Package 'likelihood'

May 8, 2026

Version 1.9

Title Methods for Maximum Likelihood Estimation

Date 2023-03-30

Author Lora Murphy [aut, cre]

Maintainer Lora Murphy <murphyl@caryinstitute.org>

Description Tools for maximum likelihood estimation of parameters
of scientific models. Based on Goffe et al (1994) <doi:10.1016/0304-4076(94)90038-8>.

Depends R (>= 2.1.1), nlme

License GPL-2

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2023-03-30 16:50:04 UTC

Contents

likelihood-package *Package for maximum likelihood estimation*

Description

This package allows you to find the maximum likelihood estimates of statistical models using simulated annealing, a global optimization algorithm.

Details

Package:	likelihood
Version:	1.5
Date:	2012-01-27
Depends:	R (>= 2.1.1, nlme
License:	GNU Public License

Several demonstration scripts are included in the demo directory.

Author(s)

Lora Murphy <murphyl@caryinstitute.org> Maintainer: Lora Murphy <murphyl@caryinstitute.org>

anneal *Perform Simulated Annealing for Maximum Likelihood Estimation*

Description

Performs simulated annealing - a global optimization algorithm - for maximum likelihood estimation of model parameters. Bounded, unbounded, and mixed searches can all be performed. See the Simulated Annealing Algorithm help page for more on how simulated annealing is actually performed.

Usage

```
anneal(model, par, var, source_data, par_lo = NULL, par_hi = NULL, pdf,
dep_var, initial_temp = 3, temp_red = 0.95, ns = 20, nt = 100,
max_iter = 50000, min_change = 0, min_drops = 100, hessian = TRUE,
delta = 100, slimit = 2, c = 2, note = "", show_display = TRUE, ...)
```

Arguments

model	Scientific model for whose parameters anneal will find maximum likelihood estimates. This is an R function.
par	List object of parameters for which to find maximum likelihood estimates using simulated annealing. The name of each component in par matches the name of an argument in one of the functions passed to anneal (either model, pdf, or any other function that you pass in). The value of each component is the initial value. All components in par must be numeric vectors. Vectors of length greater than one have each of their elements treated separately as individual parameters to estimate.
var	List object with the source for all other arguments and data used by model, pdf, and any other functions.
source_data	Data frame containing any needed source data. You can reference the data frame columns by name to anneal.
par_lo	List object with the lower search bounds for each parameter to estimate. The list component names and sizes should each match a component in par. Any individual component (up to and including the entire par_lo argument) is optional. For any component of par that is omitted, the lower search bound for that parameter is assumed to be negative infinity. (Infinity isn't quite infinity - see details section for more.)
par_hi	List object with the upper search bounds for each parameter to estimate. The list component names and sizes should each match a component in par. Any individual component (up to and including the entire par_hi argument) is optional. For any component of par that is omitted, the upper search bound for that parameter is assumed to be infinity. (Infinity isn't quite infinity - see details section for more.)
pdf	Probability density function to use in likelihood calculations. anneal depends on a log likelihood value, so you must instruct pdf to calculate the log of its result. This is an option with all of R's built-in PDFs.
dep_var	The name of the column in source_data, as a string, that contains the dependent variable (the "observed" value).
initial_temp	The temperature at which to start the annealing process.
temp_red	The rate of temperature reduction (a fractional number less than 1).
ns	Number of iterations between changes in parameter search ranges. One iteration varies all parameters one time.
nt	Controls number of iterations between drops in temperature. Temperature drops occur at nt * ns iterations. One iteration varies all parameters one time.
max_iter	Maximum number of iterations to perform. One iteration varies all parameters one time.
min_change	An alternate (and optional) way to specify quitting conditions for the run. This is the minimum amount of change in likelihood in min_drop number of temperature drops. If the change is less than min_change, execution stops even if max_iter number of iterations have not been performed.

min_drops	The companion to min_change for alternate quitting conditions. This is the number of temperature drops over which the likelihood must have changed more than min_change for execution to continue.
hessian	if TRUE, the Hessian matrix is used to calculate the standard error for each parameter and the parameter variance-covariance matrix. These are included in the output. If FALSE, this step is skipped.
delta	The number by which to divide each parameter maximum likelihood estimate value when searching for support limits. The bigger the number, the finer the search. See support_limits for more on how support limits are calculated.
slimit	When calculating support limits for the parameter maximum likelihood estimates, this is the number of likelihood units less than the optimum likelihood for which to search the parameter ranges. 2 units is standard. 1.92 units corresponds roughly to a 95 percent confidence interval.
c	Controls the reduction in parameter search range. A value of 0 would keep the search range permanently between the values set in par_lo and par_hi. A higher value will restrict the search more when range adjustments are made. A value of 2 is recommended by Goffe.
note	A note about the run. This can be any character string. This will be written to output files by write_results.
show_display	Whether or not to show the progress display.
...	Any other data needed by model, pdf, or any other function to be called by anneal. This is an alternative to providing the data in var; however, passing all values in var is strongly recommended.

