Package: BayesSUR (via r-universe)

October 15, 2026

Type Package

Title Bayesian Seemingly Unrelated Regression Models in High-Dimensional Settings

Version 2.3-3

URL https://github.com/mbant/BayesSUR

BugReports https://github.com/mbant/BayesSUR/issues

Description Bayesian seemingly unrelated regression with general variable selection and dense/sparse covariance matrix. The sparse seemingly unrelated regression is described in Bottolo et al. (2021) <doi:10.1111/rssc.12490>, the software paper is in Zhao et al. (2021) <doi:10.18637/jss.v100.i11>, and the model with random effects is described in Zhao et al. (2024) <doi:10.1093/jrsssc/qlad102>.

License MIT + file LICENSE

Copyright R function vertical.image.legend() is slightly modified from R package aqfig by Jenise Swall.

VignetteBuilder R.rsp, knitr, rmarkdown

Depends R (>= 3.5.0)

Encoding UTF-8

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, igraph, Matrix, tikzDevice, stats, utils, grDevices, graphics

Suggests R.rsp, knitr, rmarkdown, BDgraph, data.table, plyr, scrime

LazyData true

NeedsCompilation yes

SystemRequirements C++17

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libglpk-dev libpng-dev libxml2-dev

Repository https://ocbe-uio.r-universe.dev

Date/Publication 2026-05-17 19:01:01 UTC

RemoteUrl https://github.com/ocbe-uio/BayesSUR

RemoteRef HEAD

RemoteSha 4da5262d01658c4511b78c5dc18fa1630e86098f

RemoteSubdir BayesSUR

Contents

BayesSUR 2
BayesSUR_internal 7
coef.BayesSUR 8
elpd 9
exampleEQTL 10
exampleGDSC 15
fitted.BayesSUR 19
getEstimator 21
plot.BayesSUR 22
plotCPO 23
plotEstimator 25
plotGraph 27
plotManhattan 29
plotMCMCdiag 30
plotNetwork 31
predict.BayesSUR 34
print.BayesSUR 35
summary.BayesSUR 36
targetGene 37

Index **42**

`BayesSUR` *Fitting BayesSUR models*

Description

Main function of the package. Fits a range of models introduced in the package vignette `BayesSUR.pdf`. Returns an object of S3 class `BayesSUR`. There are three options for the prior on the residual covariance matrix (i.e., independent inverse-Gamma, inverse-Wishart and hyper-inverse Wishart) and three options for the prior on the latent indicator variable (i.e., independent Bernoulli, hotspot and Markov random field). So there are nine models in total. See details for their combinations.

Usage

```
BayesSUR(
  data = NULL,
  Y,
  X,
  X_0 = NULL,
  covariancePrior = "HIW",
  gammaPrior = "hotspot",
  betaPrior = "independent",
  nIter = 10000,
  burnin = 5000,
  nChains = 2,
  outFilePath = "",
  gammaSampler = "bandit",
  gammaInit = "R",
  mrfG = NULL,
  standardize = TRUE,
  standardize.response = TRUE,
  maxThreads = 1,
  tick = 1000,
  output_gamma = TRUE,
  output_beta = TRUE,
  output_Gy = TRUE,
  output_sigmaRho = TRUE,
  output_pi = TRUE,
  output_tail = TRUE,
  output_model_size = TRUE,
  output_model_visit = FALSE,
  output_CPO = FALSE,
  output_Y = TRUE,
  output_X = TRUE,
  hyperpar = list(),
  tmpFolder = "tmp/"
)
```

Arguments

`data`	a numeric matrix with variables on the columns and observations on the rows, if arguments `Y` and `X` (and possibly `X_0`) are vectors. Can be `NULL` if arguments `Y` and `X` (and possibly `X_0`) are numeric matrices
`Y`, `X`	vectors of indices (with respect to the data matrix) for the outcomes (`Y`) and the predictors to select (`X`) respectively; if the `data` argument is `NULL`, these needs to be numeric matrices containing the data instead, with variables on the columns and observations on the rows
`X_0`	vectors of indices (with respect to the data matrix) for the fixed predictors that are not selected, i.e. always included in the model; if the data argument is not provided, this needs to be a numeric matrix containing the data instead, with variables on the columns and observations on the rows

`covariancePrior`
: string indicating the prior for the covariance C; it has to be either `HIW` for the hyper-inverse-Wishar (which will result in a sparse covariance matrix), `IW` for the inverse-Wishart prior (dense covariance) or `IG` for independent inverse-Gamma on all the diagonal elements and 0 otherwise. See the details for the model specification

`gammaPrior`
: string indicating the gamma prior to use, either `hotspot` (default) for the Hotspot prior of Bottolo (2011), `MRF` for the Markov Random Field prior or `hierarchical` for a simpler hierarchical prior. See the details for the model specification

`betaPrior`
: string indicating the prior for regression coefficients; it has to be either `independent` for independent spike-and-slab priors (only slab part for `X_0` if specified), or `reGroup` for weakly normal priors for mandatory variables (random effects) and spike-and-slab priors for other variables of Zhao (2023)

`nIter`
: number of iterations for the MCMC procedure. Default 10000

`burnin`
: number of iterations to discard at the start of the chain. Default is 5000

`nChains`
: number of parallel tempered chains to run (default 2). The temperature is adapted during the burnin phase

`outFilePath`
: path to where the output files are to be written

`gammaSampler`
: string indicating the type of sampler for gamma, either `bandit` for the Thompson sampling inspired samper or `MC3` for the usual MC^3 sampler. See Russo et al.(2018) or Madigan and York (1995) for details

`gammaInit`
: gamma initialisation to either all-zeros (`0`), all ones (`1`), MLE-informed (`MLE`) or (default) randomly (`R`)

`mrfG`
: either a matrix or a path to the file containing (the edge list of) the G matrix for the MRF prior on gamma (if necessary)

`standardize`
: logical flag for X variable standardization. Default is `standardize=TRUE`. Coefficients are returned on the standardized scale

`standardize.response`
: logical flag for Y standardization. Default is `standardize.response=TRUE`

`maxThreads`
: maximum threads used for parallelization. Default is 1. Reproducibility of results with `set.seed()` is only guaranteed if `maxThreads=1`

`tick`
: an integer used for printing the iteration index and some updated parameters every tick-th iteration. Default is 1000

`output_gamma`
: allow (`TRUE`) or suppress (`FALSE`) the output for gamma. See the return value below for more information

`output_beta`
: allow (`TRUE`) or suppress (`FALSE`) the output for beta. See the return value below for more information

`output_Gy`
: allow (`TRUE`) or suppress (`FALSE`) the output for Gy. See the return value below for more information

`output_sigmaRho`
: allow (`TRUE`) or suppress (`FALSE`) the output for sigmaRho. See the return value below for more information

`output_pi`
: allow (`TRUE`) or suppress (`FALSE`) the output for pi. See the return value below for more information

`output_tail` allow (`TRUE`) or suppress (`FALSE`) the output for tail (hotspot tail probability). See the return value below for more information

`output_model_size` allow (`TRUE`) or suppress (`FALSE`) the output for model_size. See the return value below for more information

`output_model_visit` allow (`TRUE`) or suppress (`FALSE`) the output for all visited models over the MCMC iterations. Default is `FALSE`. See the return value below for more information

`output_CPO` allow (`TRUE`) or suppress (`FALSE`) the output for (scaled) conditional predictive ordinates (`*_CPO_out.txt`), CPO with joint posterior predictive of the response variables (`*_CPOsumy_out.txt`) and widely applicable information criterion (`*_WAIC_out.txt`). See the return value below for more information

`output_Y` allow (`TRUE`) or suppress (`FALSE`) the output for responses dataset Y

`output_X` allow (`TRUE`) or suppress (`FALSE`) the output for predictors dataset X

`hyperpar` a list of named hypeparameters to use instead of the default values. Valid names are mrf_d, mrf_e, a_sigma, b_sigma, a_tau, b_tau, nu, a_eta, b_eta, a_o, b_o, a_pi, b_pi, a_w and b_w. Their default values are a_w=2, b_w=5, a_omega=2, b_omega=1, a_o=2, b_o=p-2, a_pi=2, b_pi=1, nu=s+2, a_tau=0.1, b_tau=10, a_eta=0.1, b_eta=1, a_sigma=1, b_sigma=1, mrf_d=-3 and mrf_e=0.03. See the vignette for more information

`tmpFolder` the path to a temporary folder where intermediate data files are stored (will be erased at the end of the chain). It is specified relative to `outFilePath`

Details

The arguments `covariancePrior` and `gammaPrior` specify the model HRR, dSUR or SSUR with different gamma prior. Let γ_{jk} be latent indicator variable of each coefficient and C be covariance matrix of response variables. The nine models specified through the arguments `covariancePrior` and `gammaPrior` are as follows.

