Package ‘NVCSSL’

June 9, 2020

Type Package

Title Nonparametric Varying Coefficient Spike-and-Slab Lasso

Version 1.0

Date 2020-05-12

Author Ray Bai

Maintainer Ray Bai <raybaistat@gmail.com>

Description EM algorithm for fitting Bayesian varying coefficient models with the nonparametric varying coefficient spike-and-slab lasso of Bai et al. (2020) <arXiv:1907.06477>. Also fits penalized frequentist varying coefficient models with the group lasso, group smoothly clipped absolute deviation, and group minimax concave penalty.

License GPL-3

LazyData true

Depends R (>= 3.5.0)

Imports stats, splines, plyr, grpreg, Matrix

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-06-09 09:40:03 UTC

R topics documented:

`NVC_frequentist` *Nonparametric Varying Coefficient Model Fitting with Frequentist Penalties*

Description

This function fits frequentist sparse nonparametric varying coefficients (NVC) models for repeated measures data using basis expansions with regularization on groups of basis coefficients. NVC models aim to estimate the functions of time, $\beta_1(t), \ldots, \beta_p(t)$, in the model,

$$y_i(t_{ij}) = \sum_{k=1}^{p} x_{ik}(t_{ij})\beta_k(t_{ij}) + \varepsilon_{ij},$$

where the subscript ij refers to the jth observation for the ith subject.

Unlike the Bayesian NVC-SSL model, this function does *not* model the within-subject temporal correlations. The data does *not* need to be regularly spaced. In particular, we have n subjects with $n_i, i = 1, \ldots, n$ within-study observations each, for a total of $N = \sum_{i=1}^{n} n_i$ observations.

Usage

```
NVC_frequentist(y, t, X, df=8, penalty=c("gLasso","gSCAD","gMCP"), lambda = NULL)
```

Arguments

`y`	$N \times 1$ vector of all observed responses, $(y_{11}, \ldots, y_{1n_1}, \ldots, y_{n1}, \ldots, y_{nn_n})'$.
`t`	$N \times 1$ vector of all time points, $(t_{11}, \ldots, t_{1n_1}, \ldots, t_{n1}, \ldots, t_{nn_n})'$.
`X`	$N \times p$ design matrix of possibly time-varying covariates. The kth column of X is $(x_{1k}(t_{11}), \ldots, x_{1k}(t_{1n_1}), \ldots, x_{nk}(t_{n1}), \ldots, x_{nk}(t_{nn_n}))'$.
`df`	number of basis functions to use in each basis expansion. Default is `df=8`, but the user may specify degrees of freedom as any integer greater than or equal to 3.
`penalty`	group regularization method to use on the groups of basis coefficients. `"gLasso"` is the group lasso penalty, `"gSCAD"` is the group SCAD penalty, and `"gMCP"` is the group minimax concave penalty (MCP).
`lambda`	Optional grid for tuning the regularization parameter λ from cross-validation. If the user does not provide this, the program chooses the grid automatically.

Value

The function returns a list containing the following components:

`t.ordered`	all N time points in order from smallest to largest. Needed for plotting.
`beta.hat`	$N \times p$ matrix of the function estimates. The kth column is the function estimate $\beta_k(t_{ij})$ evaluated at the N time observations $t_{ij}, i = 1, ..., n, j = 1, ..., n_i$.
`gamma.hat`	*df* $\times p$ estimated basis coefficients γ. Needed for prediction.

`intercept` Estimate of an intercept (or grand mean). Needed for prediction.

`classifications`
$p \times 1$ vector of binary variables for the p covariates. "1" indicates that the covariate was selected, and "0" indicates that it was not selected.

`AICc` AIC with correction for small sample size. This value can be used for tuning the degrees of freedom `df`. The user should pick the model which obtains the lowest AIC_c.

References

Bai, R., Boland, M. R., and Chen, Y. (2020). "Fast algorithms and theory for high-dimensional Bayesian varying coefficient models." *arXiv preprint arXiv:1907.06477*.

Huang, J., Breheny, P., and Ma, S. (2012). "A selective review of group selection in high-dimensional models." *Statistical Science*, **27**: 481-499.