Details

Simulated annealing is a search algorithm that attempts to find the global maximum of the likelihood surface produced by all possible values of the parameters being estimated. The value of the maximum that anneal finds is the maximum likelihood value, and the value of the parameters that produced it are their maximum likelihood estimates. See the Simulated Annealing Algorithm page for details on how the search is performed. See the Likelihood Calculation page for details on how likelihood is calculated. Simulated annealing is an algorithm that can search any function; but anneal specifically searches likelihood.

The model function is the scientific model, which generally takes as arguments the parameters for which to estimate maximum likelihood. It returns a predicted value of the dependent variable for each record in the source_data dataset, which is compared to the actual (observed) value when likelihood is calculated. Write model so that it returns a vector of numeric values, one for each record in the dataset.

The probability density function calculates the likelihood using the predicted and observed values of the dependent variable. You can provide your own function, but R has many built-in functions that you can use. You can read more about R's probability density functions in the help file "An Introduction to R", but here is a brief list: dbeta (beta), dexp (exponential), dgamma (gamma), dlnorm (lognormal), dnbinom (negative binomial), dnorm (normal), and dpois (poisson). These all take a log argument which you should set to TRUE in var in order to calculate the log likelihood. If you write your own probability density function, it should return a vector of values, one for each record in the dataset.

If you wish, some of the arguments passed to `model` or `pdf` by `anneal` can be the results of other functions. `anneal` will make sure these functions are evaluated at each search iteration.

`anneal` handles all function calls and data. You tell `anneal` how to use your functions and data using `par` and `var`. Use `par` to give `anneal` the list of parameters for which to find maximum likelihood estimates. All values must be numeric vectors. The name of each list component must match the function argument where the value should go. For example, if your model function takes an argument called "a", and you want the maximum likelihood estimate for a, there should be a `par$a`. If any component of `par` is a vector of length greater than one, each value is treated as a separate parameter to estimate. This is useful if, for example, you wish to estimate a parameter that has a different value for different sites or species.

Use `var` to tell `anneal` where all other functions and data come from. `var` is a list, and each component's name matches the function argument it should be used for (as with `par`). The value can be of any data type that makes sense to the function. To indicate that the source of a function argument is a column of data from a dataset, set that value of `var` to the name of the data frame's column, as a character string (for example, `var$dbh<-"DBH"`). Case matters! You will get the best results if all function arguments and column names are unique, so that there is no ambiguity. You are also free to reference values directly from the global environment in your functions if you prefer.

The reserved character string "predicted", used in `var`, means the predicted value of the dependent variable, as calculated by `model`.

If you want `anneal` to pass the results of another function as an argument to the `model` or `pdf` functions, define the function and then set the appropriate argument in `var` to the name of the function. Then provide all arguments to the sub-function in `var` as well. For instance, if your model function takes an argument called `x`, and you wish `x` to be the result of function `fun1`, then set `var$x <- fun1`, and add any arguments to `fun1` to `var`. `anneal` will ensure that all functions are evaluated in the proper order.

If the likelihood is calculated as infinity or NaN (which can easily happen), the likelihood is arbitrarily set to -1000000 to preserve the ability to graph results and compare values. If your best likelihood is -1000000, it is possible that no valid likelihood value was found.

The search ranges for parameters can be set to (or allowed to default to) negative and positive infinity. In practice, the search is bounded by the largest and smallest values the computer can work with. To find out what the actual limits are on your computer, use `.Machine$double.xmax`.

When looking at the examples provided in the demos that come with this package, check those for `likeli` as well, since the parameter setup techniques are the same.

Value

A list object with information on the annealing run. If you stop the run by pressing Esc, you will get this data structure with the results of the run at the point where you stopped it.

`best_pars`	The maximum likelihood estimates for each value in `par`.
`var`	A copy of the `var` argument, to help you keep track of your analysis. To save space, any data frames are removed.
`source_data`	A copy of the `source_data` data frame, with a column added for the predicted values calculated by `model` using the maximum likelihood estimates of the parameters.
`pdf`	The name of the `pdf` function.

model	The name of the model function.
iterations	The number of annealing iterations completed. One iteration varies all parameters one time. If the run does not complete, this may not be an integer.
max_likeli	The maximum likelihood value found.
aic_corr	The value of Akaike's Information Criterion, "corrected" for small sample size. See the Simulated Annealing Algorithm help page for more.
aic	The value of Akaike's Information Criterion. See the Simulated Annealing Algorithm help page for more.
slope	Slope of observed values linearly regressed on those predicted by model, using the parameter maximum likelihood estimates. The intercept is forced at zero.
R2	Proportion of variance explained by the model relative to that explained by the simple mean of the data.
likeli_hist	Data frame with the history of likelihood change throughout the run. All changes in likelihood are recorded, along with regular periodic checkpoints. The columns are: "temp", the temperature at that point, "iter", the number of iterations completed, and "likeli", the maximum likelihood value.
par_lo	List object with the lower bounds for each of the parameters. If any value was omitted in the original arguments, it is recorded here as a value that approximates negative infinity.
par_hi	List object with upper bounds for varying parameters. If any value was omitted in the original arguments, it is recorded here as a value that approximates infinity.
par_step	List object with final size of the search range for each parameter.
note	The value of the note argument, above.
upper_limits	List object with upper support limits for each parameter. For more on support limits, see the support_limits function.
lower_limits	List object with lower support limits for each parameters. For more on support limits, see the support_limits function.
std_errs	If anneal was run with hessian = TRUE, this is a list object with the standard errors for each parameter.
var_covar_mat	If anneal was run with hessian = TRUE, this is the parameter variance / covariance matrix.

