GENETIC ALGORITHMS AND THEIR APPLICATIONS: AN OVERVIEW

MIR ASIF IQUEBAL
Ph.D. (Agricultural Statistics), Roll No. 9068
I.A.S.R.I., Library Avenue, New Delhi-110012

Chairperson: Dr. Prajneshu

Abstract: Genetic Algorithm (GA) is a calculus free optimization technique based on principles of natural selection for reproduction and various evolutionary operations such as crossover, and mutation. Various steps involved in carrying out optimization through GA are described. Three applications, viz. finding maximum of a mathematical function, obtaining estimates for a multiple linear regression model, and fitting a nonlinear statistical model through GA procedure, are discussed. Finally, results are compared to those obtained from standard (calculus based) solution techniques.

Key words: *Genetic Algorithm, Fitness Function, Selection, Crossover, Mutation.*

1. Introduction

A genetic algorithm (GA) is a search and optimization method which works by mimicking the evolutionary principles and chromosomal processing in natural genetics. A GA begins its search with a random set of solutions usually coded in binary strings. Every solution is assigned a fitness which is directly related to the objective function of the search and optimization problem. Thereafter, the population of solutions is modified to a new population by applying three operators similar to to natural genetic operators-reproduction, crossover, and mutation. It works iteratively by successively applying these three operators in each generation till a termination criterion is satisfied. Over the past decade and more, GAs have been successfully applied to a wide variety of problems, because of their simplicity, global perspective, and inherent parallel processing.

Classical search and optimization methods demonstrate a number of difficulties when faced with complex problems. The major difficulty arises when one algorithm is applied to solve a number of different problems. This is because each classical method is designed to solve only a particular class of problems efficiently. Thus, these methods do not have the breadth to solve different types of problems often faced by designers and practitioners. Moreover, most of the classical methods do not have the global perspective and often get converged to a locally optimum solution. Another difficulty is their inability to be used in a parallel computing environment efficiently. Since most classical algorithms are serial in nature, not much advantage can be achieved with them.

The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. GAs have been widely studied, experimented and applied in many fields of sciences. Not only does GAs provide an alternative method to solving problem, it consistently outperforms other traditional methods in most of the problems. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for GAs.

2. Basics of Genetic Algorithms

The idea of evolutionary computing was introduced in 1960 by Rechenberg in his work "Evolutionary strategies". GAs are computerized search and optimization algorithms based on mechanics of natural genetics and natural selection. Prof. John Holland of University of Michigan envisaged the concept of these algorithms in the mid-sixties and published (Holland, 1975).

2.1 Basic Concepts

GAs are good at taking larger, potentially huge, search spaces and navigating them looking for optimal combinations of things and solutions which we might not find in a life time.

GAs are very different from most of the traditional optimization methods. Genetic algorithms need design space to be converted into genetic space. So, Genetic algorithms work with a coding of variables. The advantage of working with a coding of variable space is that coding discretizes the search space even though the function may be continuous. A more striking difference between GAs and most of the traditional optimization method is that GA uses a population of points at one time in contrast to the single point approach by traditional optimization methods. This means that GA processes a number of designs at the same time.

2.2 Working Principle

The GA is an iterative optimization procedure. Instead of working with a single solution in each iteration, a GA works with a number of solutions (collectively known as population) in each iteration. A flowchart of the working principle of a simple GA is shown in Figure 2.1.

In the absence of any knowledge of the problem domain, a GA begins its search from a random population of solutions. We shall discuss about the detail of coding procedure a little later. But now notice how a GA processes strings in an iteration. If a termination criterion is not satisfied, three different operators – reproduction, crossover and mutation – are applied to update the population of strings. One iteration of these three operators is known as a generation in the parlance of GAs. Since the representation of a solution in a GA is similar to a natural chromosome and GA operators are similar to genetic operators, the above procedure is called a genetic algorithm.

