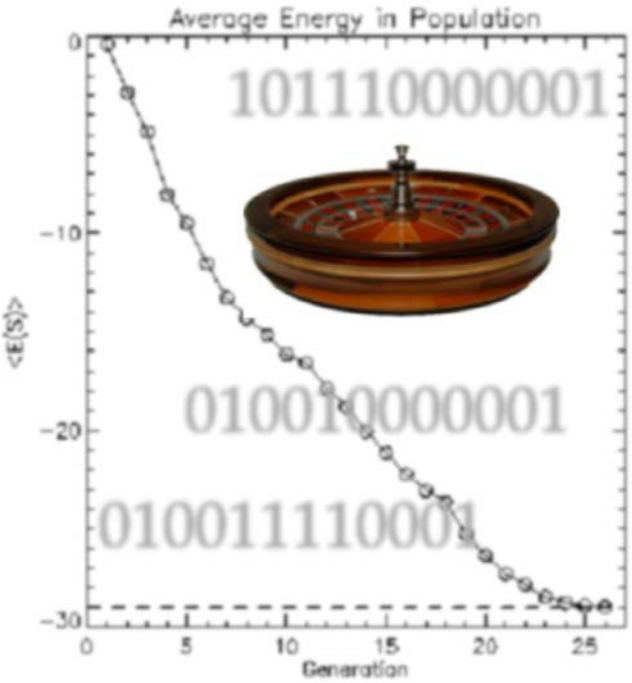
Using Genetic Algorithms for Global Optimization in IDL



Rob Dimeo
NIST Center for Neutron Research
100 Bureau Drive-MS 8562
Gaithersburg, MD 20899

Contents

Acknowledgments

This work is based upon activities supported by the National Science Foundation under Agreement No. DMR-0086210.

Introduction

Data analysis often requires a robust means of minimizing functions. In a conventional least-squares curve fitting application the quantity χ^2 (defined as the sum of the difference between the model function and the data normalized by the uncertainty of the data) is typically minimized by varying the fit parameters. In many cases the use of least-squares algorithms are adequate for fitting simple model functions to data. Success using such algorithms however often require that you provide initial parameter guesses that are close to the minimum of χ^2. If the initial parameter guesses are not in the vicinity of the global minimum of χ^2 then the algorithm can "get stuck" in a local minimum. Many minimization algorithms that require derivatives or involve hill-climbing can "get stuck" in local minima.

A number of heuristic strategies have been developed over the years that have been used in a global optimization context. Many of these are stochastic, directed searches. One such strategy that has proven successful in a number of engineering and scientific areas is the Genetic Algorithm (GA). This technique uses an evolutionary paradigm in which a random population of parameter sets are *bred* and *evolved* over a number of generations in the hopes that one will optimize the function in question (e.g. χ^2).

A function named RMD_GA was written in the IDL programming language[1] as part of a concerted effort to develop a toolbox of robust fitting routines for use in the DAVE software project. DAVE (which stands for the Data Analysis and Visualization Environment) is a reduction, visualization, and analysis package with tools for planning neutron scattering experiments written in IDL. More information on this package can be found at the DAVE website: http://www.ncnr.nist.gov/dave. RMD_GA is the first in a series of robust optimization functions written for this purpose.

In this document we will describe a simple genetic algorithm (SGA), the function RMD_GA that implements it, and describe how you can use RMD_GA in your own function optimizations. A number of examples that use RMD_GA are shown including two-dimensional function minimization, curve-fitting, and even finding the ground-state configuration of the one-dimensional Ising spin chain[2]. This function was written to be flexible enough to allow you to use your own function for minimization and also to allow you to visualize the progress of the algorithm as the evolution proceeds. The examples presented

[1] IDL stands for the Interactive Data Language and is a registered trademark of Research Systems, Inc. Manufacturers are identified in order to provide complete identification and such identification is not intended as a recommendation or endorsement by NIST.
[2] A. Prügel-Bennett and J.L. Shapiro, Analysis of Genetic Algorithms Using Statistical Mechanics, Phys.Rev.Lett. **72**, 1305-1309 (1994).

here are drawn from different areas and the emphasis is on visualization so that you can gain some intuition with usage.

All of the Genetic Algorithm files needed can be found on the following web page: http://www.ncnr.nist.gov/staff/dimeo/idl_programs.html under the heading of ANALYSIS PROGRAMS.

The Simple Genetic Algorithm

The simple genetic algorithm can be described in broad strokes as follows[3]:

(1) an initial population of NPOP members (chromosomes) is created,
(2) the fitness of each member of the population is calculated,
(3) a new population of members is created based on the fitness of the members of the previous population,
(4) members of the new population are selected at random and genetically mixed in an operation called *crossover*,
(5) a random gene within a randomly selected chromosome is modified in *mutation*.
(6) Return to (2) and iterate until either some convergence criterion has been satisfied or some maximum number of iterations (generations) have been performed.

