User Manual for “ga_refl”

1 INTRODUCTION 1
2 GENETIC ALGORITHM 1
  2.1 *Initial Population* *2*
  2.2 *Evaluation of fitness* *2*
  2.3 *Selection of parents* *3*
  2.4 *Breeding* *3*
  2.5 *Mutation* *3*
  2.6 *Creation of new population* *3*
3 INSTALLATION 3
4 SETTING UP A FIT 3
  4.1 *Headers* *4*
  4.2 *Initial set-up* *4*
  4.3 *Resolution* *5*
  4.4 *Initial model* *6*
  4.5 *Parameters* *6*
  4.6 *Constraints* *7*
5 RUNNING GA_REFL 8
  5.1 *Compiling the setup file* *8*
  5.2 *Run options* *8*
  5.3 *Interrupt options* *8*
  5.4 *Restarting* *9*
  5.5 *Seeing results* *9*

1 Introduction

“Ga_refl” is a C++-based system for modelling reflectivity data using Paratt-formalism optical matrices, and model refinement using a genetic algorithm. It has been designed from the outset to enable the simultaneous fitting of multiple datasets, which may have parameters in common or parameters linked in complex fashion, and above all it has been designed to be flexible to allow individual users to describe their own systems. It can be used with magnetic (polarised) and non-magnetic data, and with X-ray and neutron data. There is no limit to the number of datasets or parameters which can be fitted. It has so far been tested on Linux, Cygwin for Windows, and Mac OS X. It does **not** have a graphical interface, however, and requires a functioning C++ and Fortran compiler and some understanding of basic programming to make it work.

Future improvements will hopefully include: steepest descent model refinements, detailed error estimation, simplex model refinements, free-form reflectivity modelling, functional-form based SLD profiles, and a simpler user interface.

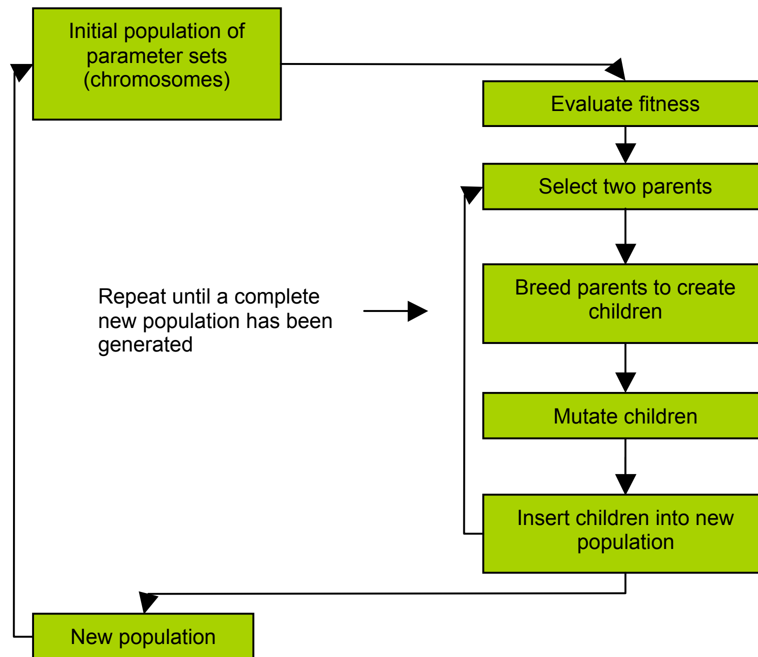
It is very much a work in progress. Suggestions / ideas / bug reports (especially bug solutions) should be addressed to Paul Kienzle or Mathieu Doucet at reflectometry-software@nist.gov.

2 Genetic Algorithm

A genetic algorithm borrows from the ideas of evolution to produce a model refinement method that explores parameter space in a semi-directed fashion, but which avoids being trapped in local minima. The general scheme of a typical genetic algorithm is illustrated below. Sets of possible values of fitting parameters are created, which, borrowing from the terminology of genetics, are known as a population of chromosomes. These chromosomes are then evaluated against a test function (for example chi-squared) for their fitness, which is used to determine the probability that a given chromosome will contribute to the next generation of the population. Pairs of chromosomes are then selected, and are “bred” to produce new

chromosomes for the next generation, a process repeated as many times as necessary to populate the new generation. Finally the new "children" chromosomes are exposed to the randomising effects of mutations, before the new population replaces the old.