References

Goffe, W.L., G.D. Ferrier, and J. Rogers. 1994. Global optimization of statistical functions with simulated annealing. Journal of Econometrics 60:65-99.

Examples

```
##
## Simulated annealing to maximize log
## likelihood for the following:
## Model: Radius = a + b * DBH
## Dataset: included crown_rad dataset
```

```
## We want to use simulated annealing to
## find maximum likelihood estimates of
## the parameters "a" and "b".
##
## Not run:
library(likelihood)

## Set up our dataset
data(crown_rad)
dataset <- crown_rad

## Create our model function
modelfun <- function (a, b, DBH) {a + b * DBH}

## Create the list for the parameters to estimate and
## set initial values for a and b
par <- list(a = 0, b = 0)

## Create a place to put all the other data needed by
## the model and PDF, and indicate that DBH comes from
## the column marked "DBH" in the dataset
var <- list(DBH = "DBH")

## Set bounds and initial search ranges within which to search for parameters
par_lo <- list(a = 0, b = 0)
par_hi <- list(a = 50, b = 50)

## We'll use the normal probability density function -
## add the options for it to our parameter list
## "x" value in PDF is observed value
var$x <- "Radius"

## Mean in normal PDF
var$mean <- "predicted"
var$sd <- 0.815585

## Have it calculate log likelihood
var$log <- TRUE

results<-anneal(model = modelfun, par = par, var = var,
  source_data = dataset, par_lo = par_lo, par_hi = par_hi,
  pdf = dnorm, dep_var = "Radius", max_iter = 20000)

## Alternately: reference crown_rad$DBH directly in the function without
## using var
modelfun <- function (a, b) {a + b * crown_rad$DBH}
var <- list(x = "Radius",
            mean = "predicted",
            sd = 0.815585,
            log = TRUE)
results<-anneal(model = modelfun, par = par, var = var,
  source_data = dataset, par_lo = par_lo, par_hi = par_hi,
  pdf = dnorm, dep_var = "Radius", max_iter = 20000)
```

```
## End(Not run)
```

`crown_rad` *Dataset of Tree DBH and Crown Radius*

Description

This is a set of imaginary data for DBH and crown radius for a set of trees.

Usage

```
crown_rad
```

Format

Tab-delimited text.

`from_sortie` *Generated Tree Allometry Dataset*

Description

This is a set of generated data for tree allometry.

Usage

```
from_sortie
```

Format

Tab-delimited text.

likeli *Calculate Likelihood*

Description

Calculate likelihood of a model, given a dataset. Typically this is log likelihood. See the Likelihood Calculation page for details on how likelihood is calculated.

Usage

```
likeli(model, par, var, source_data, pdf, ...)
```

Arguments

model	Model function for which to calculate likelihood. See details for how to set up this function.
par	List object of parameters for which to calculate likelihood. The name of each component in par matches the name of an argument in one of the functions passed to likeli (either model, pdf, or another function that does initial calculations). All elements in par must be numeric vectors. This is the same as the argument that you pass to anneal.
var	List object with the source for all other non-parameter arguments and data used by model, pdf, and any other functions. This is the same as the argument that you pass to anneal.
source_data	Data frame containing any needed source data. You can reference the data frame columns by name to likeli.
pdf	Probability density function to use in the likelihood calculation. If you want a log likelihood value, which is usual and matches what anneal does, instruct pdf to calculate the log of its result. This is an option with all of R's built-in PDFs.
...	Any other data that may be needed by model, pdf, or any other function to be called by likeli. This is an alternative to providing the data in var; however, passing values in var is strongly recommended.

Details

See the Likelihood Calculation page for details on how likelihood is calculated. anneal uses the same parameters and is set up in the same way.

The model function is the scientific model, which generally takes as arguments the parameters for which to estimate maximum likelihood. It returns a predicted value of the dependent variable for each record in the source_data dataset, which is compared to the actual (observed) value when likelihood is calculated. Write model so that it returns a vector of numeric values, one for each record in the dataset.

The probability density function calculates the likelihood using the predicted and observed values of the dependent variable. You can provide your own function, but R has many built-in functions that you can use. You can read more about R's probability density functions in the help file "An

Introduction to R", but here is a brief list: dbeta (beta), dexp (exponential), dgamma (gamma), dlnorm (lognormal), dnbinom (negative binomial), dnorm (normal), and dpois (poisson). These all take a log argument which you should set to TRUE in var in order to calculate the log likelihood. If you write your own probability density function, it should return a vector of values, one for each record in the dataset.