There are a number of challenges in adopting this algorithm for function optimization. The first challenge is to determine how to encode the parameter space into the language of the SGA, namely chromosomes that can be operated on by the crossover and mutation operators. One popular method is to use binary coding, as discussed in Goldberg's book[3]. This is the coding method used in RMD_GA so we will discuss the SGA operators in terms of their effects on binary chromosomes.

A chromosome encoded in binary is represented as a sequence of 1s and 0s. An example of a 4 bit binary encoded chromosome is 1011. In decimal this corresponds to 11 ($=2^0+2^1+2^3$). In many optimization applications you may wish to map the raw decimal value onto some other range. This is straightforward. For instance, a single parameter in a range [5,10] results in a value of 1011 of 8.67 ($=5+((10-5)/(2^4-1))\times 11$). In general, for an n-bit chromosome whose raw decimal value is x and the range onto which you wish to map the parameter is $[x_{lo},x_{hi}]$, the mapping is expressed as: $x_{lo}+((x_{hi}-x_{lo})/(2^n-1))*x$.

For search problems involving more than a single parameter, we use multi-parameter binary encoding in which a single chromosome is made up of smaller

[3] D.E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning* (Addison-Wesley, Reading, MA, 1989).

ones (representing each parameter in the search) concatenated together. So if we choose a 3-bit binary encoding in a 2-parameter search, a single chromosome is 6-bits long. Naturally we must keep track of the ordering of the parameters in the chromosome.

The first step in the SGA (1) is to create an initial population. You can specify the number of members of the population, N_{pop}, as well as the number of bits (N_{bits}) with which to represent a particular parameter. The more bits there are in a parameter representation, the more precise the result. If there are Nparams parameters in the search space then there will be N_{pop} chromosomes of length $N_{parms} \times N_{bits}$. The sequences of 1s and 0s are determined by random number generation in the initial population.

The next step in the SGA is to determine the fitness of each member of the population. This step requires evaluating the function at each point defined by the binary-encoded chromosomes. In order to do this we first decode the raw binary-encoded chromosome and then evaluate the function. In general, the SGA is a function maximization routine and the result is a maximization of the fitness function. However our main application is function minimization. Therefore we must impose a mapping of the maximization problem onto one of minimization. While you are free to implement any mapping which does this (as explained in the next section) the default method involves a simple linear transformation. If E_i represents the evaluation of the function you wish to minimize, f, with parameter set p_i, i.e. $f(p_i) = E_i$, then the fitness function F can be constructed as $F = 1+max(E)-E$. This has the advantage of maintaining a positive-definite fitness function as required by the SGA.

Another weighting method that has been used in SGA minimization is the Boltzmann weighting. In this method, the fitness is given by $F=exp(-E/\Delta E)$ where $\Delta E=max(E)-min(E)$. The linear scaling function and the Boltzmann weighting are available in RMD_GA.

Armed with the fitness values for each member in the population we can determine which members survive into the next generation. This is done using weighted roulette selection. The roulette wheel is divided up into N_{pop} regions and has total unit area. Each region on the wheel is given an area proportional to its fitness probability which is defined as the fitness of an individual member normalized to the sum fitness of all members: $A_i = f_i / \Sigma_i f_i$. The wheel is spun N_{pop} times and the results of each spin determine which member survives to the next generation. Note that it is entirely possible for a single chromosome to "survive" multiple times.

The next step is called crossover. In this operation, two chromosomes are selected randomly without replacement. Next a random position is selected among the $N_{parms} \times N_{bits}$. Both chromosomes are cut at that random position and the "tails" of each are swapped. As a simple example consider the two parent

chromosomes 1001 and 1100. If the crossing point is the second position then the resulting children are 1000 and 1101. This is not performed for all chromosomes. Rather it is performed on average $p_{cross} \times N_{pop}$ times, where p_{cross} is the probability of crossing two chromosomes. You specify this probability.

The final genetic operation performed on the population is mutation and it is performed $p_{mutate} \times N_{pop}$ times. Where p_{mutate} is the mutation probability. In this operation each of the members so selected have one of their bits changed (0 to 1 or vice versa) at random.

This process is repeated until one of two things occurs. The first could be that some convergence criteria has been attained. Since the population fitness is tracked during evolution then one possible convergence criteria is that the average fitness of the population comes within some limit of the best fit so far. Alternatively you could specify some maximum number of iterations. Note that it is usually sensible to keep the best fit population member present throughout the evolutionary process.

Before we get into the details of how to call RMD_GA so that you can minimize your own functions we will first consider a simple application of the SGA so that you can see how it works and responds to changes in p_{cross}, p_{mutate}, and N_{pop}.