There is a large body of published literature on the subject of genetic algorithms, and they have been shown to be effective minimisation tools in many systems. In part this is because the method does not depend on specific functional derivatives of the fitness function with respect to the fitting parameters, which allows us to use complicated sets of parameters and relationships between them. Computationally the largest overhead of the method is the evaluation of the fitness function, which involves the calculation of a reflectivity for each dataset using each chromosome, in each generation. Nevertheless experience so far has shown the method to converge quickly in representative systems, and the optimisation of the many aspects involved may produce further improvements



The scheme of a typical genetic algorithm.

2.1 Initial Population

The size of the population (number of different sets of parameter values considered in each generation) is a user-defined variable. The trade-off is between speed in calculating generations, and the size of the parameter-space which is explored in each generation. The default is 50.

The initial population of chromosomes uses randomly-generated values for all the parameters (although optionally one can include an initial guess as one of the chromosomes). Each chromosome in the population is created by taking each input parameter value, and normalising it to a number between 0 and 1 with respect to its fitting range. Parameters cannot leave their fitting range. The chromosome is then simply the list of these values, and the population the list of all the chromosomes. The population is regularly stored to disk during a calculation, and can also be written on demand to facilitate restarting a run.

2.2 Evaluation of fitness

Genetic algorithms are normally maximising functions, so the fitness defined for each set of parameter values is calculated as $\chi^2_{max}-\chi^2$ where χ^2_{max} is a user-defined constant value (by default 5000 × number of datasets). Parameter sets which give rise to $\chi^2>\chi^2_{max}$ are given a fitness of zero. Each time the fitness of a chromosome is better than the current best the user is notified and the parameter values are stored.

2.3 Selection of parents

The probability of a chromosome being selected as a parent for breeding purposes is linearly proportional to its rank by fitness as a fraction of the sum of the ranks of the population, so better fitting parameters are more likely to be represented in the next generation.

2.4 Breeding

Breeding is not always performed when creating children from selected parents – a parameter is defined which gives the breeding probability (default 0.85). If the parents are not bred, the children created are clones of the parents. Otherwise breeding occurs by taking the chromosome of each parent, listing the values as a single long string of numbers, choosing randomly a crossing point, and swapping all the digits after the crossing point from one chromosome to the other (algorithm type 0, default). An alternative method uses the same principle, but crosses each parameter value separately rather than the chromosome as a whole (algorithm type 2). A third method averages the parameter values of the two parents (algorithm type 1). The default appears to give the best results so far. Two children are always produced from each pair of parents.

2.5 Mutation

Each digit in each parameter value of the child chromosome is then assessed for mutation. The probability of mutation is initially set to 0.2, but varies during the routine within the range 0.01 – 0.35, increasing as the degree of variation in the population decreases (and vice versa). If a digit is mutated, it is replaced with a random number.

2.6 Creation of new population

The above process is repeated as many times as necessary to create a new population of chromosomes the same size as the initial population. With the "elite" option set (default) the fittest chromosome (best set of parameter values) is always preserved from one generation to the next, ensuring that the best fit never becomes worse (although worse fits are obviously still considered in each generation, and contribute to the exploration of parameter space). The new population then replaces the old, and a new generation begins.

3 Installation

You need to unzip and then extract the distribution provided:

```
djm% gunzip ga_refl-2005.03.11-src.tar.gz
djm% tar -xf ga_refl-2005.03.11-src.tar
```

This will produce a directory *ga_refl_nnnn.nn.nn-src/* with all necessary files and sub-directories (including sub-directories of example files).

In the ga_refl directory, type

```
djm% ./configure [--disable-magnetic] [FLIBS=-lg2c]
```

which will automatically produce the necessary configuration files [use the --disable-magnetic switch if you do not wish to analyse magnetic data. If you wish to analyse magnetic data you must also have a Fortran 77 compiler installed. For Mac OS X.3 the flibs may be necessary], and finally

```
djm% make
```

which will compile the main ga_refl code.

4 Setting up a fit

The initial model, data sets, constraints and beam details are adjusted in the *setup.c* file. The initial model is called only once in setting up the optical matrices, but the constraints are evaluated each time the fitness of a chromosome is tested. An example *setup.c* file is attached (Appendix A) but the pieces that make it up are described individually below. Anything bracketed by `/*…*/` is only a comment.

The *setup.c* file consists in order of Headers, Constraints, Initial Set-up, Initial Model, Parameters.