	γ_{jk}~Bernoulli	γ_{jk}~hotspot	γ~MRF
C~indep	HRR-B	HRR-H	HRR-M
C~IW	dSUR-B	dSUR-H	dSUR-M
C~HIW	SSUR-B	SSUR-H	SSUR-M

Value

An object of class `BayesSUR` is saved as `obj_BayesSUR.RData` in the output file, including the following components:

- status - the running status
- input - a list of all input parameters by the user
- output - a list of the all output filenames:
  - "`*_logP_out.txt`" - contains each row for the $1000t$-th iteration's log-likelihoods of parameters, i.e., Tau, Eta, JunctionTree, SigmaRho, O, Pi, Gamma, W, Beta and data conditional log-likelihood depending on the models.

  - "*_gamma_out.txt" - posterior mean of the latent indicator matrix.
  - "*_pi_out.txt" - posterior mean of the predictor effects (prospensity) by decomposing the probability of the latent indicator.
  - "*_hotspot_tail_p_out.txt" - posterior mean of the hotspot tail probability. Only available for the hotspot prior on the gamma.
  - "*_beta_out.txt" - posterior mean of the coefficients matrix.
  - "*_Gy_out.txt" - posterior mean of the response graph. Only available for the HIW prior on the covariance.
  - "*_sigmaRho_out.txt" - posterior mean of the transformed parameters. Not available for the IG prior on the covariance.
  - "*_model_size_out.txt" - contains each row for the$1000t$-th iteration's model sizes of the multiple response variables.
  - "*_model_visit_gy_out.txt" - contains each row for the nonzero indices of the vectorized estimated graph matrix for each iteration.
  - "*_model_visit_gamma_out.txt" - contains each row for the nonzero indices of the vectorized estimated gamma matrix for each iteration.
  - "*_CPO_out.txt" - the (scaled) conditional predictive ordinates (CPO).
  - "*_CPOsumy_out.txt" - the (scaled) conditional predictive ordinates (CPO) with joint posterior predictive of the response variables.
  - "*_WAIC_out.txt" - the widely applicable information criterion (WAIC).
  - "*_Y.txt" - responses dataset.
  - "*_X.txt" - predictors dataset.
  - "*_X0.txt" - fixed predictors dataset.

- call - the matched call.

References

Russo D, Van Roy B, Kazerouni A, Osband I, Wen Z (2018). *A tutorial on Thompson sampling.* Foundations and Trends in Machine Learning, 11: 1-96.

Madigan D, York J (1995). *Bayesian graphical models for discrete data.* International Statistical Review, 63: 215–232.

Bottolo L, Banterle M, Richardson S, Ala-Korpela M, Jarvelin MR, Lewin A (2020). *A computationally efficient Bayesian seemingly unrelated regressions model for high-dimensional quantitative trait loci discovery.* Journal of Royal Statistical Society: Series C, 70: 886-908.

Zhao Z, Banterle M, Bottolo L, Richardson S, Lewin A, Zucknick M (2021). *BayesSUR: An R package for high-dimensional multivariate Bayesian variable and covariance selection in linear regression.* Journal of Statistical Software, 100: 1–32.

Zhao Z, Banterle M, Lewin A, Zucknick M (2023). *Multivariate Bayesian structured variable selection for pharmacogenomic studies.* Journal of the Royal Statistical Society: Series C (Applied Statistics), qlad102.

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)
```

```
set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 5, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/", output_CPO = TRUE
)

## check output
# show the summary information
summary(fit)

# show the estimated beta, gamma and graph of responses Gy
plot(fit, estimator = c("beta", "gamma", "Gy"), type = "heatmap")

## Not run:
## Set up temporary work directory for saving a pdf figure
# td <- tempdir()
# oldwd <- getwd()
# setwd(td)

## Produce authentic math formulas in the graph
# plot(fit, estimator = c("beta", "gamma", "Gy"), type = "heatmap", fig.tex = TRUE)
# system(paste(getOption("pdfviewer"), "ParamEstimator.pdf"))
# setwd(oldwd)

## End(Not run)
```

`BayesSUR_internal` *BayesSUR_internal*

Description

Run a SUR Bayesian sampler – internal function

Arguments

`dataFile`	path to data file
`outFilePath`	path to where the output is to be written
`nIter`	number of iterations
`nChains`	number of parallel chains to run
	NOTE THAT THIS IS BASICALLY JUST A WRAPPER

coef.BayesSUR *coef method for class* BayesSUR

Description

Extract the posterior mean of the coefficients of a BayesSUR class object

Usage

```
## S3 method for class 'BayesSUR'
coef(object, beta.type = "marginal", Pmax = 0, ...)
```

Arguments

object	an object of class BayesSUR
beta.type	type of output beta. Default is marginal, giving marginal beta estimation. If beta.type="conditional", it gives beta estimation conditional on gamma=1.
Pmax	If Pmax=0.5 and beta.type="conditional", it gives median probability model betas. Default is 0.
...	other arguments

Value

Estimated coefficients are from an object of class BayesSUR. If the BayesSUR specified data standardization, the fitted values are base based on standardized data.

Examples

```
data("exampleQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check prediction
beta.hat <- coef(fit)
```

elpd *expected log pointwise predictive density*

Description

Measure the prediction accuracy by the elpd (expected log pointwise predictive density). The out-of-sample predictive fit can either be estimated by Bayesian leave-one-out cross-validation (LOO) or by widely applicable information criterion (WAIC) (Vehtari et al. 2017).

Usage

```
elpd(object, method = "LOO")
```

Arguments

object	an object of class BayesSUR
method	the name of the prediction accuracy index. Default is the "LOO" (Bayesian LOO estimate of out-of-sample predictive fit). The other index is the "WAIC" (widely applicable information criterion). For the HRR models, both "LOO" and "WAIC" are computed based on the multivate t-distribution of the posterior predictive rather than approximation of importance sampling.

Value

Return the predictiion accuracy measure from an object of class BayesSUR. It is elpd.loo if the argumnet method="LOO" and elpd.WAIC if method="WAIC".

References

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.

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/", output_CPO = TRUE
)

## check output
# print prediction accuracy elpd (expected log pointwise predictive density)
```

```
# by the Bayesian LOO estimate of out-of-sample predictive fit
elpd(fit, method = "LOO")
```

`exampleEQTL` *Simulated data set to mimic a small expression quantitative trait loci (eQTL) example*

Description

Simulated data set to mimic a small expression quantitative trait loci (eQTL) example, with p=150 single nucleotide polymorphisms (SNPs) as explanatory variables, s=10 gene expression features as response variables and data for n=100 observations. Loading the data will load the associated blockList object needed to fit the model with BayesSUR(). The R code for generating the simulated data is given in the Examples paragraph.

#importFrom BDgraph rgwish #importFrom gRbase mcsMAT #importFrom scrime simulateSNPs

Usage

```
exampleEQTL
```

Format

An object of class `list` of length 4.

