Package 'missoNet'

July 22, 2025

Type Package

Title Missingness in Multi-Task Regression with Network Estimation

Version 1.2.0

Date 2023-07-18

Maintainer Yixiao Zeng <yixiao.zeng@mail.mcgill.ca>

Description Efficient procedures for fitting conditional graphical lasso models that link a set of predictor variables to a set of response variables (or tasks), even when the response data may contain missing values. 'missoNet' simultaneously estimates the predictor coefficients for all tasks by leveraging information from one another, in order to provide more accurate predictions in comparison to modeling them individually. Additionally, 'missoNet' estimates the response network structure influenced by conditioning predictor variables using a L1-regularized conditional Gaussian graphical model. Unlike most penalized multi-task regression methods (e.g., MRCE), 'missoNet' is capable of obtaining estimates even when the response data is corrupted by missing values. The method automatically enjoys the theoretical and computational benefits of convexity, and returns solutions that are comparable to the estimates obtained without missingness.

License GPL-2

URL https://github.com/yixiao-zeng/missoNet

BugReports https://github.com/yixiao-zeng/missoNet/issues

Imports circlize (>= 0.4.14), ComplexHeatmap, glasso (>= 1.11), mvtnorm (>= 1.1.3), pbapply (>= 1.5.0), Rcpp (>= 1.0.8.3), scatterplot3d (>= 0.3.41)

Suggests knitr, rmarkdown

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Author Yixiao Zeng [aut, cre, cph],
Celia Greenwood [ths, aut],
Archer Yang [ths, aut]

Repository CRAN

Date/Publication 2023-07-19 13:40:02 UTC

Contents

`missoNet-package` *Multi-task regression and conditional network estimation with missing values in the tasks*

Description

Package:	missoNet
Type:	Package
Version:	1.0.0
Date:	2022-10-01
License:	GPL-2

missoNet functions

missoNet Fit a series of '`missoNet`' models with user-supplied regularization parameter pairs for the lasso penalties, $\{(\lambda_B, \lambda_\Theta)\}$.

cv.missoNet Perform k-fold cross-validation for '`missoNet`' over a grid of (auto-computed) regularization parameter pairs.

plot S3 method for plotting the cross-validation errors from a fitted `'cv.missoNet'` object.

predict S3 method for making predictions of response values from a fitted `'cv.missoNet'` object.

generateData Quickly generate synthetic data for simulation studies.

`cv.missoNet` *Cross-validation for missoNet*

Description

This function performs k-fold cross-validation for '`missoNet`'. The regularization path is computed for all possible combinations of values given in the two regularization parameter sequences, namely λ_B and λ_Θ. '`cv.missoNet`' will select the most suitable model among all cross-validated fits along the path. See the details of '`missoNet`' for the model definition. To help users, the '`cv.missoNet`' function is designed to automatically determine the likely ranges of the regularization parameters over which the cross-validation searches.

Usage

```
cv.missoNet(
  X,
  Y,
  kfold = 5,
  rho = NULL,
  lambda.Beta = NULL,
  lambda.Theta = NULL,
  lamBeta.min.ratio = NULL,
  lamTheta.min.ratio = NULL,
  n.lamBeta = NULL,
  n.lamTheta = NULL,
  lamBeta.scale.factor = 1,
  lamTheta.scale.factor = 1,
  Beta.maxit = 1000,
  Beta.thr = 1e-04,
  eta = 0.8,
  Theta.maxit = 1000,
  Theta.thr = 1e-04,
  eps = 1e-08,
  penalize.diagonal = TRUE,
  diag.penalty.factor = NULL,
  standardize = TRUE,
  standardize.response = TRUE,
  fit.1se = FALSE,
  fit.relax = FALSE,
  permute = TRUE,
  with.seed = NULL,
  parallel = FALSE,
  cl = NULL,
  verbose = 1
)
```

Arguments

X	Numeric predictor matrix ($n \times p$): columns correspond to predictor variables and rows correspond to samples. Missing values are not allowed. There is no need for centering or scaling of the variables. 'X' should not include a column of ones for an intercept.
Y	Numeric response matrix ($n \times q$): columns correspond to response variables and rows correspond to samples. Missing values should be coded as either 'NA's or 'NaN's. There is no need for centering or scaling of the variables.
kfold	Number of folds for cross-validation – the default is '5'.
rho	(Optional) A scalar or a numeric vector of length q: the elements are user-supplied probabilities of missingness for the response variables. The default is 'rho = NULL' and the program will compute the empirical missing rates for each of the columns of 'Y' and use them as the working missing probabilities. The default setting should suffice in most cases; misspecified missing probabilities would introduce biases into the model.
lambda.Beta	(Optional) Numeric vector: a user-supplied sequence of non-negative values for $\{\lambda_B\}$ penalizing the elements of the coefficient matrix $\mathbf{B}$ among which the cross-validation procedure searches. The default is 'lambda.Beta = NULL', in which case the program computes an appropriate range of λ_B values using 'n.lamBeta' and 'lamBeta.min.ratio'. Supplying a vector overrides this default. Note that the supplied sequence will be automatically arranged, internally, in a descending order.
lambda.Theta	(Optional) Numeric vector: a user-supplied sequence of non-negative values for $\{\lambda_\Theta\}$ penalizing the (off-diagonal) elements of the precision matrix $\mathbf{\Theta}$ among which the cross-validation procedure searches. The default is 'lambda.Theta = NULL', in which case the program computes an appropriate range of λ_Θ values using 'n.lamTheta' and 'lamTheta.min.ratio'. Supplying a vector overrides this default. Note that the supplied sequence will be automatically arranged, internally, in a descending order.
lamBeta.min.ratio	The smallest value of λ_B is calculated as the data-derived $\max(\lambda_B)$ multiplied by 'lamBeta.min.ratio'. The default depends on the sample size, n, relative to the number of predictors, p. If $n > p$, the default is '1.0E-4', otherwise it is '1.0E-3'. A very small value of 'lamBeta.min.ratio' may significantly increase runtime and lead to a saturated fit in the $n \leq p$ case. This is only needed when 'lambda.Beta = NULL'.
lamTheta.min.ratio	The smallest value of λ_Θ is calculated as the data-derived $\max(\lambda_\Theta)$ multiplied by 'lamTheta.min.ratio'. The default depends on the sample size, n, relative to the number of responses, q. If $n > q$, the default is '1.0E-4', otherwise it is '1.0E-3'. A very small value of 'lamTheta.min.ratio' may significantly increase runtime and lead to a saturated fit in the $n \leq q$ case. This is only needed when 'lambda.Theta = NULL'.
n.lamBeta	The number of λ_B values. If $n > p$, the default is '40', otherwise it is '30'. Avoid supplying an excessively large number since the program will

fit ('n.lamBeta' * 'n.lamTheta') models in total for each fold of the cross-validation. Typically we suggest 'n.lamBeta' = -log10('lamBeta.min.ratio') * c, where c $\in$ [10, 20]. This is only needed when 'lambda.Beta' = NULL'.

