IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN: 2319-765X. Volume 16, Issue 1 Ser. I (Jan – Feb 2020), PP 04-21
www.iosrjournals.org

Derivative Free Optimization by Using Genetic Algorithm Method

Firaol Asfaw Wodajo
Debreberhan University, Debreberhan, Ethiopia

Abstract: *In this paper, derivative free optimization methods specifically Genetic Algorithm is discussed. The solution of bound-constrained optimization problems by using Genetic algorithm, the concept of the project is divided into five parts: The first part is the introduction part; under this statement of the problem and objective of the study are included. The second part is the preliminary which includes some lemma and theorems which are used in the body of the seminar paper. The part includes detail explanation of derivative free optimization methods specifically, Genetic Algorithm and Simulated Annealing and also supporting examples of them. The last part presents the summary of the study what was discussed in the main part of the paper. A Non linear Mathematical model is proposed and studied the combined effect of vertical Transmission (MTCT) and variable inflow of infective immigrants on the dynamics of HIV/AIDS: Vertical transmission means, Propagation of the disease from mother to children, 'Variable in flow of infective immigrants' includes both the aware and unaware infected immigrants. The equilibrium points of the model are found and the stability analysis of the model around. These equilibrium points are conducted. The stability analysis on the model shows that the disease free equilibrium point is locally asymptotically stable when the positive endemic equilibrium point is shown to be locally asymptotically stable Further it is shown that the basic reproduction number of the present model is greater than the one which is obtained from the model without vertical transmission. Through vertical transmission the disease flows from infected mother to children. That is, Vertical transmission contributes positively to the spread of the disease. Numerical simulation of the model is carried out to assess the effect of unaware HIV infective immigrants and vertical transmission (MTCT) in the spread of HIV/AIDS disease. The result showed that HIV infective immigrants and vertical transmission (MTCT) significantly affects the spread of the disease. Screening of the disease reduces the spread of HIV and also prevents mother to child transmission. It is well accepted that both vertical transmission and immigration contribute positively to the spread of the disease and these two parameters cannot be avoided in practice. Hence, the purpose of this study is to investigate the combined effect of vertical transmission, unaware and aware infected immigrants on the spread of*
HIV/AIDS and offers possible intervention strategies:

Keywords: *HIV/AIDS, Unaware and Aware Infective Immigrant, Vertical Transmission (MTCT), Screening, Local Stability, Reproduction Number*

Date of Submission: 20-12-2019 Date of Acceptance: 03-01-2020

I. Introduction

1.1. Background

Optimization is the process of adjusting the inputs or characteristics of a device, mathematical process, or experiment to find the minimum or maximum output or result. The input consists of variables; the process of function known as the cost function, objective function, or fitness function; and the output is the cost or fitness.

The aim of optimization is to determine the best suited solution to a problem under a given set of constraints. Several researchers over the decades have come up with different solutions to linear and nonlinear optimization problems. Mathematically, an optimization problem involves a fitness function describing the problem, under a set of constraints representing the solution space for the problem. Unfortunately, most of the traditional optimization techniques are centered around evaluating the first derivatives to locate the optima on a given constrained surface. Because of the difficulties in evaluating the first derivatives, to locate the optima for many rough and discontinuous optimization surfaces, in recent times, several derivative free optimization algorithms have been emerged. The optimization problem, nowadays, is represented as an intelligent search problem, where one or more agents are employed to determinethe optima on a search landscape, representing the constrained surface for the optimization problem. In the later quarter of the twentieth century, Holland.J.H.1973 [4] (Genetic algorithms and the optimal allocation of trials), pioneered a new concept on evolutionary search algorithms, and came up with a solution to the so far opened problem to nonlinear optimization problems. This work addresses the solution of bound-constrained optimization problems using algorithms that require only the availability of objective function values but not derivative information. We refer to these algorithms as derivative free algorithms. Fueled by a growing number of applications in science and engineering, the development of derivative free optimization algorithms has long been studied, and it has found renewed interest in recent times. The development of derivative free algorithms dates back to the works of Spendley, Nelder and Mead [8] with their simplex-based algorithms. Recent works on the subject have led to significant progress by providing convergence proofs, incorporating the use of surrogate models and offering. Given a set of points, derivative free optimization identifies the point with the best objective and builds a quadratic model by interpolating a selected subset of points. The resulting model is optimized within a trust region centered at the best point.