Examples

```
# Load the eQTL sample dataset
data("exampleEQTL", package = "BayesSUR")
str(exampleEQTL)

## Not run:
# ===============
# The code below is to show how to generate the dataset "exampleEQTL.rda" above
# ===============

requireNamespace("BDgraph", quietly = TRUE)
requireNamespace("gRbase", quietly = TRUE)
requireNamespace("scrime", quietly = TRUE)

########################### Problem Dimensions
n <- 100
p <- 150
s <- 10

############################ Select a set of n x p (SNPs) covariates

## The synthetic data in the paper use a subset of the real SNPs as covariates,
# but as the NFBC66 dataset is confidential we'll use scrime to sample similar data
```

```
x <- scrime::simulateSNPs(c(n, 10), p, c(3, 2), prop.explain = c(0.9, 0.95))$data[1:n, ]
x <- cbind(rep(1, n), x)

#######################################################################

graph_pattern <- 2

snr <- 25

corr_param <- 0.9

### Create the underlying graph
if (graph_pattern == 1) {
  ### 1) Random but full
  G <- matrix(1, s, s)
  Prime <- list(c(1:s))
  Res <- Prime
  Sep <- list()
} else if (graph_pattern == 2) {
  ### 2) Block Diagonal structure
  Prime <- list(
    c(1:floor(s * 2 / 3)),
    c((floor(s * 2 / 3) + 1):(ceiling(s * 4 / 5) - 1)),
    c(ceiling(s * 4 / 5):s)
  )

  Res <- Prime
  Sep <- lapply(Res, function(x) which(x == -99))

  G <- matrix(0, s, s)
  for (i in Prime) {
    G[i, i] <- 1
  }
} else if (graph_pattern == 3) {
  ### 3) Decomposable model
  Prime <- list(
    c(1:floor(s * 5 / 12), ceiling(s * 9 / 10):s),
    c(floor(s * 2 / 9):(ceiling(s * 2 / 3) - 1)),
    c(ceiling(s * 2 / 3):(ceiling(s * 4 / 5) - 1)),
    c(ceiling(s * 4 / 5):s)
  )

  Sep <- list()
  H <- list()
  for (i in 2:length(Prime)) {
    H <- union(H, Prime[[i - 1]])
    Sep[[i - 1]] <- intersect(H, Prime[[i]])
  }

  Res <- list()
  Res[[1]] <- Prime[[1]]
  for (i in 2:length(Prime)) {
```

```
    Res[[i]] <- setdiff(Prime[[i]], Sep[[i - 1]])
  }

  G <- matrix(0, s, s)
  for (i in Prime) {
    G[i, i] <- 1
  }

  ## decomp check
  dimnames(G) <- list(1:s, 1:s)
  length(gRbase::mcsMAT(G - diag(s))) > 0
} else if (graph_pattern == 4) {
  ### 4) Non-decomposable model
  nblocks <- 5
  nElemPerBlock <- c(
    floor(s / 4), floor(s / 2) - 1 - floor(s / 4),
    ceiling(s * 2 / 3) - 1 - floor(s / 2), 7
  )
  nElemPerBlock <- c(nElemPerBlock, s - sum(nElemPerBlock))
  res <- 1:s
  blockIdx <- list()
  for (i in 1:nblocks) {
    # blockIdx[[i]] = sample(res,nElemPerBlock[i])
    blockIdx[[i]] <- res[1:nElemPerBlock[i]]
    res <- setdiff(res, blockIdx[[i]])
  }

  G <- matrix(0, s, s)
  ## add diagonal
  for (i in 1:nblocks) {
    G[blockIdx[[i]], blockIdx[[i]]] <- 1
  }
  ## add cycle
  G[blockIdx[[1]], blockIdx[[2]]] <- 1
  G[blockIdx[[2]], blockIdx[[1]]] <- 1
  G[blockIdx[[1]], blockIdx[[5]]] <- 1
  G[blockIdx[[5]], blockIdx[[1]]] <- 1
  G[blockIdx[[2]], blockIdx[[3]]] <- 1
  G[blockIdx[[3]], blockIdx[[2]]] <- 1
  G[blockIdx[[3]], blockIdx[[5]]] <- 1
  G[blockIdx[[5]], blockIdx[[3]]] <- 1

  ## decomp check
  dimnames(G) <- list(1:s, 1:s)
  length(gRbase::mcsMAT(G - diag(s))) > 0

  # Prime = blockIdx
  Res <- blockIdx ## this is not correct but not used in the non-decomp case
}

### Gamma Pattern
gamma <- matrix(0, p + 1, s)
gamma[1, ] <- 1
```

```
### 2) Extra Patterns

## outcomes (correlated in the decomp model) have some predictors in common
gamma[6:10, 6:9] <- 1

## outcomes (correlated in the decomp model) have some predictors in common
# gamma[16:20,14:15] = 1

## outcomes (sort-of correlated [pair-wise] in the decomp model)
# have predictors in common 6:15
gamma[26:30, 4:8] <- 1

## outcomes (NOT correlated in the decomp model) have predictors in common 16:17
gamma[36:40, c(3:5, 9:10)] <- 1

## these predictors are associated with ALL the outcomes
gamma[46:50, ] <- 1

combn11 <- combn(rep((6:9 - 1) * p, each = length(6:10 - 1)) + rep(6:10 - 1,
                 times = length(6:9)), 2)
combn31 <- combn(rep((4:8 - 1) * p, each = length(26:30 - 1)) + rep(26:30 - 1,
                 times = length(4:8)), 2)
combn32 <- combn(rep((4:8 - 1) * p, each = length(46:50 - 1)) + rep(46:50 - 1,
                 times = length(4:8)), 2)
combn41 <- combn(rep((3:5 - 1) * p, each = length(36:40 - 1)) + rep(36:40 - 1,
                 times = length(3:5)), 2)
combn42 <- combn(rep((3:5 - 1) * p, each = length(46:50 - 1)) + rep(46:50 - 1,
                 times = length(3:5)), 2)
combn51 <- combn(rep((9:10 - 1) * p, each = length(36:40 - 1)) + rep(36:40 - 1,
                 times = length(9:10)), 2)
combn52 <- combn(rep((9:10 - 1) * p, each = length(46:50 - 1)) + rep(46:50 - 1,
                 times = length(9:10)), 2)

Gmrf <- rbind(t(combn11), t(combn31), t(combn32), t(combn41), t(combn42), t(combn51), t(combn52))

## get for every correlated bunch in the decomposable model,

if (graph_pattern < 4) {
  # a different set of predictors
  for (i in 1:length(Prime)) {
    gamma[6:10 + (i + 6) * 10, Prime[[i]]] <- 1
  } ## for each Prime component

  ## for every Residual instead
  for (i in 1:length(Res)) {
    gamma[6:10 + (i + 10) * 10, Res[[i]]] <- 1
  }
} else {
  for (i in 1:length(Prime)) {
    gamma[6:10 + (i + 4) * 10, Prime[[i]]] <- 1
  } ## for each Prime component
```

```
  ## for every Residual instead
  for (i in 1:length(Res)) {
    gamma[6:10 + (i + 9) * 10, Res[[i]]] <- 1
  }
}

#### Sample the betas
sd_b <- 1
b <- matrix(rnorm((p + 1) * s, 0, sd_b), p + 1, s)

xb <- matrix(NA, n, s)

for (i in 1:s) {
  if (sum(gamma[, i]) > 1) {
    xb[, i] <- x[, gamma[, i] == 1] %*% b[gamma[, i] == 1, i]
  } else {
    xb[, i] <- rep(1, n) * b[1, i]
  }
}

## Sample the variance
v_r <- mean(diag(var(xb))) / snr

nu <- s + 1

M <- matrix(corr_param, s, s)
diag(M) <- rep(1, s)

P <- BDgraph::rgwish(n = 1, adj = G, b = 3, D = v_r * M)

var <- solve(P)

factor <- 10
factor_min <- 0.01
factor_max <- 1000
count <- 0
maxit <- 10000

factor_prev <- 1

repeat{
  var <- var / factor * factor_prev

  ### Sample the errors and the Ys
  cVar <- chol(as.matrix(var))
  # err = matrix(rnorm(n*s),n,s) %*% cVar
  err <- matrix(rnorm(n * s, sd = 0.5), n, s) %*% cVar
  y <- xb + err

  ## Reparametrisation ( assuming PEO is 1:s )
  cVar <- t(cVar) # make it lower-tri
  S <- diag(diag(cVar))
```

```
  sigma <- S * S
  L <- cVar %*% solve(S)
  rho <- diag(s) - solve(L)

  ### S/N Ratio
  emp_snr <- mean(diag(var(xb) %*% solve(sigma)))
  emp_g_snr <- mean(diag(var((err) %*% t(rho)) %*% solve(sigma)))

  ##############

  if (abs(emp_snr - snr) < (snr / 10) | count > maxit) {
    break
  } else {
    if (emp_snr < snr) { # increase factor
      factor_min <- factor
    } else { # decrease factor
      factor_max <- factor
    }
    factor_prev <- factor
    factor <- (factor_min + factor_max) / 2
  }
  count <- count + 1
}

#################
colnames(y) <- paste("GEX", 1:ncol(y), sep = "")
colnames(G) <- colnames(y)
Gy <- G
gamma <- gamma[-1, ]
mrfG <- Gmrf[!duplicated(Gmrf), ]
data <- cbind(y, x[, -1]) # leave out the intercept because is coded inside already

exampleEQTL <- list(data = data, blockList = list(1:s, s + 1:p))

## Write data file to the user's directory by save()

## End(Not run)
```

---

exampleGDSC *Preprocessed data set to mimic a small pharmacogenomic example*

Description

Preprocessed data set to mimic a small pharmacogenetic example from the Genomics of Drug Sensitivity in Cancer (GDSC) database, with p=850 gene features as explanatory variables, s=7 drugs sensitivity data as response variables and data for n=498 cell lines. Gene features include p1=343 gene expression features (GEX), p2=426 by copy number variations (CNV) and p3=68 mutated genes (MUT). Loading the data will load the associated blockList (and mrfG) objects needed to fit

the model with BayesSUR(). The R code for generating the simulated data is given in the Examples paragraph.

