Package 'BSL'

March 23, 2018

Type Package

Title Bayesian Synthetic Likelihood with Graphical Lasso

Version 0.1.1

Date 2018-03-23

Description
Bayesian synthetic likelihood (BSL, Price et al. (2018) <doi:10.1080/10618600.2017.1302882>) is an alternative to standard, non-
parametric approximate Bayesian computation (ABC). BSL assumes a
multivariate normal distribution for the summary statistic likelihood and it is suitable when
the distribution of the model summary statistics is sufficiently regular. This package provides
a Metropolis Hastings Markov chain Monte Carlo implementation of BSL and
BSL with graphical lasso (BSLasso, An et al. (2018) <https://eprints.qut.edu.au/102263/>),
which is computationally more efficient when the dimension of the summary statistic is high.
Extensions to this package are planned.

License GPL (>= 2)

LazyLoad yes

Imports glasso, ggplot2, stats, MASS, cvTools, grid, gridExtra,
foreach, coda, utils

Suggests elliplot

LinkingTo Rcpp, RcppArmadillo

LazyData true

RoxygenNote 6.0.1

NeedsCompilation yes

Author Ziwen An [aut, cre] (<https://orcid.org/0000-0002-9947-5182>),
Christopher C. Drovandi [ctb] (<https://orcid.org/0000-0001-9222-8763>),
Leah F. South [ctb] (<https://orcid.org/0000-0002-5646-2963>)

Maintainer Ziwen An `<ziwen.an@hdr.qut.edu.au>`

Repository CRAN

Date/Publication 2018-03-23 14:14:53 UTC

R topics documented:

`BSL-package` *Bayesian synthetic likelihood*

Description

Bayesian synthetic likelihood (BSL, Price et al (2018)) is an alternative to standard, non-parametric approximate Bayesian computation (ABC). BSL assumes a multivariate normal distribution for the summary statistic likelihood and it is suitable when the distribution of the model summary statistics is sufficiently regular.

In this package, a Metropolis Hastings Markov chain Monte Carlo (MH-MCMC) implementation of BSL is available. We also include an implementation of BSLasso (An et al., 2018) which is computationally more efficient when the dimension of the summary statistic is high.

Parallel computing is supported through the `foreach` package and users can specify their own parallel backend by using packages like `doParallel` or `doMC`.

The main functionality is available through

- `bsl`: The general function to perform BSL or BSLasso (with or without parallel computing).
- `selectPenalty`: A function to select the penalty for BSLasso.

Several examples have also been included. These examples can be used to reproduce the results of An et al. (2018).

- `ma2`: The MA(2) example from An et al. (2018).
- `mgnk`: The multivariate G&K example from An et al. (2018).
- `cell`: The cell biology example from Price et al. (2018) and An et al. (2018).

Extensions to this package are planned.

Author(s)

Ziwen An, Christopher C. Drovandi and Leah F. South

References

An, Z., South, L. F., Nott, D. J. & Drovandi, C. C. (2018). Accelerating Bayesian synthetic likelihood with the graphical lasso. https://eprints.qut.edu.au/102263/

Price, L. F., Drovandi, C. C., Lee, A., & Nott, D. J. (2018). Bayesian synthetic likelihood. To appear in Journal of Computational and Graphical Statistics. https://eprints.qut.edu.au/92795/

`bsl` *Performing BSL and BSLasso*

Description

This is the main function for performing MCMC BSL and MCMC BSLasso. Parallel computing is supported with the R package `foreach`.

