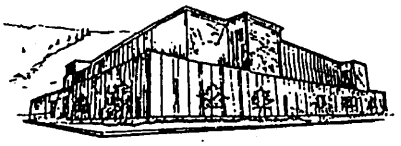
2004

Breeder algorithm for stellarator optimization

Shengping Wang
The University of Montana



Maureen and Mike
MANSFIELD LIBRARY

The University of
Montana

A Breeder Algorithm for Stellarator Optimization

By

Shengping Wang

B.S. in Mathematic Science, Hunan Normal University, China, 1988

Presented in partial fulfillment of the requirements

For the degree of

Master of Science

The University of Montana

December 2004

Directed by

Dr. Andrew S. Ware

Department of Physics and Astronomy

Approved by:

Chairman

Dean, Graduate School

12-16-04

Date

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Wang, Shengping, M.S., December 2004 Computer Science

A Breeder Algorithm for Stellarator Optimization

Committee Chair: Dr. Joel Henry

A comparison of Levenberg-Marquardt and Genetic optimization algorithms is presented and a hybrid optimization algorithm which combines these two is developed. A number of different optimization algorithms have been applied to optimization in various physical areas including both steepest descent and evolutionary optimization algorithms. Each algorithm has both advantages and disadvantages. This paper provides a comparison of optimization between the Levenberg-Marquardt routine, a steepest descent optimization method, and the Genetic Algorithm, a global evolutionary optimization algorithm, in an applied plasma physics area. To take the advantages of both steepest descent optimization and evolutionary optimization, the Breeder Algorithm which combines the Genetic Algorithm with the Levenberg-Marquardt algorithm is introduced. A description of the design and structure of the Breeder Algorithm is presented, as well as the code reviews which present the strategy and challenges of the implementation.

Contents

List of Illustrations

1. Introduction

A number of different optimization algorithms have been applied to stellarator optimization including both steepest descent and evolutionary optimization algorithms. A stellarator is a device used to confine a high-temperature plasma with magnetic fields with the long-term goal of sustaining a controlled nuclear fusion reaction. In a stellarator, the magnetic field necessary to confine the plasma is completely generated by external coils. The first such devices were built at the Princeton Plasma Physics Laboratory in 1951. In recent years, the design of stellarator experiments has been aided by extensive use of computational algorithms used to optimize confinement and stability properties of the magnetic configuration. In this paper, we present two major works: (1) a comparison of two different algorithms applied to the optimization of three-dimensional equilibria: a Levenberg-Marquardt routine (LM), and a Genetic Algorithm (GA); and (2) the development of a new optimization algorithm, the Breeder Algorithm (BA), which combines the methods of GA and LM.

The speed and efficiency of numerical codes used to calculate equilibrium, stability, and transport properties of three-dimensional plasmas has been sufficiently enhanced so that global (in parameter space) optimization methods are now feasible. In principle, the primary advantage of evolutionary algorithms such as GA is that they perform a global parameter space search. This is in contrast to steepest descent methods which perform a local parameter space search for the optimal configuration. LM, for example is prone to finding local extremum. The primary disadvantage of evolutionary algorithms is that they can be inefficient when compared to steepest descent algorithms. In the first part of this work, we present a comparison of the GA and LM methods when applied to stellarator optimization.

The Breeder Algorithm combines a global Genetic Algorithm with a local Levenberg-Marquardt optimizer used to refine each generation. The goal of the BA algorithm is to take advantage of the global parameter space search of the evolutionary algorithm while maintaining the efficiency of the LM method. Here, we present a description of the Breeder Algorithm and the first results from the application of the BA to stellarator optimization.

2. The Existing Search Algorithms

In this section we give brief descriptions of the two optimization algorithms most commonly used for stellarator optimization. An example stellarator optimization case is presented and the optimization results for both algorithms on this case are discussed.

2.1. The Levenberg-Marquardt Algorithm

2.1.1. A brief introduction to the Levenberg-Marquardt algorithm

The problem for which the LM was developed is called *nonlinear least squares minimization*. Suppose that we have a function: $f(\bar{x}) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(\bar{x})$, where $\bar{x} = (x_1, x_2, ...x_n)$ is a vector, and each χ^2 is a function from R^n to R. The χ^2 are referred to as a residual and it is assumed that $m \geq n$. Also, f can be represented as a residual vector r: R^n to R^m defined by $r(x) = (r_1(x), r_2(x), ..., r_m(x))$. Therefore, f can be written as $f(x) = \frac{1}{2} \| r(x) \|^2$. The derivatives of f can be written using the Jacobian matrix J of r with respect to x defined as $J_{ij} = \frac{\partial r_j}{\partial x_i}, 1 \leq j \leq m, 1 \leq i \leq n$.

The LM is a blend of gradient decent and Gauss-Newton iteration. Vanilla gradient descent is the simplest technique to find minima in a function. Parameter updating is performed by adding the negative of the scaled gradient at each step, $x_{i+1} = x_i + \lambda \nabla f$. There are some disadvantages using pure gradient descent updating. One well known is when the given function is not differentialable, pure gradient descent won't work properly.

Using a Tayler series, there is an update rule: $x_{i+1} = x_i - (\nabla^2 f(x_i))^{-1} \nabla f(x_i)$. Based on that, and Levenberg's update rule: $x_{i+1} = x_i - (H + \lambda I)^{-1} \nabla f(x_i)$, Marquardt introduced an improved updating formula in 1963: $x_{i+1} = x_i - (H + \lambda diag[H])^{-1} \nabla f(x_i)$. Where H is the Hessian Matrix, $H_{ij} = \frac{1}{2} \frac{\partial^2 x^2}{\partial \alpha_i \partial \alpha_j}$ [1].

To summarize, LM is an iterative method to minimize the sum of squares χ^2 ("Chi-Square") of M functions in N variables. LM requires the finite-difference approximation of the Jacobian matrix in each iteration. LM uses the Jacobian to minimize χ^2 in a local region of parameter space.

2.1.2. A typical LM search example

In this section we discuss a typical stellarator optimization case and the results of an LM optimization on the case. This plasma equilibrium has a three-dimensional fixed boundary determined by the Fourier coefficients, $R_{bc}(m,n)$ and $Z_{bc}(m,n)$, where m and n represent the poloidal and toroidal mode numbers, respectively. A schematic of the cylindrical coordinates (R,ϕ,Z) and the toroidal coordinates $(\rho,\theta,\zeta)$is shown in Figure 1.

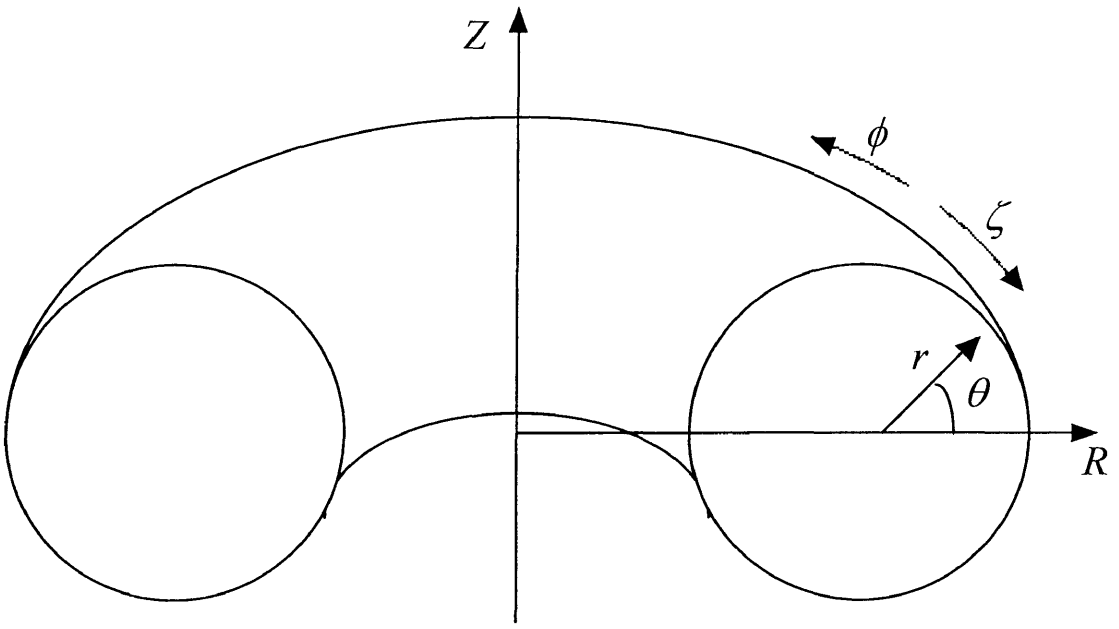


Figure 1. A schematic of cylindrical (R,ϕ,Z) and the toroidal (ρ,θ,ζ) coordinates.