n.lamTheta The number of λ_Θ values. If $n > q$, the default is '40', otherwise it is '30'. Avoid supplying an excessively large number since the program will fit ('n.lamBeta' * 'n.lamTheta') models in total for each fold of the cross-validation. Typically we suggest 'n.lamTheta' = -log10('lamTheta.min.ratio') * c, where c $\in$ [10, 20]. This is only needed when 'lambda.Theta' = NULL'.

lamBeta.scale.factor
A positive multiplication factor for scaling the entire λ_B sequence; the default is '1'. A typical usage is when the magnitudes of the auto-computed λ_B values are inappropriate. For example, this factor would be needed if the optimal value of λ_B selected by the cross-validation (i.e. $\lambda_{B\min}$ with the minimum cross-validated error) approaches either boundary of the search range. This is only needed when 'lambda.Beta' = NULL'.

lamTheta.scale.factor
A positive multiplication factor for scaling the entire λ_Θ sequence; the default is '1'. A typical usage is when the magnitudes of the auto-computed λ_Θ values are inappropriate. For example, this factor would be needed if the optimal value of λ_Θ selected by the cross-validation (i.e. $\lambda_{\Theta\min}$ with the minimum cross-validated error) approaches either boundary of the search range. This is only needed when 'lambda.Theta' = NULL'.

Beta.maxit The maximum number of iterations of the fast iterative shrinkage-thresholding algorithm (FISTA) for updating $\hat{\mathbf{B}}$. The default is 'Beta.maxit' = 1000'.

Beta.thr The convergence threshold of the FISTA algorithm for updating $\hat{\mathbf{B}}$; the default is 'Beta.thr' = 1.0E-4'. Iterations stop when the absolute parameter change is less than ('Beta.thr' * sum(abs($\hat{\mathbf{B}}$))).

eta The backtracking line search shrinkage factor; the default is 'eta' = 0.8'. Most users will be able to use the default value, some experienced users may want to tune 'eta' according to the properties of a specific dataset for a faster convergence of the FISTA algorithm. Note that 'eta' must be in (0, 1).

Theta.maxit The maximum number of iterations of the 'glasso' algorithm for updating $\hat{\mathbf{\Theta}}$. The default is 'Theta.maxit' = 1000'.

Theta.thr The convergence threshold of the 'glasso' algorithm for updating $\hat{\mathbf{\Theta}}$; the default is 'Theta.thr' = 1.0E-4'. Iterations stop when the average absolute parameter change is less than ('Theta.thr' * ave(abs(offdiag($\hat{\mathbf{\Sigma}}$)))), where $\hat{\mathbf{\Sigma}}$ denotes the empirical working covariance matrix.

eps A numeric tolerance level for the L1 projection of the empirical covariance matrix; the default is 'eps' = 1.0E-8'. The empirical covariance matrix will be projected onto a L1 ball to have min(eigen($\hat{\mathbf{\Sigma}}$)$value) == 'eps', if any of the eigenvalues is less than the specified tolerance. Most users will be able to use the default value.

penalize.diagonal
Logical: should the diagonal elements of $\mathbf{\Theta}$ be penalized? The default is 'TRUE'.

diag.penalty.factor
Numeric: a separate penalty multiplication factor for the diagonal elements of $\mathbf{\Theta}$ when 'penalize.diagonal' = TRUE'. λ_Θ is multiplied by this number to allow

a differential shrinkage of the diagonal elements. The default is 'NULL' and the program will guess a value based on an initial estimate of $\mathbf{\Theta}$. This factor could be '0' for no shrinkage (equivalent to 'penalize.diagonal = FALSE') or '1' for an equal shrinkage.

standardize — Logical: should the columns of 'X' be standardized so each has unit variance? The default is 'TRUE'. The estimated results will always be returned on the original scale. 'cv.missoNet' computes appropriate λ sequences relying on standardization, if 'X' has been standardized prior to fitting the model, you might not wish to standardize it inside the algorithm.

standardize.response — Logical: should the columns of 'Y' be standardized so each has unit variance? The default is 'TRUE'. The estimated results will always be returned on the original scale. 'cv.missoNet' computes appropriate λ sequences relying on standardization, if 'Y' has been standardized prior to fitting the model, you might not wish to standardize it inside the algorithm.

fit.1se — Logical: the default is 'FALSE'. If 'TRUE', two additional models will be fitted with the largest values of λ_B and λ_Θ respectively at which the cross-validated error is within one standard error of the minimum.

fit.relax — Logical: the default is 'FALSE'. If 'TRUE', the program will re-estimate the edges in the active set (i.e. nonzero off-diagonal elements) of the network structure $\hat{\mathbf{\Theta}}$ without penalization ($\lambda_\Theta = 0$). This debiased estimate of $\mathbf{\Theta}$ could be useful for some interdependency analyses. WARNING: there may be convergence issues if the empirical covariance matrix is not of full rank (e.g. $n < q$)).

permute — Logical: should the subject indices for the cross-validation be permuted? The default is 'TRUE'.

with.seed — A random number seed for the permutation.

parallel — Logical: the default is 'FALSE'. If 'TRUE', the program uses clusters to compute the cross-validation folds in parallel. Must register parallel clusters beforehand, see examples below.

cl — A cluster object created by 'parallel::makeCluster' for parallel evaluations. This is only needed when 'parallel = TRUE'.

verbose — Value of '0', '1' or '2'. 'verbose = 0' – silent; 'verbose = 1' (the default) – limited tracing with progress bars; 'verbose = 2' – detailed tracing. Note that displaying the progress bars slightly increases the computation overhead compared to the silent mode. The detailed tracing should be used for monitoring progress only when the program runs extremely slowly, and it is not supported under 'parallel = TRUE'.