A Simple Example: The Ising Model

As an introduction to the SGA and to get some intuition of the technique, we will consider a simple application of the SGA to a basic example from statistical mechanics. The one-dimensional Ising model is a representation of magnetic spins whose ground state energy function has many minima. Therefore this is a nice test of the search capabilities of the SGA[2].

In this model N spins are coupled via nearest-neighbor interactions. The spins S can take on values of ±1 and are coupled with an interaction parameter (coupling constant) J. The energy of the entire spin configuration is given by $E=-\Sigma_i J_i S_i S_{i+1}$ where the sum is over the (N-1) spin pairs. The ground state energy is known theoretically: $E_{min}=-\Sigma_i |J_i|$. In a spin chain composed of N spins, there are 2^N possible configurations. For example, a 30-spin chain has over 1 billion possible configurations. Therefore it is not surprising that simple gradient-descent methods might fail to determine the minimum energy configuration of this system.

When the coupling between adjacent spins is +1 then the ground state energy is $E_{min}=-(N-1)$, corresponding to an all parallel spin configuration. This situation is implemented in the program called RMD_GA_ISING. Let's jump right in and

see what happens when we try to find the ground state energy using the SGA. To start, type the following:

```
IDL> RMD_GA_ISING,10,npop = 60
```

Two graphics windows will appear. The top one shows the current best spin configuration with couplings between spins and the bottom one shows the dynamically-evolving average population fitness. These will both change as the evolution proceeds. Since this is a stochastic algorithm your final results could differ from mine, especially the final spin configuration. However when I ran this I got the graphical output shown in figure 1. Your run may have found one of the two ground state spin configurations. I found the one in which the spins are all -1 whereas the other minimum energy configuration contains spins of all +1. The display would be all black for the spin configuration of all -1.

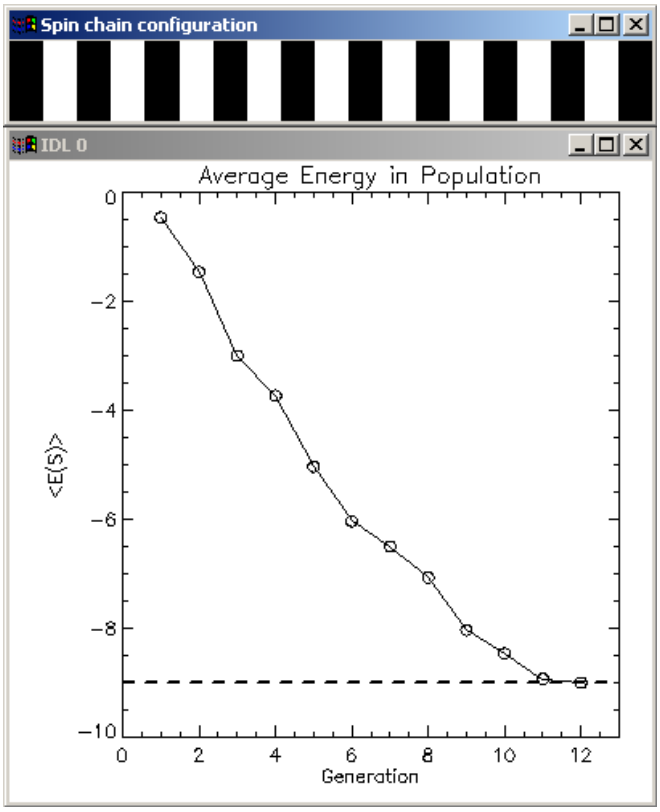


Figure 1 Graphical output resulting from running the SGA on the one-dimensional Ising model with 10 spins. In the top plot the alternating bands are the spins and bonds respectively for the final configuration. All of the bonds are +1 (white) and all of the spins are -1 (black). The bottom plot shows the average system energy as a function of generation. This is one of the two ground state spin configurations.

The text output stating the results were as follows:

```
Minimum energy (GA): -9.00000
Minimum energy (theory): -9.00000

Number of possible configurations: 1.02e+003
Number of function calls: 780

Percentage probed: 76.1718
```

This is not a particularly large set of spins so the number of possible configurations is not that large (2^{10} = 1024). Nevertheless (in this particular case) the SGA found the minimum energy configuration without needing to resort to an exhaustive search of the entire configuration space.

Next try a tougher problem. For 30 spins there are 2^{30} (more than 1 billion) possible configurations. Type the following:

```
IDL> RMD_GA_ISING,30,npop = 150
```

In this case I got the graphical output shown in figure 2.

