Package: BayesSIM (via r-universe)

October 29, 2026

Type Package

Title Integrated Interface of Bayesian Single Index Models using 'nimble'

Version 2.0.1

Author Seowoo Jung [aut, cre], Eun-kyung Lee [aut]

Maintainer Seowoo Jung <jsw1347@ewha.ac.kr>

Description Provides tools for fitting Bayesian single index models with flexible choices of priors for both the index and the link function. The package implements model estimation and posterior inference using efficient MCMC algorithms built on the 'nimble' framework, allowing users to specify, extend, and simulate models in a unified and reproducible manner. The following methods are implemented in the package: Antoniadis et al. (2004) <https://www.jstor.org/stable/24307224>, Wang (2009) <doi:10.1016/j.csda.2008.12.010>, Choi et al. (2011) <doi:10.1080/10485251003768019>, Dhara et al. (2019) <doi:10.1214/19-BA1170>, McGee et al. (2023) <doi:10.1111/biom.13569>.

License GPL (>= 2)

Depends R (>= 4.1), nimble

Imports posterior, magrittr, MASS, methods, mvtnorm, patchwork, tidyr, ggplot2, dplyr, loo, coda

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

NeedsCompilation no

Config/roxygen2/version 8.1.1

URL https://github.com/Seowoo-Jung/BayesSIM

BugReports https://github.com/Seowoo-Jung/BayesSIM/issues

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

Date/Publication 2026-10-29 03:40:02 UTC

RemoteUrl https://github.com/cran/BayesSIM

RemoteRef HEAD

RemoteSha 95cbf8f463b18c76aab10294629bb7f8524b4c04

Contents

as.data.frame.bsimPred . . . 3
as_bsim . . . 4
BayesSIM . . . 5
bsFisher . . . 8
bsPolar . . . 12
bsSphere . . . 16
bsSpike . . . 20
coef.bsim . . . 24
compileModelAndMCMC . . . 25
concrete . . . 27
DATA1 . . . 28
fitted.bsim . . . 28
getFunction . . . 30
getInit . . . 31
getModelDef . . . 32
getVarMonitor . . . 33
getWAIC . . . 34
GOF . . . 36
gpFisher . . . 37
gpPolar . . . 41
gpSphere . . . 46
gpSpike . . . 50
init_param . . . 53
nimTraceplot . . . 55
plot.bsim . . . 57
plot.bsimSetup . . . 58
predict.bsim . . . 59
prior_param . . . 62
residuals.bsim . . . 65
summary.bsim . . . 66
summary.bsimPred . . . 68
summary.bsimSetup . . . 69

Index **71**

`as.data.frame.bsimPred` *Extract the Result Data.frame*

Description

Extracts fitted result that contains posterior mean/median prediction of response variable, and standard error, credible interval of each data point in data.frame.

Usage

```
## S3 method for class 'bsimPred'
as.data.frame(x, ...)
```

Arguments

`x`	An object of class "bsimPred" which is the result of `predict.bsim`.
`...`	Additional arguments passed to other methods.

Value

data.frame of prediction information.

See Also

`predict.bsim`

Examples

```
data("DATA1")
simdata2 <- data.frame(DATA1$X, y = DATA1$y)
fit_one  <- BayesSIM(y ~ ., data = simdata2,
                     niter = 10000, nburnin = 5000, nchain = 2)

pred <- predict(fit_one, error.fit = TRUE, interval = "credible",level = 0.8)
results <- as.data.frame(pred)
summary(results)
```

as_bsim *Construct a Fitted Model Object from Model Setup and MCMC Output*

Description

Create a fitted bsim object by combining a BayesSIM setup object with MCMC samples returned by runMCMC().

Usage

```
as_bsim(object, mcmc.out, ...)

## S3 method for class 'bsimSetup'
as_bsim(object, mcmc.out, Ccompile, ...)
```

Arguments

object	A BayesSIM setup object, typically the output of a _setup function.
mcmc.out	MCMC output corresponding to the result of a call to runMCMC().
...	Additional arguments passed to other methods.
Ccompile	Result of compileModelAndMCMC() which includes compiled object of the model and the sampler.

Details

This function is mainly intended for workflows where the model structure and the MCMC sampling are performed separately. It collects the MCMC draws across chains, and returns an object of class "bsim" that can be used with generic functions such as summary(), plot(), and predict().

Value

An object of class "bsim" containing posterior samples, point estimates, fitted values, and related model information.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)
fit2 <- as_bsim(models, mcmc.out, Ccompile)
summary(fit2)
```

`BayesSIM` *Integrated Function for Bayesian Single-Index Regression*

Description

Fitting a single–index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$ in single integrated function.

Usage

```
BayesSIM(
  formula,
  data,
  indexprior = "fisher",
  link = "bspline",
  prior = NULL,
  init = NULL,
  method = "FB",
  lowerB = NULL,
  upperB = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

BayesSIM_setup(
  formula,
  data,
  indexprior = "fisher",
  link = "bspline",
  prior = NULL,
  init = NULL,
  method = "FB",
  lowerB = NULL,
  upperB = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
```

```
)

## S3 method for class 'bsim'
print(x, digits = 3, ...)

## S3 method for class 'bsimSetup'
print(x, digits = 3, ...)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
indexprior	Index vector prior among "fisher" (default), "sphere", "polar", "spike".
link	Link function among "bspline" (default), "gp"
prior	Optional named list of prior settings. Further descriptions are in every specific model's **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in every specific model's **Details** section.
method	Character, gpSphere model has 3 different types of sampling method, fully Bayesian method ("FB"), empirical Bayes approach ("EB"), and empirical Gibbs sampler ("EG"). Assign one sampler method. Empirical sampling approach is recommended in high-dimensional data. By default, fully Bayesian approach is assigned.
lowerB	This parameter is only for gpSphere model. Numeric vector of element-wise lower bounds for the "L-BFGS-B" method. When the empirical Bayes or Gibbs sampler method is used, the marginal likelihood is optimized via optim(method = "L-BFGS-B"). The vector must be ordered as c(index vector, lengthscale, amp, sigma2). Note that sigma2 is included only for the empirical Bayes method (omit it for Gibbs). By default, the lower bounds are -1 for the index vector and -1e2 for logarithm of lengthscale, amp, and (if present) sigma2.
upperB	This parameter is only for gpSphere model. Numeric vector of element-wise upper bounds for the "L-BFGS-B" method. When the empirical Bayes or Gibbs sampler method is used, the marginal likelihood is optimized via optim(method = "L-BFGS-B"). The vector must be ordered as c(index vector, lengthscale, amp, sigma2). Note that sigma2 is included only for the empirical Bayes method (omit it for Gibbs). By default, the upper bounds are 1 for the index vector and 1e2 for logarithm of lengthscale, amp, and (if present) sigma2.
monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the model_setup function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default 10000).
nburnin	Integer. Burn-in iterations (default 1000).
thin	Integer. Thinning for monitors (default 1).

`nchain`	Integer. Number of MCMC chains (default `1`).
`setSeed`	Logical or numeric argument. Further details are provided in runMCMC `setSeed` argument.
`scale`	Logical. If `TRUE`, columns of predictors are divided by their standard deviation before fitting. Default `FALSE`.
`center`	Logical. If `TRUE`, columns of predictors are mean-centered before fitting. Default `FALSE`.
`x`	A fitted `BayesSIM` object.
`digits`	Number of digits to display.
`...`	Additional arguments.

Details

Integrated function for Bayesian single-index model. Default model is von-Mises Fisher distribution for index vector with B-spline link function.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

See Also

`bsFisher()`, `bsSphere()`, `bsPolar()`, `bsSpike()`, `gpFisher()`, `gpSphere()`, `gpPolar()`, `gpPolarHigh()`, `gpSpike()`

Examples

```
set.seed(123)
n <- 200; d <- 4
theta <- c(2, 1, 1, 1); theta <- theta / sqrt(sum(theta^2))
f <- function(u) u^2 * exp(u)
sigma <- 0.5
X <- matrix(runif(n * d, -1, 1), nrow = n)
index_vals <- as.vector(X %*% theta)
y <- f(index_vals) + rnorm(n, 0, sigma)
simdata <- data.frame(x = X, y = y)
colnames(simdata) <- c(paste0("X", 1:4), "y")

# One tool version - bsFisher
fit1 <- BayesSIM(y ~ ., data = simdata,
                 niter = 10000, nburnin = 5000,
                 nchain = 2)

# Split version- bsFisher
models <- BayesSIM_setup(y ~ ., data = simdata, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)
fit2 <- as_bsim(models, mcmc.out, Ccompile)
summary(fit2)
```

`bsFisher` *Bayesian Single-Index Regression with B-Spline Link and von Mises-Fisher Prior*

Description

Fits a single-index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$ where the link $f(\cdot)$ is represented by B-spline and the index vector θ has von Mises–Fisher prior.

Usage

```
bsFisher(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
```

```
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

bsFisher_setup(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
prior	Optional named list of prior settings. Further descriptions are in **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in **Details** section.
monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the `model_setup` function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default `10000`).
nburnin	Integer. Burn-in iterations (default `1000`).
thin	Integer. Thinning for monitors (default `1`).
nchain	Integer. Number of MCMC chains (default `1`).
setSeed	Logical or numeric argument. Further details are provided in runMCMC `setSeed` argument.
scale	Logical. If `TRUE`, columns of predictors are divided by their standard deviation before fitting. Default `FALSE`.
center	Logical. If `TRUE`, columns of predictors are mean-centered before fitting. Default `FALSE`.

Details

Model The single–index model uses a m-order polynomial spline with k interior knots as follows:

$$f(t) = \sum_{j=1}^{m+k} B_j(t)\,\beta_j$$

on $[a, b]$ with $t_i = X_i'\theta, i = 1, \cdots, n$ and $\|\theta\|_2 = 1$. $\{\beta_j\}_{j=1}^{m+k}$ are spline coefficients and a_θ, b_θ are boundary knots where $a_\theta = min(t_i, i = 1, \cdots, n) - \delta$, and $b_\theta = max(t_i, i = 1, \cdots, n) + \delta$.

Priors

- von Mises–Fisher prior on the index θ with direction and concentration.
- Conditioned on θ and σ^2, the link coefficients $\beta = (\beta_1, \ldots, \beta_{m+k})^\top$ follow normal distribution with estimated mean vector $\hat{\beta}_\theta = (X_\theta' X_\theta + \rho \boldsymbol{I})^{-1} X_\theta' Y$ and covariance $\sigma^2 (X_\theta^\top X_\theta)^{-1}$, where X_θ is the B-spline basis design matrix.
- Inverse gamma prior on σ^2 with shape parameter a_σ and rate parameter b_σ.

Sampling Random walk metropolis algorithm is used for index vector θ. Given θ, σ^2 is sampled from posterior distribution. Given θ, σ^2, β is estimated using the penalized least squares estimator rather than sampled from its posterior distribution. Further sampling method is described in Antoniadis et al. (2004).

Prior hyperparameters These are the prior hyperparameters set in the function. You can define new values for each parameter in prior_param.

1. Index vector: von Mises–Fisher prior for the projection vector θ. `index_direction` is a unit direction vector of the von Mises–Fisher distribution. By default, the estimated vector from projection pursuit regression is assigned. `index_dispersion` is the positive concentration parameter (default `150`).
2. Link function: B-spline basis and coefficient of B-spline setup.
   - `basis`: For the basis of B-spline, `link_basis_df` is the number of basis functions (default `21`), `link_basis_degree` is the spline degree (default `2`) and `link_basis_delta` is a small jitter for boundary knots spacing control (default `0.001`).
   - `beta`: For the B-spline coefficients, a multivariate normal distribution with a zero mean vector and identity covariance matrix is specified in NIMBLE as a reference prior. The actual sampling, however, is performed using the prior specified above, as described in the paper.
3. Error variance: An Inverse gamma prior is assigned to σ^2 where `sigma2_shape` is shape parameter and `sigma2_rate` is rate parameter of inverse gamma distribution. (default `sigma2_shape = 0.001, sigma2_rate = 100`)

Initial values These are the initial values set in the function. You can define new values for each initial value in init_param.

1. Index vector (`index`): Initial unit index vector θ. By default, the vector is randomly sampled from the von Mises–Fisher prior.
2. Link function: Initial spline coefficients (`link_beta`). By default, $\left(X_\theta^\top X_\theta + \rho \mathrm{I}\right)^{-1} X_\theta^\top Y$ is computed, where X_θ is the B-spline basis design matrix.

3. Error variance (`sigma2`): Initial scalar error variance (default `0.01`).

Scaling the predictors. When the scale of predictors are different, the estimated index can be dominated by large scale variables and make the estimation unstable. In this case, scaling is recommended. If `scale = TRUE, center = TRUE`, it means standardization of the predictors.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

References

Antoniadis, A., Grégoire, G., & McKeague, I. W. (2004). Bayesian estimation in single-index models. *Statistica Sinica*, 1147-1164.

Hornik, K., & Grün, B. (2014). movMF: an R package for fitting mixtures of von Mises-Fisher distributions. *Journal of Statistical Software*, 58, 1-31.

Examples