#importFrom plyr mapvalues #importFrom data.table like

Usage

```
exampleGDSC
```

Format

An object of class `list` of length 3.

Examples

```
# Load the GDSC sample dataset
data("exampleGDSC", package = "BayesSUR")
str(exampleGDSC)

## Not run:
# ===============
# This code below is to do preprocessing of GDSC data and obtain the complete dataset
# "exampleGDSC.rda" above. The user needs load the datasets from
# https://cellmodelpassports.sanger.ac.uk/downloads.
# But downloading and transforming the three used datasets below to *.csv files first.
# ===============

requireNamespace("plyr", quietly = TRUE)
requireNamespace("data.table", quietly = TRUE)


features <- data.frame(read.csv("/gdsc_en_input_w5.csv", head = T))
names.fea <- strsplit(rownames(features), "")
features <- t(features)
p <- c(13321, 13747 - 13321, 13818 - 13747)
Cell.Line <- rownames(features)
features <- data.frame(Cell.Line, features)

ic50_00 <- data.frame(read.csv("gdsc_drug_sensitivity_fitted_data_w5.csv", head = T))
ic50_0 <- ic50_00[, c(1, 4, 7)]
drug.id <- data.frame(read.csv("gdsc_tissue_output_w5.csv", head = T))[, c(1, 3)]
drug.id2 <- drug.id[!duplicated(drug.id$drug.id), ]
# delete drug.id=1066 since ID1066 and ID156 both correspond drug AZD6482,
# and no ID1066 in the "suppl.Data1" by Garnett et al. (2012)
drug.id2 <- drug.id2[drug.id2$drug.id != 1066, ]
drug.id2$drug.name <- as.character(drug.id2$drug.name)
drug.id2$drug.name <- substr(drug.id2$drug.name, 1, nchar(drug.id2$drug.name) - 6)
drug.id2$drug.name <- gsub(" ", "-", drug.id2$drug.name)

ic50 <- ic50_0
# mapping the drug_id to drug names in drug sensitivity data set
ic50$drug_id <- plyr::mapvalues(ic50$drug_id, from = drug.id2[, 2], to = drug.id2[, 1])
colnames(ic50) <- c("Cell.Line", "compound", "IC50")
```

```
# transform drug sensitivity overall cell lines to a data matrix
y0 <- reshape(ic50, v.names = "IC50", timevar = "compound",
              idvar = "Cell.Line", direction = "wide")
y0$Cell.Line <- gsub("-", ".", y0$Cell.Line)

# ===============
# select nonmissing pharmacological data
# ===============
y00 <- y0
m0 <- dim(y0)[2] - 1
eps <- 0.05
# r1.na is better to be not smaller than r2.na
r1.na <- 0.3
r2.na <- 0.2
k <- 1
while (sum(is.na(y0[, 2:(1 + m0)])) > 0) {
  r1.na <- r1.na - eps / k
  r2.na <- r1.na - eps / k
  k <- k + 1
  ## select drugs with <30% (decreasing with k) missing data overall cell lines
  na.y <- apply(y0[, 2:(1 + m0)], 2, function(xx) sum(is.na(xx)) / length(xx))
  while (sum(na.y < r1.na) < m0) {
    y0 <- y0[, -c(1 + which(na.y >= r1.na))]
    m0 <- sum(na.y < r1.na)
    na.y <- apply(y0[, 2:(1 + m0)], 2, function(xx) sum(is.na(xx)) / length(xx))
  }

  ## select cell lines with treatment of at least 80% (increasing with k) drugs
  na.y0 <- apply(y0[, 2:(1 + m0)], 1, function(xx) sum(is.na(xx)) / length(xx))
  while (sum(na.y0 < r2.na) < (dim(y0)[1])) {
    y0 <- y0[na.y0 < r2.na, ]
    na.y0 <- apply(y0[, 2:(1 + m0)], 1, function(xx) sum(is.na(xx)) / length(xx))
  }
  num.na <- sum(is.na(y0[, 2:(1 + m0)]))
  message("#{NA}=", num.na, "\n", "r1.na =", r1.na, ", r2.na =", r2.na, "\n")
}

# ===============
# combine drug sensitivity, tissues and molecular features
# ===============
yx <- merge(y0, features, by = "Cell.Line")
names.cell.line <- yx$Cell.Line
names.drug <- colnames(yx)[2:(dim(y0)[2])]
names.drug <- substr(names.drug, 6, nchar(names.drug))
# numbers of gene expression features, copy number festures and muatation features
p <- c(13321, 13747 - 13321, 13818 - 13747)
num.nonpen <- 13
yx <- data.matrix(yx[, -1])
y <- yx[, 1:(dim(y0)[2] - 1)]
x <- cbind(yx[, dim(y0)[2] - 1 + sum(p) + 1:num.nonpen], yx[, dim(y0)[2] - 1 + 1:sum(p)])

# delete genes with only one mutated cell line
```

```
x <- x[,
 -c(num.nonpen + p[1] + p[2] + which(colSums(x[, num.nonpen + p[1] + p[2] + 1:p[3]]) <= 1))]
p[3] <- ncol(x) - num.nonpen - p[1] - p[2]

GDSC <- list(
  y = y, x = x, p = p, num.nonpen = num.nonpen, names.cell.line = names.cell.line,
  names.drug = names.drug
)


## ================
## ================
## select a small set of drugs
## ================
## ================

name_drugs <- c(
  "Methotrexate", "RDEA119", "PD-0325901", "CI-1040", "AZD6244", "Nilotinib",
  "Axitinib"
)

# extract the drugs' pharmacological profiling and tissue dummy
YX0 <- cbind(GDSC$y[, colnames(GDSC$y) %in% paste("IC50.", name_drugs, sep = "")]
[, c(1, 3, 6, 4, 7, 2, 5)], GDSC$x[, 1:GDSC$num.nonpen])
colnames(YX0) <- c(name_drugs, colnames(GDSC$x)[1:GDSC$num.nonpen])
# extract the genetic information of CNV & MUT
X23 <- GDSC$x[, GDSC$num.nonpen + GDSC$p[1] + 1:(p[2] + p[3])]
colnames(X23)[1:p[2]] <- paste(substr(
  colnames(X23)[1:p[2]], 1,
  nchar(colnames(X23)[1:p[2]]) - 3
), ".CNV", sep = "")

# locate all genes with CNV or MUT information
name_genes_duplicate <- c(
  substr(colnames(X23)[1:p[2]], 1, nchar(colnames(X23)[1:p[2]]) - 4),
  substr(colnames(X23)[p[2] + 1:p[3]], 1, nchar(colnames(X23)[p[2] + 1:p[3]]) - 4)
)
name_genes <- name_genes_duplicate[!duplicated(name_genes_duplicate)]

# select the GEX which have the common genes with CNV or MUT
X1 <-
 GDSC$x[, GDSC$num.nonpen + which(colnames(GDSC$x)[GDSC$num.nonpen + 1:p[1]] %in% name_genes)]

p[1] <- ncol(X1)
X1 <- log(X1)

# summary the data information
exampleGDSC <- list(data = cbind(YX0, X1, X23))
exampleGDSC$blockList <- list(
  1:length(name_drugs), length(name_drugs) + 1:GDSC$num.nonpen,
  ncol(YX0) + 1:sum(p)
)
```

```
# ========================
# construct the G matrix: edge potentials in the MRF prior
# ========================

# edges between drugs: Group1 ("RDEA119","17-AAG","PD-0325901","CI-1040" and "AZD6244")
# indexed as (2:5)
# http://software.broadinstitute.org/gsea/msigdb/cards/KEGG_MAPK_SIGNALING_PATHWAY
pathway_genes <- read.table("MAPK_pathway.txt")[[1]]
Idx_Pathway1 <- which(c(colnames(X1), name_genes_duplicate) %in% pathway_genes)
Gmrf_Group1Pathway1 <- t(combn(rep(Idx_Pathway1, each = length(2:5)) +
  rep((2:5 - 1) * sum(p), times = length(Idx_Pathway1)), 2))

# edges between drugs: Group2 ("Nilotinib","Axitinib") indexed as (6:7)
# delete gene ABL2
Idx_Pathway2 <- which(c(colnames(X1), name_genes_duplicate) %like% "BCR" |
  c(colnames(X1), name_genes_duplicate) %like% "ABL")[-c(3, 5)]
Gmrf_Group2Pathway2 <- t(combn(rep(Idx_Pathway2, each = length(6:7)) +
  rep((6:7 - 1) * sum(p), times = length(Idx_Pathway2)), 2))

# edges between the common gene in different data sources
Gmrf_CommonGene <- NULL
list_CommonGene <- list(0)
k <- 1
for (i in 1:length(name_genes)) {
  Idx_CommonGene <- which(c(colnames(X1), name_genes_duplicate) == name_genes[i])
  if (length(Idx_CommonGene) > 1) {
    Gmrf_CommonGene <- rbind(Gmrf_CommonGene,
    t(combn(rep(Idx_CommonGene, each = length(name_drugs))
    + rep((1:length(name_drugs) - 1) * sum(p), times = length(Idx_CommonGene)), 2)))
    k <- k + 1
  }
}
Gmrf_duplicate <- rbind(Gmrf_Group1Pathway1, Gmrf_Group2Pathway2, Gmrf_CommonGene)
Gmrf <- Gmrf_duplicate[!duplicated(Gmrf_duplicate), ]
exampleGDSC$mrfG <- Gmrf

# create the target gene names of the two groups of drugs
targetGenes1 <- matrix(Idx_Pathway1, nrow = 1)
colnames(targetGenes1) <- colnames(exampleGDSC$data)[seq_along(targetGene$group1)]
targetGenes2 <- matrix(Idx_Pathway2, nrow = 1)
colnames(targetGenes2) <- colnames(exampleGDSC$data)[seq_along(targetGene$group2)]

targetGene <- list(group1 = targetGenes1, group2 = targetGenes2)

## Write data file exampleGDSC.rda to the user's directory by save()

## End(Not run)
```