Details

The 'cv.missoNet' function fits 'missoNet' models ('kfold' * 'n.lamBeta' * 'n.lamTheta') times in the whole cross-validation process. That is, for the kth-fold ($k = 1, ..., K$) computation, the models are fitted at each of the all ('n.lamBeta' * 'n.lamTheta') possible combinations of the regularization parameters (λ_B, λ_Θ), with the kth fold of the training data omitted. The errors are accumulated, and the averaged errors as well as the standard deviations are computed over all folds. Note that the results of 'cv.missoNet' are random, since the samples are randomly split

into k-folds. Users can eliminate this randomness by setting 'permute = FALSE', or by explicitly assigning a seed to the permutation of samples.

A user-supplied sequence for $\{\lambda_B\}$ and/or $\{\lambda_\Theta\}$ is permitted, otherwise the program computes an appropriate range of values for the regularization parameters using other control arguments. Note that 'cv.missoNet' standardizes 'X' and 'Y' to have unit variances before computing its λ sequences (and then unstandardizes the resulting coefficients); if you wish to reproduce/compare results with those of other softwares, it is best to supply pre-standardized 'X' and 'Y'. If the algorithm is running slowly, track its progress with 'verbose = 2'. In most cases, choosing a sparser grid for the regularization parameters (e.g. smaller 'n.lamBeta' and/or 'n.lamTheta') or setting 'Beta.thr = 1.0E-3' (or even bigger) allows the algorithm to make faster progress.

After cross-validation, the regression coefficient matrix $\mathbf{B}$ and the precision matrix $\mathbf{\Theta}$ can be estimated at three special λ pairs, by reapplying 'missoNet' to the entire training dataset:

1. "lambda.min" := $(\lambda_{B\min}, \lambda_{\Theta\min})$, at which the minimum mean cross-validated error is achieved;
2. "lambda.1se.Beta" := $(\lambda_{B1\text{se}}, \lambda_{\Theta\min})$, where $\lambda_{B1\text{se}}$ is the largest λ_B at which the mean cross-validated error is within one standard error of the minimum;
3. "lambda.1se.Theta" := $(\lambda_{B\min}, \lambda_{\Theta1\text{se}})$, where $\lambda_{\Theta1\text{se}}$ is the largest λ_Θ at which the mean cross-validated error is within one standard error of the minimum.

The corresponding estimates, along with the λ values, are stored in three separate lists inside the returned object: 'est.min', 'est.1se.B' and 'est.1se.Tht' ('est.1se.B' and 'est.1se.Tht' are 'NULL' unless the argument 'fit.1se = TRUE' when calling 'cv.missoNet').

The 'cv.missoNet' function returns an R object of S3 class 'cv.missoNet' for which there are a set of accessory functions available. The plotting function 'plot.cv.missoNet' can be used to graphically identify the optimal pair of the regularization parameters, and the prediction function 'predict.cv.missoNet' can be used to make predictions of response values given new input 'X'. See the vignette for examples.

Value

This function returns a 'cv.missoNet' object containing a named 'list' with all the ingredients of the cross-validated fit:

est.min A 'list' of results estimated at "lambda.min" := $(\lambda_{B\min}, \lambda_{\Theta\min})$ that gives the minimum mean cross-validated error. It consists of the following components:

- Beta: the penalized estimate of the regression coefficient matrix $\hat{\mathbf{B}}$ $(p \times q)$.
- Theta: the penalized estimate of the precision matrix $\hat{\mathbf{\Theta}}$ $(q \times q)$.
- mu: a vector of length q storing the model intercept $\hat{\mu}$.
- lambda.Beta: the value of λ_B (i.e. $\lambda_{B\min}$) used to fit the model.
- lambda.Theta: the value of λ_Θ (i.e. $\lambda_{\Theta\min}$) used to fit the model.
- relax.net: the relaxed (debiased) estimate of the conditional network structure $\hat{\mathbf{\Theta}}_{\text{rlx}}$ $(q \times q)$ if 'fit.relax = TRUE' when calling 'cv.missoNet'.

est.1se.B A 'list' of results estimated at "lambda.1se.Beta" := $(\lambda_{B1\text{se}}, \lambda_{\Theta\min})$ if 'fit.1se = TRUE' when calling 'cv.missoNet'. "lambda.1se.Beta" refers to the largest λ_B at which the mean cross-validated error is within one standard error of the minimum, by fixing λ_Θ at $\lambda_{\Theta\min}$. This 'list' consists of the same components as 'est.min'.

`est.1se.Tht` A `'list'` of results estimated at `"lambda.1se.Theta"` := $(\lambda_{B\min}, \lambda_{\Theta 1\mathrm{se}})$ if `'fit.1se = TRUE'` when calling 'cv.missoNet'. `"lambda.1se.Theta"` refers to the largest λ_Θ at which the mean cross-validated error is within one standard error of the minimum, by fixing λ_B at $\lambda_{B\min}$. This `'list'` consists of the same components as `'est.min'`.

`rho` A vector of length q storing the working missing probabilities for the q response variables.

`fold.index` The subject indices identifying which fold each observation is in.

`lambda.Beta.vec`
A flattened vector of length (`'n.lamBeta' * 'n.lamTheta'`) storing the λ_B values along the regularization path. More specifically, `'lambda.Beta.vec' = rep('lambda.Beta', each = 'n.lamTheta')`.

`lambda.Theta.vec`
A flattened vector of length (`'n.lamBeta' * 'n.lamTheta'`) storing the λ_Θ values along the regularization path. More specifically, `'lambda.Theta.vec' = rep('lambda.Theta', times = 'n.lamBeta')`.

`cvm` A flattened vector of length (`'n.lamBeta' * 'n.lamTheta'`) storing the (standardized) mean cross-validated errors along the regularization path.

`cvup` Upper cross-validated errors.

`cvlo` Lower cross-validated errors.

`penalize.diagonal`
Logical: whether the diagonal elements of $\mathbf{\Theta}$ were penalized.

`diag.penalty.factor`
The additional penalty multiplication factor for the diagonal elements of $\mathbf{\Theta}$ when `'penalize.diagonal'` was returned as `'TRUE'`.

Author(s)

Yixiao Zeng <yixiao.zeng@mail.mcgill.ca>, Celia M.T. Greenwood and Archer Yi Yang.