```
set.seed(123)
n <- 200; d <- 4
theta <- c(2, 1, 1, 1); theta <- theta / sqrt(sum(theta^2))
f <- function(u) u^2 * exp(u)
sigma <- 0.5
X <- matrix(runif(n * d, -1, 1), nrow = n)
index_vals <- as.vector(X %*% theta)
y <- f(index_vals) + rnorm(n, 0, sigma)
simdata <- data.frame(x = X, y = y)
colnames(simdata) <- c(paste0("X", 1:4), "y")

# One tool version
```

```
fit1 <- bsFisher(y ~ ., data = simdata,
                 niter = 10000, nburnin = 5000, nchain = 3)

# Split version
models <- bsFisher_setup(y ~ ., data = simdata, nchain = 3)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 3, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)
fit2 <- as_bsim(models, mcmc.out, Ccompile)
summary(fit2)
```

---

bsPolar *Bayesian Single-Index Regression with B-Spline Link and One-to-One Polar Transformation*

Description

Fits a single-index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$ where the link $f(\cdot)$ is represented by B-spline link function and the index vector θ is parameterized on the unit sphere via a one-to-one polar transformation.

Usage

```
bsPolar(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

bsPolar_setup(
  formula,
  data,
```

```
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
prior	Optional named list of prior settings. Further descriptions are in **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in **Details** section.
monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the model_setup function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default 10000).
nburnin	Integer. Burn-in iterations (default 1000).
thin	Integer. Thinning for monitors (default 1).
nchain	Integer. Number of MCMC chains (default 1).
setSeed	Logical or numeric argument. Further details are provided in runMCMC setSeed argument.
scale	Logical. If TRUE, columns of predictors are divided by their standard deviation before fitting. Default FALSE.
center	Logical. If TRUE, columns of predictors are mean-centered before fitting. Default FALSE.

Details

Model The single–index model uses a m-order polynomial spline with k interior knots as follows:

$$f(t) = \sum_{j=1}^{m+k} B_j(t)\,\beta_j$$

on $[a, b]$ with $t_i = X_i'\theta, i = 1, \cdots, n$ and $\|\theta\|_2 = 1$. $\{\beta_j\}_{j=1}^{m+k}$ are spline coefficient and a_θ, b_θ are boundary knots where $a_\theta = min(t_i, i = 1, \cdots, n) - \delta$, and $b_\theta = max(t_i, i = 1, \cdots, n) + \delta$. θ lies on the unit sphere ($\|\theta\|_2 = 1$) to ensure identifiability and is parameterized via a one-to-one

polar transformation with angle $\psi_1, ..., \psi_{p-1}$, where p is the dimension of predictor. Sampling is performed on the angular parameters ψ defining the index vector.

The mapping is

$$\begin{aligned}\theta_1 &= \sin(\psi_1),\\ \theta_i &= \Big(\prod_{j=1}^{i-1}\cos(\psi_j)\Big)\sin(\psi_i), \quad i = 2, \ldots, p-1,\\ \theta_p &= \prod_{j=1}^{p-1}\cos(\psi_j).\end{aligned}$$

The vector is then scaled to unit length.

Priors

- ψ is $p-1$ dimension of polar angle of index vector and re-scaled Beta distribution on $[0, \pi]$ is assigned.
- Conditioned on θ and σ^2, the link coefficients $\beta = (\beta_1, \ldots, \beta_{m+k})^\top$ follow multivariate normal distribution with mean $\boldsymbol{\mu_\beta}$, and covariance $\sigma^2\tau\Sigma_0$.
- Inverse gamma prior on σ^2 with shape parameter a_σ and rate parameter b_σ.

Sampling All samplers are automatically assigned by nimble.

Prior hyperparameters These are the prior hyperparameters set in the function. You can define new values for each parameter in prior_param.

1. Index vector: Only shape parameter `index_psi_alpha` of $p-1$ dimension vector is needed since rate parameters is computed to satisfy $\theta_{j,\text{MAP}}$. By default, the shape parameter for each element of the polar vector is set to `5000`.
2. Link function: B-spline basis and coefficient of B-spline setup.
   - `basis`: For the basis of B-spline, `link_basis_df` is the number of basis functions (default `21`), `link_basis_degree` is the spline degree (default `2`) and `link_basis_delta` is a small jitter for boundary-knot spacing control (default `0.001`).
   - `beta`: For the coefficient of B-spline, multivariate normal prior is assigned with mean ($\boldsymbol{\mu_\beta}$) `link_beta_mu`, covariance (Σ_0) `link_beta_Sigma0`, scale parameter (τ) `link_beta_tau`. By default, `link_beta_mu` is zero vector, `link_beta_Sigma0` is identity matrix, and `link_beta_tau` is the number of observations.
3. Error variance: An Inverse gamma prior is assigned to σ^2 where `sigma2_shape` is shape parameter and `sigma2_rate` is rate parameter of inverse gamma distribution. (default `sigma2_shape = 0.001, sigma2_rate = 100`)

Initial values These are the initial values set in the function. You can define new values for each initial value in init_param.

1. Index vector: Initial vector of polar angle `index_psi` ($p-1$ dimension). Each element of angle is between 0 and π. By default, it is randomly draw from uniform distribution.
2. Link function: Initial spline coefficients(`link_beta`). By default, $\left(X_\theta^\top X_\theta + \rho\, \mathrm{I}\right)^{-1} X_\theta^\top Y$ is computed, where X_θ is the B-spline basis design matrix.
3. Error variance (`sigma2`): Initial scalar error variance (default `0.01`).

Scaling the predictors. When the scale of predictors are different, the estimated index can be dominated by large scale variables and make the estimation unstable. In this case, scaling is recommended. If `scale = TRUE, center = TRUE`, it means standardization of the predictors.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

References

Antoniadis, A., Grégoire, G., & McKeague, I. W. (2004). Bayesian estimation in single-index models. *Statistica Sinica*, 1147-1164.

Hornik, K., & Grün, B. (2014). movMF: an R package for fitting mixtures of von Mises-Fisher distributions. *Journal of Statistical Software*, 58, 1-31.

Dhara, K., Lipsitz, S., Pati, D., & Sinha, D. (2019). A new Bayesian single index model with or without covariates missing at random. *Bayesian analysis*, 15(3), 759.

Examples

```
set.seed(123)
n <- 200; d <- 4
theta <- c(2, 1, 1, 1); theta <- theta / sqrt(sum(theta^2))
f <- function(u) u^2 * exp(u)
sigma <- 0.5
X <- matrix(runif(n * d, -1, 1), nrow = n)
index_vals <- as.vector(X %*% theta)
y <- f(index_vals) + rnorm(n, 0, sigma)
simdata <- data.frame(x = X, y = y)
colnames(simdata) <- c(paste0("X", 1:4), "y")

# One tool version
```

```
fit1 <- bsPolar(y ~ ., data = simdata,
                niter = 10000, nburnin = 5000, nchain = 2)

# Split version
models <- bsPolar_setup(y ~ ., data = simdata, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    samplesAsCodaMCMC = TRUE)
fit2 <- as_bsim(models, mcmc.out, Ccompile)
summary(fit2)
```

---

bsSphere *Bayesian Single-Index Regression with B-Spline Link and Uniform Prior on Half-Unit Sphere*

Description

Fits a single-index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$ where the link $f(\cdot)$ is represented by B-spline link and the index vector θ is on half-unit sphere.

Usage

```
bsSphere(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

bsSphere_setup(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
```

```
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
prior	Optional named list of prior settings. Further descriptions are in **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in **Details** section.
monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the model_setup function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default 10000).
nburnin	Integer. Burn-in iterations (default 1000).
thin	Integer. Thinning for monitors (default 1).
nchain	Integer. Number of MCMC chains (default 1).
setSeed	Logical or numeric argument. Further details are provided in runMCMC setSeed argument.
scale	Logical. If TRUE, columns of predictors are divided by their standard deviation before fitting. Default FALSE.
center	Logical. If TRUE, columns of predictors are mean-centered before fitting. Default FALSE.

Details

Model The single–index model uses a m-order polynomial spline with k interior knots and intercept as follows:

$$f(t) = \sum_{j=1}^{1+m+k} B_j(t)\,\beta_j$$

on $[a, b]$ with $t_i = X_i'\theta, i = 1, \cdots, n$ and $\|\theta\|_2 = 1$. $\{\beta_j\}_{j=1}^{m+k+1}$ are spline coefficients and a_θ, b_θ are boundary knots where $a_\theta = min(t_i, i = 1, \cdots, n)$, and $b_\theta = max(t_i, i = 1, \cdots, n)$. Variable selection is encoded by a binary vector ν, equivalently setting components of θ to zero when $\nu_i = 0$.

Priors

- The variable selection indicator ν has Beta–Bernoulli hierarchy prior. Beta hyper-prior on the Bernoulli–inclusion probability w, inducing $p(\nu) \propto \text{Beta}(r_1 + n_\nu, r_2 + p - n_\nu)$ where $n_\nu = \Sigma_{i=1}^{p} I(\nu_i = 1)$. r_1, r_2 are shape and rate parameter of beta distribution.
- Free-knot prior: the number of knots k with mean λ_k. The knot locations $\xi_i, i = 1, ..., k$ have a Dirichlet prior on the scaled interval $[0, 1]$.
- Index vector prior is uniform on the half-sphere of dimension n_ν with first nonzero positive.
- Conjugate normal–inverse-gamma on (β, σ^2) enables analytic integration for RJMCMC with mean $\boldsymbol{\mu_\beta}$, and covariance $\sigma\tau\Sigma_0$.

Sampling Posterior exploration follows a hybrid RJMCMC with six move types: add/remove predictor ν, update θ, add/delete/relocate a knot. The θ update is a random-walk Metropolis via local rotations in a two-coordinate subspace. Knot changes use simple proposals with tractable acceptance ratios. Further sampling method is described in Wang (2009).

Prior hyperparameters These are the prior hyperparameters set in the function. You can define new values for each parameter in prior_param.

1. Index vector: `index_nu_r1`, `index_nu_r2` gives the shape and rate parameter of beta-binomial prior, respectively (default `index_nu_r1 = 1`, `index_nu_r2 = 1`).
2. Link function: B-spline knots, basis and coefficient setup.
   - knots: Free-knot prior for the spline. `link_knots_lambda_k` is the Poisson mean for the number of interior knot k (default `5`). `link_knots_maxknots` is the maximum number of interior knots. If `link_knots_maxknots` is `NULL`, the number of interior knots is randomly drawn from a Poisson distribution.
   - basis: For the basis of B-spline, `link_basis_degree` is the spline degree (default `2`).
   - beta: For the coefficient of B-spline, multivariate normal prior is assigned with mean ($\boldsymbol{\mu_\beta}$) `link_beta_mu`, covariance (Σ_0) `link_beta_Sigma0`, scale parameter (τ) `link_beta_tau`. By default, `link_beta_mu` is a zero vector, `link_beta_tau` is set to the sample size, and `link_beta_Sigma0` is the identity matrix of dimension $1 + k + m$, where k is the number of interior knots and m is the spline order.
3. Error variance: Inverse gamma prior is assigned to σ^2 where `sigma2_shape` is shape parameter and `sigma2_rate` is rate parameter of inverse gamma distribution. Small values for shape and rate parameters yield a weakly-informative prior on σ^2. (default `sigma2_shape = 0.001`, `sigma2_rate = 0.001`)

Initial values These are the initial values set in the function. You can define new values for each initial value in init_param.

1. Index vector: `index_nu` is binary vector indicating active predictors for the index. `index` is initial unit-norm index vector θ (automatically normalized if necessary, with the first nonzero element made positive for identifiability). The vector length must match the number of predictor. Ideally, positions where `index_nu` has a value of 1 should correspond to nonzero elements in θ; elements corresponding to `index_nu = 0` will be set to zero.
2. Link function: `link_k` is initial number of interior knots. `link_knots` is initial vector of interior knot positions in $[0, 1]$, automatically scaled to the true boundary. Length of this vector should be equal to the initial value of k. `link_beta` is initial vector of spline coefficients. Length should be equal to the initial number of basis functions with intercept $(1 + k + m)$.
3. Error variance (`sigma2`): Initial scalar error variance. (default `0.01`)

Scaling the predictors. When the scale of predictors are different, the estimated index can be dominated by large scale variables and make the estimation unstable. In this case, scaling is recommended. If `scale = TRUE, center = TRUE`, it means standardization of the predictors.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

References

Wang, H.-B. (2009). Bayesian estimation and variable selection for single index models. *Computational Statistics & Data Analysis*, 53, 2617–2627.

Hornik, K., & Grün, B. (2014). movMF: an R package for fitting mixtures of von Mises-Fisher distributions. *Journal of Statistical Software*, 58, 1-31.

Examples