In recent *y*ears, some optimization methods that are conceptually different from the traditional mathematical programming techniques have been developed. These methods are labeled as modern or non-traditional methods of optimization. Most of these methods are based on certain
Characteristics and behavior of biological, molecular, and swarm of insects: The following methods are discussed in this work.

1. Genetic algorithms
2. Simulated annealing

Most these methods have been developed only in recent years and are emerging as popular methods for the solution of complex engineering problems. Most require only the function values (and not the derivatives). The genetic algorithm is based on the principles of natural genetics and natural selection. Simulated annealing is based on the simulation of thermal annealing of critically heated solids. Both genetic algorithms and simulated annealing are stochastic methods that can find the global minimum with a high probability and are naturally applicable for the solution of discrete optimization problems.

1.2. Statement of the problem

Most of the traditional optimization techniques are centered around evaluating the first derivatives to locate the optima on a given constrained surface. Because of the difficulties in evaluating the first derivatives to locate the optima for many rough and discontinuous optimization surfaces, it is found important to use derivative free optimization algorithms. Sometimes, it is difficult to solve optimization problems by using their derivatives. It may also be tedious to find the derivatives of complex functions. This work addresses the method by which we can solve optimization problems without using derivatives but by the available objective functions values. This seminar deals with genetic algorithm and simulated annealing.

1.3. Objectives

The main objective of this work is to solve optimization problems without using derivative information of the objective function.
Accordingly, the work goes through the following specific objective:

- To describe derivative free optimization methods: Genetic Algorithm and Simulated Annealing.
- To explore real life applications of these derivative free optimization methods.
- To illustrate these methods with examples.

II. Preliminaries Concepts

In this topic, we deal with some methods, theorems and concepts which are important for the study of derivative free optimization methods specifically Genetic Algorithms and Simulated Annealing.

Derivative-free optimization has experienced a renewed interest over the past decade that has encouraged a new wave of theory and algorithms. This seminar explores the properties of these algorithms. Here, focus of our work is the unconstrained optimization problem:

$min\{f(x): x \in R^n\}$, where $f: R^n \to R$ (3.1)

It may be noisy or non-differentiable and, in particular, in the case where the evaluation is of f is computationally expensive. These expensive optimization problems arise in science and engineering because evaluation of the function $f$often requires a complex deterministic simulation based on solving the equations (for example, non-linear eigenvalue problems, ordinary or partial differential equations) that describe the underlying physical phenomena. The computational noise associated with these complex simulations means that obtaining derivatives is difficult and unreliable. Moreover, these simulations often rely on legacy or proprietary codes and hence must be treated as black-box functions, necessitating a derivative-free optimization algorithm. Several comparisons have been made of derivative-free algorithms on noisy optimization problems that arise in applications. Performance profiles, introduced by Dolan and Mor'e (2007) [2] have proved to be an important tool for benchmarking optimization solvers. Dolan and Mor'e (2007) [2] define a benchmark in terms of a set P of benchmark problems, a set S of optimization solvers, and a convergence test *T*. Benchmarking derivative-free algorithms on selected application with trajectory plots provide useful information to users related applications. In particular, users can find the solver that delivers the largest reduction within a given computational budget. However, the conclusions in these computational studies do not readily extend to other applications. Most researchers have relied on a selection of problems from the collection of problems for their work on testing and comparing derivative-free algorithms. The performance data gathered in these studies is the number of function evaluations. The convergence test is sometimes related to the accuracy of the current iterate as an approximation to a solution, while in other cases it is related to a parameter in the algorithm. For example, a typical convergence test for trust region methods requires that the trust region radius be smaller than a given tolerance. Users with expensive function evaluations are often interested in a convergence test that measures the decrease in function value. We propose the convergence test

$$f(x_0) - f(x) > (1-\tau)(f(x_0) - f(L)), where \qquad \tau > 0 \qquad (3.2)$$

It is a tolerance, x_0 is the starting point for the problem, and f_L is computed for each problem as the smallest value of f obtained by any solver within a given number μ_f of function evaluations. This convergence test is tested for derivative-free optimization because it is invariant to the affine transformation $f \mapsto \alpha f + \beta\ (\alpha > 0)$ and measures the function value reduction

$f(x_0) - f(x)$ achieved by x relative to the best possible reduction $f(x_0) - f_L$. The convergence test (3.2) was used by Marazzi and Nocedal (2002) but f_L set to an accurate estimate of f at a local minimizer obtained by a derivative-based solver. Instead of using a fixed value ofτ, we use

$\tau = 10^{-k}$With k∈ {1,3,5,7} so that a user can evaluate solver performance for different levels of accuracy: The performance profiles are useful to users who need to choose a solver that provides a given reduction in function value within a limit of μ_f function evaluations.