Examples

```
#####################################
# Example on the synthetic data set #
#####################################
data(SimulatedData)
attach(SimulatedData)
y = SimulatedData$y
t = SimulatedData$t
X = SimulatedData[,4:103]

## Set seed
set.seed(123)

## Fit frequentist NVC model with group MCP penalty

nvc.gMCP.mod = NVC_frequentist(y=y, t=t, X=X, penalty="gMCP")

## AIC with correction for sample size. Can use this to tune df
nvc.gMCP.mod$AICc

## The first 6 functionals were classified as nonzero
nvc.gMCP.mod$classifications

## Make plot of the 6 active functionals

oldpar <- par(no.readonly = TRUE)

t.ordered = nvc.gMCP.mod$t.ordered
beta.hat = nvc.gMCP.mod$beta.hat

par(mfrow=c(3,2), mar=c(4,4,4,4))
plot(t.ordered, beta.hat[,1], lwd=3, type='l', col='purple',
     xlab="T", ylim=c(-10,10), ylab=expression(beta[1]))
plot(t.ordered, beta.hat[,2], lwd=3, type='l', col='purple',
```

```
     xlab="T", ylim=c(-8,8), ylab=expression(beta[2]))
plot(t.ordered, beta.hat[,3], lwd=3, type='l', col='purple',
     xlab="T", ylim=c(-7,4), ylab=expression(beta[3]))
plot(t.ordered, beta.hat[,4], lwd=3, type='l', col='purple',
     xlab="T", ylim=c(-6,4), ylab=expression(beta[4]))
plot(t.ordered, beta.hat[,5], lwd=3, type='l', col='purple',
     xlab="T", ylim=c(0,15), ylab=expression(beta[5]))
plot(t.ordered, beta.hat[,6], lwd=3, type='l', col='purple',
     xlab="T", ylim=c(-5,0), ylab=expression(beta[6]))

par(oldpar)


#############################
# Example on the yeast cell #
# cycle data set            #
#############################

data(yeast)
attach(yeast)
y = SimulatedData$y
t = SimulatedData$t
id = SimulatedData$id
X = SimulatedData[,4:103]

y = yeast$mRNA
t = yeast$Time
id = yeast$Gene
X = yeast[,c(4:99)]

## Set seed
set.seed(12345)

## Fit frequentist NVC model with group SCAD penalty
nvc.gSCAD.mod = NVC_frequentist(y, t, X, penalty="gSCAD")

## TF's classified as significant
nvc.gSCAD.mod$classifications

## Plot the first TF's functional

oldpar <- par(no.readonly = TRUE)

t.ordered = nvc.gSCAD.mod$t.ordered
beta1.hat = nvc.gSCAD.mod$beta.hat[,1]

plot(t.ordered, beta1.hat, type='l', lwd=5, xlab="Time",
     ylab=expression(beta[1]), main=colnames(X)[1])

par(oldpar)
```

NVC_SSL *Nonparametric Varying Coefficient Spike-and-Slab Lasso (NVC-SSL)*

Description

This function implements the nonparametric varying coefficient spike-and-slab lasso (NVC-SSL) model of Bai et al. (2020) for repeated measures data. The NVC-SSL model aims to estimate the functions of time, $\beta_1(t), \ldots, \beta_p(t)$, in the model,

$$y_i(t_{ij}) = \sum_{k=1}^{p} x_{ik}(t_{ij})\beta_k(t_{ij}) + \varepsilon_{ij},$$

where the subscript ij refers to the jth observation for the ith subject.

The NVC-SSL model can model within-subject temporal correlations, and the data does *not* need to be regularly spaced. In particular, we have n subjects with $n_i, i = 1, \ldots, n$ within-study observations each, for a total of $N = \sum_{i=1}^{n} n_i$ observations.