If you wish, some of the arguments passed to model or pdf by likeli can be the results of other functions.

likeli handles all function calls and data. You tell likeli how to use your functions and data using par and var. Use par to give likeli the list of parameters for the model. Use var to tell likeli where all other functions and data come from. var and var are lists, and each component's name matches the function argument it should be used for. For example, if the model function takes an argument called "a", there should be a par$a or a var$a with the value of a. For par, all values must be numeric vectors. For var, the values can be of any data type that makes sense to the function. To indicate that the source of a function argument is a column of data from a dataset, set that value of var to the name of the data frame's column, as a character string (for example, var$dbh<-"DBH"). Case matters! You will get the best results if all function arguments and column names are unique, so that there is no ambiguity. You are also free to reference values directly from the global environment in your functions if you prefer.

The difference between par and var is important to anneal but not to likeli.

The reserved character string "predicted", used in var, means the predicted value of the dependent variable, as calculated by model.

If you want likeli to pass the results of another function as an argument to the model or pdf functions, define the function and then set the appropriate argument in var to the name of the function. Then provide all arguments to the sub-function in var as well. For instance, if your model function takes an argument called x, and you wish x to be the result of function fun1, then set var$x <- fun1, and add any arguments to fun1 to var. likeli will ensure that all functions are evaluated in the proper order.

Value

A single numeric value for the likelihood. It is possible for this to be NaN or Inf.

Examples

```
library(likelihood)

## Use the included crown_rad dataset
data( crown_rad )

## Create our model function - crown radius is a linear function of DBH.
## DBH is a column of data from the crown_rad dataset; a and b are single
## parameter values.
model <- function (a, b, DBH) {a + b * DBH}

## Create our parameters list and set values for a and b
par <- list(a = 1.12, b = 0.07)

## Create a place to put all the other data needed by
## the model and PDF, and indicate that DBH comes from
```

```
## the column marked "DBH" in the dataset
var <- list(DBH = "DBH")

## We'll use the normal probability density function dnorm - add its
## arguments to our parameter list

## "x" value in PDF is observed value
var$x <- "Radius"

## The mean is the predicted value, the outcome of the model statement. Use
## the reserved word "predicted"
var$mean <- "predicted"
## Use a fixed value of the standard deviation for this example
var$sd <- 0.815585

## Have dnorm calculate log likelihood
var$log <- TRUE

result <- likeli(model, par, var, crown_rad, dnorm)

## Alternately: reference crown_rad$DBH directly in the function without
## using var
model <- function (a, b) {a + b * crown_rad$DBH}
var <- list(x = "Radius",
            mean = "predicted",
            sd = 0.815585,
            log = TRUE)
result <- likeli(model, par, var, crown_rad, dnorm)
```

Likelihood Calculation

Details on the Calculation of Likelihood

Description

There are four inputs to a likelihood calculation: a scientific model, a probability model, parameters for the model, and data. The scientific model mathematically describes one or more relationships that have been captured by the data. The probability model describes the error in the data. The parameters are the variables of interest for the scientific and probability models, for which you are trying to find the best values.

The dataset contains a dependent variable of interest. The values for this variable in the dataset are the "observed" values. The scientific model predicts values for this same dependent variable, based on other data and parameters. The values produced by the scientific model for the dependent variable are the "predicted" values. The differences between the observed and predicted values are the residuals.

The probability model is the core of likelihood estimation. Given the scientific model and a set of specific values for its parameters, there is a certain probability of observing the actual data. The mathematical relationship that describes that probability is the probability density function. This PDF is used to calculate the likelihood of the specific parameter values, given the data.

In order to do a likelihood calculation, you must identify your scientific model, choose a probability density function, and choose values for each of your parameters. To help you identify these pieces, here is an example. Suppose research is being conducted to study how cold weather affects sales at coffee shops. A dataset is gathered, with outdoor temperature and number of coffees sold. The researcher proposes that the number of coffees sold is a linear function of the outdoor temperature. The scientific model is:

$$Sales = a + b * Temp$$

The observed values for the dependent variable (coffee sales) are the sales data gathered. The parameters are a and b. Once test values have been chosen for a and b, we can calculate the likelihood of those values. To calculate the likelihood, the test values of a and b, along with the temperature data, are plugged into the scientific model, which gives us a set of predicted values for sales.

The error, the difference between the predicted and observed values, is described by the probability model. In our example, we will assume that the error is normally distributed. The normal probability distribution function is then the probability model. The probability model compares the predicted and observed values to produce the final likelihood.

If we repeat the likelihood calculation with another set of values for a and b, we can compare the two likelihood values. The values that produce the higher likelihood value are better. The values that produce the best likelihood possible are the maximum likelihood estimates for those parameters.

