Package ‘BSL’

July 21, 2025

Type Package

Title Bayesian Synthetic Likelihood

Version 3.2.5

Date 2022-11-02

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 four methods (BSL, uBSL, semiBSL and BSLmisspec) and two shrinkage estimators (graphical lasso and Warton's estimator). uBSL (Price et al. (2018) <doi:10.1080/10618600.2017.1302882>) uses an unbiased estimator to the normal density. A semi-parametric version of BSL (semiBSL, An et al. (2018) <doi:10.48550/arXiv.1809.05800>) is more robust to non-normal summary statistics. BSLmisspec (Frazier et al. 2019 <doi:10.48550/arXiv.1904.04551>) estimates the Gaussian synthetic likelihood whilst acknowledging that there may be incompatibility between the model and the observed summary statistic. Shrinkage estimation can help to decrease the number of model simulations when the dimension of the summary statistic is high (e.g., BSLasso, An et al. (2019) <doi:10.1080/10618600.2018.1537928>). Extensions to this package are planned. For a journal article describing how to use this package, see An et al. (2022) <doi:10.18637/jss.v101.i11>.

Depends R (>= 3.3.0)

License GPL (>= 2)

LazyLoad yes

Imports glasso, ggplot2, MASS, mvtnorm, copula, whitening, graphics, gridExtra, foreach, coda, Rcpp, doRNG, methods, stringr, Rdpack (>= 0.7)

Suggests elliplot, doParallel, rbenchmark, mixtools

LinkingTo RcppArmadillo, Rcpp

LazyData true

RoxygenNote 7.1.1

Encoding UTF-8

Collate 'BSL-package.R' 'RcppExports.R' 's4-MODEL.R' 's4-BSL.R' 'bsl.R' 'cell.R' 'combinePlotsBSL.R' 'covWarton.R' 'estimateLoglike.R' 'estimateWhiteningMatrix.R' 'gaussianRankCorr.R' 'gaussianSynLike.R' 'gaussianSynLikeGhuryeOlkin.R' 'imports.R' 'kernelCDF.R' 'logitTransform.R' 'ma2.R' 'mgnk.R' 'myMiniProgressBar.R' 's4-PENALTY.R' 'selectPenalty.R' 'semiparaKernelEstimate.R' 'sliceGammaMean.R' 'sliceGammaVariance.R' 'synLikeMisspec.R' 'toad.R'

RdMacros Rdpack

NeedsCompilation yes

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

Maintainer Leah F. South <`ll.south@qut.edu.au`>

Repository CRAN

Date/Publication 2022-11-03 09:00:07 UTC

Contents

`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 implementations of four methods (BSL, uBSL, semiBSL and BSLmisspec) and two shrinkage estimators (graphical lasso and Warton's estimator).

Methods: (1) BSL (Price et al. 2018), which is the standard form of Bayesian synthetic likelihood, assumes the summary statistic is roughly multivariate normal; (2) uBSL (Price et al. 2018), which uses an unbiased estimator to the normal density; (3) semiBSL (An et al. 2019), which relaxes the normality assumption to an extent and maintains the computational advantages of BSL without any tuning; and (4) BSLmisspec (Frazier and Drovandi 2021), which estimates the Gaussian synthetic likelihood whilst acknowledging that there may be incompatibility between the model and the observed summary statistic.

Shrinkage estimators are designed particularly to reduce the number of simulations if method is BSL or semiBSL: (1) graphical lasso (Friedman et al. 2008) finds a sparse precision matrix with an L1-regularised log-likelihood. An et al. (2019) use graphical lasso within BSL to bring down the number of simulations significantly when the dimension of the summary statistic is high; and (2) Warton's estimator (Warton 2008) penalises the correlation matrix and is straightforward to compute. When using the Warton's shrinkage estimator, it is also possible to utilise the Whitening transformation (Kessy et al. 2018) to help decorrelate the summary statsitics, thus encouraging sparsity of the synthetic likelihood covariance matrix.

Parallel computing is supported through the `foreach` package and users can specify their own parallel backend by using packages like `doParallel` or `doMC`. The `n` model simulations required to estimate the synthetic likelihood at each iteration of MCMC will be distributed across multiple cores. Alternatively a vectorised simulation function that simultaneously generates `n` model simulations is also supported.

The main functionality is available through:

- `bsl`: The general function to perform BSL, uBSL, or semiBSL (with or without parallel computing).

- `selectPenalty`: A function to select the penalty when using shrinkage estimation within BSL or semiBSL.

Several examples have also been included. These examples can be used to reproduce the results of An et al. (2019), and can help practitioners learn how to use the package.

- `ma2`: The MA(2) example from An et al. (2019).
- `mgnk`: The multivariate G&K example from An et al. (2019).
- `cell`: The cell biology example from Price et al. (2018) and An et al. (2019).
- `toad`: The toad example from Marchand et al. (2017), and also considered in An et al. (2019).

Extensions to this package are planned. For a journal article describing how to use this package, including full descriptions on the MA(2) and toad examples, see An et al. (2022).

Author(s)

Ziwen An, Leah F. South and Christopher Drovandi

References

An Z, Nott DJ, Drovandi C (2019). “Robust Bayesian Synthetic Likelihood via a Semi-Parametric Approach.” *Statistics and Computing (In Press)*.

An Z, South LF, Drovandi CC (2022). “BSL: An R Package for Efficient Parameter Estimation for Simulation-Based Models via Bayesian Synthetic Likelihood.” *Journal of Statistical Software*, **101**(11), 1–33. doi: 10.18637/jss.v101.i11.

An Z, South LF, Nott DJ, Drovandi CC (2019). “Accelerating Bayesian Synthetic Likelihood With the Graphical Lasso.” *Journal of Computational and Graphical Statistics*, **28**(2), 471–475. doi: 10.1080/10618600.2018.1537928.

Frazier DT, Drovandi C (2021). “Robust Approximate Bayesian Inference with Synthetic Likelihood.” *Journal of Computational and Graphical Statistics (In Press)*. `https://arxiv.org/abs/1904.04551`.

Friedman J, Hastie T, Tibshirani R (2008). “Sparse Inverse Covariance Estimation with the Graphical Lasso.” *Biostatistics*, **9**(3), 432–441.

Kessy A, Lewin A, Strimmer K (2018). “Optimal Whitening and Decorrelation.” *The American Statistician*, **72**(4), 309–314. doi: 10.1080/00031305.2016.1277159.

Marchand P, Boenke M, Green DM (2017). “A stochastic movement model reproduces patterns of site fidelity and long-distance dispersal in a population of Fowlers toads (Anaxyrus fowleri).” *Ecological Modelling*, **360**, 63–69. ISSN 0304-3800, doi: 10.1016/j.ecolmodel.2017.06.025.

Price LF, Drovandi CC, Lee A, Nott DJ (2018). “Bayesian Synthetic Likelihood.” *Journal of Computational and Graphical Statistics*, **27**, 1–11. doi: 10.1080/10618600.2017.1302882.

Warton DI (2008). "Penalized Normal Likelihood and Ridge Regularization of Correlation and Covariance Matrices." *Journal of the American Statistical Association*, **103**(481), 340–349. doi: 10.1198/016214508000000021.

`bsl` *Performing BSL, uBSL, semiBSL and BSLmisspec*

Description

This is the main function for performing MCMC BSL (with a standard or non-standard likelihood estimator) to sample from the approximate posterior distribution. A couple of extentions to the standard approach are available by changing the following arguments, `method`, `shrinkage`, `whitening`, `misspecType`. Parallel computing is supported with the R package `foreach`.

Usage

```
bsl(
  y,
  n,
  M,
  model,
  covRandWalk,
  theta0,
  fnSim,
  fnSum,
  method = c("BSL", "uBSL", "semiBSL", "BSLmisspec"),
  shrinkage = NULL,
  penalty = NULL,
  fnPrior = NULL,
  simArgs = NULL,
  sumArgs = NULL,
  logitTransformBound = NULL,
  standardise = FALSE,
  GRC = FALSE,
  whitening = NULL,
  misspecType = NULL,
  tau = 1,
  parallel = FALSE,
  parallelArgs = NULL,
  thetaNames = NULL,
  plotOnTheFly = FALSE,
  verbose = 1L
)
```

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 for estimating the synthetic likelihood.
`M`	The number of MCMC iterations.
`model`	A "MODEL" object generated with function `newModel`. See `newModel`.
`covRandWalk`	The covariance matrix of a multivariate normal random walk proposal distribution used in the MCMC.
`theta0`	Deprecated, will be removed in the future, use `model` instead. Initial guess of the parameter value, which is used as the starting value for MCMC.
`fnSim`	Deprecated, will be removed in the future, use `model` instead. A function that simulates data for a given parameter value. The first argument should be the parameters. Other necessary arguments (optional) can be specified with `simArgs`.
`fnSum`	Deprecated, will be removed in the future, use `model` instead. A function for computing summary statistics of data. The first argument should be the observed or simulated dataset. Other necessary arguments (optional) can be specified with `sumArgs`.
`method`	A string argument indicating the method to be used. The default, "BSL", runs standard BSL. "uBSL" uses the unbiased estimator of a normal density of Ghurye and Olkin (1969). "semiBSL" runs the semi-parametric BSL algorithm and is more robust to non-normal summary statistics. "BSLmisspec" estimates the Gaussian synthetic likelihood whilst acknowledging that there may be incompatibility between the model and the observed summary statistic (Frazier and Drovandi 2021).
`shrinkage`	A string argument indicating which shrinkage method to be used. The default is `NULL`, which means no shrinkage is used. Shrinkage estimation is only available for methods "BSL" and "semiBSL". Current options are "glasso" for the graphical lasso method of Friedman et al. (2008) and "Warton" for the ridge regularisation method of Warton (2008).
`penalty`	The penalty value to be used for the specified shrinkage method. Must be between zero and one if the shrinkage method is "Warton".
`fnPrior`	Deprecated, will be removed in the future, use `model` instead. A function that computes the log prior density for a parameter. The default is `NULL`, which uses an improper flat prior over the real line for each parameter. The function must have a single input: a vector of parameter values.
`simArgs`	Deprecated, will be removed in the future, use `model` instead. A list of additional arguments to pass into the simulation function. Only use when the input `fnSim` requires additional arguments. The default is `NULL`.
`sumArgs`	Deprecated, will be removed in the future, use `model` instead. A list of additional arguments to pass into the summary statistics function. Only use when the input `fnSum` requires additional arguments. The default is `NULL`.
`logitTransformBound`	A p by 2 numeric matrix indicating the upper and lower bounds of parameters if a logit transformation is used on the parameter space, where p is the number of parameters. The default is `NULL`, which means no logit transformation is used. It is also possible to define other transformations within the simulation and prior