3. Various Steps involved in GA Procedure

3.1 Representation

In a binary coded GA, every variable is first coded in a fixed-length binary string. For example, the following is a string, representing N problem variables:

$$\underbrace{11010}_{x_1}\ \underbrace{1001001}_{x_2}\ \underbrace{010}_{x_3}\ldots\underbrace{0010}_{x_N}$$

The i^{th} problem variable is coded in a binary substring of length l_i, so that total number of alternatives allowed in that variable is 2^{l_i}. The lower bound solution x_i^{min} is represented by solution $(0,0,...,0)$ and the upper bound solution x_i^{max} is represented by the solution $(1,1,...,1)$. Any other substring s_i decodes to a solution x_i as follows:

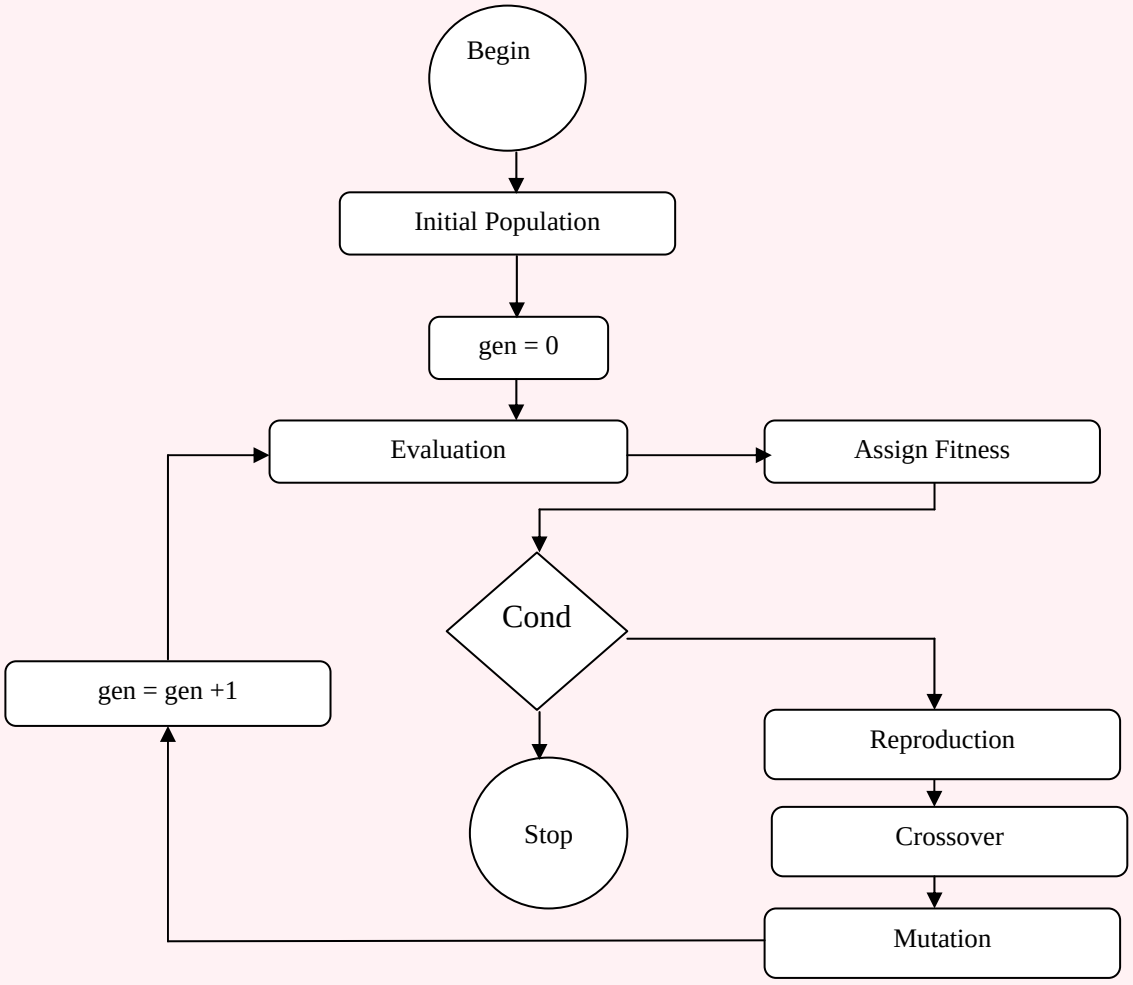

Fig. 2.1: Flowchart of working principle of a genetic algorithm

$$x_i = x_i^{min} + \frac{x_i^{max} - x_i^{min}}{2^{l_i} - 1} DV(s_i), \qquad \ldots(3.1)$$