Details

For eqni = 1...N independent observations in a vector X, with individual observations x_i, and a set of parameter values θ:

$$Likelihood = L(\theta|X) = \prod_{i=1}^{N} g(x_i|\theta)$$

where $L(\theta|X)$ is the likelihood of the set of parameters θ given the observations X, and $g(x_i|\theta)$ is the probability density function of the probability model (i.e. the probability of each observation, given the parameters). Because logarithms are easier to work with, the preferred value is log likelihood, calculated as:

$$Loglikelihood = ln[L(\theta|X)] = \sum_{i=1}^{N} ln[g(x_i|\theta)]$$

Thus to calculate likelihood, we use the parameter values and the scientific model to calculate a set of predicted values for each of the observed values in the dataset. Then we use the probability density function to calculate the natural log of the probability of each pair of predicted and observed values. Then we sum the result over all observations in the dataset. For each data point, the mean of the probability density function is the observed value. The point for which the probability is being calculated, given that mean (generally called "x" in R's PDFs), is the predicted value.

likeli_4_optim *Use Likelihood with Optim*

Description

Wraps the function likeli so you can use it with optim. This allows you to use other optimization methods to find maximum likelihood estimates.

Usage

```
likeli_4_optim(par_2_analyze, model, par_names, var, source_data, pdf)
```

Arguments

par_2_analyze	Vector of initial values for those parameters that are to be optimized. This should be a vector, NOT a list. This MUST be a one-dimensional vector - i.e. none of the vector members can be vectors themselves (in contrast to the rules for anneal). optim will pass this argument to likeli_4_optim automatically. See the example for more.
model	Model function for which to calculate likelihood.
par_names	Character vector with the name for each value in par_2_analyze.
var	List object with the source for all other non-parameter arguments and data used by the model, the PDF, and any sub-functions. This is the same as the argument that you pass to anneal or likeli.
source_data	Data frame containing any needed source data, including observed values.
pdf	Probability density function to use. If you want a log likelihood value, which is usual, the PDF must calculate the log of its result.

Details

This wraps the likeli function so that it can conform to the requirements of optim. Setting up to use this function is exactly like setting up to use likeli.

Remember to set the fnscale option in the control list for optim to -1 so that optim performs a maximization rather than the default minimization (see example for details).

Value

A single numeric value for the likelihood. It is possible for this to be NAN or Inf.

Examples

```
#################
## Set up for likeli
#################
## Use the included crown_rad dataset
data(crown_rad)
```

```
## Create our model function - crown radius is a linear function of DBH.
## DBH is a column of data from the crown_rad dataset; a and b are single
## parameter values.
model <- function (a, b, DBH) {a + b * DBH}

## We are planning to get maximum likelihood estimates for a and b.  Create
## the list that says where all other functions and data are to be found.
## Indicate that DBH comes from the column marked "DBH" in the crown_rad dataset.
var<-list(DBH = "DBH")

## We'll use the normal probability density function dnorm - add its
## arguments to our parameter list
## "x" value in PDF is observed value
var$x <- "Radius"

## The mean is the predicted value, the outcome of the model statement. Use
## the reserved word "predicted"
var$mean <- "predicted"
var$sd <- 1.0

## Have dnorm calculate log likelihood
var$log <- TRUE

## Set up a vector with initial values for a and b
par_2_analyze <- c(0.1, 0.1)

## Set up the vector with the names of a and b, so likeli_4_optim knows
## what the values in for_optim are
par_names <- c("a", "b")

## Set your choice of optim controls - pass the other likeli_4_optim arguments
## by name so optim knows they are for likeli_4_optim
## Remember to set the fnscale option of optim to a negative value to perform
## a maximization rather than a minimization

## Not run: optim(par_2_analyze, likeli_4_optim, method = "Nelder-Mead",
  control = list(fnscale = -1), model = model, par_names = par_names,
  var = var, source_data = crown_rad, pdf = dnorm)
## End(Not run)
```

`predicted_results` *Calculate Model Predicted Results*

Description

Calculate predicted results of the dependent variable from a model with parameters set up as for the `likeli` and `anneal` functions. These predicted results are useful for various statistical calculations when compared to observed results from a dataset.

Usage

```
predicted_results(model, par, var, source_data, ...)
```

Arguments

model	Model function to use to calculate predicted results.
par	List of parameters for which likelihood is being estimated. All elements in par must be numeric vectors.
var	List object with the source for all other non-parameter arguments and data used by model, pdf, or any sub-functions.
source_data	Data frame containing any needed source data.
...	Any other data that may be needed by the model or any of its sub-functions. This is an alternative to providing the data in var; however, passing values in var is strongly recommended.

Details

The parameters for this function are set up exactly as they are in anneal and likeli. See those pages for details on how to do this.

Extra list members in var are ignored, so if you have set up a var list for use with likeli or anneal, you can use that list with predicted_results without removing arguments for the PDF.

Value

A vector of predicted results, one for each observation in source_data.