	function from `model`. The first column contains the lower bound of each parameter and the second column contains the upper bound. Infinite lower or upper bounds are also supported, eg. `matrix(c(1,Inf,0,10,-Inf,0.5),3,2,byrow=TRUE)`.
`standardise`	A logical argument that determines whether to standardise the summary statistics before applying the graphical lasso. This is only valid if method is “BSL”, shrinkage is “glasso” and penalty is not `NULL`. The diagonal elements will not be penalised if the shrinkage method is “glasso”. The default is `FALSE`.
`GRC`	A logical argument indicating whether the Gaussian rank correlation matrix (Boudt et al. 2012) should be used to estimate the covariance matrix in “BSL” method. The default is `FALSE`, which uses the sample covariance by default.
`whitening`	This argument determines whether Whitening transformation should be used in “BSL” method with Warton’s shrinkage. Whitening transformation helps decorrelate the summary statistics, thus encouraging sparsity of the synthetic likelihood covariance matrix. This might allow heavier shrinkage to be applied without losing much accuracy, hence allowing the number of simulations to be reduced. By default, `NULL` represents no Whitening transformation. Otherwise this is enabled if a Whitening matrix is provided. See `estimateWhiteningMatrix` for the function to estimate the Whitening matrix.
`misspecType`	A string argument indicating which type of model misspecification to be used. The two options are "mean" and "variance". Only used when method is “BSLmisspec”. The default, `NULL`, means no model misspecification is considered.
`tau`	A numeric argument, parameter of the prior distribution for "BSLmisspec" method. For mean adjustment, `tau` is the scale of the Laplace distribution. For variance inflation, `tau` is the mean of the exponential distribution. Only used when method is “BSLmisspec”.
`parallel`	A logical value indicating whether parallel computing should be used for simulation and summary statistic evaluation. The default is `FALSE`. When model simulation is fast, it may be preferable to perform serial or vectorised computations to avoid significant communication overhead between workers. Parallel computation can only be used if not using a vectorised simulation function, see `MODEL` for options of vectorised simulation function.
`parallelArgs`	A list of additional arguments to pass into the `foreach` function. Only used when parallel computing is enabled, default is `NULL`.
`thetaNames`	Deprecated, will be removed in the future, use `model` instead. A string vector of parameter names, which must have the same length as the parameter vector. The default is `NULL`.
`plotOnTheFly`	A logical or numeric argument defining whether or by how many iterations a posterior figure will be plotted during running. If `TRUE`, a plot of approximate univariate posteriors based on the current accepted samples will be shown every one thousand iterations. The default is `FALSE`.
`verbose`	An integer indicating the verbose style. 0L means no verbose messages will be printed. 1L uses a custom progress bar to track the progress. 2L prints the iteration numbers (`1:M`) to track the progress. The default is 1L.

Value

An object of class `bsl` is returned, see `BSL` for more information of the S4 class.

Author(s)

Ziwen An, Leah F. South and Christopher Drovandi

References

Boudt K, Cornelissen J, Croux C (2012). “The Gaussian Rank Correlation Estimator: Robustness Properties.” *Statistics and Computing*, **22**(2), 471–483. doi: 10.1007/s1122201192370.

Frazier DT, Drovandi C (2021). “Robust Approximate Bayesian Inference with Synthetic Likelihood.” *Journal of Computational and Graphical Statistics (In Press)*. `https://arxiv.org/abs/1904.04551`.

Friedman J, Hastie T, Tibshirani R (2008). “Sparse Inverse Covariance Estimation with the Graphical Lasso.” *Biostatistics*, **9**(3), 432–441.

Ghurye SG, Olkin I (1969). “Unbiased Estimation of Some Multivariate Probability Densities and Related Functions.” *Ann. Math. Statist.*, **40**(4), 1261–1271.

Warton DI (2008). “Penalized Normal Likelihood and Ridge Regularization of Correlation and Covariance Matrices.” *Journal of the American Statistical Association*, **103**(481), 340–349. doi: 10.1198/016214508000000021.

Price LF, Drovandi CC, Lee A, Nott DJ (2018). “Bayesian Synthetic Likelihood.” *Journal of Computational and Graphical Statistics*, **27**, 1–11. doi: 10.1080/10618600.2017.1302882.

An Z, South LF, Nott DJ, Drovandi CC (2019). “Accelerating Bayesian Synthetic Likelihood With the Graphical Lasso.” *Journal of Computational and Graphical Statistics*, **28**(2), 471–475. doi: 10.1080/10618600.2018.1537928.

An Z, Nott DJ, Drovandi C (2019). “Robust Bayesian Synthetic Likelihood via a Semi-Parametric Approach.” *Statistics and Computing (In Press)*.

See Also

`ma2`, `cell`, `mgnk` and `toad` for examples. `selectPenalty` for a function to tune the BSLasso tuning parameter and `plot` for functions related to visualisation.

Examples

```
## Not run:
# This is just a minimal test run, please see package built-in examples for more
# comprehensive usages of the function
toy_sim <- function(n, theta) matrix(rnorm(n, theta), nrow = n)
toy_sum <- function(x) x
model <- newModel(fnSimVec = toy_sim, fnSum = toy_sum, theta0 = 0)

result_toy <- bsl(y = 1, n = 100, M = 1e4, model = model, covRandWalk = matrix(1),
    method = "BSL", plotOnTheFly = TRUE)
summary(result_toy)
plot(result_toy)

## End(Not run)
```

`BSL-class` *S4 class “BSL”.*

Description

The S4 class “BSL” is produced by running function `bsl` and contains the result of a BSL run. Basic S4 methods `show`, `summary` and `plot` are provided. `theta` and `loglike` returns the MCMC samples of parameter values and estimated log-likelihoods.

Usage

```
## S4 method for signature 'BSL'
show(object)

## S4 method for signature 'BSL'
summary(object, burnin = 0, thetaNames = NULL)

## S4 method for signature 'BSL,ANY'
plot(
  x,
  which = 1L,
  thin = 1,
  burnin = 0,
  thetaTrue = NULL,
  options.plot = NULL,
  top = "Approximate Univariate Posteriors",
  options.density = list(),
  options.theme = list()
)

## S4 method for signature 'BSL'
getTheta(object, burnin = 0, thin = 1)

## S4 method for signature 'BSL'
getLoglike(object, burnin = 0, thin = 1)

## S4 method for signature 'BSL'
getGamma(object, burnin = 0, thin = 1)
```

Arguments

`object`	A “BSL” class object.
`burnin`	the number of MCMC burn-in steps to be taken.
`thetaNames`	Parameter names to be shown in the summary table. If not given, parameter names of the “BSL” object will be used by default.

`x`	A “BSL” class object to plot.
`which`	An integer argument indicating which plot function to be used. The default, `1L`, uses the plain `plot` to visualise the result. `2L` uses ggplot2 to draw the plot.
`thin`	A numeric argument indicating the gap between samples to be taken when thinning the MCMC draws. The default is `1`, which means no thinning is used.
`thetaTrue`	A set of true parameter values to be included on the plots as a reference line. The default is `NULL`.
`options.plot`	A list of additional arguments to pass into the `plot` function. Only use when `which` is `1L`.
`top`	A character argument of the combined plot title if `which` is `2L`.
`options.density`	A list of additional arguments to pass into the `geom_density` function. Only use when `which` is `2L`.
`options.theme`	A list of additional arguments to pass into the `theme` function. Only use when `which` is `2L`.

Slots

`theta` Object of class “matrix”. MCMC samples from the joint approximate posterior distribution of the parameters.

`loglike` Object of class “numeric”. Accepted MCMC samples of the estimated log-likelihood values.

`call` Object of class “call”. The original code that was used to call the method.

`model` Object of class “MODEL”.

`acceptanceRate` Object of class “numeric”. The acceptance rate of the MCMC algorithm.

`earlyRejectionRate` Object of class “numeric”. The early rejection rate of the algorithm (early rejection may occur when using bounded prior distributions).

`errorRate` Object of class “numeric”. The error rate. If any infinite summary statistic or infinite log-likelihood estimate occurs during the process, it is marked as an error and the proposed parameter will be rejected.

`y` Object of class “ANY”. The observed data.

`n` Object of class “numeric”. The number of simulations from the model per MCMC iteration to estimate the synthetic likelihood.

`M` Object of class “numeric”. The number of MCMC iterations.

`covRandWalk` Object of class “matrix”. The covariance matrix used in multivariate normal random walk proposals.

`method` Object of class “character”. The character argument indicating the used method.

`shrinkage` Object of class “characterOrNULL”. The character argument indicating the shrinkage method.

`penalty` Object of class “numericOrNULL”. The penalty value.

`GRC` Object of class “logical”. Whether the Gaussian rank correlation matrix is used.

`logitTransform` Object of class “logical”. The logical argument indicating whether a logit transformation is used in the algorithm.

`logitTransformBound` Object of class “matrixOrNULL”. The matrix of logitTransformBound.

`standardise` Object of class “logical”. The logical argument that determines whether to standardise the summary statistics.

`parallel` Object of class “logical”. The logical value indicating whether parallel computing is used in the process.

`parallelArgs` Object of class “listOrNULL”. The list of additional arguments to pass into the `foreach` function.

`time` Object of class “difftime”. The running time.

`gamma` Object of class “numeric”. MCMC samples of gamma parameter values of the mean adjustment or variance inflation for method “BSLmisspec”.

`misspecType` Object of class “characterOrNULL”. The character argument indicating whether mean adjustment ("mean") or variance inflation ("variance") to be used in "BSLmisspec" method.

`tau` Object of class “numeric”. Parameter of the prior distribution for "BSLmisspec" method. For mean adjustment, `tau` is the scale of the Laplace distribution. For variance inflation, `tau` is the mean of the exponential distribution.

`whitening` Object of class “logicalOrMatrixOrNULL”. A logical argument determines whether Whitening transformation is used in “BSL” method with Warton’s shrinkage, or just the Whitening matrix used.

Examples

```
## Not run:
# a toy example
toy_simVec <- function(n, theta) matrix(rnorm(n, theta), nrow = n) # the simulation function
toy_sum <- function(x) x # the summary statistic function
model <- newModel(fnSimVec = toy_simVec, fnSum = toy_sum, theta0 = 0) # create the model object
result_toy <- bsl(y = 1, n = 100, M = 1e4, model = model, covRandWalk = matrix(1))
summary(result_toy)
plot(result_toy)

## End(Not run)
```

`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. (2019).

Usage

```
data(ma2)

cell_sim(theta, Yinit, rows, cols, sim_iters, num_obs)

cell_sum(Y, Yinit)

cell_prior(theta)
```

Arguments

`theta`	A vector of proposed model parameters, P_m and P_p.
`Yinit`	The initial matrix of cell presences of size `rows` × `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` × `cols` × `num_obs` array of the cell presences at times `1:num_obs` (not time 0).

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` × `cols`) of cell presences.

The model that we consider is a random walk model with parameters for the probability of cell movement (P_m) and the probability of cell proliferation (P_p) and it has no tractable likelihood function. We use the vague priors $P_m \sim U(0,1)$ and $P_p \sim U(0,1)$.

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. (2019). Johnston et al. (2014) use a different ABC method and different summary statistics for a similar example.

Functions

- `cell_sim`: The function `cell_sim(theta, Yinit, rows, cols,sim_iters, num_obs)` simulates data from the model, using C++ in the backend.
- `cell_sum`: The function `cell_sum(Y,sum_options)` calculates the summary statistics for this example.
- `cell_prior`: The function `cell_prior(theta)` evaluates the log prior density at the parameter value θ.

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 $P_m = 0.35$ and $P_p = 0.001$. 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. (2019).