```
set.seed(123)
n <- 200; d <- 4
theta <- c(2, 1, 1, 1); theta <- theta / sqrt(sum(theta^2))
f <- function(u) u^2 * exp(u)
sigma <- 0.5
X <- matrix(runif(n * d, -1, 1), nrow = n)
index_vals <- as.vector(X %*% theta)
y <- f(index_vals) + rnorm(n, 0, sigma)
simdata <- data.frame(x = X, y = y)
colnames(simdata) <- c(paste0("X", 1:4), "y")

# One tool version
fit1 <- bsSphere(y ~ ., data = simdata,
                 niter = 10000, nburnin = 3000, nchain = 2)
```

```
# Split version
models <- bsSphere_setup(y ~ ., data = simdata, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 3000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    samplesAsCodaMCMC = TRUE)
fit2 <- as_bsim(models, mcmc.out, Ccompile)
summary(fit2)
```

---

bsSpike *Bayesian Single-Index Regression with B-Spline Link and Spike-and-Slab Prior*

Description

Fits a single-index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$ where the link $f(\cdot)$ is represented by B-spline link function and the index vector θ has spike-and-slab prior.

Usage

```
bsSpike(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

bsSpike_setup(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
```

```
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
prior	Optional named list of prior settings. Further descriptions are in **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in **Details** section.
monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the model_setup function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default 10000).
nburnin	Integer. Burn-in iterations (default 1000).
thin	Integer. Thinning for monitors (default 1).
nchain	Integer. Number of MCMC chains (default 1).
setSeed	Logical or numeric argument. Further details are provided in runMCMC setSeed argument.
scale	Logical. If TRUE, columns of predictors are divided by their standard deviation before fitting. Default FALSE.
center	Logical. If TRUE, columns of predictors are mean-centered before fitting. Default FALSE.

Details

Model The single–index model uses a m-order polynomial spline with k interior knots as follows:

$$f(t) = \sum_{j=1}^{m+k} B_j(t)\,\beta_j$$

on $[a, b]$ with $t_i = X_i'\theta, i = 1, \cdots, n$ and $\|\theta\|_2 = 1$. $\{\beta_j\}_{j=1}^{m+k}$ are spline coefficient and a_θ and b_θ are boundary knots where $a_\theta = min(t_i, i = 1, \cdots, n) - \delta$, and $b_\theta = max(t_i, i = 1, \cdots, n) + \delta$. θ is a p-dimensional index vector subject to a spike-and-slab prior for variable selection with binary indicator variable ν.

Priors

- The variable selection indicator ν has Beta–Bernoulli hierarchy prior. Beta hyper-prior on the Bernoulli–inclusion probability w, inducing $p(\nu) \propto \text{Beta}(r_1 + n_\nu, r_2 + p - n_\nu)$ where $n_\nu = \Sigma_{i=1}^{p} I(\nu_i = 1)$. r_1, r_2 are shape and rate parameter of beta distribution.

- Slab coefficients θ have normal distribution with zero mean and σ_θ^2 variance.
- Conditioned on θ and σ^2, the link coefficients $\beta = (\beta_1, \ldots, \beta_{m+k})^\top$ follow multivariate normal distribution with mean $\boldsymbol{\mu_\beta}$, and covariance $\sigma^2 \tau \Sigma_0$.
- Inverse gamma prior on σ^2 with shape parameter a_σ and rate parameter b_σ.

Sampling All samplers are automatically assigned by nimble.

Prior hyperparameters These are the prior hyperparameters set in the function. You can define new values for each parameter in prior_param.

1. Index vector: `index_r1`, `index_r2` gives the shape and rate parameter of beta-binomial prior, respectively. For slab prior, normal distribution with zero mean is assigned for selected variables θ. `index_sigma_theta` is for variance of θ, and it is assigned 0.25 by default.
2. Link function: B-spline basis and coefficient of B-spline setup.
   - basis: For the basis of B-spline, `link_basis_df` is the number of basis functions (default `21`), `link_basis_degree` is the spline degree (default `2`) and `link_basis_delta` is a small jitter for boundary-knot spacing control (default `0.01`).
   - beta: For the coefficient of B-spline, multivariate normal prior is assigned with mean ($\boldsymbol{\mu_\beta}$) `link_beta_mu`, covariance (Σ_0) `link_beta_Sigma0`, scale parameter (τ) `link_beta_tau`. By default, `link_beta_mu` is zero vector, `link_beta_Sigma0` is identity matrix, and `link_beta_tau` is the number of observations.
3. Error variance: Inverse gamma prior is assigned to σ^2 where `sigma2_shape` is shape parameter and `sigma2_rate` is rate parameter of inverse gamma distribution. (default `sigma2_shape` = `0.001`, `sigma2_rate` = `100`)

Initial values These are the initial values set in the function. You can define new values for each initial value in init_param.

1. Index vector:
   - `index_pi` Initial selecting variable probability (default: `0.5`).
   - `index_nu` Initial vector of inclusion indicators. By default, each `nu` is randomly drawn by $Bernoulli(1/2)$
   - `index` Initial vector of index. By default, each element of index vector, which is chosen by ν, is proposed by normal distribution.
2. Link function: Initial spline coefficients (`link_beta`). By default, $\left(X_\theta^\top X_\theta + \rho \, \mathrm{I}\right)^{-1} X_\theta^\top Y$ is computed, where X_θ is the B-spline basis design matrix.
3. Error variance (`sigma2`): Initial scalar error variance (default `0.01`).

Scaling the predictors. When the scale of predictors are different, the estimated index can be dominated by large scale variables and make the estimation unstable. In this case, scaling is recommended. If `scale` = `TRUE`, `center` = `TRUE`, it means standardization of the predictors.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

References

Antoniadis, A., Grégoire, G., & McKeague, I. W. (2004). Bayesian estimation in single-index models. *Statistica Sinica*, 1147-1164.

Hornik, K., & Grün, B. (2014). movMF: an R package for fitting mixtures of von Mises-Fisher distributions. *Journal of Statistical Software*, 58, 1-31.

McGee, G., Wilson, A., Webster, T. F., & Coull, B. A. (2023). Bayesian multiple index models for environmental mixtures. *Biometrics*, 79(1), 462-474.

Examples

```
set.seed(123)
n <- 200; d <- 4
theta <- c(2, 1, 1, 1); theta <- theta / sqrt(sum(theta^2))
f <- function(u) u^2 * exp(u)
sigma <- 0.5
X <- matrix(runif(n * d, -1, 1), nrow = n)
index_vals <- as.vector(X %*% theta)
y <- f(index_vals) + rnorm(n, 0, sigma)
simdata <- data.frame(x = X, y = y)
colnames(simdata) <- c(paste0("X", 1:4), "y")

# One tool version
fit1 <- bsSpike(y ~ ., data = simdata,
                niter = 10000, nburnin = 3000, nchain = 1)

# Split version
models <- bsSpike_setup(y ~ ., data = simdata)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 3000, thin = 1,
                    nchains = 1, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)
```

```
fit2 <- as_bsim(models, mcmc.out, Ccompile)
summary(fit2)
```

`coef.bsim` *Extract Index Vector Coefficients from* `BayesSIM`

Description

Computes posterior summaries of the single-index model index vector from a fitted `BayesSIM`. Users may choose either the posterior mean or median as the point estimate.

Usage

```
## S3 method for class 'bsim'
coef(
  object,
  method = c("mean", "median"),
  error = FALSE,
  origscale = FALSE,
  ...
)
```

Arguments

`object`	A fitted object of `BayesSIM` or individual model.
`method`	Character string indicating the summary statistic to compute. Options are `"mean"` or `"median"`. Default is `"mean"`.
`error`	Logical value whether computing posterior standard deviation for index estimates. If `method` is `"mean"`, standard deviation of index vector MCMC samples is gained. If `method` is `"median"`, median absolute deviation of index vector MCMC samples is gained. `FALSE` is default.
`origscale`	Logical. If `origscale` is `TRUE`, you can see the index into original scale when you standardized the predictors.
`...`	Additional arguments passed to other methods.

Value

A numeric vector or data.frame of estimated coefficient and standard error of the index vector.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Check median index vector estimates with standard errors
coef(fit_one, method = "median", error = TRUE)

# Fitted index values of median prediction
fitted(fit_one, type = "linpred", method = "median")

# Residuals of median prediction
residuals(fit_one, method = "median")

# Summary of the model
summary(fit_one)

# Convergence diagnostics
nimTraceplot(fit_one)

# Goodness of fit
GOF(fit_one)

# Fitted plot
plot(fit_one)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
plot(pred)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

---

compileModelAndMCMC *Compile a 'nimble' Model and Its MCMC*

Description

Compiles a nimble model object and a corresponding (uncompiled) MCMC algorithm and returns the compiled pair.

Usage

```
compileModelAndMCMC(object)
```

Arguments

object	Class BayesSIM_setup object

Details

The function first compiles nimble model object, then compiles nimble sampler. Both compiled model and compiled MCMC samplers are returned as a list.

Value

A list with two elements:

model the compiled NIMBLE model (external pointer object).

mcmc the compiled MCMC function/algorithm bound to the model.

See Also

nimbleModel, configureMCMC, buildMCMC, compileNimble, runMCMC

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

concrete *UCI Concrete Compressive Strength (n = 1030, p = 8)*

Description

Concrete compressive strength dataset from the UCI Machine Learning Repository. No missing variables and there are 8 quantitative inputs and 1 quantitative output.