4.1 Headers

```
#include <stdio.h>
#include "refl.h"
#define MODELS 2
double vol_fract1, vol_fract2, rho_spacer, rho_alkyl;
```

MODELS defines the number of models which will be created – normally the number of datasets being evaluated. Non-standard fitting parameters should be declared here as well (in this example *vol_fract1* etc.)

4.2 Initial set-up

```
/*===========INITIAL SETUP==========================================*/
fitinfo* setup_models(int *models)
{
  static fitinfo fit[MODELS];
  int i;
  fitpars *pars = &fit[0].pars;
  fitpars *freepars = &fit[1].pars;
  *models = MODELS;

  for (i=0; i < MODELS; i++) fit_init(&fit[i]);
```

Data is loaded here. Datafiles are expected to be in white space separated columns of Q, R, Err(R) (or Q, log(R), 1/*ln*10•Err(R)). Four column data, in which the final column consists of the Err(Q), will be accepted in the future. The number of datasets expected is defined by the parameter MODELS, set in the Header section. Each dataset will have a model and beam description associated with it, in the structure fit[*i*], where *i* is the zero-base index of the dataset.

```
/*Non-magnetic data*/
fit_data(&fit[0],"/Users/djm/data/wc02.yor");
fit_data(&fit[1],"/Users/djm/data/wc03.yor");
fit_data(&fit[2],"/Users/djm/data/wc06.yor");
fit_data(&fit[3],"/Users/djm/data/wc07.yor");
```

```
/*Magnetic data*/
dataset = "f167+169bf"; /*datafile base*/
snprintf(fa,sizeof(fa),"%s.qa",dataset);
snprintf(fb,sizeof(fb),"%s.qb",dataset);
snprintf(fc,sizeof(fc),"%s.qc",dataset);
snprintf(fd,sizeof(fd),"%s.qd",dataset);
fit_polarized_data(&fit[0],fa,fb,fc,fd);
/* one for each dataset */
```

4.3 Resolution

The program allows the instrumental resolution to be defined differently for each data set, and also allows some flexibility in defining the resolution differently for different data points *within* a single data set, to allow for the variety of collection methods possible.

```
/* Assign resolution to points in the data.
 *
 * Fixed slits:
 *   data_resolution_fixed(data,L,dLoL,0.,0.,dT);
 *
 * Opening slits:
 *   data_resolution_varying(data,L,dLoL,0.,0.,dToT);
 *
 * Fixed below |Qlo| then opening:
 *   data_resolution_fv(data,L,dLoL,|Qlo|,dTlo,dToT);
 *
 * Opening between |Qlo| and |Qhi|, fixed above and below:
 *   data_resolution_fvf(data,L,dLoL,|Qlo|,|Qhi|,dTlo,dToT,dThi);
 *
 * Fixed in stages (e.g., from TOF source):
 *   data_resolution_fixed(data,L,dLoL,0.,Q1,dT1);
 *   data_resolution_fixed(data,L,dLoL,Q1,Q2,dT2);
 *   data_resolution_fixed(data,L,dLoL,Q2,Q3,dT3);
 *   ...
 *   data_resolution_fixed(data,L,dLoL,Qn,0.,dT3);
 *
 * If instrument uses angle T rather than Q:
 *   #include "reflcalc.h"
 *   Q = T2Q(L,T)
 *
 * To compute dT from slit openings and separation use:
 *   #include "reflcalc.h"
 *   dT = resolution_dT(s1,s2,d)
 *
 * For opening slits, dToT also needs an incident angle in degrees:
 *   #include "reflcalc.h"
 *   dToT = resolution_dToT(s1,s2,d,T)
 */


/* Initialize instrument parameters for each model.*/
  for (i=0; i < MODELS; i++) {
#include "reflcalc.h"
    const double L = 5.0042,dLoL=0.020,d=1890.0;
    double Qlo,Tlo,dTlo,dToT,s1,s2;
    Qlo=0.0154,Tlo=0.35;
    s1=0.21, s2=s1;
    dTlo=resolution_dT(s1,s2,d);
    dToT=resolution_dToT(s1,s2,d,Tlo);
    data_resolution_fv(&fit[i].dataA,L,dLoL,Qlo,dTlo,dToT);
    fit[i].beam.lambda = L;
    interface_create(&fit[i].rm, "erf", erf_interface, 30);
  }
```

For this example all measurements were made in the same conditions (the values are appropriate for AND/R), so a loop can be used across all the datasets. Otherwise the *fit[i].beam* information needs to altered for each dataset individually. *beam.lambda* is the radiation wavelength (Å). Other possible variables to

adjust here are *beam.frontbackratio* (default 1), *beam.background* (1e-10), and *beam.intensity* (1). Roughness is treated by calculating dividing each density step between slabs into a fixed number of smaller substeps with a given shape, here an error function of 30 substeps per density step.