Usage

```
NVC_SSL(y, t, id, X, df=8, cov.structure=c("AR1","CS","ind","unstructured"),
          lambda0=c(seq(from=5,to=30,by=5),seq(from=40,to=100,by=10)), lambda1=1,
               a=1, b=ncol(X), c0=1, d0=1, rho.support=seq(0,0.9,by=0.1),
               tol = 1e-6, print.iteration=TRUE)
```

Arguments

y	$N \times 1$ vector of all observed responses, $(y_{11}, \ldots, y_{1n_1}, \ldots, y_{n1}, \ldots, y_{nn_n})'$.
t	$N \times 1$ vector of all time points, $(t_{11}, \ldots, t_{1n_1}, \ldots, t_{n1}, \ldots, t_{nn_n})'$.
id	$N \times 1$ vector of subject identifiers (1 through n).
X	$N \times p$ design matrix of possibly time-varying covariates. The kth column of X is $(x_{1k}(t_{11}), \ldots, x_{1k}(t_{1n_1}), \ldots, x_{nk}(t_{n1}), \ldots, x_{nk}(t_{nn_n}))'$.
df	number of basis functions to use in each basis expansion. Default is `df=8`, but the user may specify degrees of freedom as any integer greater than or equal to 3.
cov.structure	within-subject covariance structure with which to fit the NVC-SSL model. The options are: `"AR1"` for first-order autoregressive, `"CS"` for compound symmetry, `"ind"` for independent errors (i.e. no within-subject temporal correlations), and `"unstructured"` for completely unstructured. The default is `"AR1"`. Note that `"unstructured"` is generally not recommended. Instead, we recommend using the `robustified_NVC_SSL` function when the covariance structure may be misspecified.
lambda0	ladder of spike hyperparameters in the spike-and-slab group lasso (SSGL) prior for dynamic posterior exploration. The user may specify either a scalar or a vector. The default is `lambda0=c(5,10,15,20,25,30,40,50,60,70,80,90,100)`.

lambda1 — slab hyperparameter in the SSGL prior, which is held fixed. The default is `lambda1=1`.

a — shape hyperparameter for the $B(a, b)$ prior on the mixing proportion θ. Default is `a=1`.

b — shape hyperparameter for the $B(a, b)$ prior on the mixing proportion θ. Default is `b=ncol(X)`.

c0 — shape hyperparameter for the $IG(c_0/2, d_0/2)$ prior on the unknown global variance σ^2. Default is `c0=1`. Ignored if the covariance structure is unstructured, i.e. `cov.structure="unstructured"`.

d0 — rate hyperparameter for the $IG(c_0/2, d_0/2)$ prior on the unknown global variance σ^2. Default is `d0=1`. Ignored if the covariance structure is unstructured, i.e. `cov.structure="unstructured"`.

rho.support — the support for the discrete uniform prior on the autocorrelation parameter ρ. Default is `rho.support=c(0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9)`, but the user may specify a finer grid (as long as the atoms are greater than or equal to 0 and strictly less than 1). This is ignored if `cov.structure="ind"` or `cov.structure="unstructured"`.

tol — convergence threshold for the EM algorithm. Default is `tol=1e-6`.

print.iteration — flag for printing the current value of `lambda0` in the ladder of spike hyperparameters. Default is `TRUE`.

Value

The function returns a list containing the following components:

t.ordered — all N time points in order from smallest to largest. Needed for plotting.

beta.hat — $N \times p$ matrix of the function estimates. The kth column is the function estimate $\beta_k(t_{ij})$ evaluated at the N time observations $t_{ij}, i = 1, ..., n, j = 1, ..., n_i$.

gamma.hat — $df \times p$ estimated basis coefficients γ. Needed for prediction.

intercept — Estimate of an intercept (or grand mean). Needed for prediction.

classifications — $p \times 1$ vector of binary variables for the p covariates. "1" indicates that the covariate was selected, and "0" indicates that it was not selected.

AICc — AIC with correction for small sample size. This value can be used for tuning the degrees of freedom `df`. The user should pick the model which obtains the lowest AIC_c.

References

Bai, R., Boland, M. R., and Chen, Y. (2020). "Fast algorithms and theory for high-dimensional Bayesian varying coefficient models." *arXiv preprint arXiv:1907.06477*.