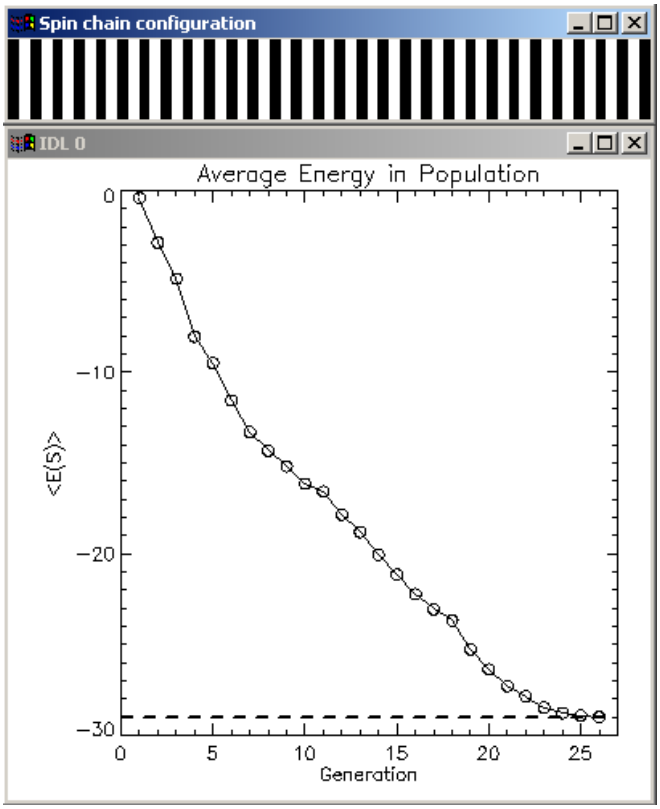


Figure 2 Same as figure 1 except for a 30 spin chain and 150 members in the population.

The text output from this run was:

```
Minimum energy (GA): -29.0000
Minimum energy (theory): -29.0000
```

```
Number of possible configurations: 1.07e+009
Number of function calls: 4050

Percentage probed: 0.000377185
```

Clearly the SGA has performed admirably in search a space of over 1 billion possible configurations and found the global minimum with only 4050 function calls. This amounts to less than 4/10000 of a percent of the possible configurations probed. Your results will differ and the global minimum might not even be found.

One point to notice from these two runs is that the efficiency clearly improves as the search space gets larger. Keeping in mind that your results may differ from mine, for a small (10-dimensional) search space we needed to evaluate the function 780 times out of 1024 possible resulting in a coverage of 76%. When the dimensionality grew to 30 the coverage reduced to less than 0.0004%. The number of members in the population for the 10-spin Ising chain was 60 whereas 150 members made up the population for the 30-spin chain. In general it is necessary to increase the number of members of the population as the dimensionality of the problem increases. This is a problem-specific dependency and there are no general rules-of-thumb of which I am aware.

You can study the effects of the various GA tuning parameters such as p_{cross}, p_{mutate}, and N_{pop} calling the procedure RMD_GA_ISING from the command line as follows:

```
IDL> RMD_GA_ISING, 60, npop = 150, pcross = 0.95,pmutate = 0.1
```

All keywords and the single parameter (N_{spins}) are optional. The defaults are $N_{spins} = 20$, $N_{pop} = 189$, $p_{cross} = 0.9$, and $p_{mutate} = 0.033$ and are used when those particular parameters are not specified. If you do not specify N_{pop} then the default value is INT($N_{spins}^{1.75}$) which was determined to be adequate using empirical means (i.e. playing around).

In addition to studying the standard Ising model in which the bonds are either +1 or -1, we can randomly assign the bond values from a Gaussian distribution whose mean is 0 and standard deviation is 0.25. This is a simple model of a spin glass[2] (a glass because it has no long-range order) and can be invoked by setting the RANDOM keyword in the call to RMD_GA_ISING:

```
IDL> RMD_GA_ISING,/random
```

The result of one run is shown in figure 3. A blue color scale was used in the display of the spins and bonds here. Because of the color scale you can clearly see that there are random bond values between spins (various shades of blue between white and black spins).

