Package ‘sbim’

March 13, 2025

Title Simulation-Based Inference using a Metamodel for Log-Likelihood Estimator

Version 1.0.0

Description Parameter inference methods for models defined implicitly using a random simulator. Inference is carried out using simulation-based estimates of the log-likelihood of the data. The inference methods implemented in this package are explained in Park, J. (2025) <doi:10.48550/arxiv.2311.09446>. These methods are built on a simulation metamodel which assumes that the estimates of the log-likelihood are approximately normally distributed with the mean function that is locally quadratic around its maximum. Parameter estimation and uncertainty quantification can be carried out using the ht() function (for hypothesis testing) and the ci() function (for constructing a confidence interval for one-dimensional parameters).

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

LinkingTo Rcpp

Imports Rcpp, stats

Suggests devtools, dplyr, ggplot2, knitr, magrittr, pomp, rmarkdown, tidyr

VignetteBuilder knitr

Depends R (>= 3.5)

NeedsCompilation yes

Author Joonha Park [aut, cre] (<`https://orcid.org/0000-0002-4493-7730`>)

Maintainer Joonha Park <`j.park@ku.edu`>

Repository CRAN

Date/Publication 2025-03-13 12:50:02 UTC

Contents

`ci` *Confidence interval for scalar parameter constructed using simulated log likelihoods*

Description

`ci` constructs confidence intervals for a scalar (one-dimensional) parameter using simulated log likelihoods. See Park (2025) for more information.

Usage

```
## S3 method for class 'simll'
ci(
  simll,
  level,
  ci = NULL,
  case = NULL,
  weights = NULL,
  autoAdjust = FALSE,
  K1_est_method = "batch",
  batch_size = NULL,
  max_lag = NULL,
  plot_acf = FALSE,
  ...
)
```

Arguments

`simll`	A class 'simll' object, containing simulated log likelihoods, the parameter values at which simulations are made, and the weights for those simulations for regression (optional). See help(simll).
`level`	A numeric vector of confidence levels.
`ci`	A character string indicating the quantity for which a confidence interval is to be constructed. Either "MESLE" or "parameter". See Details.
`case`	When `ci` is "parameter", `case` is either "iid" or "stationary" (default). `case` = "iid" means that the observations are iid, and `case` = "stationary" means that the observations form a stationary sequence. The `case` argument affects how the variance of the slope of the mean function (=K_1 in Park (2025)) is estimated.
`weights`	An optional argument. The un-normalized weights of the simulated log likelihoods for regression. A numeric vector of length equal to the 'params' attribute of the 'simll' object. See Details below.

`autoAdjust`	logical. If TRUE, simulation points at which the third order term is statistically significant in the cubic approximation to the simulated log-likelihooods have discounted weights for metamodel fitting. The weights of the points relatively far from the estimated MESLE are more heavily discounted. These weight discount factors are multiplied to the originally given weights for parameter estimation. If `autoAdjust` is FALSE, the weight discount step is skipped. Defaults to FALSE. See ?optDesign and Park (2025) for more details.
`K1_est_method`	Either "autocov" or "batch"
`batch_size`	Numeric
`max_lag`	When `test` is "parameter" and `case` is "stationary", the value of `max_lag` gives the truncation point for lagged autocovariance when estimating K1 as a sum of lagged autocovariances of estimates slopes. If not supplied, default is the maximum lag for which the lagged autocorrelation has absolute value greater than 4/sqrt(nobs), where the lagged autocorrelation is found up to lag `10*log10(nobs)`. Here `nobs` is the number of observations.
`plot_acf`	Logical. When `test` is "parameter" and `case` is "stationary", If `plot_acf` is TRUE, the autocorrelation plot of the estimated slopes of the quadratic fit to the simulated log likelihoods is shown.
`...`	Other optional arguments, not currently used.

Details

This is a generic function, taking a class 'simll' object as the first argument. Confidence intervals are constructed under a normal, locally quadratic meta model where the simulated log likelihoods given in the 'simll' object are normally distributed.