Examples

```
## Simulate a dataset with response values missing completely at random (MCAR),
## the overall missing rate is around 10%.
set.seed(123)  # reproducibility
sim.dat <- generateData(n = 300, p = 50, q = 20, rho = 0.1, missing.type = "MCAR")
tr <- 1:240  # training set indices
tst <- 241:300  # test set indices
X.tr <- sim.dat$X[tr, ]  # predictor matrix
Y.tr <- sim.dat$Z[tr, ]  # corrupted response matrix


## Perform a five-fold cross-validation WITH specified 'lambda.Beta' and 'lambda.Theta'.
## 'standardize' and 'standardize.response' are 'TRUE' by default.
lamB.vec <- 10^(seq(from = 0, to = -1, length.out = 5))
lamTht.vec <- 10^(seq(from = 0, to = -1, length.out = 5))
cvfit <- cv.missoNet(X = X.tr, Y = Y.tr, kfold = 5,
                     lambda.Beta = lamB.vec, lambda.Theta = lamTht.vec)
```

```
## Perform a five-fold cross-validation WITHOUT specified 'lambda.Beta' and 'lambda.Theta'.
## In this case, a grid of 'lambda.Beta' and 'lambda.Theta' values in a (hopefully) reasonable
## range will be computed and used by the program.
##
## < This simplest command should be a good start for most analyses. >
cvfit <- cv.missoNet(X = X.tr, Y = Y.tr, kfold = 5)


## Alternatively, compute the cross-validation folds in parallel on a cluster with 2 cores.
##
## 'fit.1se = TRUE' tells the program to make additional estimations of the parameters at the
## largest value of 'lambda.Beta' / 'lambda.Theta' that gives the most regularized model such
## that the cross-validated error is within one standard error of the minimum.
cl <- parallel::makeCluster(min(parallel::detectCores()-1, 2))
cvfit <- cv.missoNet(X = X.tr, Y = Y.tr, kfold = 5, fit.1se = TRUE,
                     parallel = TRUE, cl = cl,
                 permute = TRUE, with.seed = 486)  # permute with seed for reproducibility
parallel::stopCluster(cl)


## Use PRE-STANDARDIZED training data if you wish to compare the results with other softwares.
X.tr.std <- scale(X.tr, center = TRUE, scale = TRUE)
Y.tr.std <- scale(Y.tr, center = TRUE, scale = TRUE)
cvfit.std <- cv.missoNet(X = X.tr.std, Y = Y.tr.std, kfold = 5,
                         standardize = FALSE, standardize.response = FALSE)


## Plot the (standardized) mean cross-validated errors in a heatmap.
plot(cvfit, type = "cv.heatmap")

## Plot the (standardized) mean cross-validated errors in a 3D scatterplot.
plot(cvfit, type = "cv.scatter", plt.surf = TRUE)


## Extract the estimates at "lambda.min".
Beta.hat1 <- cvfit$est.min$Beta
Theta.hat1 <- cvfit$est.min$Theta

## Extract the estimates at "lambda.1se.Beta" (if 'fit.1se' = TRUE).
Beta.hat2 <- cvfit$est.1se.B$Beta
Theta.hat2 <- cvfit$est.1se.B$Theta

## Extract the estimates at "lambda.1se.Theta" (if 'fit.1se' = TRUE).
Beta.hat3 <- cvfit$est.1se.Tht$Beta
Theta.hat3 <- cvfit$est.1se.Tht$Theta


## Make predictions of response values on the test set.
newy1 <- predict(cvfit, newx = sim.dat$X[tst, ], s = "lambda.min")
newy2 <- predict(cvfit, newx = sim.dat$X[tst, ], s = "lambda.1se.Beta")  # 'fit.1se' = TRUE
newy3 <- predict(cvfit, newx = sim.dat$X[tst, ], s = "lambda.1se.Theta")  # 'fit.1se' = TRUE
```

generateData *Quickly generate synthetic data for simulation studies*

Description

The 'generateData' function is used to readily produce synthetic data with randomly/systematically-missing values from a conditional Gaussian graphical model. This function supports three types of missing mechanisms that can be specified by users – missing completely at random (MCAR), missing at random (MAR) and missing not at random (MNAR).

Usage

```
generateData(
  X = NULL,
  Beta = NULL,
  E = NULL,
  Theta = NULL,
  Sigma.X = NULL,
  n,
  p,
  q,
  rho,
  missing.type = "MCAR",
  Beta.row.sparsity = 0.2,
  Beta.elm.sparsity = 0.2,
  with.seed = NULL
)
```

Arguments

X	(Optional) a user-supplied predictor matrix ($n \times p$). The default is 'NULL' and the program simulates the rows of 'X' independently from $\mathcal{MVN}(0_p, \mathbf{\Sigma}_X)$. A user-supplied matrix overrides this default, and the argument 'Sigma.X' for $\mathbf{\Sigma}_X$ will be ignored.
Beta	(Optional) a user-supplied regression coefficient matrix $\mathbf{B}$ ($p \times q$). The default is 'NULL' and the program will generate a sparse $\mathbf{B}$ in which the nonzero elements are independently drawn from $\mathcal{N}(0, 1)$; the row sparsity and element sparsity of $\mathbf{B}$ are controlled by the arguments 'Beta.row.sparsity' and 'Beta.elm.sparsity', respectively. A user-supplied matrix overrides this default, and 'Beta.row.sparsity' and 'Beta.elm.sparsity' will be ignored.
E	(Optional) a user-supplied error matrix ($n \times q$). The default is 'NULL' and the program simulates the rows of 'E' independently from $\mathcal{MVN}(0_q, \mathbf{\Theta}^{-1})$. A response matrix 'Y' without missing values is given by 'Y = X %*% Beta + E'. A user-supplied matrix overrides this default, and the argument 'Theta' for $\mathbf{\Theta}$ will be ignored.

`Theta` (Optional) a user-supplied positive definite precision (inverse covariance) matrix $\mathbf{\Theta}$ ($q \times q$) for the response variables. The default is `'NULL'` and the program will generate a block-structured matrix having four blocks corresponding to four types of network structures: independent, weak graph, strong graph and chain. This is only needed when `'E = NULL'`.

`Sigma.X` (Optional) A user-supplied positive definite covariance matrix $\mathbf{\Sigma}_X$ ($p \times p$) for the predictor variables. The samples of `'X'` are independently drawn from a multivariate Gaussian distribution $\mathcal{MVN}(0_p, \mathbf{\Sigma}_X)$. If `'Sigma.X = NULL'` (default), the program uses an AR(1) covariance with 0.7 autocorrelation (i.e., $[\mathbf{\Sigma}_X]_{jk} = 0.7^{|j-k|}$). This is only needed when `'X = NULL'`.