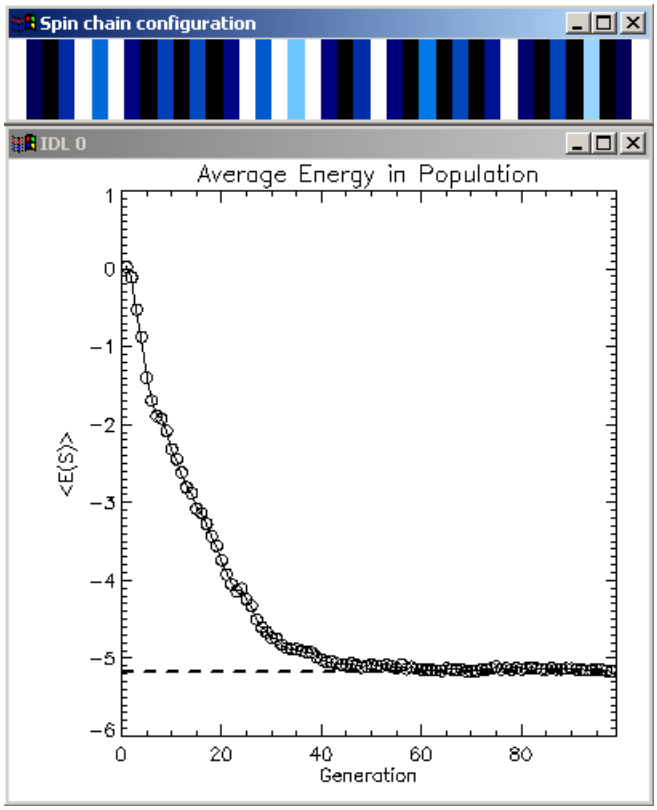


Figure 3 Graphical output for a 20-spin Ising glass in which the bonds are normally distributed about zero with a standard deviation of 0.25. The blue color scale illustrates the random bond values.

The textual output for the run shown in fig. 3 is the following:

```
Minimum energy (GA): -5.17190
Minimum energy (theory): -5.17190

Number of possible configurations: 1.05e+006
Number of function calls: 18810

Percentage probed: 1.79386
```

Function Syntax: RMD_GA

In this section we will discuss the syntax for the function RMD_GA and see how to apply it to minimization of a function of two independent variables. Moreover we will show how to implement a procedure that illustrates the progress of the algorithm at each generation.

The SGA is implemented with an object class contained in the procedure RMD_GA__DEFINE and the function RMD_GA. A typical user of the SGA would usually choose to use RMD_GA which is a function "wrapper" for the object. Therefore we will discuss the function RMD_GA at length and only briefly discuss the object class where necessary.

The function RMD_GA begins as follows:

```
function rmd_ga,  ftol,                                    $
                  function_value = function_value,         $
                  function_name = function_name,           $
                  prange = prange,                         $
                  ncalls = ncalls,                         $
                  quiet = quiet,                           $
                  pcross = pcross,                         $
                  pmutate = pmutate,                       $
                  boltzmann = boltzmann,                   $
                  itmax = itmax,                           $
                  objective_function = ofun,               $
                  gene_length = gene_length,               $
                  stretch_factor = stretch_factor,         $
                  npop = npop,                             $
                  functargs = ft
```

All of the parameters and keywords are optional (sensible defaults are used in their absence) except for `function_name` and PRANGE. The first is required because the SGA requires some/any function to optimize. In the default mode (i.e. not specifying OBJECTIVE_FUNCTION, RMD_GA will minimize the function provided by the user: FUNCTION_NAME. PRANGE is required because the function needs to know the range over which to perform the search.

The *return value* of this function is an array containing the best parameters found that minimize the function FUNCTION_NAME.

The only *parameter* passed into the function is FTOL. This parameter establishes a stopping criterion in terms of the change in the average fitness relative to the maximum fitness value. During the population evolution the average fitness value of the population will usually tend to some constant value as all of the members converge to a similar solution. This convergence is sensitive to the values of PCROSS, PMUTATE, NPOP, etc. so this must be taken into consideration when deciding on the value for FTOL. Note also that if convergence is not reached then the evolution will cease when ITMAX generations are completed (ITMAX default is 10). The default value for FTOL is 0.1.

The function to be minimized is specified as a keyword by the string FUNCTION_NAME. The requirements on the function to be minimized are that it be written as a FUNCTION and it must accept a parameter vector P and keywords passed in via the EXTRA keyword inheritance mechanism. An

example of a function to be minimized is shown below. Keywords are passed into the function using FUNCTARGS described below.

```
function test_rmd_fun,p,_Extra = extra
y = 4.0+sin(4.*p)*exp(-0.5*((p-2.)/4.0)^2)
return,y
end
```

The default optimization is minimization in RMD_GA. The user can keep the default implementation of the fitness function, specified by RMD_GA_OBJ_FUN, which is simply $F = E_{max} - E + 1$ where E is the function evaluation and E_{max} is the maximum function evaluation in a particular generation. This is a very simple linear fitness mapping that can be robust in many circumstances[3]. You can also specify your own objective function by specifying the string OBJECTIVE_FUNCTION. The only requirements on this function are that (1) it convert a maximization problem into a minimization one and (2) keywords must be allowed via the _EXTRA keyword inheritance mechanism. Finally, if you wish to replace the default objective function with a Boltzmann distribution then you can set the BOLTZMANN keyword. The fitness mapping becomes $F = \exp(-E/(E_{max}-E_{min}))$.

The keyword FUNCTION_VALUE is an output keyword that returns the function evaluated at the best fit parameter set (i.e. the return value of the function).