The outer boundary is given by

$$R(\theta,\zeta) = \sum_{m,n} R_{bc}(m,n)\cos(m\theta - n\zeta)$$
$$Z(\theta,\zeta) = \sum_{m,n} Z_{bc}(m,n)\sin(m\theta - n\zeta)$$

where a symmetry called stellarator symmetry has been assumed. These boundary coefficients are some of the independent variables, *m and n*, which LM can vary to improve the plasma properties as discussed below. The strength of the magnetic field varies throughout the plasma including on the outer surface. Figure 2 shows the outer plasma boundary for one of the test cases we used which is called QPS15. The color on the surface indicates the strength of the magnetic field at that point. Some of the basic plasma parameters of QPS15 are as follows. The average magnetic field strength is 1.0 T. The average plasma pressure is measured relative to the magnetic field strength squared

and is called the plasma beta, $\beta = \langle p/B^2 \rangle$ and for the QPS15 plasma, $\beta = 15\%$. This is a very-high plasma beta relative to existing stellarator plasma experiments. The aspect ratio, A, is the ratio of the average major radius, R, to the average minor radius, a. For QPS15, A = 3.7. This is low relative to most three-dimensional confinement devices. This configuration has an electrical current running through the loop of plasma with a total current of 176 kA. This is in contrasting to most stellarators which have zero net plasma current.

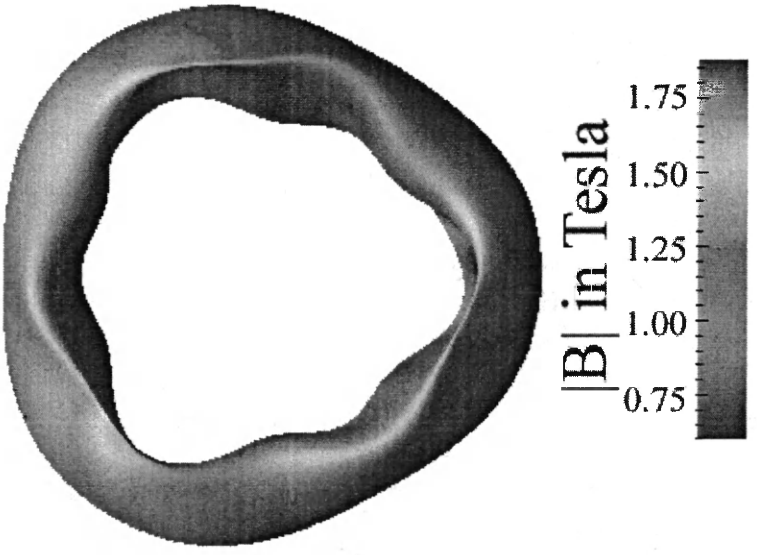


Figure 2. The outer plasma surface of the QPS15 test case.

The initial χ^2 was 1.20×10^6 and this was primarily due to the plasma being unstable to certain types of perturbations. A number of stability checks are part of the calculation for each case and a stable plasma is one of the targets. There were $N = 8$ independent variables which included the plasma boundary coefficients and coefficients describing the pressure and current profiles. The final χ^2 was 1.213×10^5 and this reduction was primarily due to improvement in the stability of the plasma.

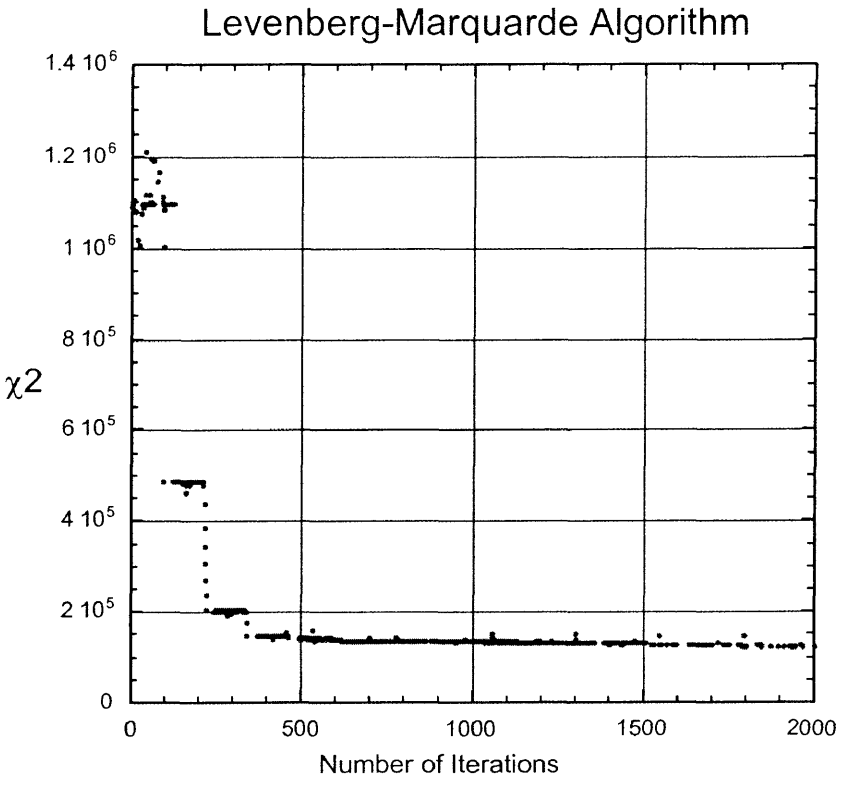


Figure 3. The evolution of the total χ^2 during an LM optimization.

Figure 3 shows the total χ^2 vs. the number of iterations in the optimization. In the first 100 iterations as the Jacobian was being determined, the average χ^2 remained around the initial χ^2 value, then between 100 to 105 iterations, the value of χ^2 drops significantly as the algorithm moves down the gradient in parameter space. The Jacobian is recalculated at the new position in parameter space during iterations 105-200 and then the algorithm again makes significant progress in reducing χ^2 as it moves down the new gradient. This process is repeated several times. After 400 iterations, χ^2 remains relatively constant as the optimizer ceases making any significant progress. This indicates

a local minimum in the χ^2 function has been obtained. This is typical for an LM optimization.

2.1.3. Advantages and disadvantages of the LM search algorithm

Since LM is a single-shot method which attempts to find the local fit-statistic minimum nearest to the starting point. Its principal advantage is that it uses information about the first derivative of the fit-statistic as a function of the parameter values to guess the location of the fit-statistic minimum. Thus this method works well (and fast) if the statistic surface is well-behaved. For the testing case shown above and other cases tested, initial results show that the LM algorithm is most effective in minimizing χ^2 after ~500 iterations

The principal disadvantages of LM are that it will not work as well with pathological statistic surfaces, the first or second derivatives of the function of the surfaces do not exist, and there is no guarantee it will find the global fit-statistic minimum.

2.2. The Genetic Algorithm

2.2.1. A brief introduction to the Genetic Algorithm

Genetic algorithms (GA) were formally introduced in the 1970s by John Holland [2]. The continuing performance improvements of computational systems have made them attractive for some types of optimization. In particular, genetic algorithms work very well on mixed (continuous and discrete), combinatorial problems. The three most important aspects of using genetic algorithms are: (1) definition of the objective function, (2) definition and implementation of the genetic representation, and (3) definition and

implementation of the genetic operators. Once these three have been defined, the generic genetic algorithm often works well. Beyond that you can try many different variations to improve performance, find multiple optima or parallelize the algorithms.

Genetic algorithms are inspired by the theory of evolution. The solution method used by genetic algorithms is an evolutionary process. The GA begins with a set of randomly selected solutions (represented by chromosomes) called a population. Solutions from one population are taken and used to form a new population. These are generated by genetically-inspired operators, of which the most well known are *crossover* and *mutation*. Crossover is performed with probability p_{cross} (the "crossover probability") between two selected individuals, called *parents*, by exchanging parts of their genomes to form two new individuals, called *offspring*; in its simplest form, substrings are exchanged after a randomly selected crossover point. This operator tends to enable the evolutionary process to move toward "promising" regions of the search space. The mutation operator is introduced to prevent premature convergence to local optima by randomly sampling new points in the search space. It is carried out by flipping bits at random, with some (small) probability p_{mut} [3] This is motivated by a hope that members of the offspring will be better (i.e., have a lower χ^2) than the old one. A new population is formed from the parents and offspring with the lowest χ^2 values (also called their fitness) - the more suitable the cases are the more chances they have to reproduce.

This is repeated until some condition (for example number of generations or improvement of the best solution) is satisfied.

2.2.2. A typical GA search example

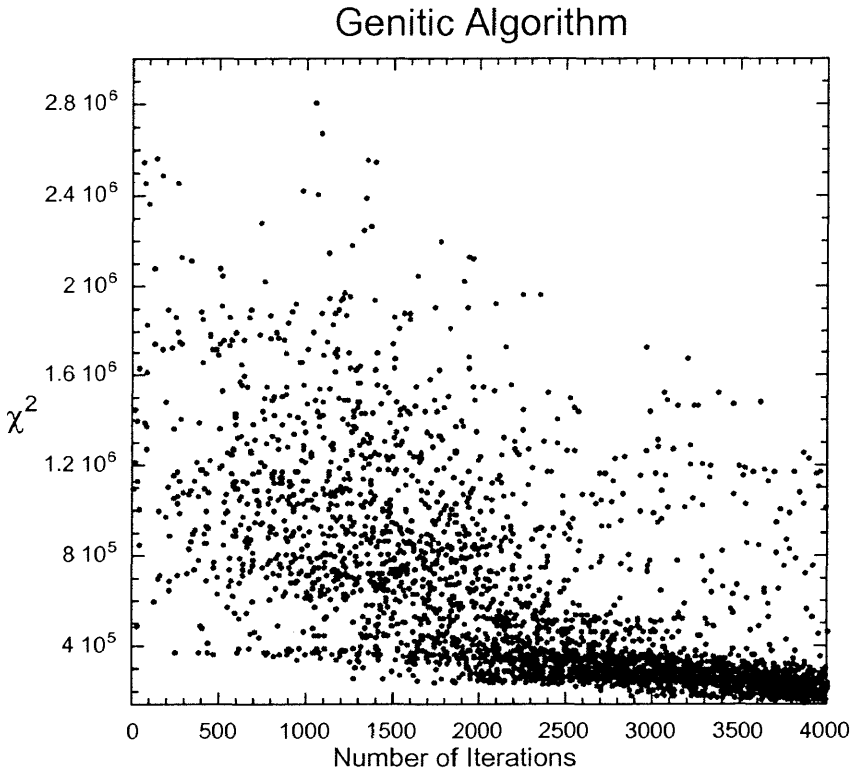


Figure 4. The evolution of the total χ^2 during an GA optimization.

