Genetic Optimisation

Andrew Nelson

November 2, 2006

1 Introduction

GeneticOptimisation.ipf is an IGOR procedure that uses a 'differential evolution' (Genetic Algorithm as detailed by Storn and Price[2] and Wormington et al[3]) technique to curve fit data, by minimising a Chi2 function.

The user needs to make a fit function that describes their model (see section 4.1, as well the IGOR Pro manual). Genetic Algorithms have a huge advantage over normal methods of analysis as they are able to find global minima in the Chi2 map very quickly (quicker than Simulated annealing for example). In contrast non-linear least squares descent methods, such as Levenberg-Marquardt, are often unable to escape local minima, although they are more effective if the descent to the global minimum in Chi2 is smooth (and contains few/little local minima).

2 Useage

2.1 Installation

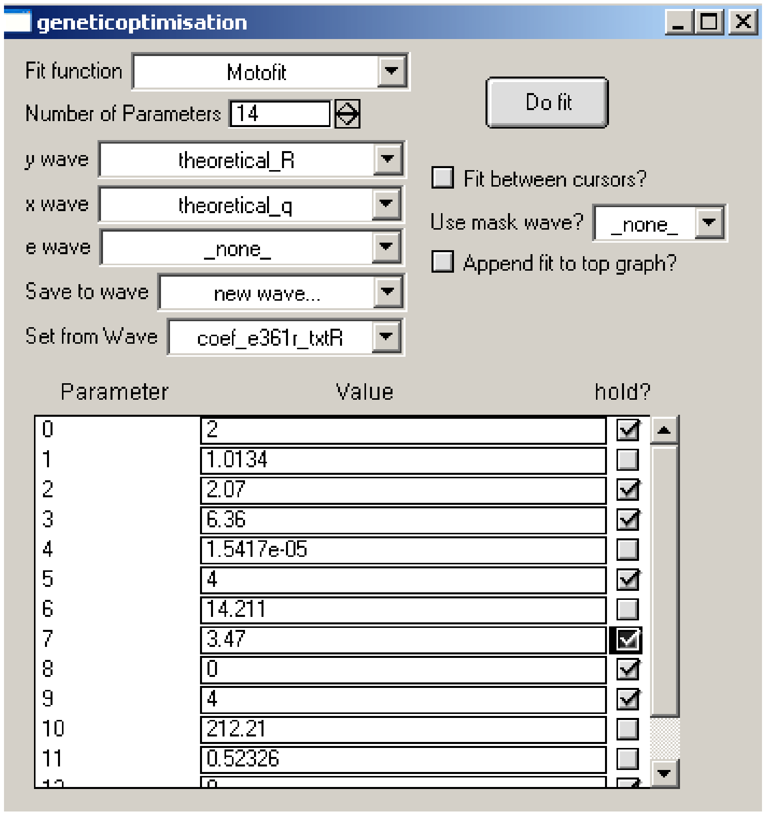
Just load the *GeneticOptimisation.ipf* file, then you should be able to use the genetic optimisation from IGOR. If you wish to perform co-refinement of multiple datasets (also called global fitting in IGOR), then you should also click on *MOTOFIT_ Global fit 2.ipf.*

2.2 Performing a Genetic Optimisation

2.2.1 Using the GUI

Simply select *Genetic Optimisation* from the *Motofit* menu. You will get a panel to help you set up your fit. You may select any user-defined fitfunction, including XOP and all-at-once functions.

The panel is fairly straightforward to use, but is not datafolder aware. In other words, the *ywave*, *xwave* (+*ewave*) need to be in the current datafolder. Once the fit has finished the fit waves and a coefficient wave will be produced in the same datafolder. You can also ask for the fit waves to be appended to



Fit function	Motofit
Number of Parameters	14
y wave	theoretical_R
x wave	theoretical_q
e wave	_none_
Save to wave	new wave...
Set from Wave	coef_e361r_txtR
Fit between cursors?	☐
Use mask wave?	_none_
Append fit to top graph?	☐

Parameter	Value	hold?
0	2	✓
1	1.0134	☐
2	2.07	✓
3	6.36	✓
4	1.5417e-05	☐
5	4	✓
6	14.211	☐
7	3.47	✓
8	0	✓
9	4	✓
10	212.21	☐
11	0.52326	☐

Figure 1: The Genetic Optimisation panel

the top graph. You can also select a fit to a selected region of data which are defined by cursors on a graph of your data.

Please also note that the error wave should contain the standard deviations of the ydata

2.2.2 Programatically

The curve fitting is called by issuing the following text in the command line:

GEN_ curvefit(functionstring,parwave,ywave,holdstring[,x=xwave]

[,w=weightwave][,c=limitwave][,mask=maskwave][,cursors=cursors]

[,popsize=populationsize][,k_ m=mutationconstant]

[,recomb=recombinationconstant][,iters=iterations][,tol=tolerance][,q=quiet])

Required parameters:

- *Functionstring*: a string containing the name of a user defined fit function (e.g. "Gaussian"). The fit function can either be a normal fit function, or an all-at-once fit function.
- *parwave*: a wave containing the initial parameters for the fit.
- *ywave*: a wave containing the y (ordinate) data.
- *holdstring*: a string which specifies which parameters you would like to vary (0), and those you would like to hold (1). The holdstring should be the same length as parwave.