When 'level' has length greater than one, a confidence interval is constructed for each value in the vector.

Quadratic regression for the simulated log likelihoods is carried out to construct confidence intervals, where the x-axis values are the 'params' values of the 'simll' object and the y-axis values are the corresponding simulated log likelihoods. In the case where 'ci' = "parameter", inference on the simulation based surrogate will be carried out under the local asymptotic normality for simulated log likelihoods (see Park (2025) for more information.) The default value of 'ci' is "parameter".

If 'ci' = "MESLE", confidence intervals are constructed for the maximum expected simulation likelihood estimate given the observed data.

When quadratic regression is carried out, the weights for the simulation based likelihood estimates can be specified. The length of 'weights' should be equal to that of the 'params' attribute of the 'simll', which is equal to the number of rows in the simulated log likelihood matrix in the 'simll' object. It is important to note that the weights are not normalized (i.e., not sum to one). Multiplying all weights by the same constant changes the estimation outputs. If not supplied, the 'weights' attribute of the 'simll' object is used. If neither is supplied, 'weights' defaults to the vector of all ones.

Value

A list consisting of the followings are returned.

- regression_estimates: point estimates for the meta model parameters, a, b, c, and sigma^2.

- meta_model_MLE_for_*: point estimate for the quantity for which confidence intervals are constructed under a normal meta model
- confidence_interval: a data frame of the lower and upper bounds of the confidence intervals and the corresponding confidence levels. Note that in some unfortunate cases (especially if the quadratic coefficient of the estimated quadratic fit of the log likelihood estimates is close to zero or nonnegative), the confidence interval may be inverted, meaning that it is of the form (-infty, bound1) U (bound2, infty). This case can happen if the signal-to-noise ratio in simulated log likelihoods is too small. The inverted confidence interval will be indicated by the additional column "inverted" in the data frame taking values of 0 or 1.
- max_lag: if `test="parameter"` and `case="stationary"`, the maximum lag for computing the autocovariance in estimating K1 is shown.
- pval_cubic: The p-value of the test about whether the cubic term in the cubic polynomial regression is significant. If `pval_cubic` is small, the constructed confidence interval may be biased. When `autoAdjust` is TRUE, `pval_cubic` is computed with the adjusted weights.
- updated_weights: When `autoAdjust` is TRUE, the modified weights for the simulation points are returned.

References

Park, J. (2025). Scalable simulation based inference for implicitly defined models using a meta-model for Monte Carlo log-likelihood estimator doi:10.48550/arxiv.2311.09446

Examples

```
# State process: X_i ~ N(theta0, tau^2), Observation process: Y_i ~ N(X_i, 1)
theta0 <- 0 # true parameter
n <- 200 # number of observations
xhidden <- rnorm(n, theta0, 30) # hidden x values
ydata <- rnorm(n, xhidden, 1) # observed y values
theta_sim <- runif(300, -10, 10) # simulation points
ll <- sapply(theta_sim, function(t) { # simulation-based log-likelihood estimates (except constant)
x <- rnorm(n, t, 30)
-(x-ydata)^2/2
})
plot(theta_sim, apply(ll, 2, sum)) # display the log-likelihood estimates
s <- simll(ll, params=theta_sim) # create a `simll` object
ci(s, level=0.95, ci="parameter", case="iid")
```

`ht` *Hypothesis tests based on simulation based log likelihood estimates*

Description

`ht` carries out hypothesis tests for models defined implicitly by a random simulator. It takes as input estimates of the log likelihood obtained via simulations of the model. Tests are carried out using a simulation meta model. See Park (2025) for more details on the method.