In order to compare GA with LM, we used the same configuration of testing files as we used for LM algorithm, namely the QPS15 case. In order to make the parallel computation more efficient, we chose 48 as the population size because we use 3 nodes and each node contains 16 processors in IBM-SP environment. The generations are 12 to 15 because after 15th generation, the χ^2 won't have significant improvement from our experiment. The total χ^2 is shown as a function of the number of iterations in Figure 4. In the first thousand iteration χ^2 is scatted around the initial value with a wide range of values and the density of χ^2 points is relatively low. After this initial period, more and more χ^2 points are clustered in a high density area near a minimum χ^2 value. The number

of excursions far from the optimal value decreases. This behavior of χ^2 is typical for a GA search.

2.2.3. Advantages and disadvantages of the GA search algorithm

Since GA is a based on a set of randomly selected solutions, its principal advantages are: (1) GA is less susceptible to getting 'stuck' at local optima than gradient search methods; (2) It's faster than the simulation evaluation; (3) It's more deterministic in terms of the evaluation provided for a given candidate.

The principal disadvantages of GA are: (1) There are few jobs that GA can do better than a heuristic-based search (i.e., if you have some ideas of how to solve your problem, you're probably better off implementing those ideas than you are turning your problem over to a GA, which relies on randomness); (2) The Genetic Algorithm was effective only on certain cases. We have been tested 4 different cases for this project, GA can not work for one of them at all; and (3) global algorithms such as GA tend to be computationally expensive relative to steepest descent methods since the number of iterations for GA is relatively greater than that for LM.

3. The Breeder Algorithm

The Breeder Algorithm combines a global Genetic Algorithm with a local Levenberg-Marquardt optimizer used to refine each generation. For each individual in GA, BA applies LM to refine it for producing the next generation. The goal of the BA algorithm is to take advantage of the global parameter space search of the evolutionary algorithm while maintaining the efficiency of the LM method.

3.1. Design Description and Diagrams

The BA combines the global coverage of the GA with intermediate optimization steps using the LM routine. The primary structure of BA is:

(1) Initial generation production by the genetic algorithm (outer loop)

(2) Refinement of the members of this generation using a Levenberg-Marquardt step (inner loop)

(3) Evolving to future generations back in the genetic algorithm.

The Figures 5-7 show the primary structure of BA. Figure 5 indicates the existing structure of the Stellarator Optimization code. It takes an input file, runs through one of three existing optimizations, Levenberg-Marquardt, GA or DE (Differential Evolution algorithm, similar to GA but not discussed here), and produces a number of output files. Figure 6 illustrates the highest level BA structure which is a modification of the Figure 5. Generally, we add a new optional optimization, BA, to the existing ORNL structure (ORNL is the existing stellarator optimization code developed by scientists from Oak Ridge National Laboratory). The major BA structure is shown in Figure 7, as well a comparison to the GA structure. GALM_DRIVER is modified from GA_DRIVER. A

subroutine called SUB_OPTIMIZE which implements LM is inserted into GA_EVALOUT subroutine. This is the core part of BA.

3.1.1. Existing Structure:

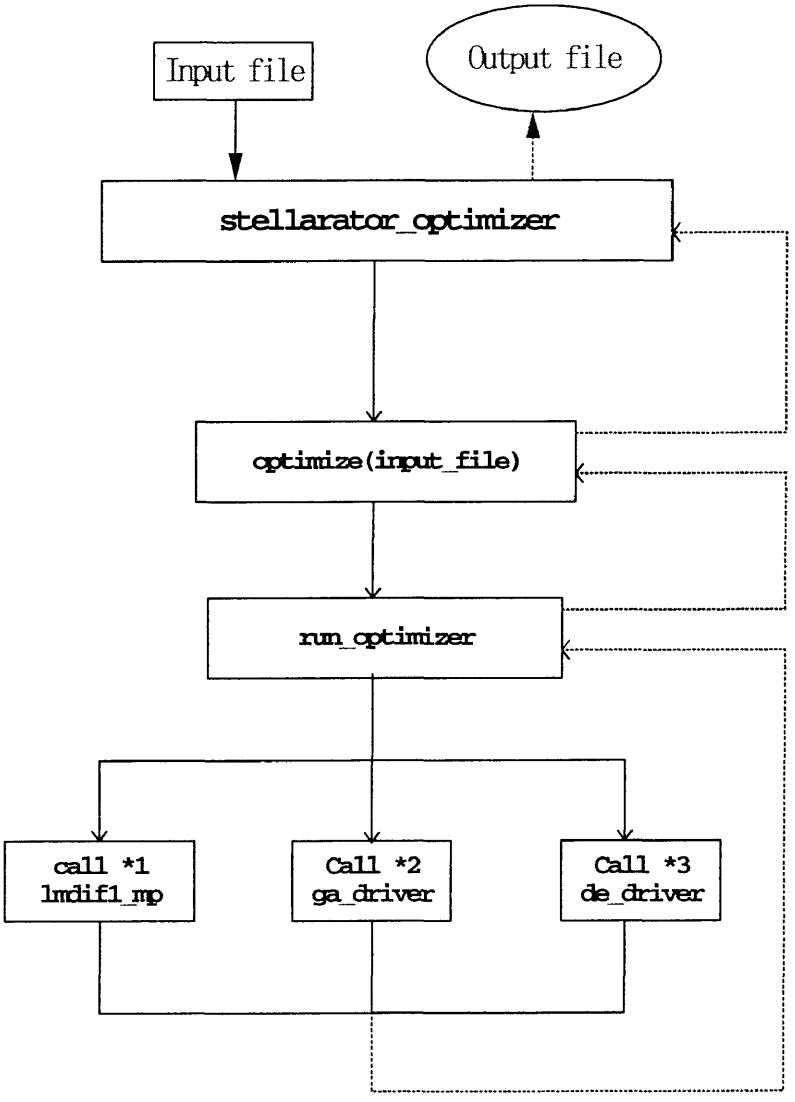
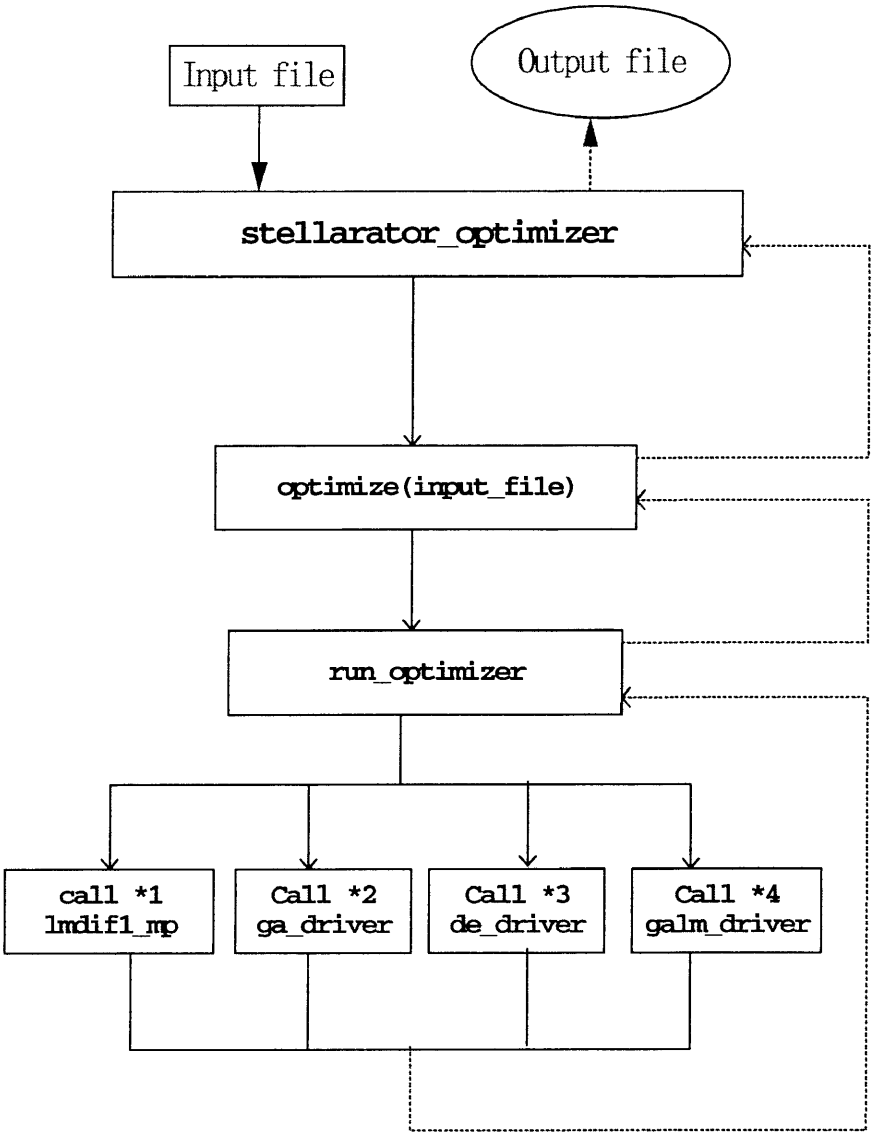