`fitted.BayesSUR` *get fitted responses*

Description

Return the fitted response values that correspond to the posterior mean estimates from a BayesSUR class object.

Usage

```
## S3 method for class 'BayesSUR'
fitted(object, Pmax = 0, beta.type = "marginal", ...)
```

Arguments

object	an object of class BayesSUR
Pmax	valid if beta.type="conditional". If beta.type="conditional" and Pmax=0.5, it gives median probability model betas. Default is 0
beta.type	type of estimated beta for the fitted model. Default is marginal, giving marginal beta estimation. If beta.type="conditional", it gives beta estimation conditional on gamma=1
...	other arguments

Value

Fitted values extracted from an object of class BayesSUR. If the BayesSUR specified data standardization, the fitted values are base based on standardized data.

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check fitted values
fitted.val <- fitted(fit)
```

getEstimator *extract the posterior mean of parameters*

Description

Extract the posterior mean of the parameters of a BayesSUR class object.

Usage

```
getEstimator(object, estimator = "gamma", Pmax = 0, beta.type = "marginal")
```

Arguments

object	an object of class BayesSUR
estimator	the name of one estimator. Default is the latent indicator estimator "gamma". Other options "beta", "Gy", "CPO" and "logP" correspond the marginal (conditional) coefficient matrix if beta.type="marginal"("conditional"), response graph and conditional predictive ordinate (CPO) respectively
Pmax	threshold that truncate the estimator "gamma" or "Gy". Default is 0. If Pmax=0.5 and beta.type="conditional", it gives median probability model betas
beta.type	the type of output beta. Default is marginal, giving marginal beta estimation. If beta.type="conditional", it gives beta estimation conditional on gamma=1

Value

Return the estimator from an object of class BayesSUR. It is a matrix if the length of argument marginal is greater than 1. Otherwise, it is a list

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check output
# extract the posterior mean of the coefficients matrix
beta_hat <- getEstimator(fit, estimator = "beta")
```

`plot.BayesSUR` *create a selection of plots*

Description

plot method for class `BayesSUR`. This is the main plot function to be called by the user. This function calls one or several of the following functions: `plotEstimator()`, `plotGraph()`, `plotMCMCdiag()`, `plotManhattan()`, `plotNetwork()`, `plotCPO()`.

Usage

```
## S3 method for class 'BayesSUR'
plot(x, estimator = NULL, type = NULL, ...)
```

Arguments

`x`	an object of class `BayesSUR`
`estimator`	It is in `c(NULL, 'beta', 'gamma', 'Gy', 'logP', 'CPO')` and works by combining with argument `type`. • If `estimator` is in `c("beta", "gamma", "Gy")` and argument `type="heatmap"`, it prints heatmaps of the specified estimator in `estimator` by a call to to function `plotEstimator()` for more other arguments. • If `estimator="Gy"` and argument `type="graph"`, it prints a structure graph of `"Gy"` by a call to function `plotGraph()` for more other arguments. • If `estimator=c("gamma", "Gy")` and argument `type="network"`, it prints the estimated network between the response variables and predictors with nonzero coefficients by a call to function `plotMCMCdiag()` for more other arguments. • If `estimator=NULL` (default) and `type=NULL` (default), it interactively prints the plots of estimators (i.e., beta, gamma and (or) Gy), response graph Gy, network, Manhattan and MCMC diagnostics.
`type`	It is one of `NULL`, `"heatmap"`, `"graph"`, `"network"`, `"Manhattan"` and `"diagnostics"`, and works by combining with argument `estimator`. • If `type="Manhattan"` and argument `estimator="gamma"`, it prints Manhattan-like plots for marginal posterior inclusion probabilities (mPIP) and numbers of associated response variables for individual predictors by a call to function `plotManhattan()` for more other arguments. • If `type="diagnostics"` and argument `estimator="logP"` it shows trace plots and diagnostic density plots of a fitted model by a call to function `plotMCMCdiag()` for more other arguments. • If `type="diagnostics"` and argument `estimator="CPO"`, it shows the conditional predictive ordinate (CPO) for each individual of a fitted model by a call to function `plotCPO()` for more other arguments.
`...`	other arguments, see functions `plotEstimator()`, `plotGraph()`, `plotNetwork()`, `plotManhattan()`, `plotMCMCdiag()` or `plotCPO()`

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 2, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check output
## Not run:
## Show the interactive plots. Note that it needs at least 2000*(nbloc+1) iterations
## for the diagnostic plots where nbloc=3 by default
# plot(fit)

## End(Not run)

## plot heatmaps of the estimated beta, gamma and Gy
plot(fit, estimator = c("beta", "gamma", "Gy"), type = "heatmap")

## plot estimated graph of responses Gy
plot(fit, estimator = "Gy", type = "graph")

## plot network between response variables and associated predictors
plot(fit, estimator = c("gamma", "Gy"), type = "network")

## print Manhattan-like plots
plot(fit, estimator = "gamma", type = "Manhattan")

## print MCMC diagnostic plots
#plot(fit, estimator = "logP", type = "diagnostics")
```

`plotCPO` *plot conditional predictive ordinate*

Description

Plot the conditional predictive ordinate (CPO) for each individual of a fitted model generated by `BayesSUR` which is a `BayesSUR` object. CPO is a handy posterior predictive check because it may be used to identify outliers, influential observations, and for hypothesis testing across different non-nested models (Gelfand 1996).

Usage

```
plotCPO(
  x,
  outlier.mark = TRUE,
  outlier.thresh = 0.01,
  scale.CPO = TRUE,
  x.loc = FALSE,
  axis.label = NULL,
  las = 0,
  cex.axis = 1,
  mark.pos = c(0, -0.01),
  mark.color = 2,
  mark.cex = 0.8,
  xlab = "Observations",
  ylab = NULL,
  ...
)
```

Arguments

x	an object of class BayesSUR
outlier.mark	mark the outliers with the response names. The default is FALSE
outlier.thresh	threshold for the CPOs. The default is 0.01.
scale.CPO	scaled CPOs which is divided by their maximum. The default is TRUE
x.loc	a vector of features distance
axis.label	a vector of predictor names which are shown in CPO plot. The default is NULL only showing the indices. The value "auto" show the predictor names from the original data.
las	graphical parameter of plot.default
cex.axis	graphical parameter of plot.default
mark.pos	location of the marked text relative to the point
mark.color	color of the marked text. The default color is red
mark.cex	font size of the marked text. The default font size is 0.8
xlab	a title for the x axis
ylab	a title for the y axis
...	other arguments

Details

The default threshold for the CPOs to detect the outliers is 0.01 by Congdon (2005). It can be tuned by the argument outlier.thresh.

References

Statisticat, LLC (2013). *Bayesian Inference.* Farmington, CT: Statisticat, LLC.

Gelfand A. (1996). *Model Determination Using Sampling Based Methods*. In Gilks W., Richardson S., Spiegelhalter D. (eds.), Markov Chain Monte Carlo in Practice, pp. 145–161. Chapman & Hall, Boca Raton, FL.

Congdon P. (2005). *Bayesian Models for Categorical Data*. John Wiley & Sons, West Sussex, England.

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/", output_CPO = TRUE
)

## check output
# plot the conditional predictive ordinate (CPO)
plotCPO(fit)
```

---

plotEstimator *plot heatmap of estimators*

Description

Plot the posterior mean estimators from a BayesSUR class object, including the coefficients beta, latent indicator variable gamma and graph of responses.