where $DV(s_i)$ is decoded value of string s_i.The decoded value of a binary substring $s \equiv (s_{l-1}\, s_{l-2} \ldots s_2\, s_1\, s_0)$ is calculated as $\sum_{j=0}^{l-1} 2^j s_j$, where $s_j \in \{0,1\}$. The length of substring is usually decided by precision needed in a variable. For example if three decimal places of accuracy are needed in the i th variable, total number of alternatives in the variable must be $\frac{x_i^{max} - x_i^{min}}{0.001}$, which can be set equal to 2^{l_i} and l_i can be computed as follows:

$$l_i = \log_2 \left(\frac{x_i^{max} - x_i^{min}}{\varepsilon_i} \right)$$

Here, the parameter ε_i is desired precision in i th variable. The total string length of a N-variable solution is then $l = \sum_{i=1}^{N} l_i$.

In the population, l-bit strings are created at random (at each of l positions, there is a equal probability of creating a 0 or 1). Once such string is created, the first l_1 bits can be extracted from the complete string and corresponding value of the variable x_1 can be calculated using Equation (3.1) and using the chosen lower and upper limits of variable x_1. This process is continued until all N-variables are obtained from complete string. Thus, an l-bit string represents a complete solution specifying all N variables uniquely. Once these values are known, the objective function $f(x_1, x_2, ..., x_N)$ can be computed.

In a GA, each string created either in the initial population or in the subsequent generations must be assigned a fitness value which is related to objective function value. For maximization problem, a string's fitness can be equal to string's objective function value. However, for minimization problems, the goal is to find a solution having minimum objective function value. Thus, the fitness can be calculated as the negative of the objective function so that solutions with similar objective function value get larger fitness.

There are number of advantages of using a string representation to code variables. First, this allows a shielding between working of GA and actual problem. The same GA code can be used for different problems by only changing definition of coding a string. This allows a GA to have widespread applicability. Second, a GA can exploit the similarities in string coding to make its search faster, a matter which is important in working of a GA.

3.2 Reproduction

Reproduction (or selection) is usually the first operator applied to a population. Reproduction selects good strings in a population and forms a mating pool. The essential idea is that above-average strings are picked from the current population and duplicates of them are inserted in the mating pool. The commonly used reproduction operator is the proportionate selection operator, where a string in the current population is selected with probability proportional to the string's fitness. Thus, the i^{th} string in the population is selected with probability proportional to f_i. Since the population size is usually kept fixed in a simple GA, the cumulative probability for all string in the population must be one. Therefore, the probability for selecting ith string is $f_j \Big/ \sum_{j=1}^{N} f_j$, where N is the population size. One way to achieve this proportionate selection is to use a roulette-wheel with the circumference marked for each string proportionate to the string's fitness.

3.3 Crossover

The crossover operator is applied next to the string of the mating pool. In crossover operator, two strings are picked from the mating pool at random and some portion of the strings is exchanged between the strings. In a single-point crossover operator, both strings are cut at an arbitrary place and right-side portion of both strings are swapped among themselves to create two new strings, as illustrated in the following:

Parent 1	0 0	0 0 0	⟹	0 0 | 1 1 1	Child 1
Parent 2	1 1	1 1 1		1 1 | 0 0 0	Child 2

It is interesting to note from the construction that good substrings from either parent string can be combined to form better child string if an appropriate site is chosen. Since the knowledge of an appropriate site is usually not known, a random site is usually chosen. However, it is important to realize that the choice of a random site does not make this search operation random. With a single-point crossover on two l-bit parent strings, the search can only find at most $2(i-1)$ different strings in the search space, whereas there are a total of 2^i strings in the search space. With a random site, the children strings produced may or may not have a combination of good substrings from parent strings depending on whether the crossing site falls in the appropriate site or not. But we do not worry about this aspect too much, because if good strings are created by crossover, there will be more copies of them in the next mating pool generated by the reproduction operator. But good strings are not created by crossover, they will not survive beyond next generation, because reproduction will not select bad strings for the next mating pool. In a two-point crossover operator, two random sites are chosen. This idea can be extended to create multi-point crossover operator and the extreme of this extension is known as a uniform crossover operator. In a uniform crossover for binary strings, each bit from either parent is selected with a probability of 0.5. The main purpose of the crossover operator is to search the parameter space. Other aspect is that the search need to be performed in a way to preserve the information stored in the parent string maximally, because these parent strings are instances of good strings selected using the reproduction operator. In the single-point crossover operator search is not extensive, but the maximum information is preserved from parent to children. On the other hand, in the uniform crossover, the search is very extensive but minimum information is preserved between parent and children strings. If a crossover probability of p_c is used then $100\,p_c$ % strings in the population are used in the crossover operation and $100(1-p_c)\%$ of the population are simply copied to the new population.

3.4 Mutation

Crossover operator is mainly responsible for the search aspect of genetic algorithms, even though the mutation operator is also used for this purpose sparingly. The mutation operator changes a 1 to a 0 and vice versa with a small mutation probability p_m:

0 0 0 0 0 ⟹ 0 0 0 1 0

In the above example, fourth gene has changed its value, thereby creating a new solution. The need for mutation is to maintain diversity in population. For example, if in a particular position along the string length all strings in the population have a value 0, and a 1 is needed in that position to obtain optimum or a near-optimum solution, then mutation operator described above will be able to create a 1 in that position. The inclution of mutation introduces some probability of turning that 0 into 1. Furthermore, for local improvement of a solution, mutation is useful.

After reproduction, crossover, and mutation are applied to whole population, one generation of GA is completed. These three operators are simple and straightforward. The reproduction operator selects good strings and the crossover operator recombines good substrings from two good strings together to hopefully form a better substring. The mutation operator alters a string locally to hopefully create a better string. Even though none of these claims guaranteed and / or tested while creating a new population strings, it is expected that if bad strings are created they will be eliminated by the reproduction operator in next generation and if good strings are created, they will be emphasized. To make a faster convergence of a GA to real-world problems, problem-specific operators are often developed and used, but the above three operators portray fundamental operations of a genetic algorithm and facilitate a comparatively easier mathematical treatment.

4. Software Packages

Whilst there exist many good public-domain genetic algorithm packages, such as GENESYS and GENITOR, none of these provide an environment that is immediately compatible with existing tools in the control domain. The MATLAB Genetic Algorithm Toolbox aims to make GAs easily accessible. This allows the retention of existing modelling and simulation tools for building objective functions and allows the user to make direct comparisons between genetic methods and traditional procedures.

4.1 Data Structures

The main data structures in the GA Toolbox are chromosomes, phenotypes, objective function values and fitness values. The chromosome structure stores an entire population in a single matrix of size Nind × Lind, where Nind is the number of individuals and Lind is the length of the chromosome structure. Phenotypes are stored in a matrix of dimensions Nind × Nvar where Nvar is the number of decision variables. An Nind × Nobj matrix stores the objective function values, where Nobj is the number of objectives. Finally, the fitness values are stored in a vector of length Nind. In all of these data structures, each row corresponds to a particular individual.

4.2 Toolbox Structure

The GA Toolbox uses MATLAB matrix functions to build a set of versatile routines for implementing a wide range of genetic algorithm methods. In this section we outline the major procedures of the GA Toolbox.