Examples

```
######################################
# Example on the synthetic data set #
######################################
data(SimulatedData)
attach(SimulatedData)
y = SimulatedData$y
t = SimulatedData$t
id = SimulatedData$id
X = SimulatedData[,4:103]

## Training set. First 20 subjects
training.indices = which(id <= 20)
y.train = y[training.indices]
t.train = t[training.indices]
id.train = id[training.indices]
X.train = X[training.indices,]

## Test set. Last 10 subjects
test.indices = which(id > 20)
y.test = y[test.indices]
t.test = t[test.indices]
id.test = id[test.indices]
X.test = X[test.indices, ]

## Set seed
set.seed(123)

###############################
## NVC-SSL with fixed lambda0 #
###############################
# We use with 6 degrees of freedom and FIXED lambda0=15,
# so we are estimating 600 total unknown basis coefficients

nvcssl.mod = NVC_SSL(y=y.train, t=t.train, id=id.train, X=X.train,
                     df=6, cov.structure="AR1", lambda0=15)

## The first 6 functionals were classified as nonzero
nvcssl.mod$classifications

## AIC with correction for sample size. Can use this to tune df
nvcssl.mod$AICc

## Make plot of the 6 active functionals

oldpar <- par(no.readonly = TRUE)

t.ordered = nvcssl.mod$t.ordered
beta.hat = nvcssl.mod$beta.hat

par(mfrow=c(3,2), mar=c(4,4,4,4))
```

```
plot(t.ordered, beta.hat[,1], lwd=3, type='l', col='blue',
     xlab="T", ylim=c(-10,10), ylab=expression(beta[1]))
plot(t.ordered, beta.hat[,2], lwd=3, type='l', col='blue',
     xlab="T", ylim=c(-8,8), ylab=expression(beta[2]))
plot(t.ordered, beta.hat[,3], lwd=3, type='l', col='blue',
     xlab="T", ylim=c(-7,4), ylab=expression(beta[3]))
plot(t.ordered, beta.hat[,4], lwd=3, type='l', col='blue',
     xlab="T", ylim=c(-6,4), ylab=expression(beta[4]))
plot(t.ordered, beta.hat[,5], lwd=3, type='l', col='blue',
     xlab="T", ylim=c(0,15), ylab=expression(beta[5]))
plot(t.ordered, beta.hat[,6], lwd=3, type='l', col='blue',
     xlab="T", ylim=c(-5,0), ylab=expression(beta[6]))

par(oldpar)

## Prediction on test set
y.preds = predict_NVC(nvcssl.mod, t.new=t.test, id.new=id.test, X.new=X.test)

## MSPE
nvcssl.mod.mspe = mean((y.test-y.preds[,2])^2)
nvcssl.mod.mspe



##############################
# NVC-SSL model with dynamic #
# posterior exploration      #
##############################

## Now we fit the NVC-SSL mod with default 8 degrees of freedom
## (800 unknown basis coefficients), AR(1) covariance structure,
## and dynamic posterior exploration for spike hyperparameter lambda0.

## Dynamic posterior exploration will generally produce a better fit
## than arbitrarily fixing lambda0.

## Note if the standard deviation of y is very small, the algorithm
## may be initially slow for values of lambda0 that are too small.

## In this case, consider increasing the lambda0 ladder to begin
## at a larger value.

data(SimulatedData)
attach(SimulatedData)
y = SimulatedData$y
t = SimulatedData$t
id = SimulatedData$id
X = SimulatedData[,4:103]

## Fit NVC-SSL model with dynamic posterior exploration
nvcssl.mod = NVC_SSL(y=y, t=t, id=id, X=X, cov.structure="AR1")

## Look at which covariates were selected
```

```
nvcssl.mod$classifications
```

---

predict_NVC *Prediction with Nonparametric Varying Coefficient Models*

Description

This function takes a model fit by a varying coefficient model,

$$y_i(t_{ij}) = \sum_{k=1}^{p} x_{ik}(t_{ij})\beta_k(t_{ij}) + \varepsilon_{ij},$$

and predicts the values $\hat{y}(t_{new})$ on a new data set. This function can be used with NVC models fit using the `NVC_SSL`, `robustified_NVC_SSL`, and `NVC_frequentist` functions.