Usage

```
data(concrete)
```

Format

Numeric data.frame of size 1030 $\times$ 8.

cement Numeric. Cement content (kg/m^3).

blast_furnace_slag Numeric. Blast furnace slag (kg/m^3).

fly_ash Numeric. Fly ash (kg/m^3).

water Numeric. Mixing water (kg/m^3).

superplasticizer Numeric. Superplasticizer (kg/m^3).

coarse_aggreate Numeric. Coarse aggregate (kg/m^3).

fine_aggregate Numeric. Fine aggregate (kg/m^3).

age Numeric. Curing age (days; typically 1–365).

strength Numeric. Concrete compressive strength (MPa).

Details

Source Concrete Compressive Strength in UCI Machine Learning Repository. This data is integrated by experimental data from 17 different sources to check the realiability of the strength. This dataset compiles experimental concrete mixes from multiple studies and is used to predict compressive strength and quantify how mixture ingredients and curing age affect that strength.

Variables.

- Cement, Blast Furnace Slag, Fly Ash, Water, Superplasticizer, Coarse Aggregate, Fine Aggregate: quantities in kg per m^3 of mixture.
- Age: curing time in days (1–365).
- Target(strength): compressive strength in MPa.

References

Yeh, I. (1998). Concrete Compressive Strength [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5PK67.

Yeh, I. (1998). Modeling of strength of high-performance concrete using artificial neural networks. Cement and Concrete research, 28(12), 1797-1808.

Examples

```
data(concrete)
str(concrete)
plot(density(concrete$strength), main = "Concrete compressive strength (MPa)")
```

DATA1 *Simulated Single–Index Data (n = 200, p = 4)*

Description

Synthetic data from a single–index model $y = f(X'\theta) + \varepsilon$ with $f(u) = u^2 \exp(u)$ and $\varepsilon \sim N(0, \sigma^2)$. The index vector is $\theta = (2, 1, 1, 1)/\|(2, 1, 1, 1)\|_2$ and $\sigma = 0.5$.

Usage

```
data(DATA1)
```

Format

X Numeric matrix of size 200×4, entries i.i.d. $Unif(-1, 1)$.

y Numeric vector of length 200.

Examples

```
data(DATA1)
str(DATA1)
```

---

fitted.bsim *Extract Fitted Values from* BayesSIM

Description

Computes fitted values from a BayesSIM, using either the posterior mean or median of the estimated link function with index values. Fitted values can be returned on the latent scale or on the linear predictor scale.

Usage

```
## S3 method for class 'bsim'
fitted(
  object,
  type = c("latent", "linpred"),
  method = c("mean", "median"),
  ...
)
```

Arguments

object	A fitted object of BayesSIM or individual model.
type	Character string indicating the scale on which fitted values are returned. Default is "latent". • "latent": fitted response values $\hat{y} = E(\mathbf{Y}\|\mathbf{X})$. • "linpred": linear predictor $X\theta$.
method	Character string specifying the summary statistic used to compute the fitted values. Options are "mean" or "median". Default is "mean".
...	Additional arguments passed to other methods.

Value

A numeric vector of fitted values.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Check median index vector estimates with standard errors
coef(fit_one, method = "median", error = TRUE)

# Fitted index values of median prediction
fitted(fit_one, type = "linpred", method = "median")

# Residuals of median prediction
residuals(fit_one, method = "median")

# Summary of the model
summary(fit_one)

# Convergence diagnostics
nimTraceplot(fit_one)

# Goodness of fit
GOF(fit_one)

# Fitted plot
plot(fit_one)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
plot(pred)


# 2. Split version
```

```
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

`getFunction` *Get nimbleModel and nimbleSampler Object from the Result of* `compileModelAndMCMC`

Description

Return compiled nimble model object and a corresponding MCMC samplers.

Usage

```
getModel(object)

getSampler(object)

getCModel(fit)

getCSampler(fit)

## S3 method for class 'bsim'
getCModel(fit)

## S3 method for class 'bsim'
getCSampler(fit)
```

Arguments

`object`	The result of `compileModelAndMCMC` function.
`fit`	Fitted object of `BayesSIM`.

Value

`getModel` returns compiled Nimble model object. `getSampler` returns compiled Nimble sampler object, directly using in `runMCMC` function.

See Also

nimbleModel, configureMCMC, buildMCMC, compileNimble, runMCMC

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)

# Extract compiled model
compiledModel <- getCModel(fit_split)

# Extract compiled sampler
compiledSampler <- getCSampler(fit_split)
```

---

getInit *Get Initial Value of the Model*

Description

Functions for getting list of initial values of the **nimble** model.

Usage

```
getInit(object)
```

Arguments

object	A fitted object of BayesSIM, BayesSIM_setup or individual model.

Details

The list of initial values are returned. This can be helpful to use when you use BayesSIM_setup. You should be aware of that if you want to get more than 1 chain of MCMC samples, you should change nchain argument in BayesSIM_setup. The output of initial values would be different, depending on the number of chain.

You can apply BayesSIM object when you want to check the list of initial values.

Value

BUGS code of the model definition.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)


# Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

---

getModelDef *Get Definition of the Model*

Description

Functions for identifying definition of the **nimble** model.

Usage

```
getModelDef(object)
```

Arguments

object	A fitted object of BayesSIM or individual model.

Details

The function that contain Bayes SIM model structure in **nimble**. This function is for advanced users. There are several functions used in the model definition.

- `transX_fisher`, `transX_sp`: Making B-spline basis.
- `dvMFnim`: Distribution of von Mises-Fisher.
- `dKnotsSimple`: Distribution of the free knots for `bsSphere`.
- `dunitSphere`: Distribution of unit sphere.
- `alphaTheta`: One-to-one polar transformation. Making index vector from individual angular vector `psi`.
- `expcov_gpSphere`, `expcov_gpPolar`, `expcov_gpSpike`: Covariance kernel of each model. `expcov_gpSphere` uses squared-exponential kernel, `expcov_gpPolar` uses OU process kernel, and `expcov_gpSpike` uses squared-exponential including its own parameter, λ^{-1}.
- `Xlinear`: Making linear combination with index vector.

Value

BUGS code of the model definition.

See Also

`getVarMonitor`

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)
models <- bsFisher_setup(y ~ ., data = simdata2)
# Get model definition
getModelDef(models)
```

`getVarMonitor` *Retrieve Monitorable Variables*

Description

Functions for retrieving the variables that can be monitored.

Usage

```
getVarMonitor(object, type = c("name", "list"))
```

Arguments

object	A fitted object of BayesSIM, BayesSIM_setup or individual model.
type	Options for variables. By default, type = "name" is used that it only prints the name of the node. If you put name of the nodes, the MCMC outputs gave you all elements of the variable, in case of the vector. If type = "list", the dimension of the nodes are printed. If you put name and dimension of the nodes, only specific location of vector or matrix can be seen in summary or nimTraceplot.

Details

The function returns a list of variables that can be used in monitors2 in the bayesSIM function. You can also refer to getModelDef to understand the model structure and designate necessary variables. Stochastic nodes of the model are recommended to be monitored.

Value

A vector of variables that can be monitored in the model.

See Also

getModelDef

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)
models <- BayesSIM_setup(y ~ ., data = simdata2)

# Get monitorable variables
getVarMonitor(models)
# Get the list of variables with dimension
getVarMonitor(models, type = "list")
```

---

getWAIC *Compute Conditional WAIC for BSIM*

Description

Computes the Watanabe-Akaike Information Criterion (WAIC) for Bayesian single-index models and related latent-function models, evaluated at the outcome-level Gaussian likelihood conditional on the latent values f and residual variance σ^2.

Usage

```
getWAIC(object)
```

Arguments

`object`	A fitted model object containing posterior samples.

Details

For each posterior draw $r = 1, \ldots, R$ and observation $i = 1, \ldots, N$, the pointwise log-likelihood is computed as

$$\log p(y_i \mid f_i^{(r)}, \sigma^{2(r)}) = \log N(y_i \mid f_i^{(r)}, \sigma^{2(r)}).$$

The resulting $R \times N$ log-likelihood matrix is passed to `waic` to obtain the standard WAIC estimates (Watanabe, 2010; Vehtari, Gelman, and Gabry, 2017).

Value

A numeric matrix of WAIC estimates containing the following rows:

- `elpd_waic`: expected log pointwise predictive density.
- `p_waic`: effective number of parameters.
- `waic`: WAIC on the deviance scale ($-2\times$ `elpd_waic`).

Each row contains the point estimate and its standard error.

References

Watanabe, S., & Opper, M. (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of machine learning research*, 11(12).

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and computing*, 27(5), 1413-1432.

See Also

`waic`

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)
# Calculate WAIC
getWAIC(fit_one)

# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
```

```
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)

# Calculate WAIC
getWAIC(fit_split)
```

GOF *Goodness of Fit for* BayesSIM

Description

Generic function applied to BayesSIM. It extracts goodness of fit of the BayesSIM.

Usage

```
GOF(object)

## S3 method for class 'bsim'
GOF(object, ...)
```

Arguments

object	A fitted object of BayesSIM or individual model.
...	Additional arguments passed to other methods.

Value

Mean squared error of model with mean of MCMC sample is applied.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Check median index vector estimates with standard errors
coef(fit_one, method = "median", error = TRUE)

# Fitted index values of median prediction
fitted(fit_one, type = "linpred", method = "median")

# Residuals of median prediction
```

```
residuals(fit_one, method = "median")

# Summary of the model
summary(fit_one)

# Convergence diagnostics
nimTraceplot(fit_one)

# Goodness of fit
GOF(fit_one)

# Fitted plot
plot(fit_one)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
plot(pred)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

---

gpFisher *Bayesian Single-Index Regression with Gaussian Process Link and von Mises-Fisher Prior*

Description

Fits a single–index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$, where the index θ lies on the unit sphere with von Mises-Fisher prior, and the link $f(\cdot)$ is represented with Gaussian process.

Usage

```
gpFisher(
  formula,
  data,
  prior = NULL,
  init = NULL,
```

```
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

gpFisher_setup(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
prior	Optional named list of prior settings. Further descriptions are in **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in **Details** section.
monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the `model_setup` function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default `10000`).
nburnin	Integer. Burn-in iterations (default `1000`).
thin	Integer. Thinning for monitors (default `1`).
nchain	Integer. Number of MCMC chains (default `1`).
setSeed	Logical or numeric argument. Further details are provided in runMCMC `setSeed` argument.
scale	Logical. If `TRUE`, columns of predictors are divided by their standard deviation before fitting. Default `FALSE`.
center	Logical. If `TRUE`, columns of predictors are mean-centered before fitting. Default `FALSE`.

Details

Model The single-index model uses Gaussian process with zero mean and and covariance kernel $\eta \cdot \exp(-\frac{(t_i - t_l)^2}{\gamma})$ as a link function, where $t_i, t_l, (i, l = 1, \ldots, n)$ is index value. Index vector should be length 1.

Priors

- von Mises–Fisher prior on the index θ with direction and concentration.
- Covariance kernel: Amplitude, η, follows log normal distribution with mean a_η and variance b_η. Length-scale parameter, γ, follows gamma distribution with shape parameter α_γ and rate parameter β_γ.
- Inverse gamma prior on σ^2 with shape parameter a_σ and rate parameter b_σ.

Sampling All sampling parameters' samplers are assigned by nimble.

Prior hyperparameters These are the prior hyperparameters set in the function. You can define new values for each parameter in prior_param.

1. Index vector: von Mises–Fisher prior for the projection vector θ. `index_direction` is a unit direction vector of the von Mises–Fisher distribution. By default, the estimated vector from projection pursuit regression is assigned. `index_dispersion` is the positive concentration parameter (default `150`).
2. Link function:
   - Length-scale:Gamma distribution is assigned for length-scale parameter, γ. `link_lengthscale_shape` is shape parameter (default `1/8`) and `link_lengthscale_rate` is rate parameter of `lengthscale` (default `1/8`.)
   - Amplitude: Log-normal distribution is assigned for amplitude parameter, η. `link_amp_a` is mean (default `-1`), and `link_amp_b` is variance (default `1`).
3. Error variance: An inverse-gamma prior is assigned to σ^2 where `sigma2_shape` is shape parameter and `sigma2_rate` is rate parameter of inverse gamma distribution (default `sigma2_shape = 1, sigma2_rate = 1`).

Initial values These are the initial values set in the function. You can define new values for each initial value in init_param.

1. Index vector (`index`): Initial unit index vector θ. By default, the vector is sampled from the von Mises–Fisher prior.
2. Link function: `link_lengthscale` is initial scalar length-scale parameter (default `0.1`). `link_amp` is initial scalar amplitude parameter (default `1`).
3. Error variance (`sigma2`): Initial scalar error variance (default `1`).

Scaling the predictors. When the scale of predictors are different, the estimated index can be dominated by large scale variables and make the estimation unstable. In this case, scaling is recommended. If `scale = TRUE, center = TRUE`, it means standardization of the predictors.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

References

Antoniadis, A., Grégoire, G., & McKeague, I. W. (2004). Bayesian estimation in single-index models. *Statistica Sinica*, 1147-1164.

Choi, T., Shi, J. Q., & Wang, B. (2011). A Gaussian process regression approach to a single-index model. *Journal of Nonparametric Statistics*, 23(1), 21-36.

Hornik, K., & Grün, B. (2014). movMF: an R package for fitting mixtures of von Mises-Fisher distributions. *Journal of Statistical Software*, 58, 1-31.

Examples