Figure 5. Existing structure of ORNL code

3.1.2. Modified Structure:



*1 Levenberg-Marquardt Method
*2 Genetic Evolution Algorithm
*3 Differential Evolution Algorithm
*4 Breeding Algorithm

Figure 6. Modified structure to ORNL code

3.1.3. BA modifies GA structure

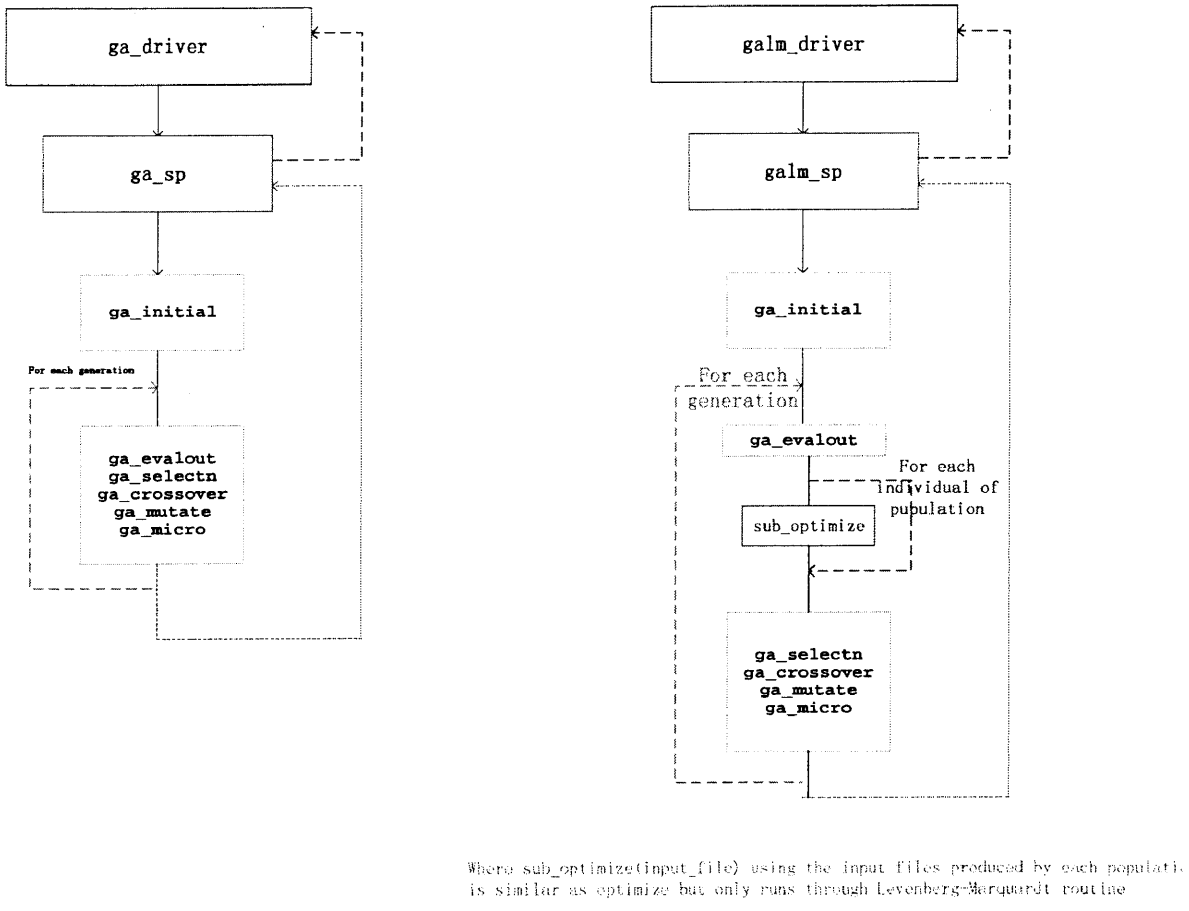


Figure 7. Comparison of GA and BA structures

3.2. Coding strategy for the BA

The original code of Stellarator Optimization was developed by Oak Ridge National Laboratory (ORNL). The Stellarator Optimization code consists of multiple packages to achieve various purposes including the calculation of magnetic fields produced by a set of coils, determination of a three-dimensional plasma equilibrium, assessment of the stability of the equilibrium, estimation of the transport properties of the plasma, and an overall optimization structure that modifies the input plasma properties to approach a user-defined goal of a stable, high-pressure plasma. The Breeder Algorithm is a modification to the structure of two of these packages called LIBSTELL and STELLOPT. The LIBSTELL library consists of subroutines used by many of the packages and includes the optimization algorithms. The STELLOPT code (this is short for Stellarator Optimizer) is the primary code which makes calls to the optimization subroutines contained in LIBSTELL and also makes system calls to execute other packages that calculate plasma properties such as equilibrium and stability.

In the STELLOPT package, we added a BA optimization option to the RUN_OPTIMIZER subroutine which implements the different optimizations using various approaches selected by the user, as well as adding a routine called RUN_SUB_OPTIMIZER which implements the LM routine when called by the GA_EVALOUT subroutine in the LIBSTELL package.

In the LIBSTELL package, we created a routine called GALM_DRIVER which performs the role of entry to BA. The most significant part of BA is the modification to the GA_EVALOUT subroutine. The primary purpose of the GA_EVALOUT subroutine is for GA to evaluate the population, assign fitness, establish the best individual, and

output essential information. The modification we made here is to insert an LM search for each individual of each population. The most challenging task in this portion of the project was making the "inner" LM work properly in parallel computation environment handling the working space including temporary working files, and correctly cleaning up memory space after each LM routine. Since both GA and LM share the same extensions of temporary files when those optimizations were implemented independently, once those two were combined together, it was necessary to develop a new system of shared extensions. An obvious example of a potential problem here is that after completing each LM step, the program will operate a cleanup process which removes most of the temporary files. In our modification, we must consider which temp files can be cleaned up and which ones must be kept since they hold the information for the future computations. To approach this goal, in addition to the existing temp filename extension for optimization, _opt extension, we introduced a new temp filename extension called _oga extension using for BA only. Therefore, after completing each LM routine for an individual, all reusable information will be copied to files with _oga extension from files with opt extension, and those files with opt extension will be removed in the clean up process. After an LM step for all individuals in a generation have been completed, all those files with either _opt extension or _oga extension will be cleaned up. The files with a _min extension that contain the minimum optimal case are saved.

3.3. Coding challenges for BA

The implementation environment of the Stellarator Optimization is an IBM SP. The language is Fortran90 using MPI libraries. In an average case, we use 3 nodes with 16 processors in each node.

As in most parallel programs, file operation is a big issue. In the entire optimization process, the program has to deal with hundreds of operations to input, output, and temporary files. When BA is added, the number of file operations increases rapidly. In the implementation of a parallel program, usually one processor performs the role or master while others perform the workers. Who, when, and how to perform file operations becomes very important. In the ORNL code, the primary coders designed a routine called SAFE_OPEN to deal with file open operation in the parallel environment which is the critical portion to the file operation. However, in some places, ORNL code still uses regular "open" system call for opening a file. It has been challenging to study code written by a physicist without professional commenting.

Memory allocation and re-allocation is another big issue. Since we merged existing LM and GA algorithms to make BA, many variables, especially global array variables, with the same name are used in both the outer loop GA and inner loop LM had to be taken care of carefully with allocation and re-allocation handled properly. One of the direct results with improperly memory allocation is the program crashes.

Another big issue in parallel programming is debugging. There is essentially no good parallel debugger forcing us to use the debugger TotalView with limited success. Due to the properties of parallel implementation, the error we located by using print/write statement is not the actual location where the error occurs. The debugging issue we are

still facing originally was considered to be an MPI call issue. After a considerable amount of effort, we determined it is a filename extension handling issue.

3.4. Code Review

3.4.1. Review 1: To handle BA temp filename properly.

Code Segment 1

Line	*Code*
0	*do 20 i=istart,maxgen+istart-1*
1	*!-----added by S.Wang ----------*
2	*!----Using fixed_seq_ext to store seq_ext at the first place----------*
3	*!----Since it is to be changed when calling LM in ga_evalout----------*
4	*if (i .eq. 1) then*
5	*fixed_seq_ext = seq_ext*
6	*end if*
	!----------
	! other code ... ...
55	*!--------added by S.Wang----------*
56	*!---getting original seq_ext back from fixed_seq_ext at end of*
56	*!---each generation ----------*
58	*seq_ext = fixed_seq_ext*
59	*!----------*

The code segment 1 shown above which is selected from the subroutine GALM_GRIVER illustrates the strategy that handles the BA temp filename properly.