Usage

```
bsl(y, n, M, start, cov_rw, fn_sim, fn_sum, penalty = NULL, fn_prior = NULL,
  sim_options = NULL, sum_options = NULL, standardise = FALSE,
  parallel = FALSE, parallel_packages = NULL, theta_names = NULL,
  verbose = TRUE)
```

Arguments

`y`	The observed data - note this should be the raw dataset NOT the set of summary statistics.
`n`	The number of simulations from the model per MCMC iteration.
`M`	The number of MCMC iterations.
`start`	Initial guess of the parameter value.
`cov_rw`	A covariance matrix to be used in multivariate normal random walk proposals.
`fn_sim`	A function that simulates data for a given parameter value. The function can have at most two arguments. The first should be the vector of tuning parameters and the second (optional) argument should be a list of other necessary arguments. See `sim_options`.
`fn_sum`	A function for computing summary statistics of data. The function can have at most two arguments. The first should be the vector of tuning parameters and the second (optional) argument should be a list of other necessary arguments. See `sum_options`.
`penalty`	The penalty value of the graphical lasso. BSLasso is performed if a numeric value is specified. The default is `NULL`, which means that BSL is performed.
`fn_prior`	A function that computes the prior density for a parameter. The default is `NULL`, which is an improper flat prior over the real line for each parameter. The function must have a single input: the vector of proposed parameters.
`sim_options`	A list of additional arguments to pass into the simulation function. Only use when the input `fn_sim` requires additional arguments. The default is `NULL`.

`sum_options`	A list of additional arguments to pass into the summary statistics function. Only use when the input `fn_sum` requires additional arguments. The default is `NULL`.
`standardise`	A logical argument that determines whether to standardise the summary statistics before applying the graphical lasso. This is only valid if penalty is not `NULL`. The diagonal elements will not be penalised if the penalty is not `NULL`. The default is `FALSE`.
`parallel`	A logical value indicating whether parallel computing should be used for simulation and summary statistic evaluation. Default is `FALSE`.
`parallel_packages`	A character vector of package names to pass into the `foreach` function as argument '.package'. Only used when parallel computing is enabled, default is `NULL`.
`theta_names`	A character vector of parameter names, which must has the same length as the parameter vector. The default is `NULL`.
`verbose`	A logical argument indicating whether the iteration numbers (`1:M`) and accepted proposal flags should be printed to track progress. The default is `FALSE`.

Value

An object of class `bsl` is returned, containing the following components:

- `theta`: MCMC samples from the joint approximate posterior distribution of the parameters.
- `loglike`: MCMC samples of the estimated log-likelihood values.
- `acceptanceRate`: The acceptance rate of the MCMC algorithm.
- `earlyRejectionRate`: The early rejection rate of the algorithm (early rejection may occur when using bounded prior distributions).
- `call`: The original code that was used to call the method.
- `theta_names`: A character vector of parameter names.

The functions print(), summary() and plot() are all available for types of class `bsl`. Multiple results can be plotted with overlapping densities using `combinePlotsBSL`.

Author(s)

Ziwen An, Christopher C. Drovandi and Leah F. South

References

An, Z., South, L. F., Nott, D. J. & Drovandi, C. C. (2018). Accelerating Bayesian synthetic likelihood with the graphical lasso. `https://eprints.qut.edu.au/102263/`

Price, L. F., Drovandi, C. C., Lee, A., & Nott, D. J. (2018). Bayesian synthetic likelihood. To appear in Journal of Computational and Graphical Statistics. `https://eprints.qut.edu.au/92795/`

See Also

`selectPenalty` for a function to tune the BSLasso tuning parameter and `plot` for functions related to visualisation.

`cell` *Cell biology example*

Description

This example estimates the probabilities of cell motility and cell proliferation for a discrete-time stochastic model of cell spreading. We provide the data and tuning parameters required to reproduce the results in An et al. (2018).

The function `cell_sim(theta, sim_options)` simulates data from the model, using C++ in the backend.

The function `cell_sum(Y,sum_options)` calculates the summary statistics for this example.

The function `cell_prior(theta)` evaluates the prior at the chosen parameters.