4.4 Initial model

```
/*=============INITIAL MODEL – Non magnetic=====================================*/
for (i=0; i < MODELS; i++) {
    model_layer(&fit[i].m, 0.000, 2.07e-6, 0.0e-8, 0.00); /* 0 silicon    */
    model_layer(&fit[i].m, 7.000, 3.40e-6, 0.0e-8, 3.00); /* 1 oxide      */
    model_layer(&fit[i].m, 5.800, 3.01e-6, 0.0e-8, 3.00); /* 2 chromium   */
    model_layer(&fit[i].m, 90.70, 4.50e-6, 0.0e-8, 3.00); /* 3 gold       */
    model_layer(&fit[i].m, 18.00, 1.00e-6, 0.0e-8, 3.00); /* 4 spacer     */
    model_layer(&fit[i].m, 28.00, -0.2e-6, 0.0e-8, 3.00); /* 5 alkyl tails*/
    model_layer(&fit[i].m, 100.0, 6.35e-6, 0.0e-8, 3.00); /* 6 solvent    */
  }
```

or

```
/*=============INITIAL MODEL – Magnetic=====================================*/
for (i=0; i < MODELS; i++) {
    model_magnetic(&fit[i].m,0,0,0,0,0,0,0,0);    /*vacuum*/
    model_magnetic(&fit[i].m,110.0,7.1e-6,2.7e-9,9.8,5.7e-07,9.8,246.0,9.82);/*X*/
    model_magnetic(&fit[i].m,270.0,6.8e-6,2.7e-9,43.2,1.0e-6,43.2,263.2,90.0);/*Fe3O4*/
    model_magnetic(&fit[i].m,134.0,8.2e-6,5.7e-9,11.8,4.7e-6,11.0,272.2,11.8);/*Fe*/
    model_magnetic(&fit[i].m,100.,5.3e-6,2.1e-10,3.5,0.0,3.5,272.2,3.5);/*MgAl2O4*/
  }
```

This creates a model from series of slabs for each dataset loaded above, and stores it in *fit[i].m*. In this example there are the same number of layers for each dataset, so a loop is used across the number of models – otherwise a separate model can be created for each dataset individually. The parameters for each slab are in order thickness (*d* / Å), SLD (*rho* / $Å^{-2}$), absorption (*mu* / $Å^{-1}$), and the FWHM roughness between this slab and the previous slab (*rough* / Å) [and for magnetic samples P, P roughness, theta, and theta roughness]. There is no limit to the number of layers.

```
/*correct layers for different models*/
fit[1].m.rho[6]=3.4e-6;
fit[2].m.rho[6]=-0.57e-6;
fit[3].m.rho[6]=4.0e-6;
```

Any differences in the models for the different datasets can be altered afterwards (here the solvent SLD varies for the four different datasets). The convention is *fit[i].m.variable[k]*, where *i* refers to the dataset number, *variable* is one of *d*, *rho*, *mu*, or *rough* [or the magnetic specific parameters], and *k* is the layer number (starting from layer 0, the incident medium).

```
rho_spacer=0.60e-6;
rho_alkyl=-0.20e-6;
vol_fract1=0.9;
vol_fract2=0.99;
```

Non-standard variables should also be initialised here.