Examples

```
## Use the included crown_rad dataset
data( crown_rad )

## Create our model function - crown radius is a linear function of DBH.
## DBH is a column of data from the crown_rad dataset; a and b are single
## parameter values.
model <- function (a, b, DBH) {a + b * DBH}

## Create our parameters list and set values for a and b
par <- list(a = 1.12, b = 0.07)

## Create a place to put all the other data needed by
## the model and PDF, and indicate that DBH comes from
## the column marked "DBH" in the dataset
var <- list(DBH = "DBH")

predicted <- predicted_results(model, par, var, crown_rad)

## Calculate R2 - proportion of variance explained by the model relative to
## that explained by the simple mean of the data
meanrad <- mean(crown_rad$Radius)
```

```
sse <- (crown_rad$Radius - predicted)^2
sst <- (crown_rad$Radius - meanrad)^2
R2 <- 1 - (sum(sse)/sum(sst))
```

`Simulated Annealing Algorithm`

Details on the Simulated Annealing Algorithm

Description

This gives details on how the simulated annealing process is performed.

Details

When you are using likelihood methods to select the best parameter values for a scientific model, you need a method for searching the space of all possible values to find the global maximum likelihood. There are several search algorithms, and many R implementations of them. The simulated annealing algorithm is a good choice for maximizing likelihood for two reasons. The likelihood function is difficult to analyze using mathematical methods, such as derivation. Also, it often has a complex topology in parameter space, with local maxima, cliffs, ridges, and holes where it is undefined. Simulated annealing is an algorithm designed to deal with these problems. The algorithm of course can be applied to all kinds of problems, but its implementation in this package is for analyzing the likelihood function only.

An analogy for the search process is walking a mountain range in the dark, trying to find the highest mountain. In the beginning, when the algorithm's "temperature" is high, the search is energetic. In addition to moving uphill, it will also regularly move downhill to try to find a better uphill path. It will also jump off the mountain it's currently on to see if it lands on another, higher mountain. Later in the search, when the temperature and energy are lower, the algorithm works on reaching the top of the best mountain it has found. It may still move downhill to try to find a better path to the top but this becomes less and less likely.

The search (hopefully) ends with the algorithm converging on the global maximum. This may happen quickly or may take a very long time. The algorithm cannot know when it has found the global maximum, so it continues searching until it reaches a predefined end point, and leaves it up to you to judge the result. The set of search controls is called the annealing schedule, and defines the search's initial conditions, its rate of energy drop, and its end point. You can change this schedule to maximize the probability of convergence with the minimum amount of computation time.

You begin an annealing run by setting up the annealing schedule and the parameter search space. For the annealing schedule, you provide:

- Initial temperature (t). The higher the temperature, the more energetic the search.
- Rate of reduction in temperature (rt). This controls how quickly the temperature falls throughout the run.
- Rate of drops in temperature (nt). This controls how long the search stays at a certain temperature before further cooling.
- Interval between changes in range (ns). This controls how often the annealing process adjusts the parameter search range.

- An end point to the search. This is generally a maximum number of search iterations.

For the parameters, you provide:

- Initial values. The values whose likelihood is the point where the search starts.
- Upper and lower bounds. If desired or mathematically necessary. The annealing can also conduct a global search on one or more parameters.

Simulated annealing searches by randomly varying one parameter value, keeping all the other parameter values the same, and calculating the new likelihood. It compares this value to the last likelihood calculated to decide whether to keep the new parameter value or discard it. It then repeats this process by randomly varying the next parameter in the set. When each parameter has been varied one time, that is one search iteration. Simulated annealing then starts over with the first parameter again, beginning a new iteration.

The latest set of parameter values represents the point in the search space where the algorithm is on its current path. The algorithm also keeps a copy of the values that produced the highest likelihood yet found, so it can go back to that point.

Each time simulated annealing picks a new parameter value to test, it must decide whether to accept or reject the change. First, it compares the new parameter's likelihood value to the likelihood before the change. If the new value is equal to or greater than the previous value, the change in the parameter is accepted and the algorithm takes a step uphill. It then checks to see if it's at a new overall high point in the search. If so, it saves this set of parameter values as its best yet.

If the new parameter value's likelihood is worse than the previous one (representing a step downhill), simulated annealing uses the Metropolis criterion to decide whether or not to accept the move. The criterion is:

$$p = e^{-\frac{L1-L2}{t}}$$

where *p* is the probability the move will be accepted, *L1* is the previous likelihood, *L2* is the new likelihood, and *T* is the current annealing temperature. The algorithm compares a random number to this probability. If the move is accepted, the algorithm steps downhill. If the move is rejected, the new parameter value is discarded and the search stays in the same place, to try a step in a different direction with the next parameter.

The parameter values are randomly chosen within a range. The search begins with any defined upper and lower bounds, or infinity if there are none. Every *ns* iterations (where *ns* is the interval between changes in range in the initial annealing schedule), simulated annealing adjusts its search bounds for each parameter so that 50% of all moves will be accepted, either enlarging the bounds to find new ground to search or shrinking them to narrow in on a maximum.

After *ns* * *nt* iterations, the temperature *T* drops as

$$T' = rt * T$$

where *rt* is the rate of temperature reduction given in the initial annealing schedule.

The search ends when simulated annealing has reached the end point defined in its annealing schedule; either a maximum number of iterations, or a failure to find a higher likelihood within a set amount of temperature drop. The search may end before the global maximum has been reached.