- `data`: The `rows` × `cols` × `num_obs` array of the cell presences at times 1:144.
- `sim_args`: Values of `sim_args` relevant to this example.
- `sum_args`: Values of `sum_args` relevant to this example, i.e. just the value of `Yinit`.
- `start`: A vector of suitable initial values of the parameters for MCMC.
- `cov`: The covariance matrix of a multivariate normal random walk proposal distribution used in the MCMC, in the form of a 2×2 matrix.

Author(s)

Ziwen An, Leah F. South and Christopher Drovandi

References

An Z, South LF, Nott DJ, Drovandi CC (2019). "Accelerating Bayesian Synthetic Likelihood With the Graphical Lasso." *Journal of Computational and Graphical Statistics*, **28**(2), 471–475. doi: 10.1080/10618600.2018.1537928.

Johnston ST, Simpson MJ, McElwain DLS, Binder BJ, Ross JV (2014). "Interpreting scratch assays using pair density dynamics and approximate Bayesian computation." *Open Biology*, **4**(9), 140097. doi: 10.1098/rsob.140097.

Price LF, Drovandi CC, Lee A, Nott DJ (2018). "Bayesian Synthetic Likelihood." *Journal of Computational and Graphical Statistics*, **27**, 1–11. doi: 10.1080/10618600.2017.1302882.

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)
model <- newModel(fnSim = cell_sim, fnSum = cell_sum, simArgs = cell$sim_args,
                  sumArgs = cell$sum_args, theta0 = cell$start, fnLogPrior = cell_prior,
                  thetaNames = expression(P[m], P[p]))
thetaExact <- c(0.35, 0.001)

# Performing BSL (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultCellBSL <- bsl(cell$data, n = 5000, M = 10000, model = model, covRandWalk = cell$cov,
                     parallel = TRUE, verbose = 1L)
stopCluster(cl)
registerDoSEQ()
show(resultCellBSL)
summary(resultCellBSL)
plot(resultCellBSL, thetaTrue = thetaExact, thin = 20)

# Performing uBSL (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultCelluBSL <- bsl(cell$data, n = 5000, M = 10000, model = model, covRandWalk = cell$cov,
                      method = "uBSL", parallel = TRUE, verbose = 1L)
stopCluster(cl)
registerDoSEQ()
show(resultCelluBSL)
summary(resultCelluBSL)
plot(resultCelluBSL, thetaTrue = thetaExact, thin = 20)

# Performing tuning for BSLasso
ssy <- cell_sum(cell$data, cell$sum_args$Yinit)
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)))
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
set.seed(100)
sp_cell <- selectPenalty(ssy, n = c(500, 1000, 1500, 2000), lambda_all, theta = thetaExact,
    M = 100, sigma = 1.5, model = model, method = "BSL", shrinkage = "glasso",
    parallelSim = TRUE, parallelMain = FALSE)
stopCluster(cl)
registerDoSEQ()
sp_cell
plot(sp_cell)

# Performing BSLasso with a fixed penalty (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultCellBSLasso <- bsl(cell$data, n = 1500, M = 10000, model = model, covRandWalk = cell$cov,
                      shrinkage = "glasso", penalty = 1.3, parallel = TRUE, verbose = 1L)
```

```
stopCluster(cl)
registerDoSEQ()
show(resultCellBSLasso)
summary(resultCellBSLasso)
plot(resultCellBSLasso, thetaTrue = thetaExact, thin = 20)

# Performing semiBSL (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultCellSemiBSL <- bsl(cell$data, n = 5000, M = 10000, model = model, covRandWalk = cell$cov,
                         method = "semiBSL", parallel = TRUE, verbose = 1L)
stopCluster(cl)
registerDoSEQ()
show(resultCellSemiBSL)
summary(resultCellSemiBSL)
plot(resultCellSemiBSL, thetaTrue = thetaExact, thin = 20)

# Plotting the results together for comparison
# plot using the R default plot function
oldpar <- par()
par(mar = c(5, 4, 1, 2), oma = c(0, 1, 2, 0))
combinePlotsBSL(list(resultCellBSL, resultCelluBSL, resultCellBSLasso, resultCellSemiBSL),
   which = 1, thetaTrue = thetaExact, thin = 20, label = c("bsl", "ubsl", "bslasso", "semiBSL"),
    col = 1:4, lty = 1:4, lwd = 1)
mtext("Approximate Univariate Posteriors", outer = TRUE, cex = 1.5)
par(mar = oldpar$mar, oma = oldpar$oma)


## End(Not run)
```

`combinePlotsBSL` *Plot the densities of multiple "bsl" class objects.*

Description

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

Usage

```
combinePlotsBSL(
  objectList,
  which = 1L,
  thin = 1,
  burnin = 0,
  thetaTrue = NULL,
  label = NULL,
```

```
  legendPosition = c("auto", "right", "bottom")[1],
  legendNcol = NULL,
  col = NULL,
  lty = NULL,
  lwd = NULL,
  cex.lab = 1,
  cex.axis = 1,
  cex.legend = 0.75,
  top = "Approximate Marginal Posteriors",
  options.color = list(),
  options.linetype = list(),
  options.size = list(),
  options.theme = list()
)
```

Arguments

`objectList`	A list of “bsl” class objects.
`which`	An integer argument indicating which plot function to be used. The default, `1L`, uses the plain `plot` to visualise the result. `2L` uses ggplot2 to draw the plot.
`thin`	A numeric argument indicating the gap between samples to be taken when thinning the MCMC draws. The default is `1`, which means no thinning is used.
`burnin`	the number of MCMC burn-in steps to be taken.
`thetaTrue`	A set of true parameter values to be included on the plots as a reference line. The default is `NULL`.
`label`	A string vector indicating the labels to be shown in the plot legend. The default is `NULL`, which uses the names from `objectList`.
`legendPosition`	One of the three string arguments, “auto”, “right” or “bottom”, indicating the legend position. The default is “auto”, which automatically choose from “right” and “bottom”. Only used when `which` is `1L`.
`legendNcol`	An integer argument indicating the number of columns of the legend. The default, `NULL`, put all legends in the same row or column depending on `legendPosition`. Only used when `which` is `1L`.
`col`	A vector argument containing the plotting color for each density curve. Each element of the vector will be passed into `lines`. Only used when `which` is `1L`.
`lty`	A vector argument containing the line type for each density curve. Each element of the vector will be passed into `lines`. Only used when `which` is `1L`.
`lwd`	A vector argument containing the line width for each density curve. Each element of the vector will be passed into `lines`. Only used when `which` is `1L`.
`cex.lab`	The magnification to be used for x and y labels relative to the current setting of cex. To be passed into `plot`. Only used when `which` is `1L`.
`cex.axis`	The magnification to be used for axis annotation relative to the current setting of cex. To be passed into `plot`. Only used when `which` is `1L`.
`cex.legend`	The magnification to be used for legend annotation relative to the current setting of cex. Only used when `which` is `1L`.

`top`	A string argument of the combined plot title. Only used when `which` is `2L`.
`options.color`	A list of additional arguments to pass into function `ggplot2::scale_color_manual`. Only used when `which` is `2L`.
`options.linetype`	A list of additional arguments to pass into function `ggplot2::scale_linetype_manual`. Only used when `which` is `2L`.
`options.size`	A list of additional arguments to pass into function `ggplot2::scale_size_manual`. Only used when `which` is `2L`.
`options.theme`	A list of additional arguments to pass into the `theme` function. Only use when `which` is `2L`.

Value

No return value, called for the plots produced.

See Also

`ma2`, `cell`, `mgnk` and `toad` for examples.

Examples

```
## Not run:
toy_sim <- function(n, theta) matrix(rnorm(2*n, theta), nrow = n)
toy_sum <- ma2_sum

model <- newModel(fnSimVec = toy_sim, fnSum = toy_sum, sumArgs = list(epsilon = 2), theta0 = 0)

result1 <- bsl(y = 1:2, n = 100, M = 5e3, model = model, covRandWalk = matrix(1),
               method = "BSL", plotOnTheFly = TRUE)
result2 <- bsl(y = 1:2, n = 100, M = 5e3, model = model, covRandWalk = matrix(1),
               method = "uBSL", plotOnTheFly = TRUE)
result3 <- bsl(y = 1:2, n = 100, M = 5e3, model = model, covRandWalk = matrix(1),
               method = "semiBSL", plotOnTheFly = TRUE)
combinePlotsBSL(list(result1, result2, result3), label = c("BSL","uBSL","semiBSL"), thin = 20)

## End(Not run)
```

`cor2cov` *Convert a correlation matrix to a covariance matrix*

Description

This function converts a correlation matrix to a covariance matrix

Usage

```
cor2cov(corr, std)
```

Arguments

`corr`	The correlation matrix to be converted. This must be symmetric.
`std`	A vector that contains the standard deviations of the variables in the correlation matrix.

Value

The covariance matrix.

`estimateLoglike` *Estimate the synthetic likelihood*

Description

This function computes the estimated synthetic (log) likelihood using one of the four methods ("BSL", "uBSL", "semiBSL" and "BSLmisspec"). Please find the links below in the see also section for more details.

Usage

```
estimateLoglike(
  ssy,
  ssx,
  method = c("BSL", "uBSL", "semiBSL", "BSLmisspec"),
  log = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`ssy`	The observed summary statisic.
`ssx`	A matrix of the simulated summary statistics. The number of rows is the same as the number of simulations per iteration.
`method`	A string argument indicating the method to be used. The default, "BSL", runs standard BSL. "uBSL" uses the unbiased estimator of a normal density of Ghurye and Olkin (1969). "semiBSL" runs the semi-parametric BSL algorithm and is more robust to non-normal summary statistics. "BSLmisspec" estimates the Gaussian synthetic likelihood whilst acknowledging that there may be incompatibility between the model and the observed summary statistic (Frazier and Drovandi 2021).
`log`	A logical argument indicating if the log of likelihood is given as the result. The default is `TRUE`.
`verbose`	A logical argument indicating whether an error message should be printed if the function fails to compute a likelihood. The default is `FALSE`.

`...`	Arguments to be passed to methods.

- `shrinkage` Available for methods “BSL” and “semiBSL”. A string argument indicating which shrinkage method to be used. The default is `NULL`, which means no shrinkage is used. Shrinkage estimation is only available for methods “BSL” and “semiBSL”. Current options are “glasso” for the graphical lasso method of Friedman et al. (2008) and “Warton” for the ridge regularisation method of Warton (2008).
- `penalty` Available for methods “BSL” and “semiBSL”. The penalty value to be used for the specified shrinkage method. Must be between zero and one if the shrinkage method is “Warton”.
- `standardise` Available for method “BSL”. A logical argument that determines whether to standardise the summary statistics before applying the graphical lasso. This is only valid if method is “BSL”, shrinkage is “glasso” and penalty is not `NULL`. The diagonal elements will not be penalised if the shrinkage method is “glasso”. The default is `FALSE`.
- `GRC` Available for method “BSL”. A logical argument indicating whether the Gaussian rank correlation matrix (Boudt et al. 2012) should be used to estimate the covariance matrix in “BSL” method. The default is `FALSE`, which uses the sample covariance by default.
- `whitening` Available for method “BSL”. This argument determines whether Whitening transformation should be used in “BSL” method with Warton’s shrinkage. Whitening transformation helps decorrelate the summary statistics, thus encourages sparsity of the synthetic likelihood covariance matrix. This might allow heavier shrinkage to be applied without losing much accuracy, hence allowing the number of simulations to be reduced. By default, `NULL` represents no Whitening transformation. Otherwise this is enabled if a Whitening matrix is provided. See `estimateWhiteningMatrix` for the function to estimate the Whitening matrix.
- `ssyTilde` Available for method “BSL”. The whitened observed summary statisic. If this is not `NULL`, it will be used to save computation effort. Only used if Whitening is enabled.
- `kernel` Available for method “semiBSL”. A string argument indicating the smoothing kernel to pass into `density` for estimating the marginal distribution of each summary statistic. Only “gaussian" and “epanechnikov" are available. The default is “gaussian".
- `type` Available for method “BSLmisspec”. A string argument indicating which method is used to account for and detect potential incompatibility. The two options are "mean" and "variance".
- `gamma` Available for method “BSLmisspec”. The additional latent parameter to account for possible incompatability between the model and observed summary statistic. In “BSLmisspec” method, this is updated with a slice sampler (Neal 2003).