Usage

```
predict_NVC(NVC.mod, t.new, id.new, X.new)
```

Arguments

`NVC.mod`	a model output by `NVC_SSL`, `robustified_NVC_SSL`, or `NVC_frequentist`.
`t.new`	a vector of new time points.
`id.new`	a vector of identifiers for the new subjects.
`X.new`	a design matrix for the new observations.

Value

The function returns a matrix with two columns. The first column `id` contains the identifiers for the new subjects. The second column `y.hat` contains the predictions $\hat{y}$ for all the new observations.

References

Bai, R., Boland, M. R., and Chen, Y. (2020). "Fast algorithms and theory for high-dimensional Bayesian varying coefficient models." *arXiv preprint arXiv:1907.06477*.

Examples

```
#####################################
# Example on the synthetic data set #
#####################################
data(SimulatedData)
attach(SimulatedData)
y = SimulatedData$y
t = SimulatedData$t
```

```
id = SimulatedData$id
X = SimulatedData[,4:103]

## Training set. First 20 subjects
training.indices = which(id <= 20)
y.train = y[training.indices]
t.train = t[training.indices]
id.train = id[training.indices]
X.train = X[training.indices,]

## Test set. Last 10 subjects
test.indices = which(id > 20)
y.test = y[test.indices]
t.test = t[test.indices]
id.test = id[test.indices]
X.test = X[test.indices, ]

## Set seed
set.seed(123)

## Fit NVC model with group lasso penalty
nvc.gLasso.mod = NVC_frequentist(y=y.train, t=t.train, X=X.train, penalty="gLasso")

## Prediction on test set
y.preds = predict_NVC(nvc.gLasso.mod, t.new=t.test, id.new=id.test, X.new=X.test)

## MSPE
nvc.gLasso.mspe = mean((y.test-y.preds[,2])^2)
nvc.gLasso.mspe
```

`robustified_NVC_SSL` *Robustified Nonparametric Varying Coefficient Spike-and-Slab Lasso (robustified NVC-SSL)*

Description

This function implements the robustified NVC-SSL model of Bai et al. (2020) for repeated measures data. The robustified NVC-SSL model aims to estimate the functions of time, $\beta_1(t), \ldots, \beta_p(t)$ in the model,

$$y_i(t_{ij}) = \sum_{k=1}^{p} x_{ik}(t_{ij})\beta_k(t_{ij}) + \varepsilon_{ij},$$

where the subscript ij refers to the jth observation for the ith subject.

The robustified NVC-SSL model can be used to consistently estimate the functions $\beta_k(t), k = 1, \ldots, p$, when the within-subject covariance matrix is possibly misspecified. The data does *not* need to be regularly spaced. In particular, we have n subjects with $n_i, i = 1, \ldots, n$ within-study observations each, for a total of $N = \sum_{i=1}^{n} n_i$ observations.

Usage

```
robustified_NVC_SSL(y, t, id, X, df=8,
                              working.cov=c("AR1","CS","ind"), frac.power=0.5,
                        lambda0=c(seq(from=5,to=30,by=5),seq(from=40,to=100,by=10)),
                        lambda1=1, a=1, b=ncol(X), tol = 1e-6, print.iteration = TRUE)
```

