Package 'jaatha'

January 8, 2023

Version 3.2.3

License GPL (>= 3)

Title Simulation-Based Maximum Likelihood Parameter Estimation

Author Paul Staab [aut],
Lisha Mathew [aut],
Dirk Metzler [aut, ths, cre]

Description An estimation method that can use computer simulations to approximate maximum-likelihood estimates even when the likelihood function can not be evaluated directly. It can be applied whenever it is feasible to conduct many simulations, but works best when the data is approximately Poisson distributed. It was originally designed for demographic inference in evolutionary biology (Naduvilezhath et al., 2011 <doi:10.1111/j.1365-294X.2011.05131.x>, Mathew et al., 2013 <doi:10.1002/ece3.722>). It has optional support for conducting coalescent simulation using the 'coala' package.

URL https://github.com/statgenlmu/jaatha

BugReports https://github.com/statgenlmu/jaatha/issues

Depends R (>= 3.0)

Imports assertthat (>= 0.1), R6 (>= 2.1.1), parallel, stats, utils

Suggests boot (>= 1.3-10), coala (>= 0.2.1), knitr, rmarkdown, testthat (>= 0.11.0), snow

VignetteBuilder knitr

RoxygenNote 7.2.2

Maintainer Dirk Metzler <metzler@bio.lmu.de>

NeedsCompilation no

Repository CRAN

Date/Publication 2023-01-08 16:30:02 UTC

R topics documented:

`boot_jaatha` *Parametric Bootstrapping of Jaatha Estimates*

Description

This function is a helper function for using the `boot` function to bootstrap Jaatha estimates. Each bootstap replication requires a complete jaatha estimation on data simulated with the original parameter estimates. Therefore, bootstrapping is normally computationally demanding and should be executed on a computing cluster.

Usage

```
boot_jaatha(results, R, cores_per_run = 1, verbose = TRUE, ...)
```

Arguments

`results`	The results of an `jaatha` analysis.
`R`	The number of bootstrapping replicates that are performed.
`cores_per_run`	The number of cores that are used for each replicate. This corresponds to the `cores` option of `jaatha`. Different replicates can be executed in parallel using the `parallel`, `ncpus` and `cl` options of `boot`. The total number of CPU cores used is `ncpus` * `cores_per_run`.
`verbose`	If TRUE (default), each bootstrap estimation is written as a message.
`...`	Additional arguments that are passed on `boot`. It is highly recommended to use its `parallel` and `ncpus` options to parallelize the bootstrap replicates.

Value

The result of `boot`. This object can be used to estimate standard errors or confidence intervals of the estimates using the functions available in package **boot**. Note that the returned object contains a vector of parameter values `t0` that is the result of an additional jaatha run for the original data, whereas the parametric bootstrap simulations used parameter values that are in the vector `mle` in the returned `boot` object. By default, the function `boot.ci` of the `boot` package uses the parameter values `t0` as a reference point. To use the values in `mle` instead, overwrite `t0` with `mle` before applying the function `boot.ci`.

See Also

boot, jaatha

Examples

```
# The original Jaatha anaylsis:
model <- create_jaatha_model(function(x) rpois(10, x),
                    par_ranges = matrix(c(0.1, 0.1, 10, 10), 2, 2),
                    sum_stats = list(create_jaatha_stat("sum", sum)))
data <- create_jaatha_data(rpois(10, 9), model)
jaatha_result <- jaatha(model, data, cores = 2)

# Bootstrapping the results using 2 CPU cores on localhost:
library(boot)
library(snow)
cl <- makeSOCKcluster(c("localhost", "localhost"))

jaatha_boot_results <- boot_jaatha(jaatha_result, 50,
                                   cores_per_run = 2,
                                   parallel = "snow",
                                   cl = cl)

stopCluster(cl)
boot.ci(jaatha_boot_results, type = "norm")
jaatha_boot_results$t0 <- jaatha_boot_results$mle
boot.ci(jaatha_boot_results, type = "norm")
```

`coarsen_jsfs` *Divides the joint site frequency spectrum (jsfs) into blocks and returns the sum of the jsfs entries for each block.*

Description

ja is the jsfs, part a list of vectors specifying for each dimension how ja should be partitioned. If part_hi!=NULL, it is a list spefifying how ja is to be paritioned on the higher end of each dimension. if part or part_hi is not a list, it is turned into a list of the same length as dim(ja), in which each entry is the original part or part_hi e.g. 2,7,9 partitions into 1:2, 3:7, 8:9, 9:N For example, with part=c(1,3) and part_hi=c(1,3) we get the classical jaatha summary statistics. Note, however, that the order in which they appear will be different than in the original jaatha package.