4.5 Parameters

Parameters are described as fit and/or free parameters. Fit parameters are adjusted by the genetic algorithm, and the adjusted value is, by default, copied into all models. To prevent a parameter being overwritten it must be added to the free parameter list – free parameters are parameters whose values are allowed to differ between models. A fit parameter may also be a free parameter (for example, the SLD of the solvent is different for each dataset, so each one must be a free parameter. To fit the SLD of the solvents as well they are simply added as fit parameters). There is no limit to the number of parameters in either case.

```
/*=============== FIT PARAMETERS ================================*/
  pars_add(pars, "Oxide thickness", &(fit[0].m.d[1]), 5., 30.);
  pars_add(pars, "Cr thickness", &(fit[0].m.d[2]), 5., 30.);
  pars_add(pars, "Au thickness", &(fit[0].m.d[3]), 40., 150.);
```

```
  pars_add(pars, "Au density", &(fit[0].m.rho[3]), 4.0e-6, 4.8e-6);
  pars_add(pars, "Spacer thickness", &(fit[0].m.d[4]), 5., 20.);
  pars_add(pars, "Spacer volume fraction", &(vol_fract1), 0, 1);
  pars_add(pars, "Alkyl chain thickness", &(fit[0].m.d[5]), 5., 30.);
  pars_add(pars, "Alkyl volume fraction", &(vol_fract2), 0, 1);
  pars_add(pars, "SLD solvent 0", &(fit[0].m.rho[6]), 5.0e-6, 6.5e-6);
  pars_add(pars, "SLD solvent 1", &(fit[1].m.rho[6]), 3.0e-6, 4.5e-6);
  pars_add(pars, "SLD solvent 2", &(fit[2].m.rho[6]), 5.0e-6, 6.5e-6);
  pars_add(pars, "SLD solvent 3", &(fit[3].m.rho[6]), 3.0e-6, 4.5e-6);
```

These lines set up the fit parameters, assigning them a "name", pointing to the parameter in question (a standard variable or a non-standard variable declared and initialised earlier, e.g. *vol_fract1* above), and setting the lower and upper limits of the parameter range. The free parameters are defined below. The name and range is unimportant for these as the *freepars* only marks them as differing from model 0.

```
  pars_add(freepars, "SLD solvent 2", &(fit[1].m.rho[6]), 0., 1.);
  pars_add(freepars, "SLD solvent 3", &(fit[2].m.rho[6]), 0., 1.);
  pars_add(freepars, "SLD solvent 4", &(fit[3].m.rho[6]), 0., 1.);
  pars_add(freepars, "rough_cr_au", &(fit[0].m.rough[3]), 0., 1.);

  constraints = constr_models;
  return fit;
}
```

Finally the constraints are assigned to complete the model formation.

4.6 Constraints

Although assigned at the end of the initialisation the constraints should be defined before, and hence appear near the beginning of the *setup.c* file. The constraints are extremely flexible – each slab of the matrix model can be altered in any way, or set equal to other slabs' values. These constraints are evaluated each time the fitness of a chromosome is tested.

```
/*=========== CONSTRAINTS =====================*/
void constr_models(fitinfo *fit)
{
  int i,k;

  /* Rescue the free parameters from the model. */
  for (i=0; i < fit[1].pars.n; i++)
    fit[1].pars.value[i] = *(fit[1].pars.address[i]);

  /* Go through all layers copying parameters from model 0 to other models */
  tied_parameters(fit);

  /* copy the global roughness to all interfaces*/
  for (i=0; i< MODELS; i++) {
    for (k=1;k<7; k++) fit[i].m.rough[k]=global_rough;
  }

  /* Restore the free parameters to the model. */
  for (i=0; i < fit[1].pars.n; i++){
    *(fit[1].pars.address[i]) = fit[1].pars.value[i];
  }

  /* allow the cr_au roughness to differ from the global rough
   * - this was saved as a free parameter for model[0], now copy to
   * other models
   */
 for (i=1; i< MODELS; i++) {
    fit[1].m.rough[3]=fit[0].m.rough[3];
 }
  /*calculate the layer densities for layers given in vol fractions*/
  for (i=0; i< MODELS; i++) {
    fit[i].m.rho[4]= vol_fract_spacer*rho_spacer+(1-vol_fract_spacer)*fit[i].m.rho[6];
    fit[i].m.rho[5]= vol_fract_alkyl*rho_alkyl+(1-vol_fract_alkyl)*fit[i].m.rho[6];
  }
}
```

The function *tied_parameters* set all values of thickness/density/mu/rho [and magnetic values] to be the same across all the models, *except* for those previously defined as free parameters. Following this is an example of the kinds of custom constraints that can be used – here the SLD of the "spacer" layer and the "alkyl chain" layer for each of the datasets is calculated from the volume fraction of the molecule and the solvent SLD, which differs for each dataset. It is thus the volume fraction of the molecule which is directly fitted, rather than the individual SLDs of the layers. Also a global roughness is modelled – one roughness value for all interfaces except the specifically excluded Cr-Au interface.