Value

The estimated synthetic (log) likelihood value.

References

Boudt K, Cornelissen J, Croux C (2012). “The Gaussian Rank Correlation Estimator: Robustness Properties.” *Statistics and Computing*, **22**(2), 471–483. doi: 10.1007/s1122201192370.

Frazier DT, Drovandi C (2021). “Robust Approximate Bayesian Inference with Synthetic Likelihood.” *Journal of Computational and Graphical Statistics (In Press)*. `https://arxiv.org/abs/1904.04551`.

Friedman J, Hastie T, Tibshirani R (2008). “Sparse Inverse Covariance Estimation with the Graphical Lasso.” *Biostatistics*, **9**(3), 432–441.

Ghurye SG, Olkin I (1969). “Unbiased Estimation of Some Multivariate Probability Densities and Related Functions.” *Ann. Math. Statist.*, **40**(4), 1261–1271.

Neal RM (2003). “Slice sampling.” *The Annals of Statistics*, **31**(3), 705–767.

Warton DI (2008). “Penalized Normal Likelihood and Ridge Regularization of Correlation and Covariance Matrices.” *Journal of the American Statistical Association*, **103**(481), 340–349. doi: 10.1198/016214508000000021.

See Also

`gaussianSynLike`, `gaussianSynLikeGhuryeOlkin`, `semiparaKernelEstimate` and `synLikeMisspec`.

Examples

```
data(ma2)
ssy <- ma2_sum(ma2$data)
m <- newModel(fnSim = ma2_sim, fnSum = ma2_sum, simArgs = ma2$sim_args,
                  theta0 = ma2$start)
ssx <- simulation(m, n = 300, theta = c(0.6, 0.2), seed = 10)$ssx
estimateLoglike(ssy, ssx, method = "BSL")
estimateLoglike(ssy, ssx, method = "uBSL")
estimateLoglike(ssy, ssx, method = "semiBSL")
estimateLoglike(ssy, ssx, method = "BSLmisspec", type = "mean", gamma = rep(0.1, 50))
```

`estimateWhiteningMatrix` *Estimate the Whitening matrix to be used in the “wBSL” method of Priddle et al. (2021)*

Description

This function estimates the Whitening matrix to be used in BSL with Warton’s shrinkage and Whitening (“wBSL” method of Priddle et al. (2021)). The Whitening transformation and decorrelation methods are detailed in Kessy et al. (2018).

Usage

```
estimateWhiteningMatrix(
  n,
  model,
  method = c("PCA", "ZCA", "Cholesky", "ZCA-cor", "PCA-cor"),
  thetaPoint = NULL,
  parallel = FALSE,
  parallelArgs = NULL
)
```

Arguments

`n`	The number of model simulations to estimate the Whitening matrix.
`model`	A "MODEL" object generated with function `newModel`. See `newModel`.
`method`	The type of Whitening method to be used. The default is "PCA".
`thetaPoint`	A point estimate of the parameter value with non-negligible posterior support.
`parallel`	A logical value indicating whether parallel computing should be used for simulation and summary statistic evaluation. The default is `FALSE`. When model simulation is fast, it may be preferable to perform serial or vectorised computations to avoid significant communication overhead between workers. Parallel computation can only be used if not using a vectorised simulation function, see `MODEL` for options of vectorised simulation function.
`parallelArgs`	A list of additional arguments to pass into the `foreach` function. Only used when parallel computing is enabled, default is `NULL`.

Value

The estimated Whitening matrix.

References

Kessy A, Lewin A, Strimmer K (2018). "Optimal Whitening and Decorrelation." *The American Statistician*, **72**(4), 309–314. doi: 10.1080/00031305.2016.1277159.

Priddle JW, Sisson SA, Frazier DT, Turner I, Drovandi C (2021). "Efficient Bayesian Synthetic Likelihood with Whitening Transformations." *Journal of Computational and Graphical Statistics (In Press)*. `https://arxiv.org/abs/1909.04857`.

Examples

```
## Not run:
data(ma2)
model <- newModel(fnSim = ma2_sim, fnSum = ma2_sum, simArgs = ma2$sim_args, theta0 = ma2$start)
W <- estimateWhiteningMatrix(20000, model, method = "PCA", thetaPoint = c(0.6, 0.2))

## End(Not run)
```

`gaussianRankCorr` *Gaussian rank correlation*

Description

This function computes the Gaussian rank correlation of Boudt et al. (2012).

Usage

```
gaussianRankCorr(x, vec = FALSE)
```

Arguments

`x`	A numeric matrix representing data where the number of rows is the number of independent data points and the number of columns is the number of variables in the dataset.
`vec`	A logical argument indicating if the vector of correlations should be returned instead of a matrix.

Value

Gaussian rank correlation matrix (default) or a vector of pair correlations.

References

Boudt K, Cornelissen J, Croux C (2012). “The Gaussian Rank Correlation Estimator: Robustness Properties.” *Statistics and Computing*, **22**(2), 471–483. doi: 10.1007/s1122201192370.

See Also

`cor2cov` for conversion from correlation matrix to covariance matrix.

Examples

```
data(ma2)
model <- newModel(fnSimVec = ma2_sim_vec, fnSum = ma2_sum, simArgs = list(TT = 10),
                  theta0 = ma2$start, fnLogPrior = ma2_prior)
set.seed(100)

# generate 1000 simualtions from the ma2 model
x <- simulation(model, n = 1000, theta = c(0.6, 0.2))$x

corr1 <- cor(x) # traditional correlation matrix
corr2 <- gaussianRankCorr(x) # Gaussian rank correlation matrix
oldpar <- par()
par(mfrow = c(1, 2))
image(corr1, main = 'traditional correlation matrix')
image(corr2, main = 'Gaussian rank correlation matrix')
par(mfrow = oldpar$mfrow)
```

```
std <- apply(x, MARGIN = 2, FUN = sd) # standard deviations
cor2cov(gaussianRankCorr(x), std) # convert to covariance matrix
```

`gaussianSynLike` *Estimate the Gaussian synthetic (log) likelihood*

Description

This function estimates the Gaussian synthetic log-likelihood (see Wood 2010 and Price et al. 2018). Several extensions are provided in this function: `shrinkage` enables shrinkage estimation of the covariance matrix and is helpful to bring down the number of model simulations (see An et al. (2019) for an example of BSL with glasso (Friedman et al. 2008) shrinkage estimation); `GRC` uses Gaussian rank correlation (Boudt et al. 2012) to find a more robust correlation matrix; `whitening` (Kessy et al. 2018) could further reduce the number of model simulations upon Warton's shrinkage (Warton 2008) by decorrelating the summary statistics.

Usage

```
gaussianSynLike(
  ssy,
  ssx,
  shrinkage = NULL,
  penalty = NULL,
  standardise = FALSE,
  GRC = FALSE,
  whitening = NULL,
  ssyTilde = NULL,
  log = TRUE,
  verbose = FALSE
)
```

Arguments

`ssy`	The observed summary statisic.
`ssx`	A matrix of the simulated summary statistics. The number of rows is the same as the number of simulations per iteration.
`shrinkage`	A string argument indicating which shrinkage method to be used. The default is `NULL`, which means no shrinkage is used. Shrinkage estimation is only available for methods "BSL" and "semiBSL". Current options are "glasso" for the graphical lasso method of Friedman et al. (2008) and "Warton" for the ridge regularisation method of Warton (2008).
`penalty`	The penalty value to be used for the specified shrinkage method. Must be between zero and one if the shrinkage method is "Warton".

`standardise`	A logical argument that determines whether to standardise the summary statistics before applying the graphical lasso. This is only valid if method is “BSL”, shrinkage is “glasso” and penalty is not `NULL`. The diagonal elements will not be penalised if the shrinkage method is “glasso”. The default is `FALSE`.
`GRC`	A logical argument indicating whether the Gaussian rank correlation matrix (Boudt et al. 2012) should be used to estimate the covariance matrix in “BSL” method. The default is `FALSE`, which uses the sample covariance by default.
`whitening`	This argument determines whether Whitening transformation should be used in “BSL” method with Warton’s shrinkage. Whitening transformation helps decorrelate the summary statistics, thus encouraging sparsity of the synthetic likelihood covariance matrix. This might allow heavier shrinkage to be applied without losing much accuracy, hence allowing the number of simulations to be reduced. By default, `NULL` represents no Whitening transformation. Otherwise this is enabled if a Whitening matrix is provided. See `estimateWhiteningMatrix` for the function to estimate the Whitening matrix.
`ssyTilde`	The whitened observed summary statisic. If this is not `NULL`, it will be used to save computation effort. Only used if Whitening is enabled.
`log`	A logical argument indicating if the log of likelihood is given as the result. The default is `TRUE`.
`verbose`	A logical argument indicating whether an error message should be printed if the function fails to compute a likelihood. The default is `FALSE`.

Value

The estimated synthetic (log) likelihood value.

References

An Z, South LF, Nott DJ, Drovandi CC (2019). “Accelerating Bayesian Synthetic Likelihood With the Graphical Lasso.” *Journal of Computational and Graphical Statistics*, **28**(2), 471–475. doi: 10.1080/10618600.2018.1537928.

Boudt K, Cornelissen J, Croux C (2012). “The Gaussian Rank Correlation Estimator: Robustness Properties.” *Statistics and Computing*, **22**(2), 471–483. doi: 10.1007/s1122201192370.

Friedman J, Hastie T, Tibshirani R (2008). “Sparse Inverse Covariance Estimation with the Graphical Lasso.” *Biostatistics*, **9**(3), 432–441.

Kessy A, Lewin A, Strimmer K (2018). “Optimal Whitening and Decorrelation.” *The American Statistician*, **72**(4), 309–314. doi: 10.1080/00031305.2016.1277159.

Price LF, Drovandi CC, Lee A, Nott DJ (2018). “Bayesian Synthetic Likelihood.” *Journal of Computational and Graphical Statistics*, **27**, 1–11. doi: 10.1080/10618600.2017.1302882.

Warton DI (2008). “Penalized Normal Likelihood and Ridge Regularization of Correlation and Covariance Matrices.” *Journal of the American Statistical Association*, **103**(481), 340–349. doi: 10.1198/016214508000000021.

Wood SN (2010). “Statistical Inference for Noisy Nonlinear Ecological Dynamic Systems.” *Nature*, **466**, 1102–1107. doi: 10.1038/nature09319.