The range of the function parameters are given by the input keyword PRANGE. This is a a 2 x N vector defining the lower and upper bounds for the N parameters to be refined in the minimization.

The keyword FUNCTARGS is expected to be a structure that contains any exogenous information necessary to evaluate the function to be minimized. This is a convenience for the user and is not required if no exogenous information is required.

If the user sets the QUIET keyword then there will be no intermediate output. However if QUIET is unset (0B) then intermediate output will be provided. If ITERPROC is not specified and QUIET is unset then the best parameters found at each generation will be printed to the IDL output log. The default is for QUIET = 1B (i.e. intermediate output is suppressed). Alternatively you can create your own intermediate reporting procedure and set the ITERPROC keyword to the string name of the procedure. You must also specify ITERARGS which contains additional parameters that can be useful for reporting. The requirements on the intermediate reporting procedure are that it be written as a PROCEDURE and it must accept the following arguments: FUNC, P (the current best parameter set), ITER (the current iteration, and INTERRUPT (a byte variable equal 1B or 0B indicating if the algorithm should cease at the next update--note that interrupt must be a variable since it is expected to be passed by reference). Optional keywords are FUNCTARGS, ITERARGS, and

OREF. OREF is provided so that you can extract information on the GA object (its state) at the current iteration. This can be useful if you want to get all of the current members of the population and display them, for instance.

The keyword ITERARGS is optional and, if present, is expected to be a structure containing exogenous information necessary for performing the intermediate reporting procedure, ITERPROC. ITERARGS is passed into ITERPROC via the _EXTRA keyword inheritance mechanism.

The input keywords PCROSS and PMUTATE specify the crossover and mutation probabilities respectively. The default values for these are 0.9 and 0.01 respectively.

The input keyword GENE_LENGTH is the length in bits of each member of the population. The more bits the better the numerical precision of the result (default: 5).

The input keyword STRETCH_FACTOR specifies the amount by which to "stretch" the fitness function. The default value is 1.0.

The output keyword NCALLS returns the number of function evaluations performed over the course of the evolution. This is very useful for determining the efficiency of this search technique.

The input keyword ITMAX specifies the maximum number of generations to proceed in the evolution. The default is 10.

NPOP is an input keyword that specifies the total number of members in the population. The default value is 20.

The next section provides a detailed example illustrating the function.

Bivariate Function Minimization Example

In this final section we show how to use RMD_GA to minimize a function of two variables. The source code for this example is found at the bottom of the program listing for RMD_GA.PRO. The source code for this minimization problem is listed in its entirety in the appendix for this document. However we will reproduce parts of the code to illustrate usage. The requirements for this problem are two-fold. First, the function to minimize is in TEST_FUNC1 and is represented mathematically as

$$z = 3(1-x)^2 \exp(-(x^2 + (y+1)^2)) - 10(\frac{x}{5} - x^3 - y^5)\exp(-x^2 - y^2) . - \frac{1}{3}\exp(-(y^2 + (x+1)^2))$$

This function has two minima but the global minimum over the range $x \in [-9,9]$ and $y \in [-9,9]$ is (0.228,-1.626). The second requirement is that the average function evaluation of the population must be displayed dynamically as the evolution proceeds. This is a relatively straightforward optimization problem in terms of the requirements stated above.

The function as defined over this interval is displayed in figure 4.