Usage

```
cell_sim(theta, sim_options)

cell_sum(Y, sum_options)

cell_prior(theta)
```

Arguments

`theta`	A vector of proposed model parameters, Pm and Pp.
`sim_options`	A list of options for simulating data from the model. For this example, the list contains • `Yinit`: The initial matrix of cell presences of size `rows` x `cols`. • `rows`: The number of rows in the lattice (rows in the cell location matrix). • `cols`: The number of columns in the lattice (columns in the cell location matrix). • `sim_iters`: The number of discretisation steps to get to when an observation is actually taken. For example, if observations are taken every 5 minutes but the discretisation level is 2.5 minutes, then `sim_iters` would be 2. Larger values of `sim_iters` lead to more "accurate" simulations from the model, but they also increase the simulation time. • `num_obs`: The total number of images taken after initialisation.
`Y`	A `rows` x `cols` x `num_obs` array of the cell presences at times `1:num_obs` (not time 0).
`sum_options`	A list of options for simulating data from the model. For this example, the list just contains `Yinit`, the same `rows` x `cols` matrix as in `sim_options`.

Details

Cell motility (movement) and proliferation (reproduction) cause tumors to spread and wounds to heal. If we can measure cell proliferation and cell motility under different situations, then we may be able to use this information to determine the efficacy of different medical treatments.

A common method for measuring in vitro cell movement and proliferation is the scratch assay. Cells form a layer on an assay and, once they are completely covering the assay, a scratch is made to separate the cells. Images of the cells are taken until the scratch has closed up and the cells are in contact again. Each image can be converted to a binary matrix by forming a lattice and recording the binary matrix (of size `rows` x `cols`) of cell presences.

The model that we consider is a random walk model with parameters for the probability of cell movement (Pm) and the probability of cell proliferation (`Pp`) and it has no tractable likelihood function. We use the uninformative priors θ_1 $U(0, 1)$ and θ_2 $U(0, 1)$.

We have a total of 145 summary statistics, which are made up of the Hamming distances between the binary matrices for each time point and the total number of cells at the final time.

Details about the types of cells that this model is suitable for and other information can be found in Price et al. (2018) and An et al. (2018). Johnston et al. (2014) use a different ABC method and different summary statistics for a similar example.

A simulated dataset

An example 'observed' dataset and the tuning parameters relevant to that example can be obtained using `data(cell)`. This 'observed' data is a simulated dataset with $Pm = 0.35$ and $Pp = 0.001$. The lattice has 27 `rows` and 36 `cols` and there are `num_obs = 144` observations after time 0 (to mimic images being taken every 5 minutes for 12 hours). The simulation is based on there initially being 110 cells in the assay.

Further information about the specific choices of tuning parameters used in BSL and BSLasso can be found in An et al. (2018).

- `data`: The `rows` x `cols` x `num_obs` array of the cell presences at times 1:144.
- `sim_options`: Values of `sim_options` relevant to this example.
- `sum_options`: Values of `sim_options` relevant to this example, i.e. just the value of `Yinit`.
- `start`: A vector of suitable initial values of the parameters for MCMC.
- `cov`: Covariance matrix of the multivariate normal random walk, in the form of a $2x2$ matrix.

Author(s)

Ziwen An, Christopher C. Drovandi and Leah F. South

References

An, Z., South, L. F., Nott, D. J. & Drovandi, C. C. (2018). Accelerating Bayesian synthetic likelihood with the graphical lasso. https://eprints.qut.edu.au/102263/

Johnston, S., Simpson, M. J., McElwain, D. L. S., Binder, B. J. & Ross, J. V. (2014). Interpreting Scratch Assays Using Pair Density Dynamic and Approximate Bayesian Computation. Open Biology, 4, 1-11.