Performance profiles were designed to compare solvers and thus use a performance ratio instead of the number of function evaluations required to solve a problem. As a result, performance profiles do not provide the percentage of problems that can be solved (for a given tolerance τ) with expensive optimization problems and thus an interest in the short-term behavior of algorithms.

III. Benchmarking Derivative-Free Optimization Solvers

Performance profiles, introduced by Dolan and More, have proved to be an important tool for benchmarking optimization solvers. Dolan and More define a benchmark in terms of a set P of benchmark problems, a set S of optimization solvers, and a convergence test T. Once these components of a benchmark are defined, performance profiles can be used to compare the performance of the solvers.

3.1 Performance profiles

Performance profiles are define in terms of a performance measure $t_{p,s} > 0$ obtained for each $p \in P$ and $s \in S$. For example, this measure could be based on the amount of computing time or the number of function evaluations required to satisfy the convergence test. Larger values of $t_{p,s}$ indicates worse performance. For any pair (p, s) of problem p and solvers, the performance ratio is defined by:

$$r_{p,s} = \frac{t_{p,s}}{min\{t_{p,s} : s \in S\}} \qquad (3.3)$$

Note that the best solver for a particular problem attains the lower bound $r_{p,s} = 1.$ The convention $r_{p,s} = \infty$ is used where the solvers fails to satisfy the convergence test on problem p.The performance profile of a solver $s \in S$ is defined as the fraction of problems where the performance ratio is at most α, that is,

$$\rho_s(\alpha) = \frac{1}{|P|}\, size\,\{p \in P : r_{p,s} \leq \alpha\} \qquad (3.4)$$

Where $|P|$denotes cardinality of P. Performance profile is the probability distribution for the ratio$r_{p,s}$. the performance profiles seek to capture how well the solver performs relative to the other solvers in S on the set of problems in$P$. Note in particular, that $\rho_s(1)$ is the fraction of problems for which solver sϵS performs the best and that for α sufficiently large,$\rho_s(\alpha)$ is the fraction problems solved by sϵS.In general,$\rho_s(\alpha)$ is the fraction of problems with a performance ratio $r_{p,s}$ bounded by α,and thus solvers with high values for $\rho_s(\alpha)$ are preferable. Benchmarking gradient-based optimization solvers is reasonably straightforward once the convergence test is chosen. The convergence test is invariably based on the gradient, for example,$\|\nabla f(x)\| \leq \tau\|\nabla f(x_0)\|$ for some $\tau > 0$ and norm $\|.\|$. This convergence test is augmented by a limit on the amount of computing time or the number of function evaluations. The latter requirement is needed to catch solvers is usually done with a fixed choice of tolerance τ that yields reasonably accurate solutions on the benchmark problems. The underlying assumption is that the performance of the solvers will not change significantly with other choices of the tolerance and that, in any case, users tend to be interested in solvers that can deliver high-accuracy solutions. In derivative-free optimization, however, users are interested in both low-accuracy and high-accuracy solutions. In particular, situations, when the evaluation of f is expensive, a low-accuracy solution is all that can be obtained within the user's computational budget. Moreover, in these situations, the accuracy of the data may warrant only a low-accuracy solution.

Benchmarking derivative-free solvers require a convergence test that does not depend on evaluation of the gradient. We propose to use the convergence test

$$f(x) \leq f_L + \tau(f(x_0) - f_L) \qquad (3.5)$$

Where $\tau > 0$ is a tolerance,x_0 is the starting point for the problem, and f_L is computed for each problem $p\epsilon P$ as the smallest value of f obtained by any solver within a given number μ_f of function evaluations. The convergence test *(3.5)* can also be written as:

$f(x_0) - f(x) \geq (1-\tau)(f(x_0) - f_L)$ and this shows that equation (3.5) requires that the reduction $f(x_0) - f(x)$ achieved by x be at least $1-\tau$ times the best possible reduction $f(x_0) - f_L$.The convergence test equation (3.5) was used by Elster and Neumaier [6] but with f_L set to an accurate estimate of f at a global minimizer. This test was also used by Marazzi and Nocedal [7] but with f_L set to an accurate estimate of f at a local minimizer obtained by a derivative-based solver. Setting f_L to an accurate estimate of f at a local a minimizer is not appropriate when the evaluation of f is expensive because no solver may be able to satisfy equation *(3.5)* within the user's computational budget. Even for problems with a cheap$f$, a derivative-free solver is not likely to achieve accuracy comparable to a derivative-based solver. On the other hand, if f_L is the smallest value of f obtained by any solver, then at least one solver will satisfy equation *(3.5)* for any $\tau \geq 0.$ An advantage of equation *(3.4)* is that it is invariant to the affine transformation f$\mapsto \alpha f + \beta$ where α$>$ 0. Hence, we can assume, for example, that f_L=0 and $f(x_0)$ =1. There is no loss in generality in this assumption because derivative-free algorithms are invariant to the affine transformation $f \mapsto \alpha f + \beta$. In deed; algorithms for gradient-based optimization (unconstrained and constrained) problems are also invariant to this affine transformation. The tolerance $\tau\epsilon[0,1]$ in equation *(3.5)* represents the percentage decrease from the starting value $f(x_0)$. A value of $\tau = 0.1$ may represent a modest decrease,a reduction that is 90% of the total possible, while smaller values of τ decreases,the accuracy of $f(x)$ as an approximation to some minimizer depends on the growth of f in a neighborhood of the minimizer. As noted, users are interested in the performance of derivative-free solvers for both low-accuracy and high-accuracy solutions. A user's expectation of decrease possible within their computational budget will vary from application to application. The following new result relates the convergence test equation (3.5) to convergence results for gradient-based optimization solvers.

Theorem 3.2

Assume that $f: R^n \mapsto R$ is a strictly convex quadratic and that x^* is the unique minimizer of f. If $f_L = f(x^*)$,then,$x \in R^n$ satisfies the convergence test equation *(3.5)* if and only if

$$\|\nabla f(x)\|_* \leq \tau^{\frac{1}{2}}\|\nabla f(x_0\|_* \qquad (3.6)$$

For the norm$\|.\|_*$ defined by: $\|v\|_* = \left\|G^{\frac{-1}{2}}v\right\|_2$ v

And G is the Hessian matrix of f and G is the Hessian matrix of f, and x^* is unique minimizer,

$$f(x) = f(x^*) + \frac{1}{2}(x - x^*)^T G(x - x^*) \qquad (3.7)$$

Hence, the convergence test (3.5) holds if and only if:

$$(x - x^*)^T G(x - x^*) \leq \tau(x_0 - x^*)^T G(x_0 - x^*)$$

Which is in terms of the square root $G^{\frac{1}{2}}$ is just:

$$\left\|G^{\frac{1}{2}}(x - x^*\right\|_2^2 \leq \tau \left\|G^{\frac{1}{2}}(x_0 - x^*)\right\|_2^2 \qquad (3.8)$$

We obtain equation (3.6) by noting that since x^* is the minimizer of the quadratic f and G is the Hessian matrix, $\nabla f(x) = G(x - x^*)$. Other variations on theorem 3.1 are of interest. For example, it is not difficult to show, by using the same proof techniques that equation *(3.3)* is also equivalent to:

$$\frac{1}{2}\|\nabla f(x)\|_*^2 \leq \tau(f(x_0) - f(x^*)) \qquad (3.9)$$

This inequality shows, in particular, that we can expect that the accuracy of x, as measured by the gradient norm$\|\nabla f(x)\|_*$, to increase with the square root of $f(x_0) - f(x)^*$. Similar estimates hold for the error in x because $\nabla f(x) = G(x - x^*)$. thus, in view of (3.6), the convergence test (3.5) is equivalent to:

$\|x - x^*\|_\diamond \leq \tau^{\frac{1}{2}}\|x_0 - x^*\|_\diamond$ Where the norm $\|.\|_\diamond$ is defined by:$\|v\|_\diamond = \left\|G^{\frac{1}{2}}\right\|_2$

In this case, the accuracy of x in the $\|.\|_\diamond$ norm increases with the distance of x_0 from x^* in the $\|.\|_\diamond$ norm. We now explore an extension of theorem (3.1) to nonlinear functions that are valid for an arbitrary starting point x_0.The following result shows that the convergence test (3.5).