4.2.1 Population Representation and Initialization: crtbase, crtbp, crtrp

The GA Toolbox supports binary, integer and floating-point chromosome representations. Binary and integer populations may be initialised using the Toolbox function to create binary populations, crtbp. An additional function, crtbase, is provided that builds a vector describing the integer representation used. Real-valued populations may be initialised using crtrp. Conversion between binary and real-values is provided by the routine bs2rv that also supports the use of logarithmic scaling.

4.2.2 Fitness Assignment: ranking, scaling

The fitness function transforms the raw objective function values into non-negative figures of merit for each individual. The Toolbox supports the offsetting and scaling method and the linear-ranking algorithm. In addition, non-linear ranking is also supported in the routine ranking.

4.2.3 Selection Functions: reins, rws, select, sus
These functions select a given number of individuals from the current population, according to their fitness, and return a column vector to their indices. Currently available routines are roulette wheel selection, rws, and stochastic universal sampling, sus. A high-level entry function, select, is also provided as a convenient interface to the selection routines, particularly where multiple populations are used. In cases where a generation gap is required, i.e. where the entire population is not reproduced in each generation, reins can be used to effect uniform random or fitness-based re-insertion.

4.2.4 Crossover Operators: recdis, recint, reclin, recmut, recombin, xovdp, xovdprs, xovmp, xovsh, xovshrs, xovsp, xovsprs

The crossover routines recombine pairs of individuals with given probability to produce offspring. Single-point, double-point and shuffle crossover are implemented in the routines xovsp, xovdp and xovsh respectively. Reduced surrogate crossover is supported with both single-, xovsprs, and double-point, xovdprs, crossover and with shuffle, xovshrs. A general multi-point crossover routine, xovmp, that supports uniform crossover is also provided. To support real-valued chromosome representations, discrete, intermediate and line recombination are supplied in the routines, recdis, recint and reclin respectively. The routine recmut performs line recombination with mutation features. A high-level entry function to all the crossover operators supporting multiple subpopulations is provided by the function recombin.

4.2.5 Mutation Operators: mut, mutate, mutbga
Binary and integer mutation are performed by the routine mut. Real-value mutation is available using the breeder GA mutation function, mutbga. Again, a high-level entry function, mutate, to the mutation operators is provided.

5. Some Applications

We shall discuss below three problems. Parameter settings of the GA for these problems is as follows:

Table 5.1: Parameter settings of GA procedure for various problems (Chatterjee *et al.* 1996)

Problem No.	Pop.	Para.	Gen.	Crpet.	Selpr.	Mupet.	Nobs.
1	1000	1	500	70	0.75	0.1	NA
2	1000	3	500	70	0.75	0.1	29
3	1000	3	500	70	0.75	0.1	67

Pop.: Population size, Para.: Parameters to be estimated, Gen.: Number of generations, Crpet.: Crossover percentage, Selpr.: Selection pressure, Mupet.: Mutation percentage, and Nobs.: Number of observations is used for GA.

5.1 Problem 1: Finding maximum of a mathematical function
We are interested in finding the global maximum of the function (Chatterjee and Laudato, 1997).

$$f(x) = x + \text{Abs}[\sin 4x], \; -\pi < x < \pi .$$

The function is plotted in Figure 5.1.