Price, L. F., Drovandi, C. C., Lee, A., & Nott, D. J. (2018). Bayesian synthetic likelihood. To appear in Journal of Computational and Graphical Statistics. https://eprints.qut.edu.au/92795/

Examples

```
## Not run:
require(doParallel) # You can use a different package to set up the parallel backend

# Loading the data for this example
data(cell)
true_cell <- c(0.35, 0.001)

# Opening up the parallel pools using doParallel
cl <- makeCluster(detectCores())
registerDoParallel(cl)

# Performing BSL
resultCellBSL <- bsl(cell$data, n = 5000, M = 10000, start = cell$start, cov_rw = cell$cov,
                 fn_sim = cell_sim, fn_sum = cell_sum, fn_prior = cell_prior,
                 sim_options = cell$sim_options, sum_options = cell$sum_options,
                 parallel = TRUE, parallel_packages = 'BSL',
                 theta_names = c('Pm', 'Pp'))
summary(resultCellBSL)
plot(resultCellBSL, true_value = true_cell, thin = 20)

# Performing tuning for BSLasso
lambda_all <- list(exp(seq(0.5,2.5,length.out=20)), exp(seq(0,2,length.out=20)),
                   exp(seq(-1,1,length.out=20)), exp(seq(-1,1,length.out=20)))

sp_cell <- selectPenalty(ssy = cell_sum(cell$data, cell$sum_options),
                 n = c(500, 1000, 1500, 2000), lambda_all, theta = true_cell,
                 M = 100, sigma = 1.5, fn_sim = cell_sim,
                 fn_sum = cell_sum, sim_options = cell$sim_options,
                 sum_options = cell$sum_options, parallel_sim = TRUE,
                 parallel_sim_packages = 'BSL', parallel_main = TRUE)

sp_cell
plot(sp_cell)

# Performing BSLasso with a fixed penalty
resultCellBSLasso <- bsl(cell$data, n = 1500, M = 10000, start = cell$start,
                 cov_rw = cell$cov, fn_sim = cell_sim, fn_sum = cell_sum,
                 penalty = 1.35, fn_prior = cell_prior,
                 sim_options = cell$sim_options, sum_options = cell$sum_options,
                 parallel = TRUE, parallel_packages = 'BSL',
                 theta_names = c('Pm', 'Pp'))
summary(resultCellBSLasso)
plot(resultCellBSLasso, true_value = true_cell, thin = 20)

# Plotting the results together for comparison
combinePlotsBSL(resultCellBSL, resultCellBSLasso, true_value = true_cell, thin = 20)

# Closing the parallel pools
```

```
stopCluster(cl)

## End(Not run)
```

---

ma2 *An MA(2) model*

Description

In this example we wish to estimate the parameters of a simple MA(2) time series model. We provide the data and tuning parameters required to reproduce the results in An et al. (2018).

The function `ma2_sim(theta,options)` simulates an MA(2) time series.

The function `ma2_sum(x)` returns the summary statistics for a given data set. Since the summary statistics are the data, this function simply returns the data.

`ma2_prior(theta)` evaluates the (unnormalised) prior, which is uniform subject to several restrictions related to invertibility of the time series.

Usage

```
data(ma2)

ma2_sim(theta, options)

ma2_sum(x)

ma2_prior(theta)
```

Arguments

`theta`	A vector of proposed model parameters, θ_1 and θ_2
`options`	A list of options for simulating data from the model. For this example, the list contains only T, the number of observations.
`x`	Observed or simulated data in the format of a vector of length T.

Details

This example is based on estimating the parameters of a basic MA(2) time series model of the form

$$y_t = z_t + \theta_1 z_{t-1} + \theta_2 z_{t-2},$$

where $t = 1, \ldots, T$ and z_t $N(0, 1)$ for $t = -1, 0, \ldots, T$. A uniform prior is used for this example, subject to the restrictions that $-2 < \theta_1 < 2$, $\theta_1 + \theta_2 > -1$ and $\theta_1 - \theta_2 < 1$ so that invertibility of the time series is satisfied. The summary statistics are simply the full data.

A simulated dataset

An example 'observed' dataset and the tuning parameters relevant to that example can be obtained using `data(ma2)`. This 'observed' data is a simulated dataset with $\theta_1 = 0.6$, $\theta_2 = 0.2$ and $T = 50$. Further information about this model and the specific choices of tuning parameters used in BSL and BSLasso can be found in An et al. (2018).