Usage

```
coarsen_jsfs(ja, part, part_hi = NULL)
```

Arguments

ja	an array containing the joint site frequency spectrum
part	a vector of integers or a list of vectors of integers. If it is a list, the vector part[[i]] specifies that the i-th dimension of ja should be partitioned into 1:(part[[i]][1]-1), part[[i]][1]:(part[[i]][2]-1), and so on. If part is a vector, it will be used for all dimensions.
part_hi	NULL or a vector of integers or a list of vector of integers indicating the partioning at the higher end of each dimension. This means, if it is a list, the values in the vector dim(ja)[i]-part_hi[[i]] will be appended to the end of part[[i]]. If part_hi is a single vector, it will be used for all dimensions. Thus, with the combination of part=c(1,3) and part_hi=c(1,3), the classical jaatha summary statistics, plus the two values ja[0] and ja[length(ja)]. Note that the order in which they appear will however be different than in the original jaatha summary statistics.

Value

vector of numbers, which are the sums over the blocks of the jsfs for all combinations of partitions

Author(s)

Dirk Metzler & Paul Staab

References

A. Tellier, P. Pfaffelhuber, B. Haubold, L. Naduvilezhath, L. E. Rose, T. Staedler, W. Stephan, and D. Metzler (2011) Estimating parameters of speciation models based on refined summaries of the joint site-frequency spectrum. PLoS One 6(5): e18155

---

create_jaatha_data *Prepare the observed data for Jaatha*

Description

By default, this function assumes that the observed data is in a format identical to the format of the simulation results, before the summary statistics are calculated.

Usage

```
create_jaatha_data(data, model, ...)

## Default S3 method:
create_jaatha_data(data, model, ...)
```

Arguments

data	The data to be analysed with Jaatha. It should be in a format identical to the simulation results (see create_jaatha_model).
model	The jaatha model, see create_jaatha_model.
...	Currently ignored.

Methods (by class)

- create_jaatha_data(default): The data's format is identicial to the simulated data.

Demographic Inference

When used with **coala**, coala::calc_sumstats_from_data() should create output that is compatible with this function.

---

create_jaatha_model *Specify a Model for a Jaatha Analysis*

Description

This function can be used to create models for an analysis with Jaatha. Models can be created using simulation function (see create_jaatha_model.function) or using a **coala** model (see create_jaatha_model.coalmodel).

Usage

```
create_jaatha_model(x, ..., scaling_factor = 1, test = TRUE)
```

Arguments

x	The primary argument. Can be a function used for simulations, or a coala model.
...	Additional parameters passed on to the dispatch function.
scaling_factor	If your model is a down-scaled version of your data, you can indicated this using this value. The estimated expectation values are multiplied with this factor before the likelihood is calculated.
test	A logical indicating whether a simulation is performed to test the model.

`create_jaatha_model.coalmodel` *Use a coala model in Jaatha*

Description

This creates a Jaatha model from a coala model. Simulation for this model model are conducted via the `simulate` function for the coala model. The parameters that are estimated must be specified via `par_range` and the model must not have any other named parameters. Summary statistics present in the coala model are used in Jaatha.

Usage

```
## S3 method for class 'coalmodel'
create_jaatha_model(
  x,
  jsfs_summary = c("sums", "folded_sums", "none", "smooth"),
  four_gamete_breaks = c(0.2, 0.5),
  mcmf_breaks = c(0.5, 0.7, 0.9),
  jsfs_part = c(1, 3),
  jsfs_part_hi = c(1, 3),
  ...,
  scaling_factor = 1,
  test = TRUE
)
```