Optional parameters:

- *xwave*: a wave containing the x (abscissa) data. If this parameter is not specified then the fit assumes that you are using the scaling of the ywave during the fit. (i.e. Genetic Optimisation supports waveform data).
- *weightwave*: a weighting containing the standard deviations (uncertainty) of the ywave. If you do not specify this wave, then the fit will weight each point equally.
- *limitwave*: a wave containing the lower and upper bounds for each of the parameters. The limitwave should have the same number of rows as the parwave, with two columns. The first column should contain the lower limit and should be less than the upper limit, which is contained in the second column.
- *maskwave*: specifies that you want to use the wave named maskwave to select points to be fit. The mask wave must match the y wave in number of points and dimensions. Setting a point in the maskwave to zero or NaN (blank in a table) eliminates that point from the fit.
- *cursors*: if cursors=1 then the cursors on the uppermost graph are used to select a region of interest in the fit.
- *populationsize*: the population size multiplier (section 2.2.3).
- *mutationconstant*: sets the mutation constant (section 2.2.3).
- *recombinationconstant*: sets the recombination constant (section 2.2.3).
- *iterations*: the number of iterations for the fit (section 2.2.3).
- *tolerance*: the fraction tolerance for the fit (section 2.2.3).
- *quiet*: if quiet is 1 then nothing is printed in the history area.

An example of the command:

GEN_curvefit("motofit",coef_e361_R,e361_R,"10110101110011",x=e361_Q,w=e361_E)

In this example there are a total of 14 parameters in the *coef_ e361_ R* wave, which is used as input to the *motofit* function.

NOTE:

- If you don't specify the optional limits wave you will be prompted to enter boundary values.
- If you don't specify *popsize,k_ m,recomb* or *tol*, then you will be prompted to enter initialisation values.
- although the genetic algorithm is quite good at finding global minima, it's important you don't choose silly numbers for your starting guesses/limits. For example, if the function you are fitting to crashes on certain values (e.g. sqrt(-1)), then the *Gen_ curvefit* will also fall over.
- The command is datafolder aware. In other words *parwave, ywave, xwave, weightwave* can all be in different folders when you call *GEN_ curvefit* (you can call from any folder). For details on what happens to the output + where is goes please see Section 2.2.4.

2.2.3 What happens next?

A datafolder called *GEN_ optimise* is created, which holds all waves, variables, etc, while fitting. The data you are trying to fit is copied to this datafolder.

The program then asks for information to set up the genetic optimisation (unless you included this information from a command line call).

- *Number of generations.* This is effectively the number of iterations you want to do. The more iterations you use, the longer it takes, but you should expect a closer answer. Obviously, if this number is too big, then you will be wasting your time. I suggest trying 100 generations to start off with.
- *Mutation Constant.* The mutation constant is used to create genetic diversity. A larger value means a larger genetic diversity (more likely to find a solution), but it will take longer for the fit to converge. This value should lie in between 0 and 1. Start out with 0.7 and see how you go.
- *Population Size multiplier.* The genetic algorithm works by starting off with a large population of initial randomised solutions (that still lie in between the limits you set), which are then examined. The total size of this population is set as by this *popsizemultiplier*the number of free parameters.* Setting this value to a larger number increases the probability of finding a good solution, but 20 has worked for me so far.

- *Recombination constant.* Should lie between 0 and 1. This controls how much of the original guess is used in the evolved guess. If you use a high number then the original guess will not evolve all that much. If you use a lower number then genetic diversity will increase, and you will be more likely to find a solution (but will take longer to converge). Try starting out with 0.5.

- *Fractional tolerance to stop fit.* If the fractional improvement in Chi2 is less than this value then the fit will stop.

The user is then presented with a table showing the initial guesses and the lower and upper limits to be placed on those parameters during the fit. When you are satisfied with the setup then press *continue* and the optimisation will start. If the limits are not set correctly you will be asked to set them up again.

While the optimisation is proceeding the updated Chi2 value will be displayed in a graph. This graph is used to show the colourised fit parameters for each of the members of the genetic population. As the fit proceeds towards convergence then the colour for a particular parameter, over all the population, should converge. In addition a progress graph is provided.