See Also

Other available synthetic likelihood estimators: `gaussianSynLikeGhuryeOlkin` for the unbiased synthetic likelihood estimator, `semiparaKernelEstimate` for the semi-parametric likelihood estimator, `synLikeMisspec` for the Gaussian synthetic likelihood estimator for model misspecification.

Examples

```
data(ma2)
ssy <- ma2_sum(ma2$data)
m <- newModel(fnSim = ma2_sim, fnSum = ma2_sum, simArgs = ma2$sim_args,
              theta0 = ma2$start)
ssx <- simulation(m, n = 300, theta = c(0.6, 0.2), seed = 10)$ssx

# the standard Gaussian synthetic likelihood (the likelihood estimator used in BSL)
gaussianSynLike(ssy, ssx)
# the Gaussian synthetic likelihood with glasso shrinkage estimation
# (the likelihood estimator used in BSLasso)
gaussianSynLike(ssy, ssx, shrinkage = 'glasso', penalty = 0.1)
# the Gaussian synthetic likelihood with Warton's shrinkage estimation
gaussianSynLike(ssy, ssx, shrinkage = 'Warton', penalty = 0.9)
# the Gaussian synthetic likelihood with Warton's shrinkage estimation and Whitening transformation
W <- estimateWhiteningMatrix(20000, m)
gaussianSynLike(ssy, ssx, shrinkage = 'Warton', penalty = 0.9, whitening = W)
```

`gaussianSynLikeGhuryeOlkin` *Estimate the Gaussian synthetic (log) likelihood with an unbiased estimator*

Description

This function computes an unbiased, nonnegative estimate of a normal density function from simulations assumed to be drawn from it. See Price et al. (2018) and Ghurye and Olkin (1969).

Usage

```
gaussianSynLikeGhuryeOlkin(ssy, ssx, log = TRUE, verbose = FALSE)
```

Arguments

`ssy`	The observed summary statisic.
`ssx`	A matrix of the simulated summary statistics. The number of rows is the same as the number of simulations per iteration.

`log`	A logical argument indicating if the log of likelihood is given as the result. The default is `TRUE`.
`verbose`	A logical argument indicating whether an error message should be printed if the function fails to compute a likelihood. The default is `FALSE`.

Value

The estimated synthetic (log) likelihood value.

References

Ghurye SG, Olkin I (1969). "Unbiased Estimation of Some Multivariate Probability Densities and Related Functions." *Ann. Math. Statist.*, **40**(4), 1261–1271.

Price LF, Drovandi CC, Lee A, Nott DJ (2018). "Bayesian Synthetic Likelihood." *Journal of Computational and Graphical Statistics*, **27**, 1–11. doi: 10.1080/10618600.2017.1302882.

See Also

Other available synthetic likelihood estimators: `gaussianSynLike` for the standard synthetic likelihood estimator, `semiparaKernelEstimate` for the semi-parametric likelihood estimator, `synLikeMisspec` for the Gaussian synthetic likelihood estimator for model misspecification.

Examples

```
data(ma2)
ssy <- ma2_sum(ma2$data)
m <- newModel(fnSim = ma2_sim, fnSum = ma2_sum, simArgs = ma2$sim_args,
              theta0 = ma2$start)
ssx <- simulation(m, n = 300, theta = c(0.6, 0.2), seed = 10)$ssx

# unbiased estimate of the Gaussian synthetic likelihood
# (the likelihood estimator used in uBSL)
gaussianSynLikeGhuryeOlkin(ssy, ssx)
```

`getGamma` *Obtain the gamma samples (the latent parameters for BSLmisspec method) from a "BSL" object*

Description

see `BSLclass`

Usage

```
getGamma(object, ...)
```

Arguments

`object`	A “BSL” class object.
`...`	Other arguments.

Value

The matrix of gamma samples (the latent parameters for BSLmisspec method), after removing burn-in and thinning.

`getLoglike` *Obtain the log-likelihoods from a "BSL" object*

Description

see `BSLclass`

Usage

```
getLoglike(object, ...)
```

Arguments

`object`	A “BSL” class object.
`...`	Other arguments.

Value

The vector of log likelihood evaluations, after removing burn-in and thinning.

`getPenalty` *Obtain the selected penalty values from a "PENALTY" object*

Description

see `PENALTYclass`

Usage

```
getPenalty(object, ...)
```

Arguments

`object`	A “PENALTY” class object.
`...`	Other arguments.

Value

The selecty penalty values.

`getTheta` *Obtain the samples from a "BSL" object*

Description

see `BSLclass`

Usage

```
getTheta(object, ...)
```

Arguments

`object`	A “BSL” class object.
`...`	Other arguments.

Value

The matrix of samples, after removing burn-in and thinning.

`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. (2019). The journal article An et al. (2022) provides a full description of how to use this package for the toad example.

Usage

```
data(ma2)

ma2_sim(theta, TT)

ma2_sim_vec(n, theta, TT)

ma2_sum(x, epsilon = 0, delta = 1)

ma2_prior(theta)
```

Arguments

`theta`	A vector of proposed model parameters, θ_1 and θ_2.
`TT`	The number of observations.
`n`	The number of simulations to run with the vectorised simulation function.
`x`	Observed or simulated data in the format of a vector of length TT.
`epsilon`	The skewness parameter in the sinh-arcsinh transformation.
`delta`	The kurtosis parameter in the sinh-arcsinh transformation.

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 $t = 1, \ldots, TT$ and $z_t \sim N(0,1)$ for $t = -1, 0, \ldots, TT$. 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.

Functions

- `ma2_sim`: Simulates an MA(2) time series.
- `ma2_sim_vec`: Simulates n MA(2) time series with a vectorised simulation function.
- `ma2_sum`: Returns the summary statistics for a given data set. The skewness and kurtosis of the summary statistics can be controlled via the ϵ and δ parameters. This is the sinh-arcsinnh transformation of Jones and Pewsey (2009). By default, the summary statistics function simply returns the raw data. Otherwise, the transformation is introduced to motivate the "semiBSL" method.
- `ma2_prior`: Evaluates the (unnormalised) log prior, which is uniform subject to several restrictions related to invertibility of the time series.

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 $TT = 50$. Further information about this model and the specific choices of tuning parameters used in BSL and BSLasso can be found in An et al. (2019).

- `data`: A time series dataset, in the form of a vector of length TT
- `sim_args`: A list containing $TT = 50$
- `start`: A vector of suitable initial values of the parameters for MCMC
- `cov`: The covariance matrix of a multivariate normal random walk proposal distribution used in the MCMC, in the form of a 2×2 matrix

Author(s)

Ziwen An, Leah F. South and Christopher Drovandi

References

An Z, South LF, Drovandi CC (2022). "BSL: An R Package for Efficient Parameter Estimation for Simulation-Based Models via Bayesian Synthetic Likelihood." *Journal of Statistical Software*, **101**(11), 1–33. doi: 10.18637/jss.v101.i11.

An Z, South LF, Nott DJ, Drovandi CC (2019). "Accelerating Bayesian Synthetic Likelihood With the Graphical Lasso." *Journal of Computational and Graphical Statistics*, **28**(2), 471–475. doi: 10.1080/10618600.2018.1537928.

Jones MC, Pewsey A (2009). "Sinh-arcsinh distributions." *Biometrika*, **96**(4), 761–780. ISSN 0006-3444.

Examples

```
## Not run:
# Load the data for this example and set up the model object
data(ma2)
model <- newModel(fnSimVec = ma2_sim_vec, fnSum = ma2_sum, simArgs = ma2$sim_args,
                  theta0 = ma2$start, fnLogPrior = ma2_prior)
thetaExact <- c(0.6, 0.2)

# reduce the number of iterations M if desired for all methods below
# Method 1: standard BSL
resultMa2BSL <- bsl(y = ma2$data, n = 500, M = 300000, model = model, covRandWalk = ma2$cov,
                    method = "BSL", verbose = 1L)
show(resultMa2BSL)
summary(resultMa2BSL)
plot(resultMa2BSL, thetaTrue = thetaExact, thin = 20)

# Method 2: unbiased BSL
resultMa2uBSL <- bsl(y = ma2$data, n = 500, M = 300000, model = model, covRandWalk=ma2$cov,
                     method = "uBSL", verbose = 1L)
show(resultMa2uBSL)
summary(resultMa2uBSL)
plot(resultMa2uBSL, thetaTrue = thetaExact, thin = 20)

# Method 3: BSLasso (BSL with glasso shrinkage estimation)
# tune the penalty parameter fisrt
ssy <- ma2_sum(ma2$data)
lambdaAll <- list(exp(seq(-5.5,-1.5,length.out=20)))
set.seed(100)
penaltyGlasso <- selectPenalty(ssy = ssy, n = 300, lambdaAll, theta = thetaExact,
                  M = 100, sigma = 1.5, model = model, method = "BSL", shrinkage = "glasso")
penaltyGlasso
plot(penaltyGlasso)

resultMa2BSLasso <- bsl(y = ma2$data, n = 300, M = 250000, model = model, covRandWalk=ma2$cov,
                     method = "BSL", shrinkage = "glasso", penalty = 0.027, verbose = 1L)
show(resultMa2BSLasso)
summary(resultMa2BSLasso)
plot(resultMa2BSLasso, thetaTrue = thetaExact, thin = 20)
```

```
# Method 4: BSL with Warton's shrinkage and Whitening
# estimate the Whtieing matrix and tune the penalty parameter first
W <- estimateWhiteningMatrix(20000, model, method = "PCA", thetaPoint = ma2$start)
gammaAll <- list(seq(0.3, 0.8, 0.02))
set.seed(100)
penaltyWarton <- selectPenalty(ssy = ssy, n = 300, gammaAll, theta = thetaExact,
                 M = 100, sigma = 1.2, model = model, method = "BSL", shrinkage = "Warton",
                        whitening = W)
penaltyWarton
plot(penaltyWarton, logscale = FALSE)

resultMa2Whitening <- bsl(y = ma2$data, n = 300, M = 250000, model = model, covRandWalk=ma2$cov,
                        method = "BSL", shrinkage = "Warton", whitening = W,
                        penalty = 0.52, verbose = 1L)
show(resultMa2Whitening)
summary(resultMa2Whitening)
plot(resultMa2Whitening, thetaTrue = thetaExact, thin = 20)

# Method 5: semiBSL, the summary statistics function is different from previous methods
model2 <- newModel(fnSimVec = ma2_sim_vec, fnSum = ma2_sum, simArgs = ma2$sim_args,
                sumArgs = list(epsilon = 2), theta0 = ma2$start, fnLogPrior = ma2_prior)
sim <- simulation(model, n = 1e4, theta = ma2$start, seed = 1) # run a short simulation
plot(density(sim$ssx[, 1])) # the first marginal summary statistic is right-skewed
resultMa2SemiBSL <- bsl(y = ma2$data, n = 500, M = 200000, model = model2, covRandWalk=ma2$cov,
                        method = "semiBSL", verbose = 1L)
show(resultMa2SemiBSL)
summary(resultMa2SemiBSL)
plot(resultMa2SemiBSL, thetaTrue = thetaExact, thin = 20)

# Method 6: BSL with consideration of model misspecification (mean adjustment)
resultMa2Mean <- bsl(y = ma2$data, n = 500, M = 200000, model = model, covRandWalk=ma2$cov,
                        method = "BSLmisspec", misspecType = "mean", verbose = 1L)
show(resultMa2Mean)
summary(resultMa2Mean)
plot(resultMa2Mean, thetaTrue = thetaExact, thin = 20)

# Method 7: BSL with consideration of model misspecification (variance inflation)
resultMa2Variance <- bsl(y = ma2$data, n = 500, M = 200000, model = model, covRandWalk=ma2$cov,
                     method = "BSLmisspec", misspecType = "variance", verbose = 1L)
show(resultMa2Variance)
summary(resultMa2Variance)
plot(resultMa2Variance, thetaTrue = thetaExact, thin = 20)

# Plotting the results together for comparison
# plot using the R default plot function
oldpar <- par()
par(mar = c(5, 4, 1, 2), oma = c(0, 1, 2, 0))
combinePlotsBSL(list(resultMa2BSL, resultMa2uBSL, resultMa2BSLasso, resultMa2SemiBSL), which = 1,
           thetaTrue = thetaExact, thin = 20, label = c("bsl", "uBSL", "bslasso", "semiBSL"),
                col = c("black", "red", "blue", "green"), lty = 1:4, lwd = 1)
mtext("Approximate Univariate Posteriors", outer = TRUE, cex = 1.5)
```

```
# plot using the ggplot2 package
combinePlotsBSL(list(resultMa2BSL, resultMa2uBSL, resultMa2BSLasso, resultMa2SemiBSL), which = 2,
    thetaTrue = thetaExact, thin = 20, label = c("bsl", "ubsl", "bslasso", "semiBSL"),
    options.color = list(values=c("black", "red", "blue", "green")),
    options.linetype = list(values = 1:4), options.size = list(values = rep(1, 4)),
    options.theme = list(plot.margin = grid::unit(rep(0.03,4), "npc"),
     axis.title = ggplot2::element_text(size=12), axis.text = ggplot2::element_text(size = 8),
        legend.text = ggplot2::element_text(size = 12)))
par(mar = oldpar$mar, oma = oldpar$oma)

## 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. (2019).

Usage