`n` Sample size.

`p` The dimensionality of the predictors.

`q` The dimensionality of the responses.

`rho` A scalar or a numeric vector of length q specifying the approximate proportion of missing values in each column of the response matrix.

`missing.type` Character string: can be `"MCAR"` (default), `"MAR"` or `"MNAR"`.

`Beta.row.sparsity`
A Bernoulli parameter between 0 and 1 controlling the approximate proportion of rows where at least one element could be nonzero in $\mathbf{B}$; the default is `'Beta.row.sparsity = 0.2'`. This is only needed when `'Beta = NULL'`.

`Beta.elm.sparsity`
A Bernoulli parameter between 0 and 1 controlling the approximate proportion of nonzero elements among the rows of $\mathbf{B}$ where not all elements are zeros; the default is `'Beta.elm.sparsity = 0.2'`. This is only needed when `'Beta = NULL'`.

`with.seed` A random number seed for the generative process.

Details

The dataset is simulated through the following steps:

1. If `'X = NULL'` (default), the function '`MASS::mvrnorm(n, mean = rep(0, p), sigma = Sigma.X)`' is used to simulate `'n'` samples from a `'p'`-variate Gaussian distribution for generating a predictor matrix `'X'`;
2. If `'Beta = NULL'` (default), the function '`stats::rnorm(p*q, 0, 1)`' is used to fill an empty ($p \times q$) dimensional matrix `'Beta'`, of which the row sparsity and element sparsity are later controlled by the auxiliary arguments `'Beta.row.sparsity'` and `'Beta.elm.sparsity'`, respectively;
3. If `'E = NULL'` (default), the function '`MASS::mvrnorm(n, mean = rep(0, q), sigma = solve(Theta))`' is used to simulate `'n'` samples from a `'q'`-variate Gaussian distribution for generating an error matrix `'E'`;
4. A complete response matrix `'Y'` without missing values is then generated by the command `'Y = X %*% Beta + E'`;
5. To get a response matrix `'Z'` := f(`'Y'`) corrupted by missing data, the values in `'Y'` are partially replaced with `'NA'`s following the strategy specified by the arguments `'missing.type'` and `'rho'`.

To better illustrate the step 5 above, suppose for all i = 1, . . . , n and j = 1, . . . , q: 'Y[i, j]' is replaced with 'NA' if 'M[i, j] == 1', where 'M' is an indicator matrix of missingness having the same dimension as 'Y'. The value of 'M[i, j]' is partially controlled by the arguments 'missing.type' and 'rho'. Below we sum up the three built-in missing mechanisms supported by the 'generateData' function:

- 'missing.type' == "MCAR": 'Y[i, j] <- NA' if 'M[i, j] == 1', where 'M[i, j] = rbinom(0, rho[j])';
- 'missing.type' == "MAR": 'Y[i, j] <- NA' if 'M[i, j] == 1', where 'M[i, j] = rbinom(0, (rho[j] * c / (1 + exp(-(X %*% Beta)[i, j]))))', in which c is a constant correcting the missing rate of the jth column of 'Y' to 'rho[j]';
- 'missing.type' == "MNAR": 'Y[i, j] <- NA' if 'M[i, j] == 1', where 'M[i, j] = 1 * (Y[i, j] < Tj)', in which 'Tj = quantile(Y[, j], rho[j])'.

Of the aforementioned missing mechanisms, "MCAR" is random, and the other two are systematic. under "MCAR", 'M[i, j]' is not related to 'Y' or to 'X'; under "MAR", 'M[i, j]' is related to 'X', but not related to 'Y' after 'X' is controlled; under "MNAR", 'M[i, j]' is related to 'Y' itself, even after 'X' is controlled.

Value

This function returns a 'list' consisting of the following components:

X	A simulated (or the user-supplied) predictor matrix ($n \times p$).
Y	A simulated response matrix without missing values ($n \times q$).
Z	A simulated response matrix with missing values coded as 'NA's ($n \times q$).
Beta	The regression coefficient matrix $\mathbf{B}$ for the generative model ($p \times q$).
Theta	The precision matrix $\mathbf{\Theta}$ for the generative model ($q \times q$).
rho	A vector of length q storing the specified missing rate for each column of the response matrix.
missing.type	Character string: the type of missing mechanism used to generate missing values in the response matrix.

Author(s)

Yixiao Zeng <yixiao.zeng@mail.mcgill.ca>, Celia M.T. Greenwood and Archer Yi Yang.

Examples

```
## Simulate a dataset with response values missing completely at random (MCAR),
## the overall missing rate is around 10%.
sim.dat <- generateData(n = 300, p = 50, q = 20, rho = 0.1, missing.type = "MCAR")
## ------------------------------------------------------------------------------
## Fit a missoNet model using the simulated dataset.
X <- sim.dat$X  # predictor matrix
Y <- sim.dat$Z  # corrupted response matrix
fit <- missoNet(X = X, Y = Y, lambda.Beta = 0.1, lambda.Theta = 0.1)
```

```
## Simulate a dataset with response values missing at random (MAR), the approximate
## missing rate for each column of the response matrix is specified through a vector 'rho'.
##
## The row sparsity and element sparsity of the auto-generated 'Beta' could be
## adjusted correspondingly by using 'Beta.row.sparsity' and 'Beta.elm.sparsity'.
n <- 300; p <- 50; q <- 20
rho <- runif(q, min = 0, max = 0.2)
sim.dat <- generateData(n = n, p = p, q = q, rho = rho, missing.type = "MAR",
                        Beta.row.sparsity = 0.3, Beta.elm.sparsity = 0.2)


## Simulate a dataset with response values missing not at random (MNAR),
## using the user-supplied 'Beta' and 'Theta'.
n <- 300; p <- 50; q <- 20
Beta <- matrix(rnorm(p*q, 0, 1), p, q)  # a nonsparse 'Beta' (p x q)
Theta <- diag(q)  # a diagonal 'Theta' (q x q)
sim.dat <- generateData(Beta = Beta, Theta = Theta, n = n, p = p, q = q,
                        rho = 0.1, missing.type = "MNAR")
## ---------------------------------------------------------------------
## Specifying just one of 'Beta' and 'Theta' is also allowed.
sim.dat <- generateData(Theta = Theta, n = n, p = p, q = q,
                        rho = 0.1, missing.type = "MNAR")


## User-supplied 'X', 'Beta' and 'E', in which case 'Y' is deterministic.
n <- 300; p <- 50; q <- 20
X <- matrix(rnorm(n*p, 0, 1), n, p)
Beta <- matrix(rnorm(p*q, 0, 1), p, q)
E <- mvtnorm::rmvnorm(n, rep(0, q), sigma = diag(q))
sim.dat <- generateData(X = X, Beta = Beta, E = E, n = n, p = p, q = q,
                        rho = 0.1, missing.type = "MCAR")
```