Line 0 shows that *i* is the do-loop variable which represents the *i*th generation of population in the outer GA loop. Here we create a variable called fixed_seq_ext to store the initial filename extension (shown in line 5). Otherwise, LM and the inner loop of BA using the same variable seq_ext results in corruption of the value of seq_ext. This strategy avoids that error. Line 58 indicates that at the end of each generation, the program will send the initial filename extension value back to seq_ext from fixed_seq_ext.

3.4.2. Review 2: The core functionality of BA, to insert inner loop, LM routine, into existed GA routine.

Code Segment 2

```
Line    Code
0       IF (NOPT_ALG .eq. 3)THEN
1       !-- Making a temp directory to store _opt files as _oga files--------
2         if (myid .eq. master) then
3           temp = 'mkdir tempdir1'
4           call system(temp)
5         end if
6         DO k=1,npopsiz
7           write (char_npopsiz,'(i5)') k
8           myoga_ext = trim(fixed_seq_ext)//'_oga'//
9       1           trim(adjustl(char_npopsiz))
10          myopt_ext = trim(fixed_seq_ext)//'_opt'//
```

```
11        1              trim(adjustl(char_npopsiz))
12              input_file = 'input.'//myopt_ext
13
14        !------- cp opt file to oga and tempdir1/oga
15              if (myid .eq. master) then
16                tempd = 'tempdir1'
17        ! -----Copy input.myopt_ext into input.myoga_ext-
18                temp = 'cp '//input_file//' ''//'input.'//myoga_ext
19                call system(temp)
20        !---------Copy input.myopt_ext into tempdir1/input.myoga_ext
21                temp = 'cp '//input_file//' '//trim(tempd)//
22           1          '/input.'//myoga_ext
23                call system(temp)
24              end if
25
26              oga_ext = myoga_ext
27        !------run sub_optimize----------------------
28              call sub_optimize(myoga_ext)
29
30              if (myid .eq. master) then
31                open(3333)
32                read(3333,iostat=istat) funcval_breed
33                close(3333)
```

```
34                fitness(k) = funcval_breed
35              end if
36        !------Copy input.oga..min files produced by LM into super directory
37        !---replace input.myopt_ext
38              temp = "cp " //"stellopt_"//trim(seq_ext)//"/"//
39           1        "input."//trim(seq_ext)//'.min'//
40           2        " "//"input."//trim(myopt_ext)
41              print *, temp
42              call system(temp)
43        !-----------------------------------------
44        !--- need to do before continue GA routine:
45        !  1. replace input._optk by input...ogak.min
46             END DO
47           END IF ! end of (nopt_alg = 3)
```

The code segment 2 shown above is selected from the subroutine GA_EVALOUT. The initial purpose of this subroutine is to evaluate the population, assign fitness, establish the best individual, and output information for the GA routine.

Per GA's structure, it creates a temporary input files for each individual for each generation. The GA_EVALOUT subroutine then evaluates the population, assigns fitness, creates the best member, and outputs the information for the following calculation of next generation.

BA then chooses to insert the inner routine, LM, into GA by adding code segment 2 into GA_EVALOUT routine.

In line 0, nopt_alg = 3 indicates the BA algorithm.

In line 2 to line 5, program creates a temporary directory called tempdir1 which stores those temporary input files with _opt extension generated by GA. Also, the duplicated of those _opt-extended files are also created as _oga-extended files.

Line 6 is the do-loop entry for each individual to implement LM routine.

The purpose of line 8 to line 11 is to avoid the name extension problem in a future computation. Therefore, program creates two temporary variables, myopt_ext and myoga_ext.

After all files and directory are setup, line 28 calls the LM routine. Instead of calling LM driver directly, we built a routine, SUB_OPTIMIZE, in the STELLOPT package. SUB_OPTIMIZE provides similar functionality to the optimize routine but is simpler because it only needs to direct the program to the LM routine. Since SUB_OPTIMIZE is in STELLOPT package, we define it as an external function.

One purpose of the GA_EVALOUT routine to GA is to store fitness values to the fitness array. Since in BA we insert LM in GA_EVALOUT, the program needs to replace those old values in the fitness array with those new values generated by LM. Line 30 to 35 performs this function. There are two steps here. In the LM routine, the program writes those values into a temporary file called fort.3333, after returning from LM, the fitness array is populated with values from that file.

The last important feature of code segment 2 is to implement the replacement of those temporary input files with _opt extension with the input files generated by LM routine with .min extension. Line 38 to line 40 performs this functionality.

4. Comments to BA

4.1. Initial results of BA

Though there is a bug existing for the BA, we still observe the improvement of BA when implementing LM routine that refines each individual in GA. Figure 8 and 9 indicate this improvement to a test case called SIMPLE. This case, we only use few generations, as well as a small population size. It is obvious that each time the x2 is getting improved for each individual.

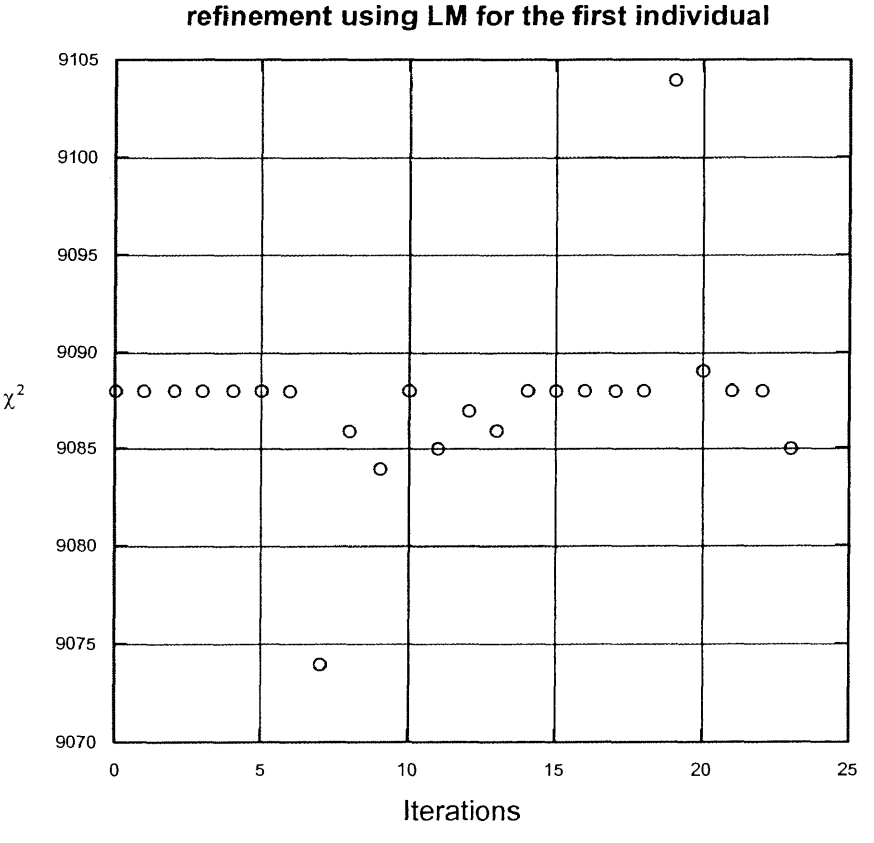


Figure 8. initial results of BA (1)

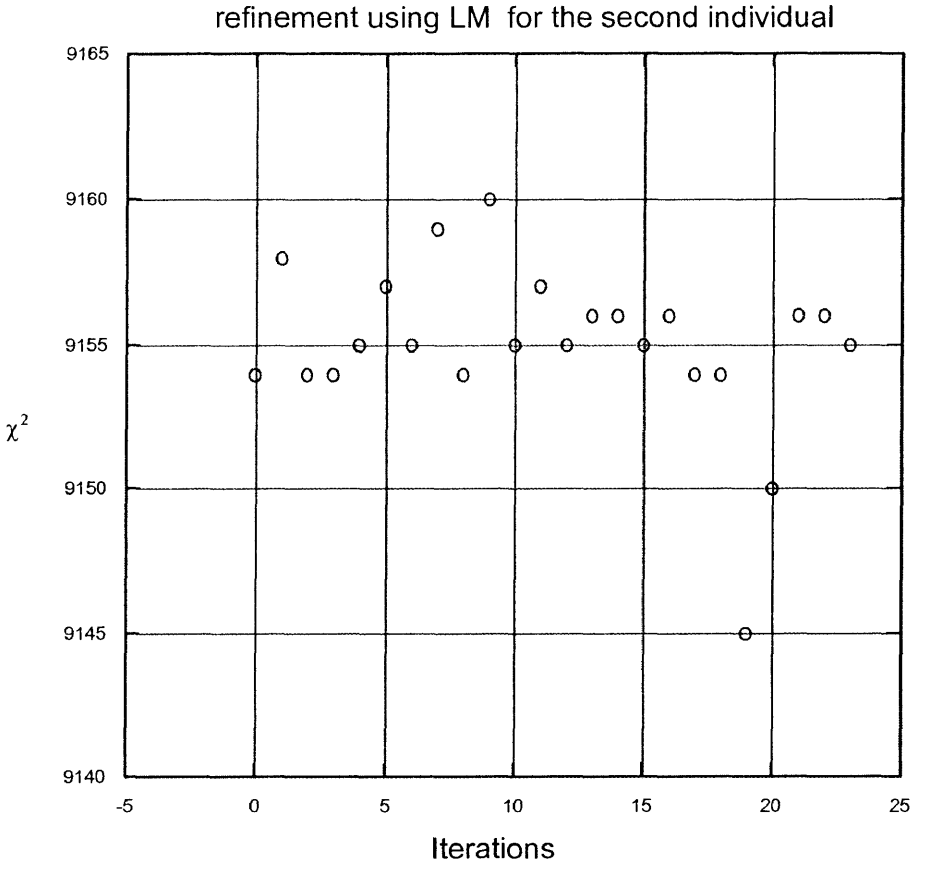


Figure 9. initial results of BA (2)

4.2. Advantages and disadvantages of BA search

Since BA combines GA with LM, the major advantages of BA are: (1) BA refines each individual in GA using LM's steepest descent search strategy which produces better offspring than only using pure GA. (2) BA avoids the local extremum to be considered as global extremum when LM encounters the surface of variables is not differentiable in first or/and second derivatives at some points (inflection points) because the evolutionary

optimization GA will produce some child points out of those inflection points which guarantees that BA can clime out of those local extremum points.