Usage

```
plotEstimator(
  x,
  estimator = NULL,
  colorScale.gamma = grey((100:0)/100),
  colorScale.beta = c("blue", "white", "red"),
  legend.cex.axis = 1,
  name.responses = NA,
  name.predictors = NA,
  xlab = "",
  ylab = "",
```

```
  fig.tex = FALSE,
  output = "ParamEstimator",
  header = "",
  header.cex = 2,
  tick = FALSE,
  mgp = c(2.5, 1, 0),
  cex.main = 1.5,
  title.beta = NA,
  title.gamma = NA,
  title.Gy = NA,
  beta.type = "marginal",
  Pmax = 0,
  ...
)
```

Arguments

x	an object of class BayesSUR
estimator	print the heatmap of estimators. The value "beta" is for the estimated coefficients matrix, "gamma" for the latent indicator matrix and "Gy" for the graph of responses
colorScale.gamma	value palette for gamma
colorScale.beta	a vector of three colors for diverging color schemes
legend.cex.axis	magnification of axis annotation relative to cex
name.responses	a vector of the response names. The default is NA only to show the locations. The value "auto" show the response names from the orginal data.
name.predictors	a vector of the predictor names. The default is NA only to show the locations. The value "auto" show the predictor names from the orginal data.
xlab	a title for the x axis
ylab	a title for the y axis
fig.tex	print the figure through LaTex. Default is FALSE
output	the file name of printed figure
header	the main title
header.cex	size of the main title for all estimators
tick	a logical value specifying whether tickmarks and an axis line should be drawn. Default is FALSE
mgp	the margin line (in mex units) for the axis title, axis labels and axis line
cex.main	size of the title for each estimator
title.beta	a title for the printed "beta"
title.gamma	a title for the printed "gamma"

title.Gy	a title for the printed "Gy"
beta.type	the type of output beta. Default is marginal, giving marginal beta estimation. If beta.type="conditional", it gives beta estimation conditional on gamma=1
Pmax	threshold that truncate the estimator "gamma" or "Gy". Default is 0. If Pmax=0.5 and beta.type="conditional", it gives median probability model betas
...	other arguments

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check output
# Plot the estimators from the fitted object
plotEstimator(fit, estimator = c("beta", "gamma", "Gy"))

## Not run:
## Set up temporary work directory for saving a pdf figure
# td <- tempdir()
# oldwd <- getwd()
# setwd(td)

## Produce authentic math formulas in the graph
# plotEstimator(fit, estimator = c("beta", "gamma", "Gy"), fig.tex = TRUE)
# system(paste(getOption("pdfviewer"), "ParamEstimator.pdf"))
# setwd(oldwd)

## End(Not run)
```

---

plotGraph *plot graph for response variables*

Description

Plot the estimated graph for multiple response variables from a BayesSUR class object.

Usage

```
plotGraph(
  x,
  Pmax = 0.5,
  main = "Estimated graph of responses",
  edge.width = 2,
  edge.weight = FALSE,
  vertex.label = NULL,
  vertex.label.color = "black",
  vertex.size = 30,
  vertex.color = "dodgerblue",
  vertex.frame.color = NA,
  ...
)
```

Arguments

x	either an object of class BayesSUR (default) or a symmetric numeric matrix representing an adjacency matrix for a given graph structure. If x is an adjacency matrix, argument main="Given graph of responses" by default.
Pmax	a value for thresholding the learning structure matrix of multiple response variables. Default is 0.5
main	an overall title for the plot
edge.width	edge width. Default is 2
edge.weight	draw weighted edges after thresholding at 0.5. The default value FALSE is not to draw weighted edges
vertex.label	character vector used to label the nodes
vertex.label.color	label color. Default is "black"
vertex.size	node size. Default is 30
vertex.color	node color. Default is "dodgerblue"
vertex.frame.color	node color. Default is "NA"
...	other arguments

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
```

```
)

## check output
# show the graph relationship between responses
plotGraph(fit, estimator = "Gy")
```

`plotManhattan` *plot Manhattan-like plots*

Description

Plot Manhattan-like plots for marginal posterior inclusion probabilities (mPIP) and numbers of responses of association for predictors of a `BayesSUR` class object.

Usage

```
plotManhattan(
  x,
  manhattan = c("mPIP", "numResponse"),
  x.loc = FALSE,
  axis.label = "auto",
  mark.responses = NULL,
  xlab1 = "Predictors",
  ylab1 = "mPIP",
  xlab2 = "Predictors",
  ylab2 = "No. of responses",
  threshold = 0.5,
  las = 0,
  cex.axis = 1,
  mark.pos = c(0, 0),
  mark.color = 2,
  mark.cex = 0.8,
  header = "",
  ...
)
```

Arguments

`x`	an object of class `BayesSUR`
`manhattan`	value(s) in `c('mPIP', 'numResponse')`. `manhattan='mPIP'` shows the Manhattan-like plot of the marginal posterior inclusion probabilities (mPIP). `manhattan='numResponse'` shows the Manhattan-like plot of the number of responses. The default is to show both figures.
`x.loc`	a vector of features distance

axis.label	a vector of predictor names which are shown in the Manhattan-like plot. The value "NULL" only showing the indices. The default "auto" show the predictor names from the original data.
mark.responses	a vector of response names which are shown in the Manhattan-like plot for the mPIP
xlab1	a title for the x axis of Manhattan-like plot for the mPIP
ylab1	a title for the y axis of Manhattan-like plot for the mPIP
xlab2	a title for the x axis of Manhattan-like plot for the numbers of responses
ylab2	a title for the y axis of Manhattan-like plot for the numbers of responses
threshold	threshold for showing number of response variables significantly associated with each feature
las	graphical parameter of plot.default
cex.axis	graphical parameter of plot.default
mark.pos	the location of the marked text relative to the point
mark.color	the color of the marked text. The default color is red.
mark.cex	the fontsize of the marked text. The default fontsize is 0.8
header	the main title
...	other arguments

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check output
# show the Manhattan-like plots
plotManhattan(fit)
```

---

plotMCMCdiag *plot MCMC diagnostic plots*

Description

Show trace plots and diagnostic density plots of a fitted model object of class BayesSUR.

Usage

```
plotMCMCdiag(x, nbloc = 3, HIWg = NULL, header = "", ...)
```

Arguments

x	an object of class BayesSUR
nbloc	number of splits for the last half iterations after substracting burn-in length
HIWg	diagnostic plot of the response graph. Default is NULL. HIW="degree" prints the diagnostic of the degrees of response nodes. HIW="edges" prints the diagnostic of every edge between two responses. HIW="lik" prints the diagnostic of the posterior likelihoods of the hyperparameters related to the response relationships
header	the main title
...	other arguments for the plots of the log-likelihood and model size

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check output
plotMCMCdiag(fit)
```

---

plotNetwork *plot network representation of the associations between responses and predictors*

Description

Plot the network representation of the associations between responses and predictors, based on the estimated gamma matrix and graph of responses from a "BayesSUR" class object.