5 Running ga_refl

5.1 Compiling the setup file

- ensure that the Makefile is in the same directory as the *setup.c* file

- type "make" (or "gmake" on Jazz) in this directory

Note that errors may be reported if you have not correctly formed your *setup.c* file. These errors are reported by the compiler with the line number of the error. Common mistakes include incorrect closures of brackets, use of variables that have not been initialised, or forgetting to use a semi-colon to end an instruction.

Warning: you must always recompile after making alterations to your *setup.c* file or the alterations will have no effect.

5.2 Run options

The command "fit -h" will provide a quick guide to the starting options. These are –

```
usage: fit [-pmgFLM] [-x n] [-c cutoff]
  -A <%> accelerated by computing at most % partial chisq
  -F fit the parameters, writing to fit.log each step (default)
  -g write profile.dat and fit.dat[ABCD] for the initial model
  -m print the initial model and profile to the screen
  -p use saved population from pop.dat
  -x <nth> print chisq landscape of the nth parameter using initial model values
  -i keep initial model parameters in initial population (default uses random population)
  -c <value> define the chisq value cut-off
  -L log all parameter sets and associated chisq to fit.log
  -t trace period for writing pop_bak.dat (0 for none, default=20)
  -N create new pop_##.dat file each trace period
  -s seed value for random number generator (allows completely reproducible runs)
  -w force unweighted fit
  -W force weighted fit (default)
```

Standard usage is either "fit" for a new fit, or "fit -p" to restart a previously interrupted fit. To restart a previous fit the population must have been saved upon exit (see below), or the backup population which is periodically written to disk must be renamed to "pop.dat". When restarting a fit the existing parameter file "par.dat" is appended to, in starting a new fit the parameters are overwritten.

5.3 Interrupt options

By default the genetic algorithm continues to run until manually stopped. To interrupt the fitting press "Ctrl-C". This will bring up a menu with a range of options –

```
Select an option:
   q (quit)
   w (write population)
   b (write and quit)
   a (run amoeba from current best)
   A (accelerate)
   r (randomize)
   x (plot chi^2)
   c (change a parameter)
   p (print current best values)
   R (Change range for a parameter)
   any other character to continue
```

"write" and "write and quit" both write the current population of chromosomes to disk in the file "pop.dat", "quit" exits the program without storing the current population. The population is automatically stored to a backup file, "pop_bak.dat" every 20 generations by default.

The other options work with specific parameters. The parameter number is the zero-based index of the parameter, as is printed on screen by using the "p" option.

"randomize" allows you to randomise the population's values for a specific parameter, but does not affect the fittest chromosome.

"plot chi-squared" prints to screen the chi-squared landscape for the selected parameter using the fixed current best values for all other parameters.

"c" and "R" allow you to change the value of a parameter and its range respectively. When the value of the parameter is changed it is changed in *all* chromosomes, including the best. This may result in a worsening of the overall chi-squared of the fit. As the value is altered in all the chromosomes, it is normally a few generations before differences between chromosomes are seen. Altering the range of a parameter should be used with care, as this may lead to inconsistencies between a stored population and the initial model when a fit is restarted.

5.4 Restarting

Fits may be restarted by using the "fit -p" option. This will cause the program to initialise based on the compiled *setup.c* file, but it will then use the stored population in "pop.dat" as its initial population. Information will be appended to the parameter file "par.dat" rather than overwriting it.

5.5 Seeing results

Output from the fitting is stored in ".dat" files in the same directory as the program is run. The datafiles used, parameters fitted, their ranges, and each improvement in the best values are all stored in the "par.dat" file. This file also records modifications to the fit in progress, such as alteration of parameter values or ranges, or notes a restart.

As noted above the population of chromosomes can be written to the "pop.dat" file, and is automatically written to the "pop_bak.dat" file every 20 generations by default.

The actual optical matrix parameters used in calculating the best fit are stored in "model*n*.dat", where *n* refers to each dataset. This is a useful place to check that special constraints are doing what they should.

The data and best fit are stored in "fit*n*.dat".

The model profiles of the best fit are stored in "profile*n*.dat", where the columns are vertical distance from the interface (z), rho, mu, [and then P and theta for magnetic systems].

To plot the the data sets a script has been written which uses gnuplot, and is called *gaplot*. This is invoked from the X-window command line.

```
djm% gaplot rho|mu|P|theta|fit|chisq|chisurf [file.ps]
```

For more detail on gaplot options type

```
djm% gaplot
```