClusters'] object

`ZINormalBlockVarUnknownClusters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ZINormalBlockVarUnknownClusters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
ex <- generate_normal_block_var_data(n = 50, p = 20, d = 1, q = 3, kappa = rep(0.3, 20))
data <- NormalBlockData$new(ex$Y, ex$X)
model <- normal_block(data, blocks = 3, zero_inflation = TRUE,
                      control = NB_control(verbose = FALSE))
model$clustering
```

Index