Usage

```
## S3 method for class 'simll'
ht(
  simll,
  null.value,
  test = c("parameter", "MESLE", "moments"),
  case = NULL,
  type = NULL,
  weights = NULL,
  autoAdjust = FALSE,
  K1_est_method = "batch",
  batch_size = NULL,
  max_lag = NULL,
  plot_acf = FALSE,
  MCcorrection = "none",
  ...
)
```

Arguments

`simll`	A class `simll` object, containing simulated log likelihoods, the parameter values at which simulations are made (may be omitted if all simulations are made at the same parameter value), and the weights for those simulations for regression (optional). See help(simll).
`null.value`	The null value(s) for the hypothesis test. The expected format depends on which teset will be carried out. See the Details section for more information.
`test`	A character string indicating which is to be tested about. One of "moments", "MESLE", or "parameter". See Details.
`case`	When `test` is "parameter", `case` needs to be either "iid" or "stationary". `case` = "iid" means that the observations are iid, and `case` = "stationary" means that the observations form a stationary sequence. The `case` argument affects how the variance of the slope of the mean function (=K_1 in Park (2025)) is estimated. The default value is "stationary".
`type`	When `test` is "moments", the `type` argument needs to be specified. `type` = "point" means that the test about the mean and the variance of simulated log likelihoods at a given parameter point is considered. `type` = "regression" means that the test about the mean function and the variance of simulated log likelihoods at various parameter values is considered. See Details.
`weights`	An optional argument. The un-normalized weights of the simulated log likelihoods for regression. A numeric vector of length equal to the `params` attribute of the `simll` object. See Details below.
`autoAdjust`	logical. If TRUE, simulation points at which the third order term is statistically significant in the cubic approximation to the simulated log-likelihooods have discounted weights for metamodel fitting. The weights of the points relatively far from the estimated MESLE are more heavily discounted. These weight discount factors are multiplied to the originally given weights for parameter esti-

	mation. If `autoAdjust` is FALSE, the weight discount step is skipped. Defaults to FALSE. See ?optDesign and Park (2025) for more details.
`K1_est_method`	Either "batch" or "autocov". Used when `test` is "parameter" and `case` is "stationary". The default is "batch". See Details for more information.
`batch_size`	Numeric. The size of the batch when `K1_est_method` is "batch". If not supplied, the default value is `round(n^0.4)` where `n` is the number of observations in the data.
`max_lag`	When `test` is "parameter" and `case` is "stationary", the value of `max_lag` gives the truncation point for lagged autocovariance when estimating K1 as a sum of lagged autocovariances of estimates slopes. If not supplied, default is the maximum lag for which at least one of the entries of the matrix of lagged autocorrelation has absolute value greater than 4/sqrt(nobs), where the lagged autocorrelation is found up to lag `10*log10(nobs/d)`. Here `nobs` is the number of observations and `d` is the dimension of the parameter space.
`plot_acf`	Logical. Should the autocorrelation plot be generated when estimating K1 for the case where `test` is "parameter" and `case` is "stationary"?
`MCcorrection`	For tests on the simulation based parameter surrogate (`test=`"parameter"), `MCcorrection` determines if and how the sampling distribution of the test statistic will be corrected by a Monte Carlo method to account for the variability in the estimate of K1. Possible values are "none" (default) and "Wishart". See the Details section and Park (2025) for more details.
`...`	Other optional arguments, not currently used.

Details

This is a generic function, taking a class `simll` object as the first argument. Hypothesis tests are carried out under a normal metamodel–that is, the simulated log likelihoods (whose values are given in the `simll` object) are normally distributed.

If `test` = "moments", the `type` argument needs to be either "point" or "regression". If `type` = "point", a test about the mean and the variance of the simulated log likelihood at a single parameter value is conducted. If `type` = "regression", the `simll` object should contain simulated log likelihoods obtained at more than one parameter values, specified by the `params` attribute of the `simll` object. A (weighted) quadratic regression for the simulated log likelihoods will be used for hypothesis tests, where the x-axis values are given by the `params` values of the `simll` object and the y-axis values are the corresponding simulated log likelihoods. The test is about the quadruple $a, b, c, sigma^2$ where a, b, c are coefficients of the polynomial describing the mean of the simulated log likelihood (i.e., $l(\theta) = a + b\theta + c\theta^2$) and σ^2 is the variance of the simulated log likelihood. If `test` = "moments" and `type` is not specified, `type` defaults to "point" if the `params` attribute of the `simll` object is not supplied or has length one, and defaults to "regression" otherwise.

