Package 'gafit'

October 13, 2022

Version 0.5.1

Date 2016-12-05

Title Genetic Algorithm for Curve Fitting

Author Telford Tendys <gafit@lnx-bsp.net>

Maintainer Telford Tendys <gafit@lnx-bsp.net>

Depends R (>= 3.0.0)

Description A group of sample points are evaluated against a
user-defined expression, the sample points are lists of
parameters with values that may be substituted into that
expression. The genetic algorithm attempts to make the result
of the expression as low as possible (usually this would be the
sum of residuals squared).

License GPL-2

URL <http://lnx-bsp.net/>

Repository CRAN

Date/Publication 2016-12-05 22:52:27

NeedsCompilation yes

R topics documented:

---

gafit	*Genetic Algorithm for Curve Fitting*

Description

Randomly iterate a group of samples (i.e. the 'gene pool') over a target function with the intent of
achieving the lowest target value. The target function is provided by the caller as an expression and
various other tuning parameters may be used to improve the convergence rate.

Usage

```
gafit( target, start, thermal=0.1, maxiter=50, samples=10, step=1e-3, tolerance=-Inf )
```

Arguments

target	An expression which returns a scalar real value. The algorithm will seek the lowest achievable value and save this in the "score" attribute (see below). Usually this would be a sum of residuals squared, so that the algorithm will seek to bring this as close as possible to zero.
start	A list of named values which will be used as the starting point for the curve fitting. This list lets the algorithm know what it is allowed to adjust so any parameters which the user wants to hold constant should be removed from this list and placed in the global environment instead. The mode of the parameters will not be changed by the curve fitting so if you provide integers or logicals then the algorithm will attempt to use parameters of that mode. Complex numbers are allowed as are vectors and matrices.
thermal	The probability that the internal bubble-sort will promote noisy samples rather than samples with a desirable score. Values above 0.1 should be used with caution. Some thermal noise is required such that the algorithm is discouraged from zooming straight into a local optima. From a user's perspective, adding thermal noise will reduce the precision of the final result but will widen the 'span' of the sample points making local optima less attractive. Often it is good to do a first run at a high thermal noise then reduce this toward zero once good starting values are are available (same principle as simulated annealing).
maxiter	In order to force the search to conclude, the number of iterations is limited. One iteration involved moving and re-evaluating all sample points. This argument allows the user to control the length of the search. There is no other termination condition except maxiter so it is also the minimum number of iterations.
samples	This controls the number of sample points in the 'gene pool' and thus the effectiveness of the algorithm. Numbers less than 5 are fairly pointless, the larger the number the better the search but the slower each iteration becomes. As a rough rule, this should be double the number of parameters in start.
step	The step size between samples is largely auto-adjusting but it has to start from somewhere. The user should put a value here which is a rough estimate of the distance (in parameter space) from the start values to the correct solution. If you have absolutely no idea what the distance might be then just put something small in comparison to the expected parameter values.
tolerance	If we find that the least squares value is less than this value then return early because the answer is considered good enough. NOTE: by default this is -Inf which will never trigger early exit.

Details

Genetic algorithms are driven by random samples so the same results may not be obtainable twice in a row. OK, I'll admit that lots of ad-hoc stuff went into this and it sometimes gets a completely wrong answer. Also there are some problems which it will never ever seem to get the exactly right answer but will reliably get something close. On the other hand, it does handle a wide range

of problems is not particularly finicky about the starting point (something in the right order of magnitude helps but is not essential). This makes it a good first stage in tackling problems which may be quite difficult to fit by more well established methods.

The results of this genetic algorithm may be used as a starting point for the nls regression algorithm (which will follow the gradient to the local optimum) so that a "nearly right" fit can be converted into a "best" fit. Often this chaining of regression algorithms requires that some deliberate error is introduced into the parameters because nls might complain about a singular gradient matrix (thinks... does nls attempt to narrow the step size for the numerical derivative when confronted by a singular gradient matrix? maybe it should).

Value

The returned value will be a list of the best parameter values that could be found. This list will be the same structure as the start list with new values inserted. The returned value will have an attribute called "score" which is the evaluation of target with those paramters, and also an attribute called "count" which counts the number of iterations completed (could be zero).

Known Bugs

There is no way to guarantee to avoid a local optima nor is there a way to be sure that any stationary point that has been discovered is the global optimum value (other than an infinite length search). As far as I know this is a theoretical limitation of all nonlinear regression, having a good overall understanding of the behaviour of the functions with which you are working with is essential.

Sometimes NaN values will be introduced into the parameters and then will go away again. Although many warnings get generated, the NaN values do not seem to turn up in the final result so this should be considered merely an harmless annoyance.

The thermal value is constant. Ideally it should gradually decrease itself but choosing the ideal "cooling curve" is too difficult, so it is left to the user to adjust this. The return value of one gafit() run can be used as the start value for the next round, making it easy to build a cascade and simulate stages of cooling.

The step size auto-adjustment can break in some situations producing amazingly wrong answers.

It is possible to generate an error which looks something like ".Random.seed[2] is not a valid integer". I blame the random generator for stuffing up but it might equally well be bugs in my code, or more likely a misunderstanding on my part as to exactly how the R API really works. If this happens, just put new values into the .Random.seed variable and try again.

Author(s)

Telford Tendys <gafit@lnx-bsp.net>

See Also

expression, nls, .Random.seed

Examples

```
# Single parameter, all real numbers (not using least squares)
e <- expression( cos( theta ) + sin( theta ))
```

```
guess.1 <- list( theta=3 )
guess.2 <- gafit( e, guess.1, step=1e-3 )  # First attempt with thermal noise
gafit( e, guess.2, step=1e-5, thermal=0 )  # usually gets close to 3.926991

# Double parameter, complex numbers (least square curve fit)
sumsq <- function( x ) { sum(( Mod( x )) ^ 2 )}
freq <- exp( 1:15 )
tpj <- 2 * pi * (0+1i)
data <- 1 / ( 10 + tpj * freq * 1e-3 )
e <- expression( sumsq( 1 / ( R + tpj * freq * C ) - data ))
guess.1 <- list( R=100, C=1e-6 );
guess.2 <- gafit( e, guess.1, step=0.1, maxiter=100, tolerance=1e-2 )
gafit( e, thermal=0, guess.2, step=1e-3, maxiter=200, tolerance=1e-5 )
```

Index