Arguments

`x`	The coala model
`jsfs_summary`	The way the Joint Site Frquency Spectrum (JSFS) is further summarized. Can be `sums` (default), `none` or `"smoothing"`. For `sums`, 23 different areas of the JSFS are summed up, and the sums are used as indepented Poission statistcs. For `folded_sums`, the same sums will be calculate from the unpolarized (folded) JSFS. This does only support two population spectra and the default partitions at the moment. For `none`, all entries are used as indepented Possion statistics. The value `smooth` is experimental so far and should not be used. This option has no effect if the JSFS is not a summary statistic of the coala model.
`four_gamete_breaks`	Quantiles of the real data that will be used as breaks for binning the Four Gamete test based statistic if present in the model.
`mcmf_breaks`	Quantiles of the real data that will be used as breaks for binning the MCMF statistic if present in the model.
`jsfs_part`	Partitions used for the summarizing the JSFS. This is only used if `jsfs_summary` is "sums". Is used as the `part` argument of `coarsen_jsfs`. Please go there for an explanation. If `folded_sums` is used as jsfs summary, the values of jsfs_part and jsfs_part_hi will be ignored, and their default values `c(1, 3)` will be used instead.

jsfs_part_hi	Same as jsfs_part, but used as part_hi argument in coarsen_jsfs.
...	Additional parameters passed on to the dispatch function.
scaling_factor	If your model is a down-scaled version of your data, you can indicated this using this value. The estimated expectation values are multiplied with this factor before the likelihood is calculated.
test	A logical indicating whether a simulation is performed to test the model.

`create_jaatha_model.function`

Specify a jaatha model using a simulation function

Description

This is the usual way to specify a jaatha model. An detailed exampled on doing so is given in the 'jaatha-intro' vignette.

Usage

```
## S3 method for class '`function`'
create_jaatha_model(
  x,
  par_ranges,
  sum_stats,
  ...,
  scaling_factor = 1,
  test = TRUE
)
```

Arguments

x	A simulation function. This function takes model parameters as input, and returns the simulated data. The function must take exactly one argument, which is a numeric vector of model parameters for which the simulation should be conducted. The function should return the simulation results in an arbitrary format, that is then passed on to the summary statistics.
par_ranges	A matrix stating the possible values for the model parameters. The matrix must have one row for each parameter, and two columns which state the minimal and maximal possible value for the parameter.
sum_stats	A list of summary statistics created with create_jaatha_stat. The simulation results will be passed to the statistics, which should convert them into a numeric vector.
...	Currently unused.
scaling_factor	If your model is a down-scaled version of your data, you can indicated this using this value. The estimated expectation values are multiplied with this factor before the likelihood is calculated.
test	A logical indicating whether a simulation is performed to test the model.

Examples

```
create_jaatha_model(function(x) rpois(10, x),
                    par_ranges = matrix(c(0.1, 0.1, 10, 10), 2, 2),
                    sum_stats = list(create_jaatha_stat("sum", sum)))
```

`create_jaatha_stat` *Create a summary statistic for Jaatha*

Description

This function creates summary statistics for Jaatha models. A summary statistic consists primarily of a function that calculates the statistic from the simulation results. Jaatha primarily supports Poisson distributed summary statistics, but can also transform summary statistics that follow a different distribution in approximately Poisson distributed statistics.

Usage

```
create_jaatha_stat(name, calc_func, poisson = TRUE, breaks = c(0.1, 0.5, 0.9))
```

Arguments

`name`	The name of the summary statistic
`calc_func`	The function that summarizes the simulation data. Must take two arguments. The first is the simulated data, and the second are options that can be calculated from the real data. Ignoring the second argument in the function body should be fine in most situations. The function must return a numeric vector if `poisson = TRUE`, and can also return a numeric matrix if `poisson = FALSE`.
`poisson`	If `TRUE`, it is assumed that the summary statistic values are (at least approximately) independent and Poisson distributed. If it is set to `FALSE`, the statistic is transformed into an approximately Poisson distributed array using a binning approach. See "Transformation of non Poisson distributed statistics" for details. If any summary statistic is only approximately Poisson distributed, Jaatha is a composite-likelihood method.
`breaks`	The probabilities for the quantiles that are used for binning the data. See the section on non Poisson distributed summary statistics for details.

Value

The summary statistic. Indented for being used with `create_jaatha_model`.