`missoNet` *Fit a series of missoNet models with user-supplied regularization parameters for the lasso penalties*

Description

This function fits the conditional graphical lasso models to datasets with missing response values. '`missoNet`' computes the regularization path for the lasso penalties sequentially along the bivariate regularization parameter sequence $\{(\lambda_B, \lambda_\Theta)\}$ provided by the user.

Usage

```
missoNet(
  X,
  Y,
  lambda.Beta,
  lambda.Theta,
```

```
  rho = NULL,
  Beta.maxit = 10000,
  Beta.thr = 1e-08,
  eta = 0.8,
  Theta.maxit = 10000,
  Theta.thr = 1e-08,
  eps = 1e-08,
  penalize.diagonal = TRUE,
  diag.penalty.factor = NULL,
  standardize = TRUE,
  standardize.response = TRUE,
  fit.relax = FALSE,
  parallel = FALSE,
  cl = NULL,
  verbose = 1
)
```

Arguments

`X`	Numeric predictor matrix ($n \times p$): columns correspond to predictor variables and rows correspond to samples. Missing values are not allowed. There is no need for centering or scaling of the variables. `'X'` should not include a column of ones for an intercept.
`Y`	Numeric response matrix ($n \times q$): columns correspond to response variables and rows correspond to samples. Missing values should be coded as either `'NA'`s or `'NaN'`s. There is no need for centering or scaling of the variables.
`lambda.Beta`	A scalar or a numeric vector: a user-supplied sequence of non-negative value(s) for $\{\lambda_B\}$ used to penalize the elements of the coefficient matrix $\mathbf{B}$. Note that the values will be sequentially visited in the given orders as inputs to the regularization parameter sequence $\{(\lambda_B, \lambda_\Theta)\}$; `'lambda.Beta'` must have the same length as `'lambda.Theta'`.
`lambda.Theta`	A scalar or a numeric vector: a user-supplied sequence of non-negative value(s) for $\{\lambda_\Theta\}$ used to penalize the (off-diagonal) elements of the precision matrix $\mathbf{\Theta}$. Note that the values will be sequentially visited in the given orders as inputs to the regularization parameter sequence $\{(\lambda_B, \lambda_\Theta)\}$; `'lambda.Theta'` must have the same length as `'lambda.Beta'`.
`rho`	(Optional) A scalar or a numeric vector of length q: the elements are user-supplied probabilities of missingness for the response variables. The default is `'rho = NULL'` and the program will compute the empirical missing rates for each of the columns of `'Y'` and use them as the working missing probabilities. The default setting should suffice in most cases; misspecified missing probabilities would introduce biases into the model.
`Beta.maxit`	The maximum number of iterations of the fast iterative shrinkage-thresholding algorithm (FISTA) for updating $\hat{\mathbf{B}}$. The default is `'Beta.maxit = 10000'`.
`Beta.thr`	The convergence threshold of the FISTA algorithm for updating $\hat{\mathbf{B}}$; the default is `'Beta.thr = 1.0E-8'`. Iterations stop when the absolute parameter change is less than (`'Beta.thr' * sum(abs(`$\hat{\mathbf{B}}$`))`).

`eta` The backtracking line search shrinkage factor; the default is `'eta = 0.8'`. Most users will be able to use the default value, some experienced users may want to tune `'eta'` according to the properties of a specific dataset for a faster convergence of the FISTA algorithm. Note that `'eta'` must be in (0, 1).

`Theta.maxit` The maximum number of iterations of the 'glasso' algorithm for updating $\hat{\mathbf{\Theta}}$. The default is `'Theta.maxit = 10000'`.

`Theta.thr` The convergence threshold of the 'glasso' algorithm for updating $\hat{\mathbf{\Theta}}$; the default is `'Theta.thr = 1.0E-8'`. Iterations stop when the average absolute parameter change is less than (`'Theta.thr' * ave(abs(offdiag(`$\hat{\mathbf{\Sigma}}$`)))`), where $\hat{\mathbf{\Sigma}}$ denotes the empirical working covariance matrix.

`eps` A numeric tolerance level for the L1 projection of the empirical covariance matrix; the default is `'eps = 1.0E-8'`. The empirical covariance matrix will be projected onto a L1 ball to have `min(eigen(`$\hat{\mathbf{\Sigma}}$`)$value) == 'eps'`, if any of the eigenvalues is less than the specified tolerance. Most users will be able to use the default value.

`penalize.diagonal`
Logical: should the diagonal elements of $\mathbf{\Theta}$ be penalized? The default is `'TRUE'`.

`diag.penalty.factor`
Numeric: a separate penalty multiplication factor for the diagonal elements of $\mathbf{\Theta}$ when `'penalize.diagonal = TRUE'`. λ_Θ is multiplied by this number to allow a differential shrinkage of the diagonal elements. The default is `'NULL'` and the program will guess a value based on an initial estimate of $\mathbf{\Theta}$. This factor could be `'0'` for no shrinkage (equivalent to `'penalize.diagonal = FALSE'`) or `'1'` for an equal shrinkage.

`standardize` Logical: should the columns of `'X'` be standardized so each has unit variance? The default is `'TRUE'`. The estimated results will always be returned on the original scale. If `'X'` has been standardized prior to fitting the model, you might not wish to standardize it inside the algorithm.

`standardize.response`
Logical: should the columns of `'Y'` be standardized so each has unit variance? The default is `'TRUE'`. The estimated results will always be returned on the original scale. If `'Y'` has been standardized prior to fitting the model, you might not wish to standardize it inside the algorithm.