The major disadvantage of BA is that the implementation is more expensive at the computing time consuming point of view. When implementing BA under a parallel computation environment, it takes twice time as implementing LM or GA alone, roughly.

4.3. Existing issue and future work

The challenges remaining in the development of BA are to deal with the memory allocation and file IO in a parallel environment. When using Totalview as the debugging tool, the existing unmatched array dimensions in the ORNL code cause unpredictable termination because unmatched array dimensions will cause memory allocation issues when assigned values between two unmatched arrays. Also, there is still a bug existing related to the file IO. We have been working to debug this issue for months.

In this work, we use MPI libraries as implementing LM, GA, and BA. As the future work, we may consider using Open MP libraries as the inner loop, LM, and MPI as the outer loop, GA for BA algorithm. This may reduce the complication and confliction when using MPI for both inner and outer loop for BA.

5. References

[1] Levenberg-Marquardt Algorithm, by Ananth Ranganathan, College of Computing, Georgia Institute of Technology, Atlanta, GA http://www.cc.gatech.edu/people/home/ananth/lmtut.pdf

[2] Introduction to Genetic Algorithms, by Matthew Wall Matthew, MIT Mechanical Engineering Department, MA

[3] A brief introduction to Genetic Algorithm, by Nikos Drakos, Computer Based Learning Unit, University of Leeds, Italy http://www.cs.bgu.ac.il/~sipper/ga.html

[4] "High-Beta Equilibria of Drift-Optimized Compact Stellarators". A. S. Ware, S. P. Hirshman, D. A. Spong, L. A. Berry, A. J. Deisher, R. Sanchez, and G. Y. Fu. Phys. Rev. Lett. 89, 125003 (2002)

[5] "Stellarator Optimization: A Comparison of Genetic, Differential Evolution, and Levenberg-Marquadt Optimization Algorithms" 2003 International Sherwood Fusion Theory Conference, April 28-30, 2003, Corpus Christi, Texas. S. Wang, A.S. Ware, S.P. Hirshman & D.A. Spong

[6] "A Breeder Algorithm for Stellarator Optimization" 45th Annual Meeting of the APS (American Physical Society) Division of Plasma Physics, October 27-31, 2003, Albuquerque, New Mexico. S. Wang, A.S. Ware, S.P. Hirshman & D.A. Spong

[7] Hybridization of Differential Evolution for Aerodynamic Design, by T. Rogalsky [Department of Mathematics, University of Manitoba] and R. W. Derksen [Department of Mechanical and Industrial Engineering, University of Manitoba]

[8] Genetic Algorithm and Optimizing Chemical Oxygen-Iodine Lasers, by David L. Carroll, Research Scientist, University of Illinois, Urbana, Illinois

[9] Levenberg-Marquardt Method, by Carleton DeTar, Physics Department, University of Utah, Salt Lake City, UT

6. Appendix: Source Code

6.1 run_optimizer.f

```
      SUBROUTINE run_optimizer(xc_opt, var_descript,
     1                  nopt, nvar, lwa, info)
      USE stel_kinds
      USE optim, ONLY: home_dir, lone_step, lrestart
      USE optim_params, ONLY: epsfcn, niter_opt, seq_ext,
     1  num_processors, num_levmar_params, nopt_alg
      USE vparams, ONLY: zero
      USE mpi_params                                        ! MPI
      IMPLICIT NONE
C-----------------------------------------------
C   D u m m y   A r g u m e n t s
C-----------------------------------------------
      INTEGER :: nopt, nvar, lwa
      REAL(rprec), DIMENSION(nvar) :: xc_opt
      CHARACTER(len=*), DIMENSION(nvar) :: var_descript                 !description of independent variables
C-----------------------------------------------
C   L o c a l   V a r i a b l e s
C-----------------------------------------------
      INTEGER, PARAMETER :: info_size = 33
      CHARACTER(len=*), PARAMETER :: describe_string =
     1        "DESCRIPTION OF INDEPENDENT VARIABLES",
     2  stop_string = 'Allocation error in STELLOPT run-optimizer!'
      INTEGER :: info, niter
      INTEGER :: mode
      REAL(rprec), DIMENSION(:), ALLOCATABLE :: fvec, diag
      REAL(rprec) :: tol
      CHARACTER*120, DIMENSION(1:info_size) :: info_array
C-----------------------------------------------
C   E x t e r n a l   F u n c t i o n s
C-----------------------------------------------
      EXTERNAL lsfun1
C-----------------------------------------------
      data info_array/
     1    'error in read_wout_opt opening wout file',

     2    'error in load_physics_codes in system call',

     3    'error opening indata file in write_indata',

     4    'error opening output file in lsfun',

     5    'error writing new input file in load_params',

     6    'vmec2000 executable file not found',

     7    'i/o error in clean_up routine',

     8    'error reading wout file in call to read_wout_opt',

     9    'allocation error in load_target',

     a    'boozer array dimension mismatch in load_target',

     b    'nvar and nopt do not match in load_target',

     c    'i/o error opening cobra file in chisq_ballooning',

     d    'i/o error opening bootstrap file in chisq_bootsj',

     e    'i/o error in open_comm_files',

     f    'error in chk_rzmnb',

     g    'boozer transform module - xbooz_xform - not found',
```

```
     h    'could not locate executable in load_physics_codes',
     i    'error reading output file in chisq_jinvar subroutine',
     j    'error in external kink computation',
     k    'error running xdkes code in chisq_dkes subroutine',
     l    'error in vacuum vessel matching subroutine',
     m    'system call to xcoilgeom failed in generate_mgrid',
     n    'coils data file was not produced by xcoilgeom',
     o    'error opening extcur file in generate_mgrid',
     p    'error opening coil_targets file in chisq_coilgeom',
     q    'error reading boozmn file in call to read_boozer_file',
     r    'error opening neo code input file neo_in',
     s    'trouble running eq3d in chisq_orbit',
     t    'trouble running mkjmc in chisq_orbit',
     u    'trouble running orbit in chisq_orbit',
     v    'error opening orbsum in chisq_orbit',
     w    'error opening ft79jmc in chisq_dsubr',
     x    'error in chisq_vac_island'
     z   /

     ALLOCATE (fvec(nopt), diag(nvar), stat = info)
     IF (info .ne. 0) STOP stop_string

     tol = 1.e-6_dp
     mode = 1
     niter = niter_opt
     IF (lone_step) niter = 1

!
!    print description of independent variables at top of screen
!
     IF (myid .eq. master) THEN
       WRITE (6, '(60("="),/,a,/,60("="),/,a)') describe_string,
    1       ' VAR #   TYPE'
       DO info = 1, nvar
         WRITE(6, '(i5,2x,a)') info, var_descript(info)
       END DO
       WRITE (6, *)
     END IF

     IF(NOPT_ALG .eq. 0) THEN
      CALL lmdif1_mp (lsfun1, nopt, nvar, xc_opt, fvec, tol, epsfcn,
    1    niter, diag, mode, info, lwa)
     ELSE IF(NOPT_ALG .eq. 1) THEN
       CALL ga_driver (lsfun1, nopt, nvar, xc_opt, fvec, tol, epsfcn,
    1  niter, num_processors, seq_ext, info, lwa, lrestart )

     ELSE IF(NOPT_ALG .eq. 2) THEN
       CALL de_driver (lsfun1, nopt, nvar, xc_opt, fvec, tol, epsfcn,
    1  niter, num_processors, seq_ext, info, lwa, lrestart )

     ELSE IF(NOPT_ALG .eq. 3) THEN                          !! BREED: A new option
       CALL galm_driver (lsfun1, nopt, nvar, xc_opt, fvec, tol,     !! BREED: A new option
    1  epsfcn, niter, num_processors, seq_ext, info, lwa, lrestart)  !! BREED: A new option
                                          !! BREED: A new option
```

```
ELSE
  IF (myid .eq. master)
1    WRITE(6,*) "NOPT_ALG = ", NOPT_ALG, "; unable to proceed"
  STOP
ENDIF
DEALLOCATE (fvec, diag)

IF (myid.eq.master .and. info.lt.0) WRITE (*, '(/,1x,a,a)')
1   'Stellopt status: ',TRIM(info_array(-info))

END SUBROUTINE run_optimizer
```