When `test` = "MESLE" or "parameter", the `simll` object should have the `params` attribute.

If `test` = "MESLE", the test is about the location of the maximum expected simulated log likelihood estimate.

If `test` = "parameter", inference on the simulation based surrogate will be carried out under the local asymptotic normality for simulated log likelihood (see Park (2025) for more information.)

The default value for `test` is "parameter".

When quadratic regression is carried out, the weights for the simulation based likelihood estimates can be specified. The length of `weights` should be equal to that of the `params` attribute of the `simll`, which is equal to the number of rows in the simulated log likelihood matrix in the `simll` object. It is important to note that the weights are not supposed to be normalized (i.e., sum to one). Multiplying all weights by the same constant changes the estimation outputs. If not supplied, the `weights` attribute of the `simll` object is used. If neither is supplied, `weights` defaults to the vector of all ones.

When `test` is "moments" and `type` is "point", `null.value` is either a vector of length two (one entry for the mean and the other for the variance of the simulated log likelihoods), a matrix of two columns (one for the mean and the other for the variance), or a list of vectors of length two (each entry of the list gives a null value consisting of the mean and the variance.) When `test` is "moments" and `type` is "regression", `null.value` can be a list of length four, or a list of lists of length four. The first case corresponds to when a single null hypothesis is tested. The four components are a) the constant term in the quadratic mean function (scalar), b) the linear coefficient term in the mean function (vector of length d where d is the dimension of the parameter vector), c) the quadratic coefficient term in the mean function (symmetric matrix of dimension $d \times d$), and d) the variance of the simulated log likelihood (scalar). The second case is when more than one null values are tested. In this case each component of the list is a list having four entries as described for the case of a single null value. When `test` is "MESLE" or "parameter", `null.value` is a vector of length d (a single null value), a matrix having d columns (each row giving a vector for a null value), or a list of vectors of length d (more than one null values).

Value

A list consisting of the following components are returned.

- regression_estimates: point estimates for the meta model parameters, a, b, c, and sigma^2. Given only when test="MESLE" or "parameter".
- meta_model_MLE_for_*: point estimate for the tested quantity under a normal meta model
- Hypothesis_Tests: a data frame of the null values and the corresponding p-values. When `test=`"moments" and `type=`"regression", each null value is given in the form of c(a,b,c,sigma^2) where a, b, c, sigma^2 are first, second, third, and fourth entries of the given null value.
- pvalue_numerical_error_size: When `test=`"moments", approximate size of error in numerical evaluation of p-values (automatically set to approximately 0.01 or 0.001). For these case, p-values are found using the SCL distributions, whose cumulative distribution functions are numerically evaluated using random number generations. Thus p-values have some stochastic error. The size of the numerical error is automatically set to approximately 0.01, but if any of the p-values found is less than 0.01, more computations are carried out to reduce the numerical error size to approximately 0.001. Note that when `test=`"MESLE" or "parameter", the (standard) F distribution is used, so this list component is omitted.
- max_lag: if `test=`"parameter" and `case=`"stationary", the maximum lag for computing the autocovariance in estimating K1 is shown.
- pval_cubic: The p-value of the test about whether the cubic term in the cubic polynomial regression is significant. If `pval_cubic` is small, the result of the `ht` function may be biased. The test on the cubic term is carried out only when the number of simulated log likelihoods is greater than $(d+1)*(d+2)*(d+3)/6$ where d is the dimension of the parameter vector. When `autoAdjust` is TRUE, `pval_cubic` is computed with the adjusted weights.

- updated_weights: When `autoAdjust` is TRUE, the modified weights for the simulation points are returned.