```
## Not run:
set.seed(123)
N <- 60; p <- 2
x1 <- runif(N, -3, 5)
x2 <- runif(N, -3, 5)
beta1 <- 0.45; beta2 <- sqrt(1-beta1^2)
sigma <- sqrt(0.0036)
xlin <- x1*beta1 + x2*beta2
eta <- 0.1*xlin + sin(0.5*xlin)^2
y <- rnorm(N, eta, sigma)
x <- matrix(c(x1, x2), ncol = 2)
simdata <- data.frame(x = x, y = y)
colnames(simdata) <- c("X1", "X2", "y")

# One tool version
fit1 <- gpFisher(y ~ ., data = simdata, nchain = 1, niter = 1000, nburnin = 100)
```

```
# Split version
models <- gpFisher_setup(y ~ ., data = simdata, nchain = 1)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 1000, nburnin = 100, thin = 1,
                    nchains = 1, setSeed = TRUE, inits = initList,
                     summary = TRUE, samplesAsCodaMCMC = TRUE)
fit2 <- as_bsim(models, mcmc.out, Ccompile)
summary(fit2)

## End(Not run)
```

---

gpPolar *Bayesian Single-Index Regression with Gaussian Process Link and One-to-One Polar Transformation*

Description

Fits a single–index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$, where the index θ is specified and computed via a one-to-one polar transformation, and the link $f(\cdot)$ is represented with a Gaussian process.

Usage

```
gpPolar(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

gpPolar_setup(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
```

```
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

gpPolarHigh(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

gpPolarHigh_setup(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
prior	Optional named list of prior settings. Further descriptions are in **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in **Details** section.

monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the model_setup function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default 10000).
nburnin	Integer. Burn-in iterations (default 1000).
thin	Integer. Thinning for monitors (default 1).
nchain	Integer. Number of MCMC chains (default 1).
setSeed	Logical or numeric argument. Further details are provided in runMCMC setSeed argument.
scale	Logical. If TRUE, columns of predictors are divided by their standard deviation before fitting. Default FALSE.
center	Logical. If TRUE, columns of predictors are mean-centered before fitting. Default FALSE.

Details

Model The single–index model is specified as $Y_i = f(X_i'\theta) + \epsilon_i$, where the index vector θ lies on the unit sphere with ($\|\theta\|_2 = 1$) with non-zero first component to ensure identifiability and is parameterized via a one-to-one polar transformation with angle $\psi_1, ..., \psi_{p-1}$.

The mapping is

$$\theta_1 = \sin(\psi_1),$$
$$\theta_i = \Big(\prod_{j=1}^{i-1} \cos(\psi_j)\Big) \sin(\psi_i), \quad i = 2, \ldots, p-1,$$
$$\theta_p = \prod_{j=1}^{p-1} \cos(\psi_j).$$

The vector is then scaled to unit length.

Sampling is performed on the angular parameters θ defining the index vector. The link function $f(\cdot)$ is modeled by a Gaussian process prior with zero mean and an Ornstein–Uhlenbeck (OU) covariance kernel $\exp(-\kappa \cdot |t_i - t_l|), i, l = 1, \ldots, n$, where κ is a bandwidth (smoothness) parameter and t_i, t_l is index value ($t_i = X_i'\theta$).

Priors

- ψ is $p-1$ dimension of polar angle of index vector and re-scaled Beta distribution on $[0, \pi]$ is assigned for prior.
- Prior for κ (bandwidth parameter) is discrete uniform of equally spaced grid points in $[\kappa_{\min}, \kappa_{\max}]$.
- Inverse gamma prior on σ^2 with shape parameter a_σ and rate parameter b_σ.

Sampling For gpPolar, θ is sampled by Metropolis-Hastings and updated with f, κ is chosen by grid search method that maximizes likelihood, σ^2 are sampled from their full conditional distributions using Gibbs sampling. Since κ is sampled by grid search, more than 5 dimension of variables gpPolarHigh is recommended. For gpPolarHigh, all sampling parameters' samplers are assigned by nimble.

Prior hyperparameters These are the prior hyperparameters set in the function. You can define new values for each parameter in prior_param.

1. Index vector: Only shape parameter `index_psi_alpha` of $p-1$ dimension vector is needed since rate parameters is computed to satisfy $\theta_{j,\text{MAP}}$. By default, the shape parameter for each element of the polar vector is set to `5000`.
2. Link function: `link_kappa_min` is minimum value of kappa (default `0.5`), `link_kappa_max` is maximum value of kappa (default `4`), and `link_kappa_grid_width` is space (default `0.1`).
3. Error variance: An Inverse gamma prior is assigned to σ^2 where `sigma2_shape` is shape parameter and `sigma2_rate` is rate parameter of inverse gamma distribution (default `sigma2_shape = 2, sigma2_rate = 0.01`).

Initial values These are the initial values set in the function. You can define new values for each initial value in init_param.

1. Index vector: Initial vector of polar angle `index_psi` with $p-1$ dimension. Each element of angle is between 0 and π.
2. Link function: Initial scalar scale parameter of covariance kernel `link_kappa` (default `2`).
3. Error variance (`sigma2`): Initial scalar error variance (default `0.01`).

Scaling the predictors. When the scale of predictors are different, the estimated index can be dominated by large scale variables and make the estimation unstable. In this case, scaling is recommended. If `scale = TRUE, center = TRUE`, it means standardization of the predictors.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

References

Dhara, K., Lipsitz, S., Pati, D., & Sinha, D. (2019). A new Bayesian single index model with or without covariates missing at random. *Bayesian analysis*, 15(3), 759.

Examples

```
library(MASS)
N <- 100    # Sample Size
p <- 3
mu <- c(0,0,0)
rho <- 0
Cx <- rbind(c(1,rho,rho), c(rho,1,rho), c(rho, rho,1))
X <- mvrnorm(n = N, mu=mu, Sigma=Cx, tol=1e-8)
alpha <- c(1,1,1)
alpha <- alpha/sqrt(sum(alpha^2))
z <- matrix(0,N)
z <- X %*% alpha
z <- z[,1]
sigma <- 0.3
f <- exp(z)
y <- f + rnorm(N, 0, sd=sigma) # adding noise
y <- y-mean(y)
f_all <- f
x_all <- X
y_all <- y
simdata <- cbind(x_all, y, f)
simdata <- as.data.frame(simdata)
colnames(simdata) = c('x1', 'x2', 'x3', 'y','f')

prior_gpPolar <- prior_param(indexprior = "polar", link = "gp",
                             link_kappa_max = 2, link_kappa_grid_width = 0.05)

# One tool version
fit1 <- gpPolar(y ~ x1 + x2 + x3, data = simdata,
                prior = prior_gpPolar,
                niter = 3000, nburnin = 2000, nchain = 1)
fit2 <- gpPolarHigh(y ~ x1 + x2 + x3, data = simdata,
                    niter = 3000, nburnin = 2000, nchain = 1)

# Split version
models1 <- gpPolar_setup(y ~ x1 + x2 + x3, data = simdata,
                         prior = prior_gpPolar)
models2 <- gpPolarHigh_setup(y ~ x1 + x2 + x3, data = simdata,
                         prior = prior_gpPolar)
Ccompile1 <- compileModelAndMCMC(models1)
Ccompile2 <- compileModelAndMCMC(models2)
sampler1 <- getSampler(Ccompile1)
sampler2 <- getSampler(Ccompile2)
initList1 <- getInit(models1)
initList2 <- getInit(models2)
mcmc.out1 <- runMCMC(sampler1, niter = 3000, nburnin = 2000, thin = 1,
                    nchains = 1, setSeed = TRUE, init = initList1,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)
mcmc.out2 <- runMCMC(sampler2, niter = 3000, nburnin = 2000, thin = 1,
                    nchains = 1, setSeed = TRUE, init = initList2,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)
fit1_split <- as_bsim(models1, mcmc.out1, Ccompile1)
```

```
fit2_split <- as_bsim(models2, mcmc.out2, Ccompile2)
```

---

gpSphere *Bayesian Single-Index Regression with Gaussian Process Link and Uniform Prior on Unit Sphere*

Description

Fits a single–index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$, where the index θ lies on the unit sphere, and the link $f(\cdot)$ is represented with Gaussian process.