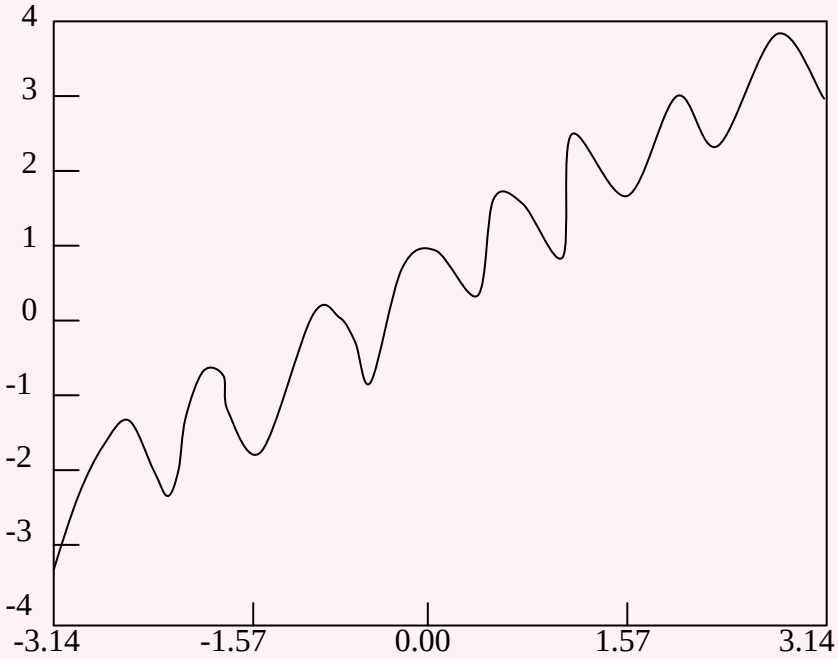

Figure 5.1: Graph of the function $f(x) = x + \text{Abs}[\sin 4x],\ -\pi < x < \pi$

The global maximum may be obtained manually by partitioning the entire range into subranges and then applying rules of calculus. A straightforward algebra shows that the global maximum is at x = 2.81206. However, it is not possible to get the solution by using a software package as the problem involves transcendental function. To get solution using genetic algorithm, we have to create a M-file containing the objective function (fitness function). Objective function is that function which we want to minimize. There is no direct option in MATLAB software for maximization of any function. But, it can easily be done by using minimization of the negative form of objective function. For finding minimization of any function (function may be of any form) with single variable, fminbnd function is used. In our problem, we wish to maximize the above defined function. The optimset command to fminbnd is used for a tabular display of output and iter is used to show iteration history. Using MATLAB, we get the following results.

MATLAB command for finding the estimates of parameters:

```
f = @(x)(-x-abs (sin(4*x)));
x = fminbnd(f, -3.14, 3.14, (optimset('Display','iter')))
```

Table 5.2: Minimum of the function in various iterations

Iteration	x	f (x)
1	-0.7413	0.5656
2	0.7413	-0.9169
3	1.6575	-1.9974
4	2.2238	-2.7291
5	2.5737	-3.3382
6	2.7900	-3.7765

7	2.9237	-3.6890
8	2.8142	-3.7818
9	2.8126	-3.7809
10	2.8121	-3.7803
11	2.8120	-3.7803
12	2.8121	-3.7803

Optimization terminated: the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004 is **x = 2.8121**.

The genetic algorithm gives the solution which is correct to 3 decimal places. The main point of this is that conventional calculus based numerical optimization problems can fail very easily for a problem of this type but GA gives almost exact solution in few number of iterations.

5.2 Problem 2: Multiple linear regression model
A multiple regression model under usual assumption (Chatterjee et. al., 1996) is given by

$$y=\alpha+\beta x_1+\gamma x_2+\varepsilon$$

is estimated for data consisting of Angell's index of "moral integration" (y) for 29 US cities with two explanatory variables, heterogeneity (x_1) and mobility indices (x_2) (Fox, 1984). Here, errors are following usual assumptions of linear model. Least squares are used for the fitness function. First, M-file is created with objective function (fitness function). Here the objective function is of minimization type. For more than single variable objective function fminsearch command is used to find estimates which minimizes the objective function. Since the GA is random, it is important to run GA multiple times before accepting the solution. Using GA we can obtain the parameter estimates. Standard errors are obtained using bootstrap technique.