References

Park, J. (2025). Scalable simulation-based inference for implicitly defined models using a meta-model for Monte Carlo log-likelihood estimator doi:10.48550/arxiv.2311.09446

Examples

```
# State process: X_i ~ N(theta0, tau^2), Observation process: Y_i ~ N(X_i, 1)
theta0 <- 0 # true parameter
n <- 200 # number of observations
xhidden <- rnorm(n, theta0, 30) # hidden x values
ydata <- rnorm(n, xhidden, 1) # observed y values
theta_sim <- runif(300, -10, 10) # simulation points
ll <- sapply(theta_sim, function(t) { # simulation-based log-likelihood estimates (except constant)
x <- rnorm(n, t, 30)
-(x-ydata)^2/2
})
plot(theta_sim, apply(ll, 2, sum)) # display the log-likelihood estimates
s <- simll(ll, params=theta_sim) # create a `simll` object
ht(s, null.value=list(-1,0,1), test="parameter", case="iid")
```

`optDesign` *Find the next optimal design point for simulation-based inference*

Description

`optDesign` finds the next design point at which simulation should be carried out for approximately best efficiency in a metamodel-based inference. See Park (2025) for more details on this method. It takes a class `simll` object.

Usage

```
## S3 method for class 'simll'
optDesign(
  simll,
  init = NULL,
  weight = 1,
  autoAdjust = TRUE,
  refgap = Inf,
  refgap_for_comp = NULL,
  ...
)
```

Arguments

`simll`	A class `simll` object, containing simulation log likelihoods, the parameter values at which simulations are made, and the weights for those simulations for regression (optional). See help(simll).
`init`	(optional) An initial parameter vector at which a search for optimal point starts.
`weight`	(optional) A positive real number indicating the user-assigned weight for the new design point. The default value is 1. This value should be chosen relative to the weights in the provided simll object.
`autoAdjust`	logical. If TRUE, simulation points at which the third order term is statistically significant in the cubic approximation to the simulated log-likelihooods have discounted weights for metamodel fitting. The weights of the points relatively far from the estimated MESLE are more heavily discounted. These weight discount factors are multiplied to the originally given weights for parameter estimation. See Park (2025) for more details. If `autoAdjust` is FALSE, the weight discount step is skipped. Defaults to TRUE.
`refgap`	A positive real number that determines the weight discount factor for the significance of the third order term in Taylor approximation. The weight of a point `theta` is discounted by a factor of exp(-(qa(theta)-qa(MESLEhat))/refgap), where MESLEhat is the estimated MESLE and qa is the quadratic approximation to the simulated log-likelihoods. If `autoAdjust` is TRUE, `refgap` is interpreted as the initial value for the tuning algorithm. If `autoAdjust` is FALSE, `refgap` is used for weight adjustments without further tuning. The default value is Inf.
`refgap_for_comp`	(optional) A value of refgap with which to compute the log(STV) to be reported at the end. A potential use for this argument is to compare log(STV) values across iterative applications of this function, as the reported logSTV value can vary significantly depending on the tuned value of refgap.
`...`	Other optional arguments, not currently used.

Details

This is a generic function, taking a class `simll` object as the first argument. Parameter inference for implicitly defined simulation models can be carried out under a metamodel for the distribution of the log-likelihood estimator. See function `ht` for hypothesis testing and `ci` for confidence interval construction for a one-dimensional parameter. This function, `optDesign`, proposes the next point at which a simulation is to be carried out such that the variance of the parameter estimate is reduced approximately the most. In order to balance efficiency and accuracy, the point is selected as far as possible from the current estimate of the parameter while ensuring that the quadratic approximation to the simulated log-likelihoods remain valid. Specifically, the weights for the existing simulation points are adjusted such that the third order term in a cubic approximation is statistically insignificant. The weight discount factor for point `theta` is given by exp(-(qa(theta)-qa(MESLEhat))/g), where qa is the quadratic approximation, MESLEhat is the estimated MLE, and g is a scaling parameter. These discount factors are multipled to the original `weights` given to the simulation points specified in the `simll` object. Moreover, in order to ensure that the cubic regression can be carried out without numerical issues, g is guaranteed not to fall below a value that makes the effective sample size (ESS) below (d+1)*(d+2)*(d+3)/6, which is the total number of parameter estimated in cubic regression, where d is the parameter dimension. Here ESS is calculated as (sum of adjusted weights)^2/(sum of squared adjusted weights).