Usage

```
gpSphere(
  formula,
  data,
  prior = NULL,
  init = NULL,
  method = "FB",
  lowerB = NULL,
  upperB = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

gpSphere_setup(
  formula,
  data,
  prior = NULL,
  init = NULL,
  method = "FB",
  lowerB = NULL,
  upperB = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
```

```
  scale = FALSE,
  center = FALSE
)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
prior	Optional named list of prior settings. Further descriptions are in **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in **Details** section.
method	Character, gpSphere model has 3 different types of sampling method, fully Bayesian method ("FB"), empirical Bayes approach ("EB"), and empirical Gibbs sampler ("EG"). Assign one sampler method. Empirical sampling approach is recommended in high-dimensional data. By default, fully Bayesian approach is assigned.
lowerB	This parameter is only for gpSphere model. Numeric vector of element-wise lower bounds for the "L-BFGS-B" method. When the empirical Bayes or Gibbs sampler method is used, the marginal likelihood is optimized via optim(method = "L-BFGS-B"). The vector must be ordered as c(index vector, lengthscale, amp, sigma2). Note that sigma2 is included only for the empirical Bayes method (omit it for Gibbs). By default, the lower bounds are -1 for the index vector and -1e2 for logarithm of lengthscale, amp, and (if present) sigma2.
upperB	This parameter is only for gpSphere model. Numeric vector of element-wise upper bounds for the "L-BFGS-B" method. When the empirical Bayes or Gibbs sampler method is used, the marginal likelihood is optimized via optim(method = "L-BFGS-B"). The vector must be ordered as c(index vector, lengthscale, amp, sigma2). Note that sigma2 is included only for the empirical Bayes method (omit it for Gibbs). By default, the upper bounds are 1 for the index vector and 1e2 for logarithm of lengthscale, amp, and (if present) sigma2.
monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the model_setup function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default 10000).
nburnin	Integer. Burn-in iterations (default 1000).
thin	Integer. Thinning for monitors (default 1).
nchain	Integer. Number of MCMC chains (default 1).
setSeed	Logical or numeric argument. Further details are provided in runMCMC setSeed argument.
scale	Logical. If TRUE, columns of predictors are divided by their standard deviation before fitting. Default FALSE.
center	Logical. If TRUE, columns of predictors are mean-centered before fitting. Default FALSE.

Details

Model The single-index model uses Gaussian process with zero mean and and covariance kernel $\eta \cdot \exp(-\frac{(t_i - t_l)^2}{\gamma})$ as a link function, where $t_i, t_l, (i, l = 1, \ldots, n)$ is index value. Index vector should be length 1.

Priors

- Uniform prior on the unit sphere for the index $\boldsymbol{\theta}$.
- Covariance kernel: Amplitude, η, follows log normal distribution with mean a_η and variance b_η. Length-scale parameter, γ, follows gamma distribution with shape parameter α_γ and rate parameter β_γ.
- Inverse-Gamma prior on σ^2 with shape parameter a_σ and rate parameter b_σ.

Sampling In the fully Bayesian approach, θ, γ, and η are updated via the Metropolis–Hastings algorithm, while f and σ^2 are sampled using Gibbs sampling.

In the empirical Bayes approach, θ, γ, η, and σ^2 are estimated by maximum a posteriori (MAP), and f is sampled from its full conditional posterior distribution.

In the empirical Gibbs sampler, θ, γ, and η are estimated by MAP, whereas f and σ^2 are sampled via Gibbs sampling.

For estimation via MAP, effective sample size or potential scale reduction factor is meaningless.

Prior hyperparameters These are the prior hyperparameters set in the function. You can define new values for each parameter in prior_param.

1. Index vector: Nothing to assign.
2. Link function:
   - Length-scale:Gamma distribution is assigned for length-scale parameter, γ. `link_lengthscale_shape` is shape parameter (default `1/8`) and `link_lengthscale_rate` is rate parameter of `lengthscale` (default `1/8`).
   - Amplitude: Log-normal distribution is assigned for amplitude parameter, η. `link_amp_a` is mean (default `-1`), and `link_amp_b` is variance (default `1`).
3. Error variance: inverse gamma prior is assigned to σ^2 where `sigma2_shape` is shape parameter and `sigma2_rate` is rate parameter of inverse gamma distribution (default `sigma2_shape = 1, sigma2_rate = 1`).

Initial values These are the initial values set in the function. You can define new values for each initial value in init_param.

1. Index vector (`index`): Initial unit index vector. By default, vector is randomly drawn from normal distribution and standardized.
2. Link function: `link_lengthscale` is initial scalar length-scale parameter (default `0.1`). `link_amp` is initial scalar amplitude parameter (default `1`).
3. Error variance (`sigma2`): Initial scalar error variance (default `1`).

Scaling the predictors. When the scale of predictors are different, the estimated index can be dominated by large scale variables and make the estimation unstable. In this case, scaling is recommended. If `scale = TRUE, center = TRUE`, it means standardization of the predictors.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

References

Choi, T., Shi, J. Q., & Wang, B. (2011). A Gaussian process regression approach to a single-index model. *Journal of Nonparametric Statistics*, 23(1), 21-36.

Examples

```
set.seed(123)
n <- 200; d <- 4
theta <- c(2, 1, 1, 1); theta <- theta / sqrt(sum(theta^2))
f <- function(u) u^2 * exp(u)
sigma <- 0.5
X <- matrix(runif(n * d, -1, 1), nrow = n)
index_vals <- as.vector(X %*% theta)
y <- f(index_vals) + rnorm(n, 0, sigma)
simdata <- data.frame(x = X, y = y)
colnames(simdata) <- c(paste0("X", 1:4), "y")

# One tool version
fit1 <- gpSphere(y ~ ., method = "EB", data = simdata,
                 niter = 1000, nburnin = 100)

# Split version
models <- gpSphere_setup(y ~ ., method = "EB", data = simdata)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 1000, nburnin = 100, thin = 1,
```

```
                    nchains = 1, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)
fit2 <- as_bsim(models, mcmc.out, Ccompile)
# The estimates computed by MAP - standard error of the esitmate is meaningless.
summary(fit2)
```

---

gpSpike *Bayesian Single-Index Regression with Gaussian Process Link and Spike-and-Slab Prior*

Description

Fits a single-index model $Y_i \sim \mathcal{N}(f(X_i'\theta), \sigma^2), i = 1, \cdots, n$, where index vector θ has a spike-and-slab prior and the link $f(\cdot)$ is represented by Gaussian process and the

Usage

```
gpSpike(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)

gpSpike_setup(
  formula,
  data,
  prior = NULL,
  init = NULL,
  monitors = NULL,
  niter = 10000,
  nburnin = 1000,
  thin = 1,
  nchain = 1,
  setSeed = FALSE,
  scale = FALSE,
  center = FALSE
)
```

Arguments

formula	an object of class formula. Interaction term is not allowed for single-index model.
data	an data frame.
prior	Optional named list of prior settings. Further descriptions are in **Details** section.
init	Optional named list of initial values. If the values are not assigned, they are randomly sampled from prior or designated value. Further descriptions are in **Details** section.
monitors	Optional character vector of additional monitor nodes. To check the names of the nodes, fit the `model_setup` function and then inspect the variable names stored in the model object using getVarMonitor.
niter	Integer. Total MCMC iterations (default `10000`).
nburnin	Integer. Burn-in iterations (default `1000`).
thin	Integer. Thinning for monitors (default `1`).
nchain	Integer. Number of MCMC chains (default `1`).
setSeed	Logical or numeric argument. Further details are provided in runMCMC `setSeed` argument.
scale	Logical. If `TRUE`, columns of predictors are divided by their standard deviation before fitting. Default `FALSE`.
center	Logical. If `TRUE`, columns of predictors are mean-centered before fitting. Default `FALSE`.

Details

Model The single–index model is specified as $Y_i = f(X_i'\theta) + \epsilon_i$, where θ is a p-dimensional index vector subject to a spike-and-slab prior for variable selection. The link function $f(\cdot)$ is modeled using a Gaussian process prior with zero mean and squared exponential covariance kernel $K(x_1, x_2) = \exp\{-\frac{(x_1-x_2)^T\theta^2}{\gamma}\}$, where γ determines the smoothness of f. The covariance kernel is re-parameterized to $\exp\{-(x_1 - x_2)^T{\theta^*}^2\}$ where $\gamma = \frac{1}{||\theta^*||}$ and $\theta = ||\theta||^{-1}\theta^*$. Therefore, θ^* is sampled in MCMC.

Priors

- The variable selection indicator ν has Beta–Bernoulli hierarchy prior. Beta hyper-prior on the Bernoulli–inclusion probability w, inducing $p(\nu) \propto \text{Beta}(r_1 + n_\nu, r_2 + p - n_\nu)$ where $n_\nu = \Sigma_{i=1}^p I(\nu_i = 1)$. r_1, r_2 are shape and rate parameter of beta distribution.
- Slab coefficients θ have normal distribution with zero mean and σ_θ^2 variance.
- GP precision λ^{-1} follows gamma distribution with shape parameter a_λ, and rate parameter b_λ.
- Error precision $(\sigma^2)^{-1}$ follows gamma distribution with shape parameter a_σ, and rate parameter b_σ.

Sampling A random walk Metropolis algorithm is used to sample λ^{-1} and a Metropolis-Hastings algorithm is used for the main parameters (θ^*, ν). The variance σ^2 is directly sampled from posterior distribution. f is not directly sampled by MCMC. A large number of total iterations is required to achieve convergence, and thinning may therefore be necessary.

Prior hyperparameters These are the prior hyperparameters set in the function. You can define new values for each parameter in prior_param.

1. Index vector: `index_r1`, `index_r2` gives the shape and rate parameter of beta-binomial prior, respectively. For slab prior, normal distribution with zero mean is assigned for selected variables θ. `index_sigma_theta` is for variance of θ, and it is assigned 0.25 by default.
2. Link function: Inverse gamma prior is assigned for hyper-parameters λ^{-1} where `link_inv_lambda_shape` is shape parameter and `link_inv_lambda_rate` is rate parameter of inverse gamma distribution (default `link_inv_lambda_shape` = `1`, `link_inv_lambda_rate` = `0.1`).
3. Error variance: An Inverse gamma prior is assigned to σ^2 where `sigma2_shape` is shape parameter and `sigma2_rate` is rate parameter of inverse gamma distribution (default `sigma2_shape` = `0.001`, `sigma2_rate` = `0.001`).

Initial values These are the initial values set in the function. You can define new values for each initial value in init_param

1. Index vector:
   - `index_pi`: Initial selecting variable probability (default: `0.5`).
   - `index_nu`: Initial vector of inclusion indicators. By default, each `index_nu` is randomly drawn by $Bernoulli(1/2)$.
   - `index`: Initial vector of index. By default, each element of index vector, which is chosen by indicator, is proposed by normal distribution.
2. Link function: Initial scalar of lambda (`link_inv_lambda`) for covariance kernel of Gaussian process.
3. Error variance (`sigma2`): Initial scalar error variance (default: `0.01`).

Scaling the predictors. When the scale of predictors are different, the estimated index can be dominated by large scale variables and make the estimation unstable. In this case, scaling is recommended. If `scale` = `TRUE`, `center` = `TRUE`, it means standardization of the predictors.

Value

A `list` typically containing:

`coefficients` Mean estimates of index vector. Return list of `model_setup` does not include it.

`ses` Mean standard error of index vector. Return list of `model_setup` does not include it.

`residuals` Residuals with mean estimates of fitted values. Return list of `model_setup` does not include it.

`fitted.values` Mean estimates of fitted values. Return list of `model_setup` does not include it.

`linear.predictors` Mean estimates of single-index values. Return list of `model_setup` does not include it.

`gof` Goodness-of-fit. Return list of `model_setup` function does not include it.

`samples` Posterior draws of variables for computing fitted values of the model, including θ, σ^2. Return list of `model_setup` does not include it.

`input` List of data used in modeling, formula, input values for prior and initial values, and computation time without compiling.

`defModel` Nimble model object.

`defSampler` Nimble sampler object.

`modelName` Name of the model.

References

McGee, G., Wilson, A., Webster, T. F., & Coull, B. A. (2023). Bayesian multiple index models for environmental mixtures. *Biometrics*, 79(1), 462-474.

Examples

```
set.seed(123)
n <- 200; d <- 4
theta <- c(2, 1, 1, 1); theta <- theta / sqrt(sum(theta^2))
f <- function(u) u^2 * exp(u)
sigma <- 0.5
X <- matrix(runif(n * d, -1, 1), nrow = n)
index_vals <- as.vector(X %*% theta)
y <- f(index_vals) + rnorm(n, 0, sigma)
simdata <- data.frame(x = X, y = y)
colnames(simdata) <- c(paste0("X", 1:4), "y")

# One tool version
fit1 <- gpSpike(y ~ ., data = simdata,
               niter = 10000, nburnin = 5000, thin = 40)

# Split version
models <- gpSpike_setup(y ~ ., data = simdata)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 40,
                    nchains = 1, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)
fit2 <- as_bsim(models, mcmc.out, Ccompile)
summary(fit2)
```

`init_param` *Build an Initial Value List for BayesSIM Models*

Description

`init_param` is a convenience helper that constructs a nested initial value list for a given combination of index vector and link function. It starts from the model-specific default prior, and then overwrites only those components for which the user supplies non-`NULL` arguments.

This allows users to modify selected hyper-parameters without having to know or manually reconstruct the full nested prior list structure.

Usage