6.2 galm_driver.f

```
      SUBROUTINE GALM_driver(fcn, n_opt, n_var, x, fvec, tol, eps,  !! BREED: This is the Breed driver
     1    num_iter_opt, max_processors, filename, info, lwa, lrestart )
      USE ga_mod
      USE system_mod
      USE safe_open_mod
      USE mpi_params, ONLY: master, myid
      IMPLICIT NONE
      include 'mpif.h'                          !mpi stuff
      INTEGER :: ierr
      INTEGER :: n_opt, n_var, info, lwa, num_iter_opt, max_processors
      REAL(rprec), DIMENSION(n_opt), TARGET :: fvec
      REAL(rprec), DIMENSION(n_var) :: x
      REAL(rprec), DIMENSION(n_var) :: partemp
      REAL(rprec) :: tol, eps, chi_sq, tmp              !ga_evaluate
      EXTERNAL fcn
      CHARACTER*(*) ::  filename
      LOGICAL :: lrestart

      INTEGER :: num_iter_max
      INTEGER :: i, iflag, nfev

c ***********************************************************************
c  entries for the 'ga' NAMELIST
c
c npopsiz   - population size
c idum      - if < 0, then |idum| is used as seed for random-number gen.
c pmutate   - probability for random jump mutation
c pcross    - crossover probability
c ielite    - /=0  make sure best parent is preserved into decendent populations
c icreep    - creep mutation flag:  only do creep mutations if .ne. 0
c pcreep    - probability for random creep mutation
c iunifrm   - =0 single point crossover at random chromosome point
c           /=0 uniform crossover
c iniche    - /=0 turn on niching
c nichflg   - array of flags for the free-parameters,
c           each non-zero entry enables niching for that free-parameter
c iskip
c iend
c nchild    - default=1; if =2, then each crossover creates 2 children,
c           the second child having the second parents genes
c parmin    - array specifying minimum value for each free-parameter,
c parmax    - array specifying maximum value for each free-parameter,
c ibound    - =1 then interpret parmin and parmax as scale-factors to be
c           multiplied by the initial guess values for each parameter
c nposibl
c nowrite   - =0 then write output during optimization
c microga   - =0 perform random mutations
c           /=0 perform micro-GA
c unique_ind
c itourny
c
c    IMPORTANT: MPI_PARAMS MODULE MUST BE LOADED BY EXTERNAL CALLS
c             TO MPI_COMM_RANK PRIOR TO THIS SUBROUTINE CALL
c ***********************************************************************
      info = 0
      num_iter_max = num_iter_opt

      itoumy=1
      maxgen=ngen
      kountmx=maxgen
      nparam=n_var
      num_obj = n_opt

      IF( ibound .eq. 1 ) THEN
        par_max(:n_var) = x(:n_var)*parmax(:n_var)
        par_min(:n_var) = x(:n_var)*parmin(:n_var)
```

```
        WHERE (par_max(:n_var) < par_min(:n_var) )
          partemp(:n_var) = par_max(:n_var)
          par_max(:n_var) = par_min(:n_var)
          par_min(:n_var) = partemp(:n_var)
        END WHERE

      ELSE
        par_max = parmax
        par_min = parmin
      END IF

      IF (ALL(nposibl .eq. 0 )) nposibl=15

      nfev=1
      f_obj => fvec

!     IF (myid .eq. master) WRITE(6, nml = ga_de)

      irestrt = 0
      IF( lrestart ) irestrt = 1
c
c      store initial parameter values as a unique individual
      parent = 0
      child = 0
      iparent = 0
      ichild = 0
      IF(unique_ind .gt. 0 ) THEN
        unique_ind=MIN(unique_ind, npopsiz)
        parent(1:nparam,unique_ind) = x(1:nparam)
      END IF

      CALL galm_sp(fcn, n_opt, fvec, chi_sq, filename, nfev,      !! BREED: Calling galm_sp
     >          iflag, max_processors, myid)                    !! BREED: Calling galm_sp

      IF (myid .eq. master) THEN
        WRITE(6,*) "final solution: "
        WRITE(6,*) "best individual : ", jbest
        WRITE(6,*) "x ", (parent(i,jbest),i=1,nparam)
c       WRITE(6,*) "fvec ",(fvec(i),i=1,n_opt)
        WRITE(6,*) "y ", chi_sq
      END IF

      x(1:n_var) = parent(1:n_var,jbest)

      iflag=-100
      CALL fcn(n_opt, npopsiz, parent(1,jbest), fvec, iflag, nfev)

      END SUBROUTINE GALM_driver                              !! BREED: End of Breed driver
```

6.3 galm_sp.f

```
      SUBROUTINE galm_sp(fcn, nopt, fvec, best, filename, nfev, iflag, !! BREED: Start of galm_sp
     1            max_num_processors, myid)
      USE ga_mod
      USE safe_open_mod
      USE mpi_params, ONLY: master
      USE optim_params, ONLY: fixed_seq_ext, seq_ext              !! BREED: *DANGER* Using optim_params in library
      IMPLICIT NONE
      include 'mpif.h'                          !mpi stuff
      EXTERNAL fcn

      INTEGER :: nopt
      REAL(rprec), DIMENSION(nopt) :: fvec

      INTEGER :: kount, npossum,ig2sum, istart, istore
      INTEGER :: ncross, ipick, mate1, mate2, istat
      INTEGER :: i, j, nfev, iflag, max_num_processors, myid
      REAL(rprec) :: fbar, best, evals
      CHARACTER*(*) :: filename
      CHARACTER*(len_trim(filename)+10) :: temp
      SAVE
c
c     CALL input
c
c  Perform necessary initialization and read the ga.restart file.
      CALL ga_initial(istart,npossum,ig2sum,filename,myid)
c
c  $$$$$ Main generational processing loop. $$$$$
      kount=0
      nfit_eval=nfev
      istore=0
      iunit_ga_out = 24
      IF (myid .eq. master) THEN
       temp = "ga_out." // filename
       CALL safe_open(iunit_ga_out, istat, TRIM(temp),
     1            'unknown', 'formatted')
      END IF


      DO 20 i=istart,maxgen+istart-1
        IF (i .eq. 1) THEN                          !! BREED: Using fixed_seq_ext to store seq_ext
          fixed_seq_ext = seq_ext                   !! BREED: since seq_ext will be changed when
        END IF                                      !! BREED: calling LM in ga_evalout
        iflag=-1
        IF (myid .eq. master) THEN
          WRITE (6,1111) i
          WRITE (iunit_ga_out,1111) i
c
c  Evaluate the population, assign fitness, establish the best
c  individual, and write output information.
          WRITE(6,*) 'pre ga_evalout', max_num_processors
          WRITE(6,*) fbar,best,nopt,nfev,max_num_processors,iflag
        END IF
        CALL ga_evalout(fbar, best, fcn, nopt, fvec, nfev,      !! BREED: Added filename
     >       max_num_processors, iflag, myid, filename)         !! BREED: to ga_evalout call
        istore=istore+1
        geni(istore) = REAL(i, kind=rprec)
        genavg(istore)=fbar
        genmax(istore)=best
        IF (npopsiz.eq.1 .or. iskip.ne.0) THEN
          IF (myid .eq. master) CLOSE(iunit_ga_out)
          CALL ga_restart(i,istart,kount,filename, myid)
          RETURN
        END IF
c
c  niching
        IF (iniche.ne.0) CALL ga_niche(myid)
c
```

```
c  selection, crossover and mutation
       ncross=0
       ipick=npopsiz
       DO 45 j=1,npopsiz,nchild
c
c  Perform selection.
          CALL ga_selectn(ipick,j,mate1,mate2)
c
c  Now perform crossover between the randomly selected pair.
          CALL crosovr(ncross,j,mate1,mate2)
 45     CONTINUE

        IF (myid .eq. master) THEN
          WRITE(6,1225) ncross
          WRITE(iunit_ga_out,1225) ncross
        END IF
c
c  Now perform random mutations.  If running micro-GA, skip mutation.
        IF (microga.eq.0) CALL ga_mutate (myid)
c
c  Write child array back into parent array for new generation.  Check
c  to see IF the best parent was replicated.
        CALL ga_newgen(npossum,ig2sum,myid)
c
c  Implement micro-GA IF enabled.
        IF (microga.ne.0) CALL ga_micro(i,npossum,ig2sum,myid)
c
c  Write to restart file.
        CALL ga_restart(i,istart,kount,filename,myid)
        seq_ext = fixed_seq_ext                    !! BREED: Setting seq_ext back to the original value
 20   CONTINUE

c  $$$$$ End of main generational processing loop. $$$$$

      IF (myid .eq. master) THEN
        WRITE(iunit_ga_out,3000)
        DO 100 i=1,maxgen
          evals = npopsiz*geni(i)
          WRITE(iunit_ga_out,3100) geni(i),evals,genavg(i),genmax(i)
 100    CONTINUE
        CLOSE (iunit_ga_out)
      END IF

 1050 FORMAT(1x,'   Binary Code',16x,'Parameter Values and  Fitness')
 1111 FORMAT(///,'################  Generation',i5,
     1       ' ################')
 1225 FORMAT(/' Number of Crossovers     =',i5)
 3000 FORMAT(2x//'Summary of Output'/
     +      2x,'Generation   Evaluations   Avg.Fitness   Best Fitness')
 3100 FORMAT(2x,3(e10.4,4x),e11.5)

      END SUBROUTINE galm_sp                       !! BREED: End of galm_sp
```