Using the algorithm

You set up the annealing schedule and search bounds to maximize the probability of convergence on the global maximum while minimizing the computation time. Unfortunately, there are no firm rules for establishing that convergence has occurred. You can conclude that the algorithm has not converged if the maximum likelihood is still trending upwards when the run ends. If the maximum likelihood is stable for many iterations, this is evidence for convergence. Better evidence is multiple runs finding approximately the same maximum likelihood.

If an annealing run has not converged by the time it finishes, you can change the annealing schedule to improve the chances of convergence on a subsequent run. If the likelihood is changing at a rapid rate when the run finishes, give it more time by increasing the maximum iterations, and possibly increasing *ns* and *nt* as well. You can also begin subsequent runs by setting the parameter initial values to the best values found in the previous run, to allow it to continue searching where it left off.

If the maximum likelihood value does not change much throughout the run, but the maximum likelihood estimates for the parameters are not very good and you suspect that better values exist but were not found, it's possible the run was not effectively searching the parameter space. Try increasing the parameter bounds and the initial temperature to start a more energetic search.

Other information calculated

The simulated annealing algorithm returns many pieces of information to allow evaluation of the maximum likelihood estimates and comparison between models.

Akaike's Information Criterion. Akaike's Information Criterion is a measure of how well a model approximates reality. Its most common use is to compare models (based on the same dataset) that differ in their number of parameters. Models with more parameters will generally have higher likelihood, and AIC provides a means to incorporate principles of parsimony in model comparison.

AIC is calculated as:

$$AIC = -2ln(L(\theta|y)) + 2K$$

where $ln(L(\theta|y))$ is the log likelihood and *K* is the number of model parameters.

Unless the sample size is large relative to the number of model parameters, AIC corrected for small sample size (AICc) is recommended as an alternative. This is calculated as:

$$AIC_c = -2ln(L(\theta|y)) + 2K(\frac{n}{n-K-1})$$

where *n* = dataset size.

Slope. Slope is calculated as:

$$slope = \frac{\sum_{i=1}^{N}(obs_i)(exp_i)}{\sum_{i=1}^{N} exp_i^2}$$

where exp_i is the expected value of observation i in the dataset (obs_i) given the model.

R2. R^2 (different from r^2) is the proportion of variance explained by the model relative to that explained by the simple mean of the data. It is not bounded between 0 and 1. It is calculated as:

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{N}(obs_i - exp_i)^2}{\sum_{i=1}^{N}(obs_i - \overline{obs})^2}$$

where exp_i is the expected value of observation i in the dataset (obs_i) given the model, and $\overline{obs_i}$ is the mean of the observations.

Support limits. Support limits help you evaluate the strength of support for each parameter's maximum likelihood estimate. For details on how support limits are calculated, see the help page for the `support_limits` function.

Standard errors, variance and covariance. Standard errors are calculated using the Hessian matrix, which is a matrix of numerical approximations of the second partial derivatives of the likelihood function with respect to parameters, evaluated at the maximum likelihood estimates. Inverting the negative of the Hessian matrix gives the parameter variance-covariance matrix. The square roots of the diagonals of the variance-covariance matrix are the parameter standard errors.

References

Goffe, W.L., G.D. Ferrier, and J. Rogers. 1994. Global optimization of statistical functions with simulated annealing. Journal of Econometrics 60:65-99.

`support_limits` *Calculate Support Limits*

Description

Calculates asymptotic support limits for parameter maximum likelihood estimates. For a parameter, support limits are the values above and below the maximum likelihood estimate that cause the likelihood to drop by a given number of units, while holding all other parameters at their maximum likelihood values. Two units is standard. 1.92 units roughly corresponds to a 95% confidence interval.

Usage

```
support_limits(model, par, var, source_data, pdf, par_lo = NULL,
par_hi = NULL, delta = 100, slimit = 2)
```

Arguments

`model`	Model function for which to calculate likelihood. This is the same as the argument that you pass to `anneal` or `likeli`.
`par`	List of parameters for which to find the support limits. The name of each component in `par` matches the name of an argument in one of the functions passed to `support_limits` (either `model`, `pdf`, or another function that does initial calculations). The value of each component is the maximum likelihood estimate. All components in `par` must be numeric vectors. Vectors of length greater than one get a set of support limits calculated separately for each vector value. This is the same as the argument that you pass to `anneal` or `likeli`.
`var`	List object with the source for all other arguments and data used by `model`, `pdf`, and any other functions. This is the same as the argument that you pass to `anneal` or `likeli`.