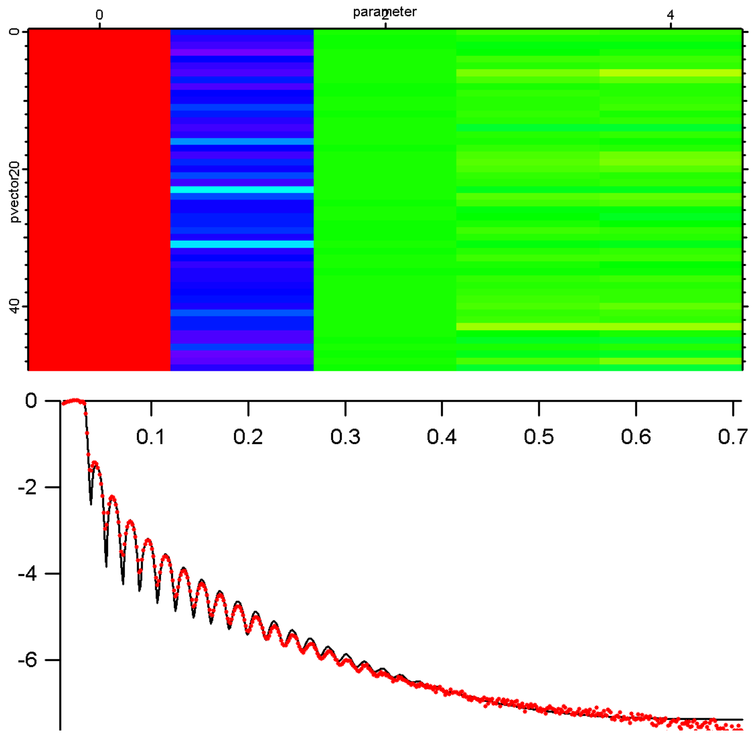


Figure 2: Watching a genetic optimisation converge

The fit will normally terminate when the fractional change in Chi2 is less than the user specified value or the optimisation goes through all the generations requested.

2.2.4 The output from Genetic Optimisation

On finishing the optimisation (or aborting it part way through) fitwaves will be produced in the datafolder that you were in when you started the genetic optimisation. If you used the GUI to perform the fit a coefficient wave is produced in the same datafolder. If you used the command line to perform the fit then the coefficient wave you used to supply the original guesses is updated with the results of the fit.

These will be named using the wavename of the original *ywave* you used in the fit. For example:

You are in datafolder *root:somewhere:*, and you call *GEN_ curvefit* as:

GEN_ curvefit("motofit",root:foo:testcoef,root:bar:yhello_ txt,

"10110101110011",x=root:bar:xhello_ txt,w=root:bar:ehello_ txt)

The fit waves *fit_ yhello_ txt* and *fitx_ yhello_ txt* are produced in *root:somewhere:,* the coefficient wave *testcoef* is updated in *root:foo:.*

At the end of the fit you will be in datafolder *root:somewhere:,* the Chi2 value will be printed in the history area, and placed in the global variable V_chisq. In addition a wave containing uncertainties of the fitted parameters, *W_ Sigma*, is produced in the originating datafolder. These uncertainties are estimated as the change in parameter that causes Chi2 to increase by 2%.

2.3 Co-refinement/global/simultaneous fitting

If you wish to simultaneously analyse multiple datasets at once (co-refinement with linked parameters), then also load *MOTOFIT_ Global fit 2.ipf.* This procedure is derived from the *Globalfit2* procedures of John Weekes, but should co-exist and run separately from that version. This type of fitting is often used in neutron and X-ray scattering problems. You can use this functionality by selecting *MotoGlobal fit* from the *Motofit* menu.

3 Licensing

GeneticOptimisation.ipf uses a publicly available algorithm, however it took me time to write it. Therefore the procedure is released with the GNU General Public Licence (GPL) `http://www.opensource.org/licenses/gpl-license.php`. The licence is distributed with the procedure file, please read and understand before you use it.

This procedure was developed as part of the *Motofit* package. Therefore, if you publish data (e.g. scientific journal) which was analysed using this *GeneticOptimisation* procedure (even if you didn't use *Motofit*), I would really appreciate it (i.e. not compulsory, but I would like it) if you could quote the *Journal of Applied Crystallography* paper[1] in the work.

4 Appendix

4.1 An example fit function

This fit function fits a Gaussian to a set of data:

```
Function MyGaussian(w,y,x): Fitfunc
Wave w,y,x
y = w[0] + w[1] * exp(-((x-w[2])/w[3])^2)
End
```

For more examples please see the IGOR Pro Manual.

References

[1] A. Nelson. Co-refinement of multiple contrast neutron / X-ray reflectivity data using MOTOFIT. *Journal of Applied Crystallography*, 39:273–276, 2006.

[2] R Storn and K.V. Price. Differential Evolution - a simple and efficient scheme for global optimization over continuous spaces. Technical report, 1995.

[3] M. Wormington, C. Panaccione, K.M. Matney, and D.K. Bowen. Characterization of structures from X-ray scattering data using genetic algorithms. *Philosophical Transactions of the Royal Society of London*, 357:2827–2848, 1999.