```
init_param(
  indexprior,
  link,
  index = NULL,
  index_nu = NULL,
  index_psi = NULL,
  index_pi = NULL,
  link_beta = NULL,
  link_k = NULL,
  link_knots = NULL,
  link_lengthscale = NULL,
  link_amp = NULL,
  link_kappa = NULL,
  link_inv_lambda = NULL,
  sigma2 = NULL
)
```

Arguments

`indexprior`	Character scalar indicating the prior for the index. Typically one of `"fisher"`, `"sphere"`, `"polar"`, or `"spike"`. The valid options mirror those used in the corresponding model functions.
`link`	Character scalar indicating the link function family. Typically `"bspline"` for B-spline link functions or `"gp"` for Gaussian process link functions. The valid options mirror those used in the corresponding model functions.
`index`, `index_nu`, `index_psi`, `index_pi`	Optional initial values for index and related parameter values.
`link_beta`, `link_k`, `link_knots`, `link_lengthscale`, `link_amp`, `link_kappa`, `link_inv_lambda`	Optional initial values for components under link functions.
`sigma2`	Optional numeric scalar giving the initial value of σ^2.

Details

`init_param(indexprior, link)` can be used to obtain the random initial values list for the requested combination of index prior and link function. For any argument that is not `NULL`, the corresponding field in the nested prior list is overwritten.

The detailed meaning and recommended choices for each initial values depend on the specific model, index vector and link function. For those details, please refer to the documentation of the corresponding model-fitting functions.

Value

A nested list with components `index`, `link`, and `sigma2`.

See Also

bsFisher(), bsSphere(), bsPolar(), bsSpike(), gpFisher(), gpSphere(), gpPolar(), gpPolarHigh(), gpSpike()

Examples

```
## Default initial values for Fisher index + B-spline link:
i0 <- init_param("fisher", "bspline")

## Modify only a few initial values:
i1 <- init_param(
  indexprior = "fisher",
  link       = "bspline",
  index      = c(1, 0, 0),      # initial direction of the index
  link_beta  = rep(0, 21),      # initial values for spline coefficients
  sigma2     = 0.1              # initial value for sigma^2
)

## Example with GP link:
i2 <- init_param(
  indexprior        = "sphere",
  link              = "gp",
  link_lengthscale  = 0.2,      # initial GP length-scale
  link_amp          = 1.5,      # initial GP amplitude
  sigma2            = 1         # initial variance
)
```

---

nimTraceplot *Traceplot for* BayesSIM

Description

Provides traceplot for objects of BayesSIM.

Usage

```
nimTraceplot(x, origscale = FALSE, ...)
```

Arguments

x	A fitted object of BayesSIM or individual model.
origscale	Logical. If origscale is TRUE, you can see the index into original scale when you standardized the predictors.
...	Further arguments passed to plot.

Value

Traceplots for MCMC samples are displayed. By default, the index vector and error variance are only included in the summary. Extra monitor variables are included, if specified.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Check median index vector estimates with standard errors
coef(fit_one, method = "median", error = TRUE)

# Fitted index values of median prediction
fitted(fit_one, type = "linpred", method = "median")

# Residuals of median prediction
residuals(fit_one, method = "median")

# Summary of the model
summary(fit_one)

# Convergence diagnostics
nimTraceplot(fit_one)

# Goodness of fit
GOF(fit_one)

# Fitted plot
plot(fit_one)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
plot(pred)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

plot.bsim *Plot Method for* BayesSIM

Description

Produce diagnostic plots for a fitted Bayesian single-index model.

Usage

```
## S3 method for class 'bsim'
plot(x, method = c("mean", "median"), interval = TRUE, level = 0.95, ...)

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

Arguments

x	A fitted object of BayesSIM or individual model.
method	Character string specifying the summary used for the posterior fitted values. Options are "mean" or "median". Default is "mean".
interval	A logical value indicating whether a credible interval is included in the fitted plot. Default is TRUE.
level	Numeric value between 0 and 1 specifying the credible level. By default, level = 0.95 produces a 95% credible interval.
...	Additional arguments passed to underlying plotting functions.

Details

The function internally calls predict() on the fitted model object to obtain posterior summaries of $\hat{y}$. Predicted value of y is $\hat{f}(X'\hat{\theta})$.

- If interval = TRUE, the function requests posterior credible intervals and overlays them on the fitted curve.
- If interval = FALSE, only the posterior mean or median curve is drawn.

Value

The output consists of two plots:

1. **Observed vs Predicted plot**: a diagnostic scatter plot comparing actual outcomes with posterior fitted values to visually assess model fit.
2. **Fitted curve plot**: posterior $\hat{y}$ as a function of the estimated single index, optionally with $100 \times \alpha$ % credible intervals.

See Also

predict.bsim(), summary.bsim()

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Check median index vector estimates with standard errors
coef(fit_one, method = "median", error = TRUE)

# Fitted index values of median prediction
fitted(fit_one, type = "linpred", method = "median")

# Residuals of median prediction
residuals(fit_one, method = "median")

# Summary of the model
summary(fit_one)

# Convergence diagnostics
nimTraceplot(fit_one)

# Goodness of fit
GOF(fit_one)

# Fitted plot
plot(fit_one)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
plot(pred)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

---

plot.bsimSetup *Plot Method for* BayesSIM_setup

Description

Produce diagnostic plots for a raw data.

Usage

```
## S3 method for class 'bsimSetup'
plot(x, select = NULL, ...)
```

Arguments

x	A fitted object of BayesSIM or individual model.
select	A vector of index or name of variables in data.
...	Additional arguments passed to underlying plotting functions.

Details

The function displays scatter plots of the response variable against each predictor variable. Each panel includes a smoothed trend line to help assess the marginal relationship before fitting the model.

Value

A ggplot object is returned invisibly, allowing further modification if needed.

See Also

summary.bsimSetup()

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

models <- BayesSIM_setup(y ~ ., data = simdata2)

plot(models, select = c("X1", "X2"))
```

---

predict.bsim *Prediction Method for* BayesSIM

Description

Generate predictions from a fitted Bayesian single-index model.

Usage

```
## S3 method for class 'bsim'
predict(
  object,
  newdata = NULL,
  error.fit = FALSE,
  type = c("response", "latent", "index"),
  method = c("mean", "median"),
  interval = c("none", "credible"),
  n_draws = NULL,
  level = 0.95,
  ...
)

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

Arguments

object	A fitted object of BayesSIM or individual model.
newdata	Optional data frame for which predictions should be made. If NULL, predictions are returned for the training data.
error.fit	A logical value indicating whether uncertainty is required. Default is FALSE. If method == "mean", posterior standard deviation is calculated. If method == "median", posterior median absolute deviation is calculated.
type	Character string specifying the type of prediction. "response" is the default.
	"response" Posterior predictive summaries of the response $\boldsymbol{Y}$. This corresponds to draws from the posterior predictive distribution $\boldsymbol{Y}^{(m)} \sim N(f(\boldsymbol{X}\boldsymbol{\theta}^{(m)}), \sigma^{2(m)})$ and therefore incorporates both the uncertainty in the link function and the variability of the error term for each m^{th} MCMC sample.
	"latent" Posterior summaries of the latent mean structure $E(\boldsymbol{Y} \mid \boldsymbol{X}) = f^{(m)}(t^{(m)})$, where $t^{(m)} = \boldsymbol{X}\theta^{(m)}$. Unlike "response", it excludes the noise term and calculated by $f^{(m)}(\boldsymbol{X}\boldsymbol{\theta}^{(m)})$ for each m^{th} MCMC sample of $\boldsymbol{\theta}$.
	"index" Posterior summaries of the single index $\boldsymbol{t}^{(m)} = \boldsymbol{X}\boldsymbol{\theta}^{(m)}$.
method	Character string determining the posterior summary used for point predictions. Options are "mean" or "median". Default is "mean".
interval	Character string indicating whether a credible interval should be returned. Default is "none".
	"none" Return only point predictions.
	"credible" Return a $100 \times$ level$\%$ credible interval.
n_draws	Numeric value for designating number of draws for posterior predictive samples. If the value is NULL, function uses all MCMC samples from the fitted model.
level	Numeric value between 0 and 1 specifying the credible level. level = 0.95 yields a 95% credible interval. Default is 0.95.

...	Additional arguments.
x	An object of class "bsimPred" which is the result of predict.

Details

This method extracts MCMC posterior samples stored in a BayesSIM object and computes posterior summaries as follows:

- type = "response": the **posterior predictive response** $\boldsymbol{Y} \mid \boldsymbol{X}$
- type = "latent": the **latent link function** evaluation $E(\boldsymbol{Y} \mid \boldsymbol{X}) = f(\boldsymbol{X}\boldsymbol{\theta})$
- type = "index": the **single index** $\boldsymbol{X}\boldsymbol{\theta}$

The key distinction is that "response" incorporates the posterior variability of the error term ϵ, whereas "latent" represents the noiseless conditional expectation $E(\boldsymbol{Y} \mid \boldsymbol{X})$ computed directly from the link function and the posterior draws of $\boldsymbol{\theta}$.

When interval = "credible", the returned object includes lower and upper credible bounds computed via posterior quantiles for the chosen prediction scale.

If newdata is supplied, predictions are evaluated at the new covariate values by computing the corresponding posterior index $\boldsymbol{t} = \boldsymbol{X}\boldsymbol{\theta}$ and applying the link function.

Value

A list containing at least the following components:

fitted Posterior mean or median predictions. If error.fit = TRUE or interval = "credible", standard error and lower, upper bounds of the credible interval is also included.

truey True response value of test data. When true response value is not available, NULL is saved.

idxValue Linear index value is saved where θ is estimated by method.

posterior Posterior preditive or MCMC samples for prediction from the model.

level Credible level.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Check median index vector estimates with standard errors
coef(fit_one, method = "median", error = TRUE)

# Fitted index values of median prediction
fitted(fit_one, type = "linpred", method = "median")

# Residuals of median prediction
residuals(fit_one, method = "median")

# Summary of the model
```

```
summary(fit_one)

# Convergence diagnostics
nimTraceplot(fit_one)

# Goodness of fit
GOF(fit_one)

# Fitted plot
plot(fit_one)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
plot(pred)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

`prior_param` *Build a Prior List for BayesSIM Models*

Description

`prior_param` is a convenience helper that constructs a nested prior list for a given combination of index prior and link function. It starts from the model-specific default prior, and then overwrites only those components for which the user supplies non-`NULL` arguments.

This allows users to modify selected hyper-parameters without having to know or manually reconstruct the full nested prior list structure.

Usage

```
prior_param(
  indexprior,
  link,
  index_direction = NULL,
  index_dispersion = NULL,
```

```
  index_nu_r1 = NULL,
  index_nu_r2 = NULL,
  index_psi_alpha = NULL,
  index_sigma_theta = NULL,
  index_r1 = NULL,
  index_r2 = NULL,
  link_basis_df = NULL,
  link_basis_degree = NULL,
  link_basis_delta = NULL,
  link_knots_lambda_k = NULL,
  link_knots_maxknots = NULL,
  link_beta_mu = NULL,
  link_beta_cov = NULL,
  link_beta_tau = NULL,
  link_beta_Sigma0 = NULL,
  link_lengthscale_shape = NULL,
  link_lengthscale_rate = NULL,
  link_amp_a = NULL,
  link_amp_b = NULL,
  link_kappa_min = NULL,
  link_kappa_max = NULL,
  link_kappa_grid_width = NULL,
  link_inv_lambda_shape = NULL,
  link_inv_lambda_rate = NULL,
  sigma2_shape = NULL,
  sigma2_rate = NULL
)
```

Arguments

`indexprior`	Character scalar indicating the prior for the index. Typically one of `"fisher"`, `"sphere"`, `"polar"`, or `"spike"`. The valid options mirror those used in the corresponding model functions.
`link`	Character scalar indicating the link function family. Typically `"bspline"` for B-spline link functions or `"gp"` for Gaussian process link functions. The valid options mirror those used in the corresponding model functions.
`index_direction`, `index_dispersion`, `index_nu_r1`, `index_nu_r2`, `index_psi_alpha`, `index_sigma_theta`, `index_r1`, `index_r2`	Optional overrides for hyper-parameters related to the index prior.
`link_basis_df`, `link_basis_degree`, `link_basis_delta`	Optional overrides for the B-spline basis hyper-parameters, such as the effective degrees of freedom, spline degree, and penalty parameter.
`link_knots_lambda_k`, `link_knots_maxknots`	Optional overrides for the B-spline knot-selection hyper-parameters in, used for models with adaptive knot placement.
`link_beta_mu`, `link_beta_cov`, `link_beta_tau`, `link_beta_Sigma0`	Optional overrides for the prior on spline coefficients. The detailed interpretation of these hyper-parameters depends on the specific model and is described

in the documentation of each model-fitting function.