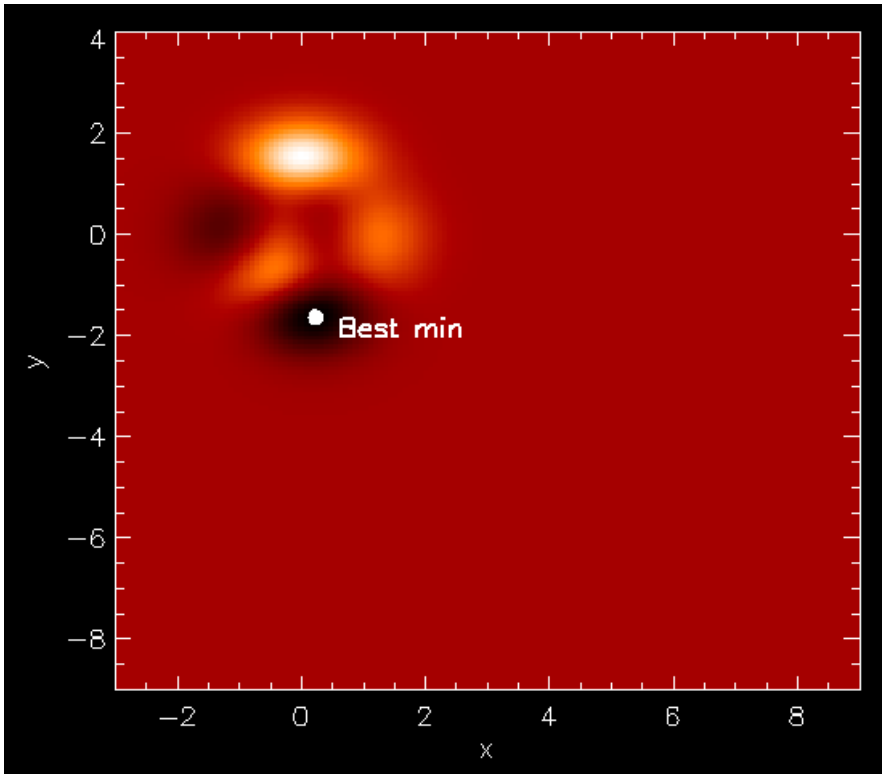


Figure 4 Image display of the bivariate multi-modal function to be minimized using RMD_GA. The global minimum is shown as the white circle marked "Best min".

The main "driver" program is called TEST_RMD_GA. This procedure sets up the parameter ranges, specifies the function to minimize, specifies the intermediate reporting procedure, and creates the window for displaying the average function evaluation with each successive generation.

The range for the search, $x \in [-9,9]$ and $y \in [-9,9]$, is specified as a 2 by 2 array PRANGE = [[-9.0,9.0],[-9.0,9.0]]. The objective function is specified as 'my_obj_fun' which replaces a simple linear scaling with a linear scaling that is weighted by generation: $F = (ITER+1) \bullet (E_{max}-E)$. This function was selected in the hopes that it would provide more efficient coverage and, hence, rapid convergence than simple linear scaling. Whether this is the case or not is left up to you to determine. The code is shown below.

```
function my_obj_fun,z,_Extra = extra
return,(extra.iter+1.) * (max(z)-z)
end
```

Since we wish to display the average function evaluation as a function of generation, we create a window (both visible and a pixmap) as a destination. This window will be updated within ITERPROC. We add both the visible and pixmap window IDs to ITERARGS, the structure of exogenous information that is passed into ITERPROC. The code snippet that accomplishes this is given below.

```
quiet = 0B
if ~quiet then begin
   xsize = 400 & ysize = 400
   window,0,xsize = xsize,ysize = ysize
   winvis = 0
   window,/free,/pixmap,xsize = xsize,ysize = ysize
   winpix = !d.window
   iterargs = {winvis:winvis,winpix:winpix}
   iterproc = 'test_ga_iterproc'
endif
```

The intermediate reporting procedure, TEST_GA_ITERPROC, is responsible for only one thing: it must display the average fitness. As shown in the procedure reproduced below this is accomplished by getting the current average fitness via the get_property method of the GA object reference, OGA. Actually the average fitness is just the population average of the function. This is an example where it is particularly helpful to have the object reference available so that current information can be obtained on the algorithm. After obtaining the average fitness from the object the only other thing that this procedure does is display the current (and preceding) average fitnesses in the pixmap and copy it over to the visible window (for smooth animation).

```
pro test_ga_iterproc,func,                  $
                     p,                     $
                     iter,                  $
                     interrupt,             $
                     functargs = functargs, $
                     oref = oga,            $
                     _Extra = extra
compile_opt hidden,idl2
oga->get_property,ave_fitness = ave_fitness
x = 1+indgen(iter+1)
y = ave_fitness[0:iter]
wset,extra.winpix
plot,[x],[y],psym = -4,title = 'Function evaluation',xtitle = 'Generation', $
   ytitle = '<F(p)>'
wset,extra.winvis
device,copy = [0,0,!d.x_size,!d.y_size,0,0,extra.winpix]
end
```

We wish to change the defaults of the GA for our own purposes so we call RMD_GA as follows.

```
ftol = 1.e-2
p = rmd_ga(       ftol,                                $
                  function_value = function_value,     $
                  function_name = func,                $
                  prange = prange,                     $
                  ncalls = ncalls,                     $
                  quiet = quiet,                       $
                  objective_function = ofun,           $
                  pcross = 0.95,                       $
                  gene_length = 25,                    $
                  pmutate = 0.01,                      $
                  itmax = 100,                         $
                  iterproc = iterproc,                 $
                  iterargs = iterargs,                 $
                  npop = 250)
```

We are interested in a smaller tolerance (i.e. more complete convergence of the population) so the fitness tolerance is 0.01. We also seek an answer with greater precision so the bit length (gene_length) of each parameter is set to 25. The crossover and mutation probabilities are set to 0.95 and 0.01 respectively. The maximum number of iterations is set to 100. Remember that the evolution will only proceed this long if the fractional fitness (average fitness/best fitness) is less than 0.01. Finally we override the default population of 20 and set it to 250.