`fit.relax` Logical: the default is `'FALSE'`. If `'TRUE'`, the program will re-estimate the edges in the active set (i.e. nonzero off-diagonal elements) of the network structure $\hat{\mathbf{\Theta}}$ without penalization ($\lambda_\Theta = 0$). This debiased estimate of $\mathbf{\Theta}$ could be useful for some interdependency analyses. WARNING: there may be convergence issues if the empirical covariance matrix is not of full rank (e.g. $n < q$)).

`parallel` Logical: the default is `'FALSE'`. If `'TRUE'`, the program uses clusters to fit models with each element of the λ sequence $\{(\lambda_B, \lambda_\Theta)\}$ in parallel. Must register parallel clusters beforehand, see examples below.

`cl` A cluster object created by 'parallel::makeCluster' for parallel evaluations. This is only needed when `'parallel = TRUE'`.

`verbose` Value of `'0'`, `'1'` or `'2'`. `'verbose = 0'` – silent; `'verbose = 1'` (the default) – limited tracing with progress bars; `'verbose = 2'` – detailed tracing. Note that displaying the progress bars slightly increases the computation overhead

compared to the silent mode. The detailed tracing should be used for monitoring progress only when the program runs extremely slowly, and it is not supported under 'parallel = TRUE'.

Details

'missoNet' is the main model-fitting function which is specifically proposed to fit the conditional graphical lasso models / penalized multi-task Gaussian regressions to (corrupted) datasets with response values missing at random (MAR). To facilitate the interpretation of the model, let's temporarily assume that there are no missing values in the data used to fit the model. Suppose we have n observations of both a p-variate predictor $X \in \mathcal{R}^p$ and a q-variate response $Y \in \mathcal{R}^q$, for the ith sample ($i = 1, ..., n$), 'missoNet' assumes the model

$$Y_i = \mu + X_i\mathbf{B} + E_i, \;\; E_i \sim \mathcal{MVN}(0_q, (\mathbf{\Theta})^{-1}),$$

where $Y_i \in \mathcal{R}^{1\times q}$ and $X_i \in \mathcal{R}^{1\times p}$ are one realization of the q responses and the p predictors, respectively. $E_i \in \mathcal{R}^{1\times q}$ is an error vector drawn from a multivariate Gaussian distribution.

The regression coefficient matrix $\mathbf{B} \in \mathcal{R}^{p\times q}$ that mapping predictors to responses and the precision (inverse covariance) matrix $\mathbf{\Theta} \in \mathcal{R}^{q\times q}$ that revealing the responses' conditional dependencies are the parameters to be estimated by solving a penalized MLE problem

$$(\hat{\mathbf{\Theta}}, \hat{\mathbf{B}}) = \text{argmin}_{\mathbf{\Theta}\succeq 0,\, \mathbf{B}}\; g(\mathbf{\Theta}, \mathbf{B}) + \lambda_\Theta(\|\mathbf{\Theta}\|_{1,\text{off}} + 1_{n\leq \max(p,q)}\|\mathbf{\Theta}\|_{1,\text{diag}}) + \lambda_B\|\mathbf{B}\|_1,$$

where

$$g(\mathbf{\Theta}, \mathbf{B}) = \text{tr}\left[\frac{1}{n}(\mathbf{Y} - \mathbf{XB})^\top(\mathbf{Y} - \mathbf{XB})\mathbf{\Theta}\right] - \log|\mathbf{\Theta}|.$$

The response matrix $\mathbf{Y} \in \mathcal{R}^{n\times q}$ has ith row ($Y_i - \frac{1}{n}\sum_{j=1}^n Y_j$), and the predictor matrix $\mathbf{X} \in \mathcal{R}^{n\times p}$ has ith row ($X_i - \frac{1}{n}\sum_{j=1}^n X_j$). The intercept $\mu \in \mathcal{R}^{1\times q}$ is canceled out because of centering of the data matrices $\mathbf{Y}$ and $\mathbf{X}$. $1_{n\leq \max(p,q)}$ denotes the indicator function for whether penalizing the diagonal elements of $\mathbf{\Theta}$ or not. When $n \leq \max(p, q)$, a global minimizer of the objective function defined above does not exist without the diagonal penalization.

Missingness in real data is inevitable. In this instance, the estimates based only on complete cases are likely to be biased, and the objective function is likely to no longer be a biconvex optimization problem. In addition, many algorithms cannot be directly employed since they require complete datasets as inputs. 'missoNet' aims to handle the specific situation where the response matrix $\mathbf{Y}$ contains values that are missing at random (MAR. Please refer to the vignette or other resources for more information about the differences between MAR, missing completely at random (MCAR) and missing not at random (MNAR)). As it should be, 'missoNet' is also applicable to datasets with MCAR response values or without any missing values. The method provides a unified framework for automatically solving a convex modification of the multi-task learning problem defined above, using corrupted datasets. Moreover, 'missoNet' enjoys the theoretical and computational benefits of convexity and returns solutions that are comparable/close to the clean conditional graphical lasso estimates. Please refer to the original manuscript (coming soon) for more details of our method.

Value

This function returns a 'list' consisting of the following components:

`est.list` A named 'list' storing the lists of results estimated at each of the λ pairs, (λ_B, λ_Θ). Each sub-'list' contains:

- `Beta`: the penalized estimate of the regression coefficient matrix $\hat{\mathbf{B}}$ ($p \times q$).
- `Theta`: the penalized estimate of the precision matrix $\hat{\mathbf{\Theta}}$ ($q \times q$).
- `mu`: a vector of length q storing the model intercept $\hat{\mu}$.
- `lambda.Beta`: the value of λ_B used to fit the model.
- `lambda.Theta`: the value of λ_Θ used to fit the model.
- `relax.net`: the relaxed (debiased) estimate of the conditional network structure $\hat{\mathbf{\Theta}}_{\text{rlx}}$ ($q \times q$) if `'fit.relax = TRUE'` when calling '`missoNet`'.

`rho` A vector of length q storing the working missing probabilities for the q response variables.

`penalize.diagonal`
Logical: whether the diagonal elements of $\mathbf{\Theta}$ were penalized.

`diag.penalty.factor`
The additional penalty multiplication factor for the diagonal elements of $\mathbf{\Theta}$ when `'penalize.diagonal'` was returned as `'TRUE'`.

Author(s)

Yixiao Zeng <yixiao.zeng@mail.mcgill.ca>, Celia M.T. Greenwood and Archer Yi Yang.