MATLAB command for finding the estimates of parameters:

```
mulr = @(b)((19 - b(1)-b(2)*20.6-b(3)*15)^2+ (17 - b(1)-b(2)*15.6-b(3)*20.2)^2+ . . .
        + (8 - b(1)-b(2)*27.4-b(3)*25.0)^2+ (7.2 - b(1)-b(2)*16.4-b(3)*35.8)^2);
[b,fval] = fminsearch(mulr,[0.0, 0.0, 0.0],(optimset('Display','iter')))
```

Table 5.3: Minimum of the objective function in different iterations

Iteration	min f(x)	Iteration	min f(x)	Iteration	min f(x)
1	5031.530	85	628.983	170	143.040
5	5014.920	90	628.982	175	139.885
10	4874.590	95	628.981	180	139.590
15	3818.510	100	628.980	185	139.545
20	696.029	105	628.979	190	139.536
25	641.001	110	628.973	195	139.517
30	632.439	115	628.940	200	139.506

35	631.379	120	628.731	205	139.501
40	631.238	125	628.173	210	139.498
45	631.234	130	627.043	215	139.489
50	631.233	135	621.336	220	139.486
55	631.231	140	595.380	225	139.481
60	631.219	145	520.485	230	139.476
65	631.168	150	481.336		
70	630.734	155	408.173		
75	629.015	160	243.556		
80	628.994	165	158.235		

Optimization terminated:
the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004 and f(x) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-004

b = (**21. 797 -0.167 -0.214)**

fval = **139.476**

Table 5.4: Comparison of solutions for multiple linear regression problem obtained using standard and GA solutions

Problem	Norm	Standard solution			GA solution		
		α	β	γ	α	β	γ
Angell's data	Least Square	21.800 (2.190)	-0.167 (0.055)	-0.214 (0.051)	21.797 (2.142)	-0.167 (0.049)	-0.214 (0.059)

The results obtained using least square techniques and GA are almost identical. The possible use of GA in data analysis and statistical inference are shown through this example. This GA algorithm can essentially be used when the assumptions of multiple linear regression models are not valid. For any linear model the GA technique is useful. Thus the GA, with the aid of a bootstrap like tool, may enable us to make statistical inference of any quantity of interest without severe restrictions on the models employed or the norms used in their estimation.

5.3 Problem 3: Nonlinear least squares regression
Given data (y_t and t, t = 1, 2, . . ., T) we are interested in the nonlinear least square estimate of α, β, γ of the model

$$y_t = \alpha + \beta \exp(-\gamma t) + \varepsilon_t$$

The dataset is the ratio of women's world record time to men's world record time at time t from 1923 (t = 1) to 1992 (t = 69) for 800 metres freestyle swimming. The results obtained standard procedure and GA are given in Table 5.

Table 5.5: Comparison of solutions for nonlinear least squares regression problem using standard and GA solutions

Problem	Norm	Standard solution			GA solution		
		α	β	γ	α	β	γ
Swimming ratio	Least square	0.953 (0.036)	0.246 (0.045)	0.014 (0.004)	0.954 (0.123)	0.245 (0.117)	0.014 (0.009)

The agreement of GA results and the results from the corresponding parametric theory is excellent.

4. Conclusions

GA is a dynamic stochastic optimization technique and its utility for a variety of research problems has been demonstrated in the seminar. GAs are of particular importance when one or more of underlying assumptions in a statistical model are not satisfied. Allthough a lot of theoretical work, particularly in engineering literature, has already being done, its application to solving agricultural research problems involving real data, is still a challenging task. Accordingly, it is highly desirable to make serious efforts in applying this optimization technique for solving various agricultural research problems.

References

Chatterjee, S. and Laudato, M. (1997). Genetic algorithms in statistics: procedures and applications. *Commun. Statist. Simula.*, **26** (4), 1617-1630.

Chatterjee, S., Laudato, M. and Lynch, L. A. (1996). Genetic algorithm and their statistical applications: an introduction. *Computl. Stat. Data Anal.*, **22**, 633-51.

Deb, K. (2004). Genetic algorithms for optimization. In “Statistical Computing. Ed. D. Kundu and A. Basu”. Narosa Publishing House, New Delhi, pp.85-123.

Fox, J. (1984). Linear statistical models and related methods. Wiley, New York.

Holland, J. H. (1975). Adaptation in natural and artificial systems. Univ. of Michigan Press, Ann Arbor. USA.

MATLAB (2005). Version 7.1. The MathWorks, Inc., USA