Arguments

y	$N \times 1$ vector of all observed responses, $(y_{11}, \ldots, y_{1n_1}, \ldots, y_{n1}, \ldots, y_{nn_n})'$.
t	$N \times 1$ vector of all time points, $(t_{11}, \ldots, t_{1n_1}, \ldots, t_{n1}, \ldots, t_{nn_n})'$.
id	$N \times 1$ vector of subject identifiers (1 through n).
X	$N \times p$ design matrix of possibly time-varying covariates. The kth column of X is $(x_{1k}(t_{11}), \ldots, x_{1k}(t_{1n_1}), \ldots, x_{nk}(t_{n1}), \ldots, x_{nk}(t_{nn_n}))'$.
df	number of basis functions to use in each basis expansion. Default is `df=8`, but the user may specify degrees of freedom as any integer greater than or equal to 3.
working.cov	working within-subject covariance structure with which to fit the robustified NVC-SSL model. The options are: `"AR1"` for first-order autoregressive, `"CS"` for compound symmetry, and `"ind"` for independent errors (i.e. no within-subject temporal correlations). The default is `"AR1"`.
lambda0	ladder of spike hyperparameters in the spike-and-slab group lasso (SSGL) prior for dynamic posterior exploration. The user may specify either a scalar or a vector. The default is `lambda0=c(5,10,15,20,25,30,40,50,60,70,80,90,100)`.
lambda1	slab hyperparameter in the SSGL prior, which is held fixed. The default is `lambda1=1`.
a	shape hyperparameter for the $B(a, b)$ prior on the mixing proportion θ. Default is `a=1`.
b	shape hyperparameter for the $B(a, b)$ prior on the mixing proportion θ. Default is `b=ncol(X)`.
frac.power	The fractional power $\xi \in (0, 1)$ to use in order to fit the robustified NVC-SSL model. `frac.power` must be strictly between 0 and 1.
tol	convergence threshold for the EM algorithm. Default is `tol=1e-6`.
print.iteration	flag for printing the current value of `lambda0` in the ladder of spike hyperparameters. Default is `TRUE`.

Value

The function returns a list containing the following components:

t.ordered	all N time points in order from smallest to largest. Needed for plotting.
beta.hat	$N \times p$ matrix of the function estimates. The kth column is the function estimate $\beta_k(t_{ij})$ evaluated at the N time observations $t_{ij}, i = 1, ..., n, j = 1, ..., n_i$.
gamma.hat	*df* $\times p$ estimated basis coefficients γ. Needed for prediction.

`intercept` Estimate of an intercept (or grand mean). Needed for prediction.

`classifications`
$p \times 1$ vector of binary variables for the p covariates. "1" indicates that the covariate was selected, and "0" indicates that it was not selected.

`AICc` AIC with correction for small sample size. This value can be used for tuning either the fractional power `frac.power` or the degrees of freedom `df` or both. The user should pick the model which obtains the lowest AIC_c.

References

Bai, R., Boland, M. R., and Chen, Y. (2020). "Fast algorithms and theory for high-dimensional Bayesian varying coefficient models." *arXiv preprint arXiv:1907.06477*.

Examples

```
#####################################
# Example on the synthetic data set #
#####################################
data(SimulatedData)
attach(SimulatedData)
y = SimulatedData$y
t = SimulatedData$t
id = SimulatedData$id
X = SimulatedData[,4:103]

## Training set. First 20 subjects
training.indices = which(id <= 20)
y.train = y[training.indices]
t.train = t[training.indices]
id.train = id[training.indices]
X.train = X[training.indices,]

## Test set. Last 10 subjects
test.indices = which(id > 20)
y.test = y[test.indices]
t.test = t[test.indices]
id.test = id[test.indices]
X.test = X[test.indices, ]

## Set seed
set.seed(123)

## Fit robustified NVC-SSL model with 8 degrees of freedom
## (so we are estimating 800 unknown basis coefficients),
## working AR(1) covariance structure, and fractional power 0.6

robust.nvcssl.mod = robustified_NVC_SSL(y=y.train, t=t.train,
                                       id=id.train, X=X.train, df=8,
                                       working.cov="AR1",
                                       frac.power=0.6)
```

```
## AIC with correction for sample size. Can use to tune df or frac.power
robust.nvcssl.mod$AICc

## The first 6 functionals were classified as nonzero
robust.nvcssl.mod$classifications

## Make plot of the 6 active functionals

oldpar <- par(no.readonly = TRUE)

t.ordered = robust.nvcssl.mod$t.ordered
beta.hat = robust.nvcssl.mod$beta.hat

par(mfrow=c(3,2), mar=c(4,4,4,4))
plot(t.ordered, beta.hat[,1], lwd=3, type='l', col='red',
     xlab="T", ylim=c(-10,10), ylab=expression(beta[1]))
plot(t.ordered, beta.hat[,2], lwd=3, type='l', col='red',
     xlab="T", ylim=c(-8,8), ylab=expression(beta[2]))
plot(t.ordered, beta.hat[,3], lwd=3, type='l', col='red',
     xlab="T", ylim=c(-7,4), ylab=expression(beta[3]))
plot(t.ordered, beta.hat[,4], lwd=3, type='l', col='red',
     xlab="T", ylim=c(-6,4), ylab=expression(beta[4]))
plot(t.ordered, beta.hat[,5], lwd=3, type='l', col='red',
     xlab="T", ylim=c(0,15), ylab=expression(beta[5]))
plot(t.ordered, beta.hat[,6], lwd=3, type='l', col='red',
     xlab="T", ylim=c(-5,0), ylab=expression(beta[6]))

par(oldpar)

## Prediction on the test set
y.preds = predict_NVC(robust.nvcssl.mod, t.new=t.test, id.new=id.test, X.new=X.test)

## MSPE for test set
robust.nvcssl.mod.mspe = mean((y.test-y.preds[,2])^2)
robust.nvcssl.mod.mspe
```