The next simulation point is selected by approximately minimizing the scaled total Monte Carlo variation of the parameter estimate. The scaled total variation (STV) is defined as the trace of `c_hat^{-1} V` where `c_hat` is the quadratic coefficient matrix of the fitted quadratic polynomial and `V` is an approximate Monte Carlo variance of the estimate of the MESLE given by `-(1/2) * c_hat^{-1} b_hat` (here `b_hat` is the linear coefficient vector of the fitted quadratic polynomial.) The optimization is carried out using the BFGS algorithm via the `optim` function. See Park (2025) for more details.

Value

A list containing the following entries.

- par: a proposal for the next simulation point.
- logSTV: the logarithm of the approximate scaled total variation (STV) evaluated at the proposed simulation point.
- wadj_new: the adjusted weight for the newly proposed simulation point.
- Wadj: the vector of all adjusted weights for the existing simulation points.
- refgap: the tuned value of g for weight adjustments.
- logSTV_for_comp: when `refgap_for_comp` is not NULL, log(STV) is evaluated using the provided value of `refgap_for_comp` and reported as `logSTV_for_comp`.

References

Park, J. (2025). Scalable simulation-based inference for implicitly defined models using a metamodel for Monte Carlo log-likelihood estimator doi:10.48550/arxiv.2311.09446

`SCL` *The SCL distribution*

Description

Quantile function, distribution function, and random generation for the SCL distribution family. See Park (2025) for information about the SCL distributions.

Usage

```
qscl(
  p,
  M,
  k,
  num_error_size = 0.01,
  lower = TRUE,
  log_p = FALSE,
  force = FALSE
)
```

```
pscl(
  q,
  M,
  k,
  num_error_size = 0.01,
  lower = TRUE,
  log_p = FALSE,
  force = FALSE
)

rscl(n, M, k)
```

Arguments

`p`	vector of probabilities
`M`	the first parameter for the SCL distributions
`k`	the second parameter for the SCL distribution
`num_error_size`	The requested size of numerical error for the outputs of qscl and pscl functions, in terms of the estimated standard deviation of the output. For example num_error_size of 0.01 will output values with the standard deviation of approximately equal to 0.01.
`lower`	logical; if TRUE, probabilities are P(X <= x), otherwise, P(X > x).
`log_p`	logical; if TRUE, probabilities p are given as log(p).
`force`	logical; if TRUE, the function will run regardless of how long it will take. If FALSE, the function will ask if you want to continue, stop, or give a new num_error_size value whenever the expected run time is longer than 15 seconds.
`q`	vector of quantiles
`n`	number of draws

Value

a list consisting of the numeric vector of quantiles and the num_error_size (numeric) used.

`simll` *Simulation Log Likelihood class*

Description

Simulation Log Likelihood class

Usage

```
simll(ll, params = NULL, weights = NULL)
```

Arguments

`ll`	A matrix of simulation log likelihoods. The (i,m)-th entry is given by the simulation log likelihood for y_i obtained by simulating X at theta_m (e.g., the log density of y_i given X).
`params`	A matrix or a vector of parameter values. If a matrix, the m-th row gives the parameter vector theta_m. If theta is one dimensional, 'params' is can be a numeric vector or a matrix with one column. 'params' can be omitted if simulation log likelihoods are obtained at a single one parameter value.
`weights`	A numeric vector of weights, inversely proportional to the variance of simulation log likelihoods (optional)

Value

A class 'sll' object

Index