- `data`: A time series dataset, in the form of a vector of length T
- `sim_options`: A list containing $T = 50$
- `start`: A vector of suitable initial values of the parameters for MCMC
- `cov`: Covariance matrix of the multivariate normal random walk, in the form of a $2x2$ matrix

Author(s)

Ziwen An, Christopher C. Drovandi and Leah F. South

References

An, Z., South, L. F., Nott, D. J. & Drovandi, C. C. (2018). Accelerating Bayesian synthetic likelihood with the graphical lasso. https://eprints.qut.edu.au/102263/

Examples

```
## Not run:
# Loading the data for this example
data(ma2)
true_ma2 <- c(0.6,0.2)

# Performing BSL
resultMa2BSL <- bsl(y = ma2$data, n = 500, M = 300000, start = ma2$start, cov_rw = ma2$cov,
                 fn_sim = ma2_sim, fn_sum = ma2_sum, fn_prior = ma2_prior,
                 sim_options = ma2$sim_options, theta_names = c('theta1', 'theta2'))
summary(resultMa2BSL)
plot(resultMa2BSL, true_value = true_ma2, thin = 20)

# Performing tuning for BSLasso
lambda_all <- list(exp(seq(-3,0.5,length.out=20)), exp(seq(-4,-0.5,length.out=20)),
                 exp(seq(-5.5,-1.5,length.out=20)), exp(seq(-7,-2,length.out=20)))

sp_ma2 <- selectPenalty(ssy = ma2_sum(ma2$data), n = c(50, 150, 300, 500), lambda_all,
                 theta = true_ma2, M = 100, sigma = 1.5, fn_sim = ma2_sim,
                 fn_sum = ma2_sum, sim_options = ma2$sim_options)
sp_ma2
plot(sp_ma2)

# Performing BSLasso with a fixed penalty
resultMa2BSLasso <- bsl(y = ma2$data, n = 300, M = 250000, start = ma2$start, cov_rw = ma2$cov,
               fn_sim = ma2_sim, fn_sum = ma2_sum, penalty = 0.027, fn_prior = ma2_prior,
                 sim_options = ma2$sim_options, theta_names = c('theta1', 'theta2'))
summary(resultMa2BSLasso)
```

```
plot(resultMa2BSLasso, true_value = true_ma2, thin = 20)

# Plotting the results together for comparison
combinePlotsBSL(resultMa2BSL, resultMa2BSLasso, true_value = true_ma2, thin = 20)

## End(Not run)
```

---

mgnk *The multivariate G&K example*

Description

Here we provide the data and tuning parameters required to reproduce the results from the multivariate G & K (Drovandi and Pettitt, 2011) example from An et al. (2018).

The function `mgnk_sim(theta_tilde,sim_options)` simulates from the multivariate G & K model.

The function `mgnk_sum(y)` calculates the summary statistics for the multivariate G & K example.

Usage

```
mgnk_sim(theta_tilde, sim_options)

mgnk_sum(y)
```

Arguments

`theta_tilde`	A vector with 15 elements for the proposed model parameters.
`sim_options`	A list of options for simulating data from the model. For this example, the list contains • `T`: The number of observations in the data. • `J`: The number of variables in the data. • bound: A matrix of boundaries for the uniform prior.
`y`	A `T` x `J` matrix of data.

Details

It is not practical to give a reasonable explanation of this example through R documentation given the number of equations involved. We refer the reader to the BSLasso paper (An et al., 2018) at https://eprints.qut.edu.au/102263/ for information on the model and summary statistic used in this example.

An example dataset

We use the foreign currency exchange data available from http://www.rba.gov.au/statistics/historical-data.html as in An et al. (2018).

- data: A 1651 x 3 matrix of data.
- sim_options: Values of sim_options relevant to this example.
- start: A vector of suitable initial values of the parameters for MCMC.
- cov: Covariance matrix of the multivariate normal random walk, in the form of a $15x15$ matrix.