`SimulatedData` *Simulated data with irregularly spaced time points and different number of within-subject observations.*

Description

This data set contains simulated repeated measurements data on $n = 30$ subjects, each with a different number of time points and irregularly spaced time points. There are a total of $N = 238$ observations and $p = 100$ time-varying covariates.

Usage

```
data(SimulatedData)
```

Format

A dataframe with the following columns.

`y`: $N = 238$ responses for all $n = 30$ subjects.

`t`: $N = 238$ time points for all $n = 30$ subjects.

`id`: identifier for subject (1-30).

`X1-X100`: columns 4-103 are the 100 covariates for all N observations.

References

Bai, R., Boland, M. R., and Chen, Y. (2020). "Fast algorithms and theory for high-dimensional Bayesian varying coefficient models." *arXiv preprint arXiv:1907.06477*.

`yeast` *Yeast cell cycle data set*

Description

This data set contains the yeast cell cycle data that is analyzed in Bai et al. (2020), which is a subset of the original yeast data analyzed by Spellman et al. (1998). There are $n = 47$ yeast genes that were determined to periodically expressed by De Lichtenberg et al. (2005). For each of these genes, mRNA levels were measured at $n_i = 18$ time points from time 0 to 119 minutes (7 minutes apart), for a total of $N = 846$ data points. For covariates, we have the binding information for 96 transcription factors (TFs) taken from the chromatin immunoprecipitation data set of Lee et al. (2002).

Usage

```
data(yeast)
```

Format

A dataframe with the following columns.

`mRNA`: mRNA levels for 47 yeast genes at 18 different time points.

`Time`: time points from 0-119 minutes.

`Gene`: gene identifiers (1-47).

`ABF1-ZMS1`: Columns 4-99 contain the binding information on 96 TFs that are possibly associated with the yeast genes.

References

Bai, R., Boland, M. R., and Chen, Y. (2020). "Fast algorithms and theory for high-dimensional Bayesian varying coefficient models." *arXiv preprint arXiv:1907.06477*.

De Lichtenberg, U., Jensen, L. J., Fausboll, A., Jensen T. S., Bork, P., and Brunak, S. (2005). "Comparison of computational methods for the identification of cell cycle-regulated genes." *Bioinformatics*, **21**: 1164-1171.

Lee, T.I., Rinaldi, N.J., Robert, F., Odom, D.T., Bar-Joseph, Z., Gerber, G.K., Hannett, N.M., Harbison, C.T., Thomson, C.M., Simon, I., Zeitlinger, J., Jennings, E.G., Murray, H.L., Gordon, D.B., Ren, B., Wyrick, J.J., Tagne, J.B., Volkert, T.L, Fraenkel, E, Gifford, D.K., and Young, R.A (2002). "Transcriptional regulatory networks in Saccharomyces cerevisiae." *Science*, **298**: 799-804.

Spellman, P.T., Sherlock, G., Zhang, M.Q., Iyer, V.R., Anders, K., Eisen, M.B., Brown, P.O., Botstein, D., and Futcher, B. (1998). "Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hydrization. *Molecular Biology of the Cell*, **9**: 3273-3279.

Index