Usage

```
plotNetwork(
  x,
  includeResponse = NULL,
  excludeResponse = NULL,
```

```
  includePredictor = NULL,
  excludePredictor = NULL,
  MatrixGamma = NULL,
  PmaxPredictor = 0.5,
  PmaxResponse = 0.5,
  nodesizePredictor = 2,
  nodesizeResponse = 15,
  no.isolates = FALSE,
  lineup = 1.2,
  gray.alpha = 0.6,
  edgewith.response = 5,
  edgewith.predictor = 2,
  edge.weight = FALSE,
  label.predictor = NULL,
  label.response = NULL,
  color.predictor = NULL,
  color.response = NULL,
  name.predictors = NULL,
  name.responses = NULL,
  color.edge.positive = "gray",
  color.edge.negative = "gray",
  vertex.frame.color = NA,
  layoutInCircle = FALSE,
  header = "",
  beta.type = "conditional",
  ...
)
```

Arguments

x	an object of class BayesSUR
includeResponse	A vector of the response names which are shown in the network
excludeResponse	A vector of the response names which are not shown in the network
includePredictor	A vector of the predictor names which are shown in the network
excludePredictor	A vector of the predictor names which are not shown in the network
MatrixGamma	A matrix or dataframe of the latent indicator variable. Default is NULL and to extrate it from object of class inheriting from an object of class BayesSUR
PmaxPredictor	cutpoint for thresholding the estimated latent indicator variable. Default is 0.5
PmaxResponse	cutpoint for thresholding the learning structure matrix of multiple response variables. Default is 0.5
nodesizePredictor	node size of Predictors in the output graph. Default is 15

`nodesizeResponse`
: node size of response variables in the output graph. Default is 25

`no.isolates`
: remove isolated nodes from responses graph and full graph, may get problem if there are also isolated Predictors

`lineup`
: A ratio of the heights between responses' area and predictors'

`gray.alpha`
: the opacity. The default is 0.6

`edgewith.response`
: the edge width between response nodes

`edgewith.predictor`
: the edge width between the predictor and response node

`edge.weight`
: draw weighted edges after thresholding at 0.5. The default value `FALSE` is not to draw weighted edges

`label.predictor`
: A vector of the names of predictors

`label.response`
: A vector of the names of response variables

`color.predictor`
: color of the predictor nodes

`color.response`
: color of the response nodes

`name.predictors`
: A subtitle for the predictors

`name.responses`
: A subtitle for the responses

`color.edge.positive`
: color of the edges corresponding to estimated positive regression coefficients

`color.edge.negative`
: color of the edges corresponding to estimated negative regression coefficients

`vertex.frame.color`
: color of the frame of the vertices. If you don't want vertices to have a frame, supply NA as the color name

`layoutInCircle`
: place vertices on a circle, in the order of their vertex ids. The default is `FALSE`

`header`
: the main title

`beta.type`
: the type of output beta. Default is `marginal`, giving marginal beta estimation. If `beta.type="conditional"`, it gives beta estimation conditional on gamma=1

`...`
: other arguments

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
```

```
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check output
# draw network representation of the associations between responses and covariates
plotNetwork(fit)
```

`predict.BayesSUR` *predict method for class* `BayesSUR`

Description

Predict responses corresponding to the posterior mean of the coefficients, return posterior mean of coefficients or indices of nonzero coefficients of a `BayesSUR` class object.

Usage

```
## S3 method for class 'BayesSUR'
predict(object, newx, type = "response", beta.type = "marginal", Pmax = 0, ...)
```

Arguments

`object`	an object of class `BayesSUR`
`newx`	Matrix of new values for x at which predictions are to be made
`type`	Type of prediction required. `type="response"` gives the fitted responses; `type="coefficients"` returns the estimated coefficients depending on the arguments `beta.type` and `Pmax`. `type="nonzero"` returns a list of the indices of the nonzero coefficients corresponding to the estimated latent indicator variable thresholding at `Pmax`
`beta.type`	the type of estimated coefficients beta for prediction. Default is `marginal`, giving marginal beta estimation. If `beta.type="conditional"`, it gives conditional beta estimation
`Pmax`	If `type="nonzero"`, it is a threshold for the estimated latent indicator variable. If `type="coefficients"`, `beta.type="conditional"` and `Pmax=0.5`, it gives median probability model betas. Default is 0
`...`	other arguments

Value

Predicted values extracted from an object of class `BayesSUR`. If the `BayesSUR` specified data standardization, the fitted values are base based on standardized data.

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 20, burnin = 10, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/"
)

## check prediction
predict.val <- predict(fit, newx = exampleEQTL[["blockList"]][[2]])
```

---

print.BayesSUR *print method for class* BayesSUR

Description

Print a short summary of a BayesSUR class object. It includes the argument matching information, number of selected predictors based on thresholding the posterior mean of the latent indicator variable at 0.5 by default.

Usage

```
## S3 method for class 'BayesSUR'
print(x, Pmax = 0.5, ...)
```

Arguments

x	an object of class BayesSUR
Pmax	threshold that truncates the estimated coefficients based on thresholding the estimated latent indicator variable. Default is 0.5
...	other arguments

Value

Return a short summary from an object of class BayesSUR, including the number of selected predictors with mPIP>Pmax and the expected log pointwise predictive density estimates (i.e., elpd.LOO and elpd.WAIC).

Examples

```
data("exampleEQTL", package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/", output_CPO = TRUE
)

## check output
# show the print information
print(fit)
```

---

summary.BayesSUR *summary method for class* BayesSUR

Description

Summary method for class BayesSUR. It includes the argument matching information, Top predictors/responses on average mPIP across all responses/predictors, elpd estimates, MCMC specification, model specification and hyper-parameters. The summarized number of the selected variable corresponds to the posterior mean of the latent indicator variable thresholding at 0.5 by default.

Usage

```
## S3 method for class 'BayesSUR'
summary(object, Pmax = 0.5, ...)
```

Arguments

object	an object of class BayesSUR
Pmax	threshold that truncates the estimated coefficients based on thresholding the estimated latent indicator variable. Default is 0.5
...	other arguments

Value

Return a result summary from an object of class BayesSUR, including the CPOs, number of selected predictors with mPIP>Pmax, top 10 predictors on average mPIP across all responses, top 10 responses on average mPIP across all predictors, Expected log pointwise predictive density (elpd) estimates, MCMC specification, model specification (i.e., covariance prior and gamma prior) and hyper-parameters.

Examples

```
data(exampleEQTL, package = "BayesSUR")
hyperpar <- list(a_w = 2, b_w = 5)

set.seed(9173)
fit <- BayesSUR(
  Y = exampleEQTL[["blockList"]][[1]],
  X = exampleEQTL[["blockList"]][[2]],
  data = exampleEQTL[["data"]], outFilePath = tempdir(),
  nIter = 10, burnin = 0, nChains = 1, gammaPrior = "hotspot",
  hyperpar = hyperpar, tmpFolder = "tmp/", output_CPO = TRUE
)

## check output
# show the summary information
summary(fit)
```

`targetGene` *targetGene*

Description

Indices list of target genes corresponding the `example_GDSC` data set. It has two components representing the gene indices of the MAPK/ERK pathway and BCR-ABL gene fusion in the `example_GDSC` data set.

Usage

```
targetGene
```

Format

An object of class `list` of length 2.