Examples

```
## Simulate a dataset with response values missing completely at random (MCAR),
## the overall missing rate is around 10%.
set.seed(123)  # reproducibility
sim.dat <- generateData(n = 300, p = 50, q = 20, rho = 0.1, missing.type = "MCAR")
tr <- 1:240  # training set indices
tst <- 241:300  # test set indices
X.tr <- sim.dat$X[tr, ]  # predictor matrix
Y.tr <- sim.dat$Z[tr, ]  # corrupted response matrix


## Fit one missoNet model with two scalars for 'lambda.Beta' and 'lambda.Theta'.
fit1 <- missoNet(X = X.tr, Y = Y.tr, lambda.Beta = 0.1, lambda.Theta = 0.2)


## Fit a series of missoNet models with the lambda pairs := (lambda.Beta, lambda.Theta)
## sequentially extracted from the 'lambda.Beta' and 'lambda.Theta' vectors, note that the
## two vectors must have the same length.
lamB.vec <- 10^(seq(from = 0, to = -1, length.out = 5))
lamTht.vec <- rep(0.1, 5)
fit2 <- missoNet(X = X.tr, Y = Y.tr, lambda.Beta = lamB.vec, lambda.Theta = lamTht.vec)


## Parallelization on a cluster with two cores.
cl <- parallel::makeCluster(2)
fit2 <- missoNet(X = X.tr, Y = Y.tr, lambda.Beta = lamB.vec, lambda.Theta = lamTht.vec,
                 parallel = TRUE, cl = cl)
parallel::stopCluster(cl)
```

```
## Extract the estimates at ('lamB.vec[1]', 'lamTht.vec[1]').
## The estimates at the subsequent lambda pairs could be accessed in the same way.
Beta.hat <- fit2$est.list[[1]]$Beta
Theta.hat <- fit2$est.list[[1]]$Theta
lambda.Beta <- fit2$est.list[[1]]$lambda.Beta  # equal to 'lamB.vec[1]'
lambda.Theta <- fit2$est.list[[1]]$lambda.Theta  # equal to 'lamTht.vec[1]'


## Fit a series of missoNet models using PRE-STANDARDIZED training data
## if you wish to compare the results with other softwares.
X.tr.std <- scale(X.tr, center = TRUE, scale = TRUE)
Y.tr.std <- scale(Y.tr, center = TRUE, scale = TRUE)
fit3 <- missoNet(X = X.tr.std, Y = Y.tr.std, lambda.Beta = lamB.vec, lambda.Theta = lamTht.vec,
                 standardize = FALSE, standardize.response = FALSE)
```

`plot.cv.missoNet` *Plot the cross-validation errors produced by cv.missoNet*

Description

S3 method for plotting the cross-validation error surface from a fitted `'cv.missoNet'` object.

Usage

```
## S3 method for class 'cv.missoNet'
plot(
  x,
  type = c("cv.heatmap", "cv.scatter"),
  detailed.axes = TRUE,
  plt.surf = TRUE,
  ...
)
```

Arguments

`x`	A fitted `'cv.missoNet'` object.
`type`	A character string for the type of plot, can be either "`cv.heatmap`" (default) or "`cv.scatter`".
`detailed.axes`	Logical: whether the detailed axes should be plotted. The default is `'TRUE'`.
`plt.surf`	Logical: whether to draw the error surface. The default is `'TRUE'`. This is only needed when `'type' = "cv.scatter"`.
`...`	Other graphical arguments used by 'ComplexHeatmap::Heatmap' (`'type' = "cv.heatmap"`) or 'scatterplot3d::scatterplot3d' (`'type' = "cv.scatter"`).

Value

The plot object.

Author(s)

Yixiao Zeng <yixiao.zeng@mail.mcgill.ca>, Celia M.T. Greenwood and Archer Yi Yang.

Examples

```
## Simulate a dataset.
set.seed(123)  # reproducibility
sim.dat <- generateData(n = 200, p = 10, q = 10, rho = 0.1, missing.type = "MCAR")


## Perform a five-fold cross-validation on the simulated dataset.
cvfit <- cv.missoNet(X = sim.dat$X, Y = sim.dat$Z, kfold = 5,
                     fit.1se = TRUE, permute = TRUE, with.seed = 486)


## Plot the (standardized) mean cross-validated errors in a heatmap.
plot(cvfit, type = "cv.heatmap")

## Plot the (standardized) mean cross-validated errors in a 3D scatterplot.
plot(cvfit, type = "cv.scatter", plt.surf = TRUE)
```

---

predict.cv.missoNet *Make predictions from a cv.missoNet object*

Description

S3 method for making predictions of response values from a fitted 'cv.missoNet' object.

Usage

```
## S3 method for class 'cv.missoNet'
predict(object, newx = NULL, s = "lambda.min", ...)
```

Arguments

object	A fitted 'cv.missoNet' object.
newx	A predictor matrix of new values at which predictions are to be made. The columns of 'newx' should have the same standardization flags as the original input for training the model. Missing values are not allowed. 'newx' should not include a column of ones for an intercept.
s	Character string, the regularization parameter pair $\lambda = (\lambda_B, \lambda_\Theta)$ at which the coefficients are extracted for making predictions. It supports three special strings, named "lambda.min" (default), "lambda.1se.Beta" and "lambda.1se.Theta".
...	Not used. Other arguments for predicting.

Value

The matrix of predicted values: 'newy = mu_hat + newx %*% Beta_hat'.

Author(s)

Yixiao Zeng <yixiao.zeng@mail.mcgill.ca>, Celia M.T. Greenwood and Archer Yi Yang.

Examples

```
## Simulate a dataset.
set.seed(123)  # reproducibility
sim.dat <- generateData(n = 300, p = 10, q = 10, rho = 0.1, missing.type = "MCAR")
tr <- 1:240  # training set indices
tst <- 241:300  # test set indices


## Perform a five-fold cross-validation on the training set.
cvfit <- cv.missoNet(X = sim.dat$X[tr, ], Y = sim.dat$Z[tr, ], kfold = 5,
                     fit.1se = TRUE, permute = TRUE, with.seed = 486)


## Make predictions of response values on the test set.
newy1 <- predict(cvfit, newx = sim.dat$X[tst, ], s = "lambda.min")
newy2 <- predict(cvfit, newx = sim.dat$X[tst, ], s = "lambda.1se.Beta")  # 'fit.1se' = TRUE
newy3 <- predict(cvfit, newx = sim.dat$X[tst, ], s = "lambda.1se.Theta")  # 'fit.1se' = TRUE
```

Index