`link_lengthscale_shape`, `link_lengthscale_rate`
: Optional overrides for the hyper-parameters of the GP length-scale prior.

`link_amp_a`, `link_amp_b`
: Optional overrides for the hyper-parameters of the GP amplitude prior.

`link_kappa_min`, `link_kappa_max`, `link_kappa_grid_width`
: Optional overrides for the hyper-parameters in used in models with polar index and GP-type link, to control the grid or support for the concentration parameter κ in `gpPolar`.

`link_inv_lambda_shape`, `link_inv_lambda_rate`
: Optional overrides for spike-and-slab–type GP link priors.

`sigma2_shape`, `sigma2_rate`
: Optional overrides for the inverse-gamma prior on the observation variance σ^2.

Details

`prior_param(indexprior, link)` can be used to obtain the default prior list for the requested combination of index prior and link function. For any argument that is not `NULL`, the corresponding field in the nested prior list is overwritten.

The detailed meaning and recommended choices for each hyper-parameter depend on the specific model, prior of index vector and link function. For those details, please refer to the documentation of the corresponding model-fitting functions.

Value

A nested list with top-level elements `index`, `link`, and `sigma2`, suitable for passing to the `prior` argument of the various BayesSIM model fitting functions.

See Also

`bsFisher()`, `bsSphere()`, `bsPolar()`, `bsSpike()`, `gpFisher()`, `gpSphere()`, `gpPolar()`, `gpPolarHigh()`, `gpSpike()`

Examples

```
## Default prior for Fisher index + B-spline link:
p0 <- prior_param("fisher", "bspline")

## Modify only a few hyper-parameters:
p1 <- prior_param(
  indexprior          = "fisher",
  link                = "bspline",
  sigma2_shape        = 0.5,
  link_basis_df       = 30,
  index_direction     = c(1, 0, 0)
)
```

`residuals.bsim` *Extract Residuals from* `BayesSIM`

Description

Returns the model residuals based on the posterior fitted values of a

Usage

```
## S3 method for class 'bsim'
residuals(object, method = c("mean", "median"), ...)
```

Arguments

`object`	A fitted object of `BayesSIM` or individual model.
`method`	Character string specifying the summary statistic used to compute the fitted values. Options are `"mean"` or `"median"`. Default is `"mean"`.
`...`	Additional arguments passed to other methods.

Value

A numeric vector of residuals. ($\mathbf{r} = \mathbf{Y} - \hat{\mathbf{Y}}$) $\hat{\mathbf{Y}}$ can be mean or median of MCMC samples.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Check median index vector estimates with standard errors
coef(fit_one, method = "median", error = TRUE)

# Fitted index values of median prediction
fitted(fit_one, type = "linpred", method = "median")

# Residuals of median prediction
residuals(fit_one, method = "median")

# Summary of the model
summary(fit_one)

# Convergence diagnostics
nimTraceplot(fit_one)

# Goodness of fit
GOF(fit_one)
```

```
# Fitted plot
plot(fit_one)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
plot(pred)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

---

summary.bsim *Summarize* BayesSIM

Description

Provides a summary for BayesSIM.

Usage

```
## S3 method for class 'bsim'
summary(object, origscale = FALSE, ...)

## S3 method for class 'summary.bsim'
print(x, digits = 3, ...)
```

Arguments

object	A fitted object of BayesSIM or individual model.
origscale	Logical. If origscale is TRUE, you can see the index into original scale when you standardized the predictors.
...	Further arguments passed.
x	A summary output of BayesSIM or individual model.
digits	The minimum number of significant digits to be printed.

Details

A `list` of summary statistics for MCMC samples, including `data.frame` table for the results. Each row corresponds to a model parameter, and columns report the statistics.

Value

The function summarizes posterior MCMC samples by reporting key statistics, including:

- Posterior mean and median
- Empirical standard deviation, median absolute deviation
- 95% credible interval (lower and upper quantiles)
- Potential scale reduction factor (`Rhat`) for multiple chains
- Effective sample size (`ESS`)

By default, the index vector and error variance are only included in the summary. If variable selection methods are used, such as uniform sphere and spike-and-slab prior, the indicator vector (`nu`) is also included. Note that the potential scale reduction factor for `nu` can be reported as `NaN` or `Inf`, since the indicator rarely changes during the MCMC run.

If the model is fitted with single chain, both `all.chain` and `chain` have identical information.

See Also

`gelman.diag`, `effectiveSize`

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

# 1. One tool version
fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Check median index vector estimates with standard errors
coef(fit_one, method = "median", error = TRUE)

# Fitted index values of median prediction
fitted(fit_one, type = "linpred", method = "median")

# Residuals of median prediction
residuals(fit_one, method = "median")

# Summary of the model
summary(fit_one)

# Convergence diagnostics
nimTraceplot(fit_one)

# Goodness of fit
GOF(fit_one)
```

```
# Fitted plot
plot(fit_one)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
plot(pred)


# 2. Split version
models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes exactly the same as the class of "fit_one" object and apply generic functions.
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

---

summary.bsimPred *Summarize Predictions*

Description

Provides a summary for BayesSIM prediction.

Usage

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

## S3 method for class 'summary.bsimPred'
print(x, digits = 3, ...)
```

Arguments

object	An object of class "bsimPred" which is the result of predict.
...	Further arguments passed.
x	A summary output of class "bsimPred".
digits	The minimum number of significant digits to be printed.

Details

Performance of prediction with point predicted values.

Value

Metrics are including mean squared error (MSE), mean absolute error (MAE), mean bias, and coverage. In terms of coverage, it is only available when the "bsimPred" includes interval, corresponding to `interval = "credible"` options for `predict` function. It indicates mean coverage rate of data points whether confidence intervals include true response values. In this case, all chains are combined for the computation.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

fit_one <- BayesSIM(y ~ ., data = simdata2,
                    niter = 10000, nburnin = 5000, nchain = 2)

# Prediction with 95% credible interval at new data
newx <- data.frame(X1 = rnorm(10), X2 = rnorm(10), X3 = rnorm(10), X4 = rnorm(10))
pred <- predict(fit_one, newdata = newx, interval = "credible", level = 0.95)
summary(pred)
plot(pred)
```

---

summary.bsimSetup *Summarize* BayesSIM_setup

Description

Provides a summary for BayesSIM_setup.

Usage

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

## S3 method for class 'summary.bsimSetup'
print(x, digits = 1, ...)
```

Arguments

object	A fitted object of BayesSIM_setup.
...	Further arguments passed.
x	A summary output of BayesSIM_setup.
digits	The minimum number of significant digits to be printed.

Details

Summarize the model setup information of an object created by a setup function (e.g., BayesSIM_setup), including the model name, data dimensions, prior settings, and sampling options.

Value

An object of class `summary.bsimSetup`, which is a list containing:

`modelName`	The name of the fitted model.
`n`	The number of observations.
`p`	The number of predictors.
`prior`	A list of prior settings used in the model.
`samplingOptions`	A list of MCMC sampling options, including `niter`, `nburnin`, `thin`, and `nchain`.
`model`	The compiled nimble model object.

Examples

```
simdata2 <- data.frame(DATA1$X, y = DATA1$y)

models <- BayesSIM_setup(y ~ ., data = simdata2, nchain = 2)
summary(models)

Ccompile <- compileModelAndMCMC(models)
nimSampler <- getSampler(Ccompile)
initList <- getInit(models)
mcmc.out <- runMCMC(nimSampler, niter = 10000, nburnin = 5000, thin = 1,
                    nchains = 2, setSeed = TRUE, inits = initList,
                    summary = TRUE, samplesAsCodaMCMC = TRUE)

# "fit_split" becomes the class of "bsim".
fit_split <- as_bsim(models, mcmc.out, Ccompile)
```

Index

∗ **datasets**
concrete, 27
DATA1, 28

as.data.frame.bsimPred, 3
as_bsim, 4

BayesSIM, 5
BayesSIM_setup, *69*
BayesSIM_setup *(*BayesSIM*)*, 5
bsFisher, 8
bsFisher(), *7*, *55*, *64*
bsFisher_setup *(*bsFisher*)*, 8
bsPolar, 12
bsPolar(), *7*, *55*, *64*
bsPolar_setup *(*bsPolar*)*, 12
bsSphere, 16
bsSphere(), *7*, *55*, *64*
bsSphere_setup *(*bsSphere*)*, 16
bsSpike, 20
bsSpike(), *7*, *55*, *64*
bsSpike_setup *(*bsSpike*)*, 20
buildMCMC, *26*, *31*

coef.bsim, 24
compileModelAndMCMC, 25
compileNimble, *26*, *31*
concrete, 27
configureMCMC, *26*, *31*

DATA1, 28

effectiveSize, *67*

fitted.bsim, 28
formula, *6*, *9*, *13*, *17*, *21*, *38*, *42*, *47*, *51*

gelman.diag, *67*
getCModel *(*getFunction*)*, 30
getCSampler *(*getFunction*)*, 30
getFunction, 30
getInit, 31
getModel *(*getFunction*)*, 30
getModelDef, 32, *34*
getSampler *(*getFunction*)*, 30
getVarMonitor, *6*, *9*, *13*, *17*, *21*, 33, *33*, *38*, *43*, *47*, *51*
getWAIC, 34
GOF, 36
gpFisher, 37
gpFisher(), *7*, *55*, *64*
gpFisher_setup *(*gpFisher*)*, 37
gpPolar, 41
gpPolar(), *7*, *55*, *64*
gpPolar_setup *(*gpPolar*)*, 41
gpPolarHigh *(*gpPolar*)*, 41
gpPolarHigh(), *7*, *55*, *64*
gpPolarHigh_setup *(*gpPolar*)*, 41
gpSphere, 46
gpSphere(), *7*, *55*, *64*
gpSphere_setup *(*gpSphere*)*, 46
gpSpike, 50
gpSpike(), *7*, *55*, *64*
gpSpike_setup *(*gpSpike*)*, 50

init_param, *10*, *14*, *18*, *22*, *39*, *44*, *48*, *52*, 53

nimbleModel, *26*, *31*
nimTraceplot, 55

plot, *55*
plot.bsim, 57
plot.bsimPred *(*plot.bsim*)*, 57
plot.bsimSetup, 58
predict(), *57*
predict.bsim, *3*, 59
predict.bsim(), *57*
print.bsim *(*BayesSIM*)*, 5
print.bsimPred *(*predict.bsim*)*, 59
print.bsimSetup *(*BayesSIM*)*, 5
print.summary.bsim *(*summary.bsim*)*, 66

print.summary.bsimPred
(summary.bsimPred), 68
print.summary.bsimSetup
(summary.bsimSetup), 69
prior_param, *10*, *14*, *18*, *22*, *39*, *43*, *48*, *52*, 62

residuals.bsim, 65
runMCMC, *7*, *9*, *13*, *17*, *21*, *26*, *31*, *38*, *43*, *47*, *51*

summary.bsim, 66
summary.bsim(), *57*
summary.bsimPred, 68
summary.bsimSetup, 69
summary.bsimSetup(), *59*

waic, *35*