```
data(mgnk)

mgnk_sim(theta_tilde, TT, J, bound)

mgnk_sum(y)
```

Arguments

`theta_tilde`	A vector with 15 elements for the proposed model parameters.
`TT`	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 `TT` × `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. 2019) at <doi:10.1080/10618600.2018.1537928> for information on the model and summary statistic used in this example.

An example dataset

We use the foreign currency exchange data available from https://www.rba.gov.au/statistics/historical-data.html as in An et al. (2019).

- `data`: A 1651 × 3 matrix of data.
- `sim_args`: Values of `sim_args` relevant to this example.
- `start`: A vector of suitable initial values of the parameters for MCMC.
- `cov`: The covariance matrix of a multivariate normal random walk proposal distribution used in the MCMC, in the form of a 15 by 15 matrix

Author(s)

Ziwen An, Leah F. South and Christopher Drovandi

References

An Z, South LF, Nott DJ, Drovandi CC (2019). "Accelerating Bayesian Synthetic Likelihood With the Graphical Lasso." *Journal of Computational and Graphical Statistics*, **28**(2), 471–475. doi: 10.1080/10618600.2018.1537928.

Drovandi CC, Pettitt AN (2011). "Likelihood-free Bayesian estimation of multivariate quantile distributions." *Computational Statistics & Data Analysis*, **55**(9), 2541–2556. ISSN 0167-9473, doi: 10.1016/j.csda.2011.03.019.

Examples

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

# Loading the data for this example
data(mgnk)
model <- newModel(fnSim = mgnk_sim, fnSum = mgnk_sum, simArgs = mgnk$sim_args, theta0 = mgnk$start,
    thetaNames = expression(a[1],b[1],g[1],k[1],a[2],b[2],g[2],k[2],
                            a[3],b[3],g[3],k[3],delta[12],delta[13],delta[23]))

# Performing BSL (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultMgnkBSL <- bsl(mgnk$data, n = 60, M = 80000, model = model, covRandWalk = mgnk$cov,
    method = "BSL", parallel = FALSE, verbose = 1L, plotOnTheFly = TRUE)
stopCluster(cl)
registerDoSEQ()
show(resultMgnkBSL)
summary(resultMgnkBSL)
plot(resultMgnkBSL, which = 2, thin = 20)

# Performing uBSL (reduce the number of iterations M if desired)
```

```
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultMgnkuBSL <- bsl(mgnk$data, n = 60, M = 80000, model = model, covRandWalk = mgnk$cov,
    method = "uBSL", parallel = FALSE, verbose = 1L)
stopCluster(cl)
registerDoSEQ()
show(resultMgnkuBSL)
summary(resultMgnkuBSL)
plot(resultMgnkuBSL, which = 2, thin = 20)


# Performing tuning for BSLasso
ssy <- mgnk_sum(mgnk$data)
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)))

# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
set.seed(100)
sp_mgnk <- selectPenalty(ssy, n = c(15, 20, 30, 50), lambda = lambda_all, theta = mgnk$start,
   M = 100, sigma = 1.5, model = model, method = "BSL", shrinkage = "glasso", standardise = TRUE,
    parallelSim = TRUE, parallelSimArgs = list(.packages = "MASS", .export = "ninenum"),
    parallelMain = TRUE)
stopCluster(cl)
registerDoSEQ()
sp_mgnk
plot(sp_mgnk)

# Performing BSLasso with a fixed penalty (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultMgnkBSLasso <- bsl(mgnk$data, n = 20, M = 80000, model = model, covRandWalk = mgnk$cov,
   method = "BSL", shrinkage = "glasso", penalty = 0.3, standardise = TRUE, parallel = FALSE,
    verbose = 1L)
stopCluster(cl)
registerDoSEQ()
show(resultMgnkBSLasso)
summary(resultMgnkBSLasso)
plot(resultMgnkBSLasso, which = 2, thin = 20)


# Performing semiBSL (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultMgnkSemiBSL <- bsl(mgnk$data, n = 60, M = 80000, model = model, covRandWalk = mgnk$cov,
    method = "semiBSL", parallel = FALSE, verbose = 1L)
stopCluster(cl)
registerDoSEQ()
show(resultMgnkSemiBSL)
```

```
summary(resultMgnkSemiBSL)
plot(resultMgnkSemiBSL, which = 2, thin = 20)

# Plotting the results together for comparison
# plot using the R default plot function
oldpar <- par()
par(mar = c(4, 4, 1, 1), oma = c(0, 1, 2, 0))
combinePlotsBSL(list(resultMgnkBSL, resultMgnkuBSL, resultMgnkBSLasso, resultMgnkSemiBSL),
                which = 1, thin = 20, label = c("bsl", "ubsl", "bslasso", "semiBSL"),
                col = c("red", "yellow", "blue", "green"), lty = 2:5, lwd = 1)
mtext("Approximate Univariate Posteriors", outer = TRUE, line = 0.75, cex = 1.2)

# plot using the ggplot2 package
combinePlotsBSL(list(resultMgnkBSL, resultMgnkuBSL, resultMgnkBSLasso, resultMgnkSemiBSL),
    which = 2, thin = 20, label=c("bsl","ubsl","bslasso","semiBSL"),
    options.color=list(values=c("red","yellow","blue","green")),
    options.linetype = list(values = 2:5), options.size = list(values = rep(1, 4)),
    options.theme = list(plot.margin = grid::unit(rep(0.03,4),"npc"),
     axis.title = ggplot2::element_text(size=12), axis.text = ggplot2::element_text(size = 8),
        legend.text = ggplot2::element_text(size = 12)))
par(mar = oldpar$mar, oma = oldpar$oma)

## End(Not run)
```

`MODEL-class` *S4 class “MODEL”*

Description

The S4 class contains the simulation and summary statistics function and other necessary arguments for a model to run in the main `bsl` function.

`newModel` is the constructor function for a `MODEL` object.

`simulation` runs a number of simulations and computes the correponding summary statistics with the provided model.

`summStat` computes the summary statistics with the given data and model object. The summary statistics function and relevant arguments are obtained from the model.

Usage

```
newModel(
  fnSim,
  fnSimVec,
  fnSum,
  fnLogPrior,
  simArgs,
  sumArgs,
  theta0,
```

```
  thetaNames,
  test = TRUE,
  verbose = TRUE
)

## S4 method for signature 'MODEL'
simulation(
  model,
  n = 1,
  theta = model@theta0,
  summStat = TRUE,
  parallel = FALSE,
  parallelArgs = NULL,
  seed = NULL
)

## S4 method for signature 'ANY,MODEL'
summStat(x, model)
```

Arguments

`fnSim`	A function that simulates data for a given parameter value. The first argument should be the parameters. Other necessary arguments (optional) can be specified with `simArgs`.
`fnSimVec`	A vectorised function that simulates a number of datasets simultaneously for a given parameter value. The first two arguments should be the number of simulations to run and parameters, respectively. Other necessary arguments (optional) can be specified with `simArgs`. The output must be a list of each simulation result or a matrix with each row corresponding to a simulation.
`fnSum`	A function for computing summary statistics of data. The first argument should be the observed or simulated dataset. Other necessary arguments (optional) can be specified with `sumArgs`.
`fnLogPrior`	A function that computes the log of prior density for a parameter. If this is missing, the prior by default is an improper flat prior over the real line for each parameter. The function must have a single input: a vector of parameter values.
`simArgs`	A list of additional arguments to pass into the simulation function. Only use when the input `fnSim` requires additional arguments.
`sumArgs`	A list of additional arguments to pass into the summary statistics function. Only use when the input `fnSum` requires additional arguments.
`theta0`	Initial guess of the parameter value.
`thetaNames`	A string vector of parameter names, which must have the same length as the parameter vector.
`test`	Logical, whether a short simulation test will be ran upon initialisation.
`verbose`	Logical, whether to print verbose messages when initialising a “MODEL” object.
`model`	A “MODEL” class object.

`n`	The number of simulations to run.
`theta`	The parameter value.
`summStat`	Logical indicator whether the correpsonding summary statistics should be returned or not. The default is `TRUE`.
`parallel`	A logical value indicating whether parallel computing should be used for simulation and summary statistic evaluation. The default is `FALSE`. When model simulation is fast, it may be preferable to perform serial or vectorised computations to avoid significant communication overhead between workers. Parallel computation can only be used if not using a vectorised simulation function, see `MODEL` for options of vectorised simulation function.
`parallelArgs`	A list of additional arguments to pass into the `foreach` function. Only used when parallel computing is enabled, default is `NULL`.
`seed`	A seed number to pass to the `set.seed` function. The default is `NULL`, when no seed number is specified. Please note `parallel` also affects the result even with the same seed.
`x`	The data to pass to the summary statistics function.

Value

A list of simulation results using the given parameter. `x` contains the raw simulated datasets. `ssx` contains the summary statistics.

A vector of the summary statistics.

Slots

`fnSim` A function that simulates data for a given parameter value. The first argument should be the parameters. Other necessary arguments (optional) can be specified with `simArgs`.

`fnSimVec` A vectorised function that simulates a number of datasets simultaneously for a given parameter value. If this is not `NULL`, vectorised simulation function will be used instead of `fnSim`. The first two arguments should be the number of simulations to run and parameters, respectively. Other necessary arguments (optional) can be specified with `simArgs`. The output must be a list of each simulation result.