Author(s)

Ziwen An, Christopher C. Drovandi and Leah F. South

References

An, Z., South, L. F., Nott, D. J. & Drovandi, C. C. (2018). Accelerating Bayesian synthetic likelihood with the graphical lasso. https://eprints.qut.edu.au/102263/

Drovandi, C. C. and Pettitt, A. N. (2011). Likelihood-free Bayesian estimation of multivariate quantile distributions. Computational Statistics and Data Analysis, 55(9):2541-2556.

Examples

```
## Not run:
require(doParallel) # You can use a different package to set up the parallel backend

# Loading the data for this example
data(mgnk)

# Opening up the parallel pools using doParallel
cl <- makeCluster(detectCores())
registerDoParallel(cl)

# Performing BSL
resultMgnkBSL <- bsl(mgnk$data, n = 60, M = 80000, start = mgnk$start, cov_rw = mgnk$cov,
                 fn_sim = mgnk_sim, fn_sum = mgnk_sum, sim_options = mgnk$sim_options,
                 parallel = TRUE, parallel_packages = c('BSL', 'MASS', 'elliplot'),
              theta_names = c('a1','b1','g1','k1','a2','b2','g2','k2','a3','b3','g3','k3'
                 ,'delta12','delta13','delta23'))
summary(resultMgnkBSL)
plot(resultMgnkBSL, thin = 20)

# Performing tuning for BSLasso
lambda_all <- list(exp(seq(-2.5,0.5,length.out=20)), exp(seq(-2.5,0.5,length.out=20)),
                   exp(seq(-4,-0.5,length.out=20)), exp(seq(-5,-2,length.out=20)))

sp_mgnk <- selectPenalty(ssy = mgnk_sum(mgnk$data), n = c(15, 20, 30, 50), lambda_all,
                 theta = mgnk$start, M = 100, sigma = 1.5, fn_sim = mgnk_sim,
                 fn_sum = mgnk_sum, sim_options = mgnk$sim_options, standardise = TRUE,
               parallel_sim = TRUE, parallel_sim_packages = c('BSL', 'MASS', 'elliplot'),
                 parallel_main = TRUE)
sp_mgnk
plot(sp_mgnk)

# Performing BSLasso with a fixed penalty
resultMgnkBSLasso <- bsl(mgnk$data, n = 20, M = 80000, start = mgnk$start, cov_rw = mgnk$cov,
                 fn_sim = mgnk_sim, fn_sum = mgnk_sum, sim_options = mgnk$sim_options,
                 penalty = 0.3, standardise = TRUE, parallel = TRUE,
                 parallel_packages = c('BSL', 'MASS', 'elliplot'),
              theta_names = c('a1','b1','g1','k1','a2','b2','g2','k2','a3','b3','g3','k3',
                 'delta12','delta13','delta23'))
summary(resultMgnkBSLasso)
plot(resultMgnkBSLasso, thin = 20)

# Plotting the results together for comparison
combinePlotsBSL(resultMgnkBSL, resultMgnkBSLasso, thin = 20)

# Closing the parallel pools
stopCluster(cl)

## End(Not run)
```

plot *Plotting BSL and BSLasso Results*

Description

There are several functions included in this package to help visualise the results of the MCMC run and of the tuning process.

The function `combinePlotsBSL` can be used to plot multiple BSL and BSLasso densities together, optionally with the true values for the parameters.

The function `plot.BSL` can be used to plot BSL and BSLasso densities, optionally with the true values for the parameters.

The function `plot.penbsl` can be used to plot the results from tuning to select the optimal penalty for BSLasso.