Examples

```
# Load the indices of gene targets from the GDSC sample dataset
data("targetGene", package = "BayesSUR")
str(targetGene)

## Not run:
# ===============
# This code below is to do preprocessing of GDSC data and obtain the complete dataset
# "targetGene.rda" above. The user needs load the datasets from
# https://cellmodelpassports.sanger.ac.uk/downloads.
# But downloading and transforming the three used datasets below to *.csv files first.
# ===============
```

```
requireNamespace("plyr", quietly = TRUE)
requireNamespace("data.table", quietly = TRUE)


features <- data.frame(read.csv("/gdsc_en_input_w5.csv", head = T))
names.fea <- strsplit(rownames(features), "")
features <- t(features)
p <- c(13321, 13747 - 13321, 13818 - 13747)
Cell.Line <- rownames(features)
features <- data.frame(Cell.Line, features)

ic50_00 <- data.frame(read.csv("gdsc_drug_sensitivity_fitted_data_w5.csv", head = T))
ic50_0 <- ic50_00[, c(1, 4, 7)]
drug.id <- data.frame(read.csv("gdsc_tissue_output_w5.csv", head = T))[, c(1, 3)]
drug.id2 <- drug.id[!duplicated(drug.id$drug.id), ]
# delete drug.id=1066 since ID1066 and ID156 both correspond drug AZD6482,
# and no ID1066 in the "suppl.Data1" by Garnett et al. (2012)
drug.id2 <- drug.id2[drug.id2$drug.id != 1066, ]
drug.id2$drug.name <- as.character(drug.id2$drug.name)
drug.id2$drug.name <- substr(drug.id2$drug.name, 1, nchar(drug.id2$drug.name) - 6)
drug.id2$drug.name <- gsub(" ", "-", drug.id2$drug.name)

ic50 <- ic50_0
# mapping the drug_id to drug names in drug sensitivity data set
ic50$drug_id <- plyr::mapvalues(ic50$drug_id, from = drug.id2[, 2], to = drug.id2[, 1])
colnames(ic50) <- c("Cell.Line", "compound", "IC50")

# transform drug sensitivity overall cell lines to a data matrix
y0 <- reshape(ic50, v.names = "IC50", timevar = "compound",
                    idvar = "Cell.Line", direction = "wide")
y0$Cell.Line <- gsub("-", ".", y0$Cell.Line)

# ===============
# select nonmissing pharmacological data
# ===============
y00 <- y0
m0 <- dim(y0)[2] - 1
eps <- 0.05
# r1.na is better to be not smaller than r2.na
r1.na <- 0.3
r2.na <- 0.2
k <- 1
while (sum(is.na(y0[, 2:(1 + m0)])) > 0) {
  r1.na <- r1.na - eps / k
  r2.na <- r1.na - eps / k
  k <- k + 1
  ## select drugs with <30% (decreasing with k) missing data overall cell lines
  na.y <- apply(y0[, 2:(1 + m0)], 2, function(xx) sum(is.na(xx)) / length(xx))
  while (sum(na.y < r1.na) < m0) {
    y0 <- y0[, -c(1 + which(na.y >= r1.na))]
    m0 <- sum(na.y < r1.na)
    na.y <- apply(y0[, 2:(1 + m0)], 2, function(xx) sum(is.na(xx)) / length(xx))
  }
```

```
  ## select cell lines with treatment of at least 80% (increasing with k) drugs
  na.y0 <- apply(y0[, 2:(1 + m0)], 1, function(xx) sum(is.na(xx)) / length(xx))
  while (sum(na.y0 < r2.na) < (dim(y0)[1])) {
    y0 <- y0[na.y0 < r2.na, ]
    na.y0 <- apply(y0[, 2:(1 + m0)], 1, function(xx) sum(is.na(xx)) / length(xx))
  }
  num.na <- sum(is.na(y0[, 2:(1 + m0)]))
  message("#{NA}=", num.na, "\n", "r1.na =", r1.na, ", r2.na =", r2.na, "\n")
}

# ===============
# combine drug sensitivity, tissues and molecular features
# ===============
yx <- merge(y0, features, by = "Cell.Line")
names.cell.line <- yx$Cell.Line
names.drug <- colnames(yx)[2:(dim(y0)[2])]
names.drug <- substr(names.drug, 6, nchar(names.drug))
# numbers of gene expression features, copy number festures and muatation features
p <- c(13321, 13747 - 13321, 13818 - 13747)
num.nonpen <- 13
yx <- data.matrix(yx[, -1])
y <- yx[, 1:(dim(y0)[2] - 1)]
x <- cbind(yx[, dim(y0)[2] - 1 + sum(p) + 1:num.nonpen], yx[, dim(y0)[2] - 1 + 1:sum(p)])

# delete genes with only one mutated cell line
x <- x[, -c(num.nonpen + p[1] + p[2] +
            which(colSums(x[, num.nonpen + p[1] + p[2] + 1:p[3]]) <= 1))]
p[3] <- ncol(x) - num.nonpen - p[1] - p[2]

GDSC <- list(
  y = y, x = x, p = p, num.nonpen = num.nonpen, names.cell.line = names.cell.line,
  names.drug = names.drug
)


## ================
## ================
## select a small set of drugs
## ================
## ================

name_drugs <- c(
  "Methotrexate", "RDEA119", "PD-0325901", "CI-1040", "AZD6244", "Nilotinib",
  "Axitinib"
)

# extract the drugs' pharmacological profiling and tissue dummy
YX0 <- cbind(GDSC$y[, colnames(GDSC$y) %in% paste("IC50.", name_drugs, sep = "")]
[, c(1, 3, 6, 4, 7, 2, 5)], GDSC$x[, 1:GDSC$num.nonpen])
colnames(YX0) <- c(name_drugs, colnames(GDSC$x)[1:GDSC$num.nonpen])
# extract the genetic information of CNV & MUT
X23 <- GDSC$x[, GDSC$num.nonpen + GDSC$p[1] + 1:(p[2] + p[3])]
```

```
colnames(X23)[1:p[2]] <- paste(substr(
  colnames(X23)[1:p[2]], 1,
  nchar(colnames(X23)[1:p[2]]) - 3
), ".CNV", sep = "")

# locate all genes with CNV or MUT information
name_genes_duplicate <- c(
  substr(colnames(X23)[1:p[2]], 1, nchar(colnames(X23)[1:p[2]]) - 4),
  substr(colnames(X23)[p[2] + 1:p[3]], 1, nchar(colnames(X23)[p[2] + 1:p[3]]) - 4)
)
name_genes <- name_genes_duplicate[!duplicated(name_genes_duplicate)]

# select the GEX which have the common genes with CNV or MUT
X1 <- GDSC$x[, GDSC$num.nonpen +
            which(colnames(GDSC$x)[GDSC$num.nonpen + 1:p[1]] %in% name_genes)]

p[1] <- ncol(X1)
X1 <- log(X1)

# summary the data information
example_GDSC <- list(data = cbind(YX0, X1, X23))
example_GDSC$blockList <- list(
  1:length(name_drugs), length(name_drugs) + 1:GDSC$num.nonpen,
  ncol(YX0) + 1:sum(p))

# ========================
# construct the G matrix: edge potentials in the MRF prior
# ========================

# edges between drugs: Group1 ("RDEA119","17-AAG","PD-0325901","CI-1040" and "AZD6244")
# indexed as (2:5)
# http://software.broadinstitute.org/gsea/msigdb/cards/KEGG_MAPK_SIGNALING_PATHWAY
pathway_genes <- read.table("MAPK_pathway.txt")[[1]]
Idx_Pathway1 <- which(c(colnames(X1), name_genes_duplicate) %in% pathway_genes)
Gmrf_Group1Pathway1 <- t(combn(rep(Idx_Pathway1, each = length(2:5)) +
  rep((2:5 - 1) * sum(p), times = length(Idx_Pathway1)), 2))

# edges between drugs: Group2 ("Nilotinib","Axitinib") indexed as (6:7)
# delete gene ABL2
Idx_Pathway2 <- which(c(colnames(X1), name_genes_duplicate) %like% "BCR" |
  c(colnames(X1), name_genes_duplicate) %like% "ABL")[-c(3, 5)]
Gmrf_Group2Pathway2 <- t(combn(rep(Idx_Pathway2, each = length(6:7)) +
  rep((6:7 - 1) * sum(p), times = length(Idx_Pathway2)), 2))

# edges between the common gene in different data sources
Gmrf_CommonGene <- NULL
list_CommonGene <- list(0)
k <- 1
for (i in 1:length(name_genes)) {
  Idx_CommonGene <- which(c(colnames(X1), name_genes_duplicate) == name_genes[i])
  if (length(Idx_CommonGene) > 1) {
    Gmrf_CommonGene <-
    rbind(Gmrf_CommonGene, t(combn(rep(Idx_CommonGene, each = length(name_drugs)) +
```

```
    rep((1:length(name_drugs) - 1) * sum(p), times = length(Idx_CommonGene)), 2)))
    k <- k + 1
  }
}
Gmrf_duplicate <- rbind(Gmrf_Group1Pathway1, Gmrf_Group2Pathway2, Gmrf_CommonGene)
Gmrf <- Gmrf_duplicate[!duplicated(Gmrf_duplicate), ]
example_GDSC$mrfG <- Gmrf

# create the target gene names of the two groups of drugs
targetGenes1 <- matrix(Idx_Pathway1, nrow = 1)
colnames(targetGenes1) <- colnames(example_GDSC$data)[seq_along(targetGene$group1)]
targetGenes2 <- matrix(Idx_Pathway2, nrow = 1)
colnames(targetGenes2) <- colnames(example_GDSC$data)[seq_along(targetGene$group2)]

targetGene <- list(group1 = targetGenes1, group2 = targetGenes2)

## Write data file targetGene.rda to the user's directory by save()

## End(Not run)
```

Index

∗ **datasets**
exampleEQTL, 10
exampleGDSC, 15
targetGene, 37

BayesSUR, 2
BayesSUR-package *(*BayesSUR*)*, 2
BayesSUR_internal, 7

coef.BayesSUR, 8

elpd, 9
exampleEQTL, 10
exampleGDSC, 15

fitted.BayesSUR, 19

getEstimator, 21

plot.BayesSUR, 22
plotCPO, 23
plotEstimator, 25
plotGraph, 27
plotManhattan, 29
plotMCMCdiag, 30
plotNetwork, 31
predict.BayesSUR, 34
print.BayesSUR, 35

summary.BayesSUR, 36

targetGene, 37