`fnSum` A function for computing summary statistics of data. The first argument should be the observed or simulated dataset. Other necessary arguments (optional) can be specified with `sumArgs`. The users should code this function carefully so the output have fixed length and never contain any `Inf` value.

`fnLogPrior` A function that computes the log of prior density for a parameter. The default is `NULL`, which uses an improper flat prior over the real line for each parameter. The function must have a single input: a vector of parameter values.

`simArgs` A list of additional arguments to pass into the simulation function. Only use when the input `fnSim` or `fnSimVec` requires additional arguments. The default is `NULL`.

`sumArgs` A list of additional arguments to pass into the summary statistics function. Only use when the input `fnSum` requires additional arguments. The default is `NULL`.

`theta0` Initial guess of the parameter value, which is used as the starting value for MCMC.

`thetaNames` Expression, parameter names.

`ns` The number of summary statistics of a single observation. Note this will be generated automatically, thus is not required for initialisation.

`test` Logical, whether a short simulation test will be ran upon initialisation.

`verbose` Logical, whether to print verbose messages when initialising a “MODEL” object.

Examples

```
# set up the model for the ma2 example
data(ma2)
m <- newModel(fnSim = ma2_sim, fnSum = ma2_sum, simArgs = ma2$sim_args,
                  theta0 = ma2$start, fnLogPrior = ma2_prior, verbose = FALSE)
validObject(m)

# benchmark the serial and vectorised simulation function (require the rbenchmark package)
m1 <- newModel(fnSim = ma2_sim, fnSum = ma2_sum, simArgs = ma2$sim_args,
            theta0 = ma2$start, fnLogPrior = ma2_prior)
m2 <- newModel(fnSimVec = ma2_sim_vec, fnSum = ma2_sum, simArgs = ma2$sim_args,
            theta0 = ma2$start, fnLogPrior = ma2_prior)
require("rbenchmark")

## Not run:
benchmark(serial  = simulation(m1, n = 1000, theta = c(0.6, 0.2)),
          vectorised  = simulation(m2, n = 1000, theta = c(0.6, 0.2)))

## End(Not run)
```

`obsMat2deltax` *Convert an observation matrix to a vector of n-day displacements*

Description

Convert an observation matrix to a vector of n-day displacements. This is a function for the toad example.

Usage

```
obsMat2deltax(X, lag)
```

Arguments

`X`	The observation matrix to be converted.
`lag`	Interger, the number of day lags to compute the displacement.

Value

A vector of displacements.

`PENALTY-class` *S4 class “PENALTY”*

Description

This S4 class contains the penalty selection result from function `selectPenalty`. `show` display the penalty selection result. `plot` plot the penalty selection result using ggplot2.

Usage

```
## S4 method for signature 'PENALTY'
show(object)

## S4 method for signature 'PENALTY,ANY'
plot(x, logscale = TRUE)

## S4 method for signature 'BSL'
getPenalty(object)
```

Arguments

`object`	The S4 object of class “PENALTY” to show.
`x`	The S4 object of class “PENALTY” to plot.
`logscale`	A logical argument whether the x-axis (penalty) should be log transformed. The default is `TRUE`.

Slots

`loglike` A list of the log-likelihood values. The list contains multiple matrices (each corresponds to the result for a specific `n` value). The number of row of the matrix equals to the number of repeats `M`. The columns of the matrix stands for different penalty values.

`n` A vector of `n`, the number of simulations from the model per MCMC iteration for estimating the synthetic likelihood.

`lambda` A list, with each entry containing the vector of penalty values for the corresponding choice of `n`.

`M` The number of repeats used in estimating the standard deviation of the estimated log synthetic likelihood.

`sigma` The standard deviation of the log synthetic likelihood estimator to aim for, usually a value between 1 and 2. This reflects the mixing of a Markov chain.

`model` A “MODEL” object generated with function `newModel`. See `newModel`.

`stdLoglike` A list contains the estimated standard deviations of log-likelihoods.

`penalty` The vector stores the selected penalty values for each given `n` by choosing from the candidate `lambda` list. The selected values produce closest standard deviations `stdLoglike` to the target `sigma`.

`result` The result data frame.

`call` The original code used to run `selectPenalty`.

See Also

`selectPenalty` for the function that selects the penalty parameter.

`selectPenalty` *Selecting the Penalty Parameter*

Description

This is the main function for selecting the shrinkage (graphical lasso or Warton's estimator) penalty parameter for method BSL or semiBSL based on a point estimate of the parameters. Parallel computing is supported with the R package `foreach`. The penalty selection method is outlined in An et al. (2019).

Usage

```
selectPenalty(
  ssy,
  n,
  lambda,
  M,
  sigma = 1.5,
  model,
  theta = NULL,
  method = c("BSL", "semiBSL"),
  shrinkage = c("glasso", "Warton"),
  parallelSim = FALSE,
  parallelSimArgs = NULL,
  parallelMain = FALSE,
  verbose = 1L,
  ...
)
```

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 for estimating the synthetic likelihood.
`lambda`	A list, with each entry containing the vector of penalty values to test for the corresponding choice of `n`.
`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 estimator to aim for, usually a value between 1 and 2. This parameter helps to control the mixing of a Markov chain.
`model`	A "MODEL" object generated with function `newModel`. See `newModel`.

`theta`	A point estimate of the parameter value which all of the simulations will be based on. By default, if `theta` is `NULL`, it will be replaced by `theta0` from the given `model`.
`method`	A string argument indicating the method to be used. If the method is "BSL", the shrinkage is applied to the Gaussian covariance matrix. Otherwise if the method is "semiBSL", the shrinkage is applied to the correlation matrix of the Gaussian copula.
`shrinkage`	A string argument indicating which shrinkage method to be used. Current options are "glasso" for the graphical lasso method of Friedman et al. (2008) and "Warton" for the ridge regularisation method of Warton (2008).
`parallelSim`	A logical value indicating whether parallel computing should be used for simulation and summary statistic evaluation. Default is `FALSE`.
`parallelSimArgs`	A list of additional arguments to pass into the `foreach` function. Only used when `parallelSim` is `TRUE`, default is `NULL`.
`parallelMain`	A logical value indicating whether parallel computing should be used to computing the graphical lasso function. Notice that this should only be turned on when there are a lot of candidate values in `lambda`. Default is `FALSE`.
`verbose`	An integer indicating the verbose style. 0L means no verbose messages will be printed. 1L uses a custom progress bar to track the progress. 2L prints the iteration numbers (`1:M`) to track the progress. The default is 1L.
`...`	Other arguments to pass to `gaussianSynLike` ("BSL" method) or `semiparaKernelEstimate` ("semiBSL" method).

Value

An S4 object `PENALTY` of the penalty selection results. The `show` and `plot` methods are provided with the S4 class.

Author(s)

Ziwen An, Leah F. South and Christopher Drovandi

References

An Z, South LF, Nott DJ, Drovandi CC (2019). "Accelerating Bayesian Synthetic Likelihood With the Graphical Lasso." *Journal of Computational and Graphical Statistics*, **28**(2), 471–475. doi: 10.1080/10618600.2018.1537928.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441.

Warton DI (2008). "Penalized Normal Likelihood and Ridge Regularization of Correlation and Covariance Matrices." *Journal of the American Statistical Association*, **103**(481), 340–349. doi: 10.1198/016214508000000021.

See Also

`PENALTY` for the usage of the S4 class. `ma2`, `cell` and `mgnk` for examples. `bsl` for the main function to run BSL.

Examples

```
## Not run:
data(ma2)
model <- newModel(fnSimVec = ma2_sim_vec, fnSum = ma2_sum, simArgs = ma2$sim_args,
                  theta0 = ma2$start, fnLogPrior = ma2_prior)
theta <- c(0.6,0.2)

# Performing tuning for BSLasso (BSL with glasso shrinkage estimation)
ssy <- ma2_sum(ma2$data)
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)))
set.seed(100)
sp_ma2 <- selectPenalty(ssy = ssy, n = c(50, 150, 300, 500), lambda_all, theta = theta,
    M = 100, sigma = 1.5, model = model, method = 'BSL', shrinkage = 'glasso')
sp_ma2
plot(sp_ma2)

## End(Not run)
```

`semiparaKernelEstimate`

Estimate the semi-parametric synthetic (log) likelihood

Description

This function computes the semi-parametric synthetic likelihood estimator of (An et al. 2019). The advantage of this semi-parametric estimator over the standard synthetic likelihood estimator is that the semi-parametric one is more robust to non-normal summary statistics. Kernel density estimation is used for modelling each univariate marginal distribution, and the dependence structure between summaries are captured using a Gaussian copula. Shrinkage on the correlation matrix parameter of the Gaussian copula is helpful in decreasing the number of simulations.

Usage

```
semiparaKernelEstimate(
  ssy,
  ssx,
  kernel = "gaussian",
  shrinkage = NULL,
  penalty = NULL,
  log = TRUE
)
```

Arguments

`ssy`	The observed summary statisic.
`ssx`	A matrix of the simulated summary statistics. The number of rows is the same as the number of simulations per iteration.
`kernel`	A string argument indicating the smoothing kernel to pass into `density` for estimating the marginal distribution of each summary statistic. Only “gaussian" and “epanechnikov" are available. The default is “gaussian".
`shrinkage`	A string argument indicating which shrinkage method to be used. The default is `NULL`, which means no shrinkage is used. Current options are “glasso” for the graphical lasso method of Friedman et al. (2008) and “Warton” for the ridge regularisation method of Warton (2008).
`penalty`	The penalty value to be used for the specified shrinkage method. Must be between zero and one if the shrinkage method is “Warton”.
`log`	A logical argument indicating if the log of likelihood is given as the result. The default is `TRUE`.

Value

The estimated synthetic (log) likelihood value.

References

An Z, Nott DJ, Drovandi C (2019). “Robust Bayesian Synthetic Likelihood via a Semi-Parametric Approach.” *Statistics and Computing (In Press)*.

Friedman J, Hastie T, Tibshirani R (2008). “Sparse Inverse Covariance Estimation with the Graphical Lasso.” *Biostatistics*, **9**(3), 432–441.

Warton DI (2008). “Penalized Normal Likelihood and Ridge Regularization of Correlation and Covariance Matrices.” *Journal of the American Statistical Association*, **103**(481), 340–349. doi: 10.1198/016214508000000021.

Friedman J, Hastie T, Tibshirani R (2008). “Sparse Inverse Covariance Estimation with the Graphical Lasso.” *Biostatistics*, **9**(3), 432–441.

Warton DI (2008). “Penalized Normal Likelihood and Ridge Regularization of Correlation and Covariance Matrices.” *Journal of the American Statistical Association*, **103**(481), 340–349. doi: 10.1198/016214508000000021.

Boudt K, Cornelissen J, Croux C (2012). “The Gaussian Rank Correlation Estimator: Robustness Properties.” *Statistics and Computing*, **22**(2), 471–483. doi: 10.1007/s1122201192370.

See Also

Other available synthetic likelihood estimators: `gaussianSynLike` for the standard synthetic likelihood estimator, `gaussianSynLikeGhuryeOlkin` for the unbiased synthetic likelihood estimator, `synLikeMisspec` for the Gaussian synthetic likelihood estimator for model misspecification.