The program is run by typing TEST_RMD_GA at the command line.

```
IDL> TEST_RMD_GA
```

The results I obtained when I ran it are shown in figure 5. In this case the global minimum was found.

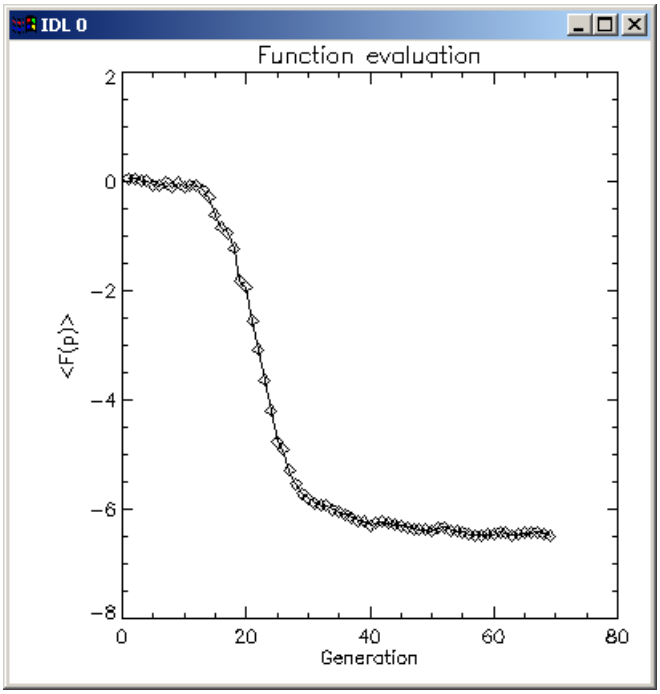


Figure 5 Graphical output for the bivariate function minimization executed in TEST_RMD_GA. This plot represents the average function evaluation as a function of generation and illustrates the convergence of the entire population to the global minimum.

Source Code Listing: TEST_RMD_GA

```
; ********************************** ;
function test_func_1,p,_EXTRA = extra
x = p[0] & y = p[1]
z = 3.*(1.-x)^2 * exp(-x^2 - (y+1.)^2) - $
    10.*(x/5 - x^3 - y^5)*exp(-x^2-y^2) - $
    (1./3)*exp(-(x+1)^2 - y^2)
return,z
end
; ********************************** ;
pro test_ga_iterproc,func,                  $
                     p,                     $
                     iter,                  $
                     interrupt,             $
                     functargs = functargs, $
                     oref = oga,            $
                     _Extra = extra
compile_opt hidden,idl2
oga->get_property,ave_fitness = ave_fitness
x = 1+indgen(iter+1)
y = ave_fitness[0:iter]
wset,extra.winpix
plot,[x],[y],psym = -4,title = 'Function evaluation',xtitle = 'Generation', $
   ytitle = '<F(p)>'
wset,extra.winvis
device,copy = [0,0,!d.x_size,!d.y_size,0,0,extra.winpix]
end
; ********************************** ;
function my_obj_fun,z,_Extra = extra
return,(extra.iter+1.) * (max(z)-z)
end
; ********************************** ;
pro test_rmd_ga
; Example implementation of the SGA to minimize a function of
; two variables.  Here the range of the parameters is defined
; as follows:
;  -9.<P[0]<9. and -9.<P[1]<9.
; The actual global maximum (within the specified range) is
; (x,y) = (0.228,-1.626)

prange = [[-9.0,9.0],[-9.,9.]]
ofun = 'my_obj_fun'
func = 'test_func_1'
quiet = 0B

if ~quiet then begin
   xsize = 400 & ysize = 400
   window,0,xsize = xsize,ysize = ysize
   winvis = 0
   window,/free,/pixmap,xsize = xsize,ysize = ysize
   winpix = !d.window
   iterargs = {winvis:winvis,winpix:winpix}
   iterproc = 'test_ga_iterproc'
endif
ftol = 1.e-2
p = rmd_ga(       ftol,                                 $
                  function_value = function_value,      $
                  function_name = func,                 $
                  prange = prange,                      $
                  ncalls = ncalls,                      $
                  quiet = quiet,                        $
                  objective_function = ofun,            $
                  pcross = 0.95,                        $
                  gene_length = 25,                     $
                  pmutate = 0.01,                       $
                  itmax = 100,                          $
                  iterproc = iterproc,                  $
                  iterargs = iterargs,                  $
                  npop = 250)

if ~quiet then wdelete,winpix
this_format = '(f15.3)'
print,'Best parameters: '+strtrim(string(p[0],format = this_format),2)+ $
      ','+strtrim(string(p[1],format = this_format),2)
print,'Function value: '+strtrim(string(function_value,format = this_format),2)
```

```
end
```