Usage

```
combinePlotsBSL(..., true_value = NULL, thin = 1)

## S3 method for class 'bsl'
plot(x, true_value = NULL, thin = 1, ...)

## S3 method for class 'penbsl'
plot(x, logscale = TRUE, ...)
```

Arguments

`...`	The first inputs to the `combinePlotsBSL` function should be two or more results of type `bsl`, separated by commas.
`true_value`	A set of values to be included on the plots as a reference line. The default is `NULL`.
`thin`	The gap between samples to be taken when thinning the MCMC draws. The default is 1 (no thinning).
`x`	An object to be plotted.
`logscale`	An indicator for whether the penalty values should be shown on the log scale in plots of `penbsl` objects. The default is `TRUE`.

Author(s)

Ziwen An, Christopher C. Drovandi and Leah F. South

`selectPenalty` *Selecting BSLasso Penalty*

Description

This is the main function for selecting the penalty for BSLasso based on a point estimate of the parameters. Parallel computing is supported with the R package `foreach`.

Usage

```
selectPenalty(ssy, n, lambda_all, theta, M, sigma, fn_sim, fn_sum,
  sim_options = NULL, sum_options = NULL, standardise = FALSE,
  parallel_sim = FALSE, parallel_sim_packages = NULL,
  parallel_main = FALSE, verbose = TRUE)
```

Arguments

`ssy`	A summary statistic vector for the observed data.
`n`	A vector of possible values of `n`, the number of simulations from the model per MCMC iteration, to test.
`lambda_all`	A list, with each entry containing the vector of penalty values to test for the corresponding choice of `n`.
`theta`	A point estimate of the parameter value which all of the simulations will be based on.
`M`	The number of repeats to use in estimating the standard deviation of the estimated log synthetic likelihood.
`sigma`	The standard deviation of the log synthetic likelihood to aim for.

`fn_sim`	A function that simulates data for a given parameter value. The function can have at most two arguments. The first is the vector of tuning parameters and the second (optional) argument is a list of other necessary arguments. See `sim_options`.
`fn_sum`	A function for computing summary statistics of data. The function can have at most two arguments. The first is the vector of tuning parameters and the second (optional) argument is a list of other necessary arguments. See `sum_options`.
`sim_options`	A list of additional arguments to pass into the simulation function. Only use when the input `fn_sim` requires additional arguments. The default is `NULL`.
`sum_options`	A list of additional arguments to pass into the summary statistic function. Only use when the input `fn_sum` requires additional arguments. The default is `NULL`.
`standardise`	A logical argument that determines whether to standardise the summary statistics before applying the graphical lasso. This is only valid if penalty is not `NULL`. The diagonal elements will not be penalised if the penalty is not `NULL`. The default is `FALSE`.
`parallel_sim`	A logical value indicating whether parallel computing should be used for simulation and summary statistic evaluation. Default is `FALSE`.
`parallel_sim_packages`	A character vector of package names to pass into the `foreach` function as argument '.package'. Only used when `parallel_sim` is `TRUE`, default is `NULL`.
`parallel_main`	A logical value indicating whether parallel computing should be used to computing the graphical lasso function. Default is `FALSE`.
`verbose`	A logical argument indicating whether the iteration numbers (`1:M`) should be printed to track progress. The default is `FALSE`.

Value

An object of class `penbsl` is returned, containing the following components:

- `resultsDF`: A data frame containing the following:
  - `n`: The choices of `n` that were specified.
  - `penalty`: The choices of the penalty that were specified.
  - `sigma`: The standard deviation of the log synthetic likelihood under the above choices.
  - `sigmaOpt`: An indicator of whether it was the closest `sigma` to the desired one for each choice of `n`.
- `call`: The original code that was used to call the method.

The functions print() and plot() are both available for types of class `penbsl`.

Author(s)

Ziwen An, Christopher C. Drovandi and Leah F. South

References

An, Z., South, L. F., Nott, D. J. & Drovandi, C. C. (2018). Accelerating Bayesian synthetic likelihood with the graphical lasso. https://eprints.qut.edu.au/102263/

See Also

`bsl` for a function to run BSLasso after selecting the tuning parameter and `plot` for functions related to visualisation.

Index