Examples

```
data(ma2)
ssy <- ma2_sum(ma2$data)
m <- newModel(fnSim = ma2_sim, fnSum = ma2_sum, simArgs = ma2$sim_args,
              theta0 = ma2$start, sumArgs = list(delta = 0.5))
ssx <- simulation(m, n = 300, theta = c(0.6, 0.2), seed = 10)$ssx

# check the distribution of the first summary statistic: highly non-normal
plot(density(ssx[, 1]))

# the standard synthetic likelihood estimator over-estimates the likelihood here
gaussianSynLike(ssy, ssx)
# the semi-parametric synthetic likelihood estimator is more robust to non-normality
semiparaKernelEstimate(ssy, ssx)
# using shrinkage on the correlation matrix of the Gaussian copula is also possible
semiparaKernelEstimate(ssy, ssx, shrinkage = "Warton", penalty = 0.8)
```

`simulate_cell` *Simulation function of the cell biology example*

Description

Simulation function of the cell biology example.

Usage

```
simulate_cell(x, rows, cols, Pm, Pp, sim_iters, num_obs)
```

Arguments

`x`	The initial matrix of cell presences of size `rows` × `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).
`Pm`	Parameter P_m, the probability of cell movement.
`Pp`	Parameter P_p, the probability of cell proliferation.
`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.

Value

A `rows` × `cols` × `num_obs` array of the cell presences at times `1:num_obs` (not time 0).

`simulation` *Run simulations with a give "MODEL" object*

Description

see `MODEL`

Usage

```
simulation(model, ...)
```

Arguments

`model`	A "MODEL" object.
`...`	Other arguments.

`sim_toad` *The simulation function for the toad example*

Description

The simulation function for the toad example.

Usage

```
sim_toad(params, ntoad, nday, model = 1L, d0 = 100)
```

Arguments

`params`	A vector of proposed model parameters, α, $gamma$ and p_0.
`ntoad`	The number of toads to simulate in the observation.
`nday`	The number of days lasted of the observation.
`model`	Which model to be used. 1 for the random return model, 2 for the nearest return model, and 3 for the distance-based return probability model.
`d0`	Characteristic distance for model 3. Only used if model is 3.

Value

A data matrix.

Examples

```
sim_toad(c(1.7,36,0.6), 10, 8, 1)
```

`summStat` *Compute the summary statistics with the given data*

Description

see `MODEL`

Usage

```
summStat(x, model)
```

Arguments

`x`	The data to pass to the summary statistics function.
`model`	A “MODEL” object.

`synLikeMisspec` *Estimate the Gaussian synthetic (log) likelihood whilst acknowledging model incompatibility*

Description

This function estimates the Gaussian synthetic likelihood whilst acknowledging that there may be incompatibility between the model and the observed summary statistic. The method has two different ways to account for and detect incompatibility (mean adjustment and variance inflation). An additional free parameter `gamma` is employed to account for the model misspecification. See the R-BSL methods of Frazier and Drovandi (2021) for more details. Note this function is mainly designed for interal use as the latent variable `gamma` need to be chosen otherwise. Alternatively, `gamma` is updated with a slice sampler (Neal 2003), which is the method of Frazier and Drovandi (2021).

Usage

```
synLikeMisspec(
  ssy,
  ssx,
  type = c("mean", "variance"),
  gamma = numeric(length(ssy)),
  log = TRUE,
  verbose = FALSE
)
```

Arguments

`ssy`	The observed summary statisic.
`ssx`	A matrix of the simulated summary statistics. The number of rows is the same as the number of simulations per iteration.
`type`	A string argument indicating which method is used to account for and detect potential incompatibility. The two options are "mean" and "variance" for mean adjustment and variance inflation, respectively.
`gamma`	The additional latent parameter to account for possible incompatability between the model and observed summary statistic. In "BSLmisspec" method, this is updated with a slice sampler (Neal 2003). The default gamma implies no model misspecification and is equivalent to the standard `gaussianSynLike` estimator.
`log`	A logical argument indicating if the log of likelihood is given as the result. The default is `TRUE`.
`verbose`	A logical argument indicating whether an error message should be printed if the function fails to compute a likelihood. The default is `FALSE`.

Value

The estimated synthetic (log) likelihood value.

References

Frazier DT, Drovandi C (2021). "Robust Approximate Bayesian Inference with Synthetic Likelihood." *Journal of Computational and Graphical Statistics (In Press)*. `https://arxiv.org/abs/1904.04551`.

Neal RM (2003). "Slice sampling." *The Annals of Statistics*, **31**(3), 705–767.

See Also

Other available synthetic likelihood estimators: `gaussianSynLike` for the standard synthetic likelihood estimator, `gaussianSynLikeGhuryeOlkin` for the unbiased synthetic likelihood estimator, `semiparaKernelEstimate` for the semi-parametric likelihood estimator, `synLikeMisspec` for the Gaussian synthetic likelihood estimator for model misspecification. Slice sampler to sample gamma `sliceGammaMean` and `sliceGammaVariance` (internal functions).

Examples

```
# a toy model (for details see section 4.1 from Frazier et al 2019)
# the true underlying model is a normal distribution with standard deviation equals to 0.2
# whist the data generation process has the standard deviation fixed to 1
set.seed(1)
y <- rnorm(50, 1, sd = 0.2)
ssy <- c(mean(y), var(y))
m <- newModel(fnSim = function(theta) rnorm(50, theta), fnSum = function(x) c(mean(x), var(x)),
              theta0 = 1, fnLogPrior = function(x) log(dnorm(x, sd = sqrt(10))))
ssx <- simulation(m, n = 300, theta = 1, seed = 10)$ssx
```

```
# gamma is updated with a slice sampler
gamma <- rep(0.1, length(ssy))
synLikeMisspec(ssy, ssx, type = "variance", gamma = gamma)
```

`toad` *Toad example*

Description

This example estimates the parameter for the toad example. The model simulates the movement of an amphibian called Fowler's toad. The model is proposed by Marchand et al. (2017). This example includes both simulated and real data. The real data is obtained from the supplementary material of Marchand et al. (2017). The journal article An et al. (2022) provides a full description of how to use this package for the toad example.

Usage

```
data(toad)

toad_sim(
  theta,
  ntoads,
  ndays,
  model = 1,
  d0 = 100,
  na = matrix(FALSE, ndays, ntoads)
)

toad_sum(X, lag = c(1, 2, 4, 8), p = seq(0, 1, 0.1))

toad_prior(theta)
```

Arguments

`theta`	A vector of proposed model parameters, α, γ and p_0.
`ntoads`	The number of toads to simulate in the observation.
`ndays`	The number of days observed.
`model`	Which model to be used: 1 for the random return model, 2 for the nearest return model, and 3 for the distance-based return probability model. The default is 1.
`d0`	Characteristic distance for model 3. Only used if `model` is 3.
`na`	Logical. This is the index matrix for missing observations. By default, `matrix(FALSE, ndays, ntoads)` indicates there is no missingness in the observation matrix.
`X`	The data matrix.
`lag`	The lag of days to compute the summary statistics, default as 1, 2, 4 and 8.
`p`	The numeric vector of probabilities to compute the quantiles.

Details

The example includes the three different returning models of Marchand et al. (2017). Please see Marchand et al. (2017) for a full description of the toad model, and also An et al. (2019) for Bayesian inference with the semi-BSL method.

Functions

- `toad_sim`: Simulates data from the model, using C++ in the backend.
- `toad_sum`: Computes the summary statistics for this example. The summary statistics are the log differences between adjacent quantiles and also the median.
- `toad_prior`: Evaluates the log prior at the chosen parameters.

datasets (simulated and real)

A simulated dataset and a real dataset are provided in this example. Both datasets contain observations from 66 toads for 63 days. The simulated dataset is simulated with parameter $\theta = (1.7, 35, 0.6)$. This is the data used in An et al. (2019). The real dataset is obtained from the supplementary data of Marchand et al. (2017).

- `data_simulated`: A 63 × 66 matrix of the observed toad locations (simulated data).
- `data_real`: A 63 × 66 matrix of the observed toad locations (real data).
- `cov`: The covariance matrix of a multivariate normal random walk proposal distribution used in the MCMC, in the form of a 3 × 3 matrix.
- `theta0`: A vector of suitable initial values of the parameters for MCMC.
- `sim_args_simulated` and `sim_args_real`: A list of the arguments to pass into the simulation function.
  - `ndays`: The number of days observed.
  - `ntoads`: The total number of toads being observed.
  - `model`: Indicator of which model to be used.
  - `na`: Indicator matrix for missingness.

Author(s)

Ziwen An, Leah F. South and Christopher Drovandi

References

An Z, Nott DJ, Drovandi C (2019). “Robust Bayesian Synthetic Likelihood via a Semi-Parametric Approach.” *Statistics and Computing (In Press)*.

An Z, South LF, Drovandi CC (2022). “BSL: An R Package for Efficient Parameter Estimation for Simulation-Based Models via Bayesian Synthetic Likelihood.” *Journal of Statistical Software*, **101**(11), 1–33. doi: 10.18637/jss.v101.i11.

Marchand P, Boenke M, Green DM (2017). “A stochastic movement model reproduces patterns of site fidelity and long-distance dispersal in a population of Fowlers toads (Anaxyrus fowleri).” *Ecological Modelling*, **360**, 63–69. ISSN 0304-3800, doi: 10.1016/j.ecolmodel.2017.06.025.()

Examples

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

data(toad)

## run standard BSL for the simulated dataset
model1 <- newModel(fnSim = toad_sim, fnSum = toad_sum, theta0 = toad$theta0,
                   fnLogPrior = toad_prior, simArgs = toad$sim_args_simulated,
                   thetaNames = expression(alpha,gamma,p[0]))
paraBound <- matrix(c(1,2,0,100,0,0.9), 3, 2, byrow = TRUE)

# Performing BSL (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultToadSimulated <- bsl(toad$data_simulated, n = 1000, M = 10000, model = model1,
                           covRandWalk = toad$cov, logitTransformBound = paraBound,
                           parallel = TRUE, verbose = 1L, plotOnTheFly = 100)
stopCluster(cl)
registerDoSEQ()
show(resultToadSimulated)
summary(resultToadSimulated)
plot(resultToadSimulated, thetaTrue = toad$theta0, thin = 20)

## run standard BSL for the real dataset
model2 <- newModel(fnSim = toad_sim, fnSum = toad_sum, theta0 = toad$theta0,
                   fnLogPrior = toad_prior, simArgs = toad$sim_args_real,
                   thetaNames = expression(alpha,gamma,p[0]))
paraBound <- matrix(c(1,2,0,100,0,0.9), 3, 2, byrow = TRUE)

# Performing BSL (reduce the number of iterations M if desired)
# Opening up the parallel pools using doParallel
cl <- makeCluster(min(detectCores() - 1,2))
registerDoParallel(cl)
resultToadReal <- bsl(toad$data_real, n = 1000, M = 10000, model = model2,
                      covRandWalk = toad$cov, logitTransformBound = paraBound,
                      parallel = TRUE, verbose = 1L, plotOnTheFly = 100)
stopCluster(cl)
registerDoSEQ()
show(resultToadReal)
summary(resultToadReal)
plot(resultToadReal, thetaTrue = toad$theta0, thin = 20)

## End(Not run)
```

Index