source_data	Data frame containing any needed source data. This is the same as the argument that you pass to anneal or likeli.
pdf	Probability density function to use in likelihood calculations. This is the same as the argument that you pass to anneal or likeli.
par_lo	List object with lower bounds for the support limit search. The support limit bounds are in general the same as the simulated annealing search bounds. The list component names and sizes should each match a component in par. Any individual component (up to and including the entire par_lo argument) is optional. For any component of par that is omitted, the lower search bound for that parameter is assumed to be negative infinity. (Infinity isn't quite infinity - see details section for more.) This is the same as the argument that you pass to anneal.
par_hi	List object with upper bounds for the support limit search. The support limit bounds are in general the same as the simulated annealing search bounds. The list component names and sizes should each match a component in par. Any individual component (up to and including the entire par_hi argument) is optional. For any component of par that is omitted, the lower search bound for that parameter is assumed to be infinity. (Infinity isn't quite infinity - see details section for more.) This is the same as the argument that you pass to anneal.
delta	Controls the fineness of the search for support limits. Each parameter is divided by this number to arrive at a step size used for "walking" the likelihood function. Bigger numbers mean a finer search. See details for more on how the support limits are determined.
slimit	The number of units of likelihood that define the support limits. If slimit is 2, then the limits are those values that cause the likelihood to drop by 2 on either side of the parameter maximum likelihood estimate.

Details

Support limits are the values on either side of a parameter's maximum likelihood estimate that make the likelihood drop by slimit units, holding all other parameters at their maximum likelihood estimate value. Of course, support limits are only meaningful if the values in par are indeed maximum likelihood estimates. The distance from the maximum likelihood estimate of a parameter to its support limits is an indication of the "pointiness" of the maximum on the likelihood surface.

The algorithm produces support limits for a parameter by holding all other values at their maximum likelihood value and "walking" the likelihood function in the plane of that parameter, seeking to find the first spot that is slimit units below the peak likelihood. It starts by walking in big steps, then in progressively smaller steps, until it reaches that point. The smallest step it takes is found by dividing the parameter value by delta. This controls the overall fineness of the search.

The support limits search is bounded by the values in par_lo and par_hi. The search uses these bounds to control how it searches. This means that different bounds values may produce slightly different results. If a bounds value is omitted, support_limits will attempt an unbounded search, up to infinity. This will work fine as long as the likelihood surface is not completely flat. In practice, "infinity" means the largest and smallest values the computer can work with. To find out what the actual limits are on your computer, use .Machine$double.xmax.

This algorithm works best if the surface produced by the likelihood function is continuous and monotonic from the maximum likelihood value out to the support limits of all parameters. This is

often not true. However, in most cases, this will produce reasonably good results with a low amount of total computation time.

Support limits are calculated automatically at the end of an anneal run.

Value

A list object with two components: "upper_limits" and "lower_limits". upper_limits has the upper support limits for each member in par, with the maximum possible value being that parameter's value in par_hi; lower_limits has the lower support limits, with the minimum possible value being that parameter's value in par_lo.

If the likelihood calculated from par is infinite or NA, then the support limits will also be NA.

Note

The parameter maximum likelihood estimates found by anneal are in the list component called best_pars. These are the values to pass to support_limits for the par argument.

See Also

likeli, anneal

Examples

```
#################
## Set up for an annealing run
#################
## Use the included crown_rad dataset
data(crown_rad)

## Create our model function - crown radius is a linear function of DBH.
## DBH is a column of data from the crown_rad dataset; a and b are single
## parameter values.
model <- function (a, b, DBH) {a + b * DBH}

## Create our parameters list and set values for a and b, and indicate
## that DBH comes from the column marked "DBH" in the crown_rad dataset
par <- list(a = 1.12, b = 0.07)
var <- list(DBH = "DBH")

## We'll use the normal probability density function dnorm - add its
## arguments to our parameter list

## "x" value in PDF is observed value
var$x <- "Radius"

## The mean is the predicted value, the outcome of the model statement. Use
## the reserved word "predicted"
var$mean <- "predicted"
var$sd <- 0.815585

## Set bounds within which to search for parameters
```

```
par_lo <- list(a = 0, b = 0)
par_hi <- list(a = 50, b = 50)

## Have dnorm calculate log likelihood
var$log <- TRUE

## Not run:
results <- anneal(model, par, var, crown_rad, par_lo, par_hi, dnorm, "Radius", max_iter=20000)

## End(Not run)

##################
## Do support limits - even though there are a set already in results
##################

## Not run:
limits <- support_limits(model, results$best_pars, var, crown_rad, dnorm, par_lo, par_hi)

## End(Not run)
```

`write_results` *Write the Results of Simulated Annealing to File*

Description

Takes the results produced by the function `anneal` and writes them to a tab-delimited text file.

Usage

```
write_results(results, filename, data = TRUE, print_whole_hist = FALSE)
```

Arguments

`results`	The output list produced by the function `anneal`.
`filename`	A string with the file and path to the file you wish to write. This will overwrite any existing files of that name. This will not add any file extensions so remember to put on the appropriate one.
`data`	If TRUE, the `source_dataset` member of `results` is written to the file; if FALSE, it is not. Large datasets can inflate the size of output files.
`print_whole_hist`	If TRUE, the entire likelihood history of the run is printed; if FALSE, it is not. Long runs can have rather long histories.

Value

A file with the contents of `results` written as tab-delimited text.

See Also

anneal

Examples

```
## Assuming you have performed a simulated annealing run and placed the
## results in an object called "my_results"...
## Not run: write_results(my_results, "c:\results.txt")
```

Index