Transformation of non Poisson distributed statistics

To transform a statistic into approximately Poisson distributed values, we first calculate the empirical quantiles of the real data for the probabilities given in `breaks`. These are used as break points for divining the range of the statistic into disjunct intervals. We then count who many of the values for the simulated data fall into each intervals, and use this counts as summary statistic. The counts are multinomial distributed, and should be close to the required Poisson distribution in most cases.

`estimate_llh` *Estimate the Log-Likelihood for a given parameter combination*

Description

This function estimates the Log-likelihood value for a given parameter combination. It conducts a number of simulations for the parameter combination, averages the summary statistics to esimate their expected values, and uses them to calculate the likelihood. For a resonable number of simulation, this is more precise than the glm fitting used in the main algorithm.

Usage

```
estimate_llh(
  model,
  data,
  parameter,
  sim = 100,
  cores = 1,
  normalized = FALSE,
  sim_data = NULL
)
```

Arguments

`model`	The model used for the estimation. See `create_jaatha_model`.
`data`	The data used for the estimation. See `create_jaatha_data`.
`parameter`	The parameter combination for which the loglikelihood will be estimated.
`sim`	The number of simulations that will be used for averaging the expectation values of the summary statistics.
`cores`	The number of CPU cores that will be used for the simulations. The relies on the **parallel** package, and consequently only one core is supported on Windows.
`normalized`	For internal use. Indicates whether the parameter combination is normalized to [0, 1]-scale, or on its natural scale.
`sim_data`	For internal use. Use existing simulations.

`jaatha` *Simulation based maximum likelihood estimation*

Description

Simulation based maximum likelihood estimation

Usage

```
jaatha(
  model,
  data,
  repetitions = 3,
  sim = model$get_par_number() * 25,
  max_steps = 100,
  init_method = c("zoom-in", "initial-search", "random", "middle"),
  cores = 1,
  verbose = TRUE,
  sim_cache_limit = 10000,
  block_width = 0.1,
  final_sim = 100,
  zoom_in_steps = 3
)
```

Arguments

model	The model used for the estimation. See create_jaatha_model.
data	The data used for the estimation. See create_jaatha_data.
repetitions	The number of independent optimizations that will be conducted. You should use a value greater than one here, to minimize the chance that the algorithms is stuck in a local maximum.
sim	The number of simulations conducted for each step.
max_steps	The maximal number of steps, in case Jaatha fails to converge.
init_method	Determines how the starting position of each repetition is chosen. See below for a description of the different options.
cores	The number of CPU cores that will be used for the simulations. The relies on the **parallel** package, and consequently only one core is supported on Windows.
verbose	If TRUE, information about the optimization algorithm is printed.
sim_cache_limit	The maximal number of simulations results that will be cached. Cached results may be reused in following estimation steps if they are within the current block. Reduce this value to save memory. Setting this to a value smaller than sim disables caching.
block_width	The relative width of a block within jaatha will fit its local GLM. The default value is usually fine. Increasing this value may help in case jaatha fails to converge, while you can try decreasing it if the estimates of the likelihoods differ from the corrected values in the 'Correcting likelihoods for best estimates' phase.
final_sim	The number of simulations conducted for calculating precise likelihoods for the best estimates found in the optimization procedure. These number of simulations is conducted for the best five estimates from each repetition. Using the default value is usually fine.

`zoom_in_steps` The number of steps conducted in the `zoom-in` initialization method. Has no effect if a different initialization method is used. Using the default value is usually fine.

Value

A list contain the results. The list has the following entries:

estimate The (approximated) maximum likelihood estimate

loglikelihood The estimate log-likelihood of the estimate.

converged A boolean indicating whether the optimization procedure converged or not

args The arguments provided to the jaatha function

Initialization Methods

Jaatha has different options for determining the starting positions for it's optimization procedure. The option `initial-search` will divide the parameter space in a number of equally sized block, estimate parameters within each block and use the estimates with the highest likelihood as starting positions. The option `zoom-in` starts with a block that is equal to the complete parameter space, estimate parameters in there, and then iteratively creates a smaller block around the estimates. Finally, `random` chooses random starting positions and `middle` will just start all repetitions at the middle of the parameter space.

Author(s)

Paul Staab, Lisha Mathew and Dirk Metzler

Index