6.4 ga_evalout.f

```
      SUBROUTINE ga_evalout(fbar, best, fcn, nopt, fvec, nfev,     !! BREED: filename added to
     >                num_processors, iflag, myid, filename)    !! BREED: argument list of ga_evalout
c#######################################################################
c
c  this subroutine evaluates the population, assigns fitness,
c  establishes the best individual, and outputs information.
      USE optim_params, ONLY: opt_ext, oga_ext, seq_ext, nopt_alg, !! BREED: *DANGER* Using optim_params
     1                  fixed_seq_ext, funcval_breed        !! BREED: *DANGER* in ga_evalout
      USE ga_mod
      USE system_mod                                  !! BREED: ga_evalout now uses system calls
      USE mpi_params, ONLY: master
      IMPLICIT NONE
      EXTERNAL fcn, sub_optimize                          !! BREED: New external sub_optimize
      INTEGER :: nopt, n, j, k, kk, iflag, myid
      REAL(rprec), DIMENSION(nopt) :: fvec
      REAL(rprec), DIMENSION(nparmax) :: paramsm,paramav
      INTEGER :: nfev, num_processors
      REAL(rprec) :: fitsum, funcval, fbar, best
      INTEGER :: jstart, jend, istat, jstat
      LOGICAL :: ldiag_opt
      CHARACTER(5) :: char_npopsiz                        !! BREED: Added character variable char_npopsiz
      CHARACTER(10) :: tempd                             !! BREED: Added character variable tempd
      CHARACTER(60) :: temp_seq_ext, temp_oga_ext, temp_opt_ext    !! BREED: Added temp character variables
      CHARACTER*(*) :: filename                           !! BREED: Added character variable filename
      CHARACTER*(len_trim(filename)+12) :: input_file          !! BREED: Added character variable input_file
      CHARACTER*(100) :: temp                            !! BREED: Added character variable temp
      CHARACTER*(100) :: myopt_ext, myoga_ext                  !! BREED: Added character variables myopt_ext, myoga_ext

      SAVE
c
c


      fitsum = 0
      best=-1.0e30_dp

      ldiag_opt = .false.

      IF (myid .eq. master) WRITE(6,*) 'in ga_evalout',num_processors
      IF (myid .eq. master) WRITE(6,*) fbar,best,nopt,nfev,iflag

      DO 29 n=1,nparam
        paramsm(n)=0
 29   CONTINUE
      jstart=1
      jend=npopsiz
      IF(iskip.ne.0) jstart=iskip
      IF(iend.ne.0) jend=iend
c
      DO  j=jstart,jend

        CALL ga_decode(j,parent,iparent)

        IF(LDIAG_OPT .and. myid.eq.master) THEN
          IF(nchrome .le. 120) THEN
            WRITE(iunit_ga_out,1075) j,(iparent(k,j),k=1,nchrome)
          ELSE
            WRITE(iunit_ga_out,1075) j,(iparent(k,j),k=1,120)
            WRITE(iunit_ga_out,1077) (iparent(k,j),k=121,nchrome)
          END IF
          WRITE(iunit_ga_out,1076) (parent(kk,j),kk=1,nparam)
        END IF
      END DO
      CALL ga_fitness_mpi (jend-jstart+1, f_obj, num_obj,
     1    fcn, nfev, fitness)
      nfev=nfev+jend-jstart+1
      nfit_eval=nfev
```

```
      IF (NOPT_ALG .eq. 3) THEN                          !! BREED: File manipulation in Breed
        IF (myid .eq. master) THEN                       !! BREED: Only the master makes directories
          temp = 'mkdir tempdir1'                        !! BREED: Stores _opt files as _oga files
          CALL system(temp)                              !! BREED: Making the directory here
        END IF                                           !! BREED: The master loop is done
        DO k=1,npopsiz                                   !! BREED: Starting loop over the population members
          write (char_npopsiz,'(i5)') k                  !! BREED: Writing the population member # to an integer
          myoga_ext = trim(fixed_seq_ext)//'_oga'//      !! BREED: Set up local variable myoga_ext
     1          trim(adjustl(char_npopsiz))              !! BREED: using fixed_seq_ext and member #
          myopt_ext = trim(fixed_seq_ext)//'_opt'//      !! BREED: Set up local variable myopt_ext
     1          trim(adjustl(char_npopsiz))              !! BREED: using fixed_seq_ext and member #
          input_file = 'input.'//myopt_ext               !! BREED: Avoiding using global variable opt_ext
                                                         !! BREED
          IF (myid .eq. master) THEN                     !! BREED: Only the master copies files
            tempd = 'tempdir1'                           !! BREED: Define the temp directory name
            temp = 'cp '//input_file//' '//'input.'//myoga_ext   !! BREED: Storing the input_file in the file input.myoga_ext
            CALL system(temp)                            !! BREED: Make it so
            temp = 'cp '//input_file//' '//trim(tempd)//     !! BREED: Make a copy in the temp
     I          '/input.'//myoga_ext                    !! BREED: directory as well. Why?
            CALL system(temp)                            !! BREED: MAke it so
          END IF                                         !! BREED: Done with the master loop
          oga_ext = myoga_ext                            !! BREED: TESTING---------08302004
          CALL sub_optimize(myoga_ext)                   !! BREED: *DANGER* library calling sub_optimize
                                                         !! BREED
          IF (myid .eq. master) THEN                     !! BREED: Only the master stores the funcval
            open(3333)                                   !! BREED: from file 3333 (written in sub_optimize)
            read(3333,iostat=istat) funcval_breed        !! BREED: Read in the value
            close(3333)                                  !! BREED: Close the fort.3333 file
            fitness(k) = funcval_breed                   !! BREED: Store this value in the fitness array
          END IF                                         !! BREED: Done with the master loop
          temp = "cp " //"stellopt_"//trim(seq_ext)//"/"//     !! BREED: Copy the .min oga files back to the main directory
     1        "input."//trim(seq_ext)//".min"//          !! BREED: the main directory but with a different
     2        " "//"input."//trim(myopt_ext)             !! BREED: extensiuon, the myopt_ext
          CALL system(temp)                              !! BREED: Make it so
!! BREED
!! BREED: need to do before continue GA routine:
!! BREED:   1. replace input._optk by input...ogak.min
!! BREED:   2. clean tempdir
!! BREED
        END DO                                           !! BREED: End of the loop over the population
!! BREED
!! BREED: A couple of questions here:
!! BREED:   1. only consider these input._opt files?
!! BREED:   2. for using each LM routine, when doing clean up operation
!! BREED:      whether it impact other input._opt(oga) files or not?
!! BREED
      END IF                                             !! BREED: End of Breed file loop

!     Clean up...
      iflag=-100
      CALL fcn(nopt, npopsiz, parent(1,jbest), fvec, iflag, nfev)
      IF (NOPT_ALG .eq. 3) THEN                          !! BREED: Clean up for Breed
        IF (myid .eq. master) THEN                       !! BREED: Only the master removes the temp directory
          temp = 'rm -rf '//trim(tempd)                  !! BREED: Remove the temp directory
          CALL system(temp)                              !! BREED: Make it so
        END IF                                           !! BREED: Done with the master loop
      END IF                                             !! BREED: Done with clean up for Breed
      DO j = jstart, jend
        fitsum=fitsum+fitness(j)
        DO n=1,nparam
          paramsm(n)=paramsm(n)+parent(n,j)
        END DO
c
c Check to see IF fitness of individual j is the best fitness.
        IF (fitness(j).gt.best) THEN
          best=fitness(j)
          jbest=j
          DO k=1,nchrome
            ibest(k)=iparent(k,j)
          END DO
```

```
      END IF
    END DO

c  compute parameter and fitness averages.
    fbar=fitsum/npopsiz
    DO n=1,nparam
      paramav(n)=paramsm(n)/npopsiz
    END DO
c
c  write output information
    IF (myid.eq.master) THEN
      IF (ldiag_opt) THEN
        IF (npopsiz.eq.1) THEN
          IF(nchrome .le. 120) THEN
            WRITE(iunit_ga_out,1075) 1,(iparent(k,1),k=1,nchrome)
          ELSE
            WRITE(iunit_ga_out,1075) 1,(iparent(k,1),k=1,120)
            WRITE(iunit_ga_out,1077) (iparent(k,j),k=121,nchrome)
          END IF
          WRITE(iunit_ga_out,1076) (parent(k,1),k=1,nparam)
          WRITE(iunit_ga_out,1078)  fitness(1)
          WRITE(iunit_ga_out,*) ' Average Values:'
          WRITE(iunit_ga_out,1275) (parent(k,1),k=1,nparam)
          WRITE(iunit_ga_out,1276) fbar
        ELSE
          WRITE(iunit_ga_out,1275) (paramav(k),k=1,nparam)
          WRITE(iunit_ga_out,1276) (fitness(j),j=1,npopsiz)
        END IF
      END IF
      WRITE(6,1100) fbar
      WRITE(iunit_ga_out,1100) fbar
      WRITE(6,1200) best
      WRITE(iunit_ga_out,1200) best
    END IF

1075 FORMAT(i3,1x,(120i1))
1077 FORMAT(3x,1x,(120i1))
1076 FORMAT(3x,1x,10(1x,e10.4))
1078 FORMAT(10x,e12.5)
1100 FORMAT(1x,'Average Function Value of Generation=',e12.5)
1200 FORMAT(1x,'Maximum Function Value          =',e12.5/)
1275 FORMAT(/' Average Values:',18x,10(1x,e10.4))
1276 FORMAT(10x,10e12.5)

    END SUBROUTINE ga_evalout
```