Lemma 3.2
If $f{:}R^n \to R$ is twice continuously differentiable in a neighborhood of a minimizer $x^*$with $\nabla^2 f(x^*)$ positive definite, then
$$lim_{x\to x^*} \frac{f(x)-f(x^*)}{\|\nabla f(x)\|_*^2} = \frac{1}{2} \qquad (3.10)$$
Where the norm $\|.\|_*$ is defined in (3.2) and $G = \nabla^2 f(x^*)$.

Proof:
We first proof that:
$$lim_{x\to x^*} \frac{\left\|\nabla^2 f(x^*)^{\frac{1}{2}}(x-x^*)\right\|}{\|\nabla f(x)\|_*} = 1. \qquad (3.11)$$
This result can be established by noting that since $\nabla^2 f$ is continuous at $x^*$and $\nabla f(x^*) = 0, \nabla f(x) = \nabla^2 f(x^*)(x-x^*) + r_1(x), r_1(x) = o(\|x-x^*\|)$. If λ_1 is the smallest eigenvalue of $\nabla^2 f(x^*)$, then, this relationship implies, in particular that
$$\|\nabla f(x)\|_* \geq \frac{1}{2}\lambda^{\frac{1}{2}}\|x-x^*\| \qquad (3.12)$$
For all x near x^*.This inequality and the previous relationship prove.
We can now proof by noting that since $\nabla^2 f$ is continuous at $x^*$and
$$\nabla f(x^*) = 0, f(x) = f(x^*) + \frac{1}{2}\left\|\nabla^2 f(x)(x)^{\frac{1}{2}}(x-x^*)^2\right\| + r_2(x), r_2(x) = o(\|x-x^*\|^2).$$
This relationship together with (3.11) and (3.12) complete the proof.
Lemma (3.1) shows that there is a neighborhood (x^*) of x^*such that if $x \in \mathcal{N}(x^*)$ satisfies the convergence test (3.5) with $f_L = f(x^*)$, then,
$$\|\nabla \mathrm{f(x)}\|_* \leq \Upsilon\tau^{\frac{1}{2}}(f(x_0)-f(x^*))^{\frac{1}{2}} \qquad (3.13)$$
where the constant $\Upsilon$is a slight overestimate of $2^{\frac{1}{2}}$. Conversely, if Υ is a slight underestimate of $2^{\frac{1}{2}}$, then (3.13) implies that (3.5) holds in some neighborhood of x^*. Thus, in this sense, the gradient (3.13) is asymptotically equivalent to (3.5) for smooth functions.

IV. Genetic Algorithms

4.1 Basic Concepts of GA

Genetic Algorithms (GAs) are the main paradigm of evolutionary computing. G As are inspired by Darwin's theory about evolution the ''survival of the fittest". In nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

GAs are the ways of solving problems by mimicking processes nature uses; i.e. selection, crosses over, Mutation and Accepting, to evolve a solution to a problem.

GAs are adaptive heuristic search based on the evolutionary ideas of natural selection and genetics. GAs are intelligent exploitation of random search used in optimization problems.

GAs, although randomized, exploit historical information to direct the search into the region of better performance within the search space.

4.2 Biological Background

Basic Genetics

- Every organism has a set of rules, describing how that organism is built. All living organism consist of cells.
- In each cell there is some set of chromosomes. Chromosomes are strings of DNA and serve as a model for the whole organism.
- A chromosome consists of genes, blocks of DNA.
- Each gene encodes a particular protein that represents a trait (feature), e.g., colour of eyes.
- Possible settings for a trait (e.g., blue, brown) are called alleles.
- Each gene has its own position in the chromosome called locus.
- Complete set of genetic material (all chromosomes) is called a genome.
- When two organisms mate they share their genes; the resultant offspring may end up having half the genes from one parent and half from the other. This process is called recombination (crossover).

The new created offspring can then be mutated. Mutation means, that the elements of DNA are a bit changed. These changes are mainly caused by errors in copying genes from parents.

4.2.1. Working Principles
Before getting into GAs, it is necessary to explain few terms.

- Chromosome: a set of genes; a chromosomes contains the solution in form of genes.
- Gene: a part of chromosome; a gene contains a part of solution. It determines the solution. It determines the solution. E.g. 16743 is a chromosome and 1,6,7,4, and 3 are its genes.
- Individual: same as chromosome.
- Population: number of individuals present with the same length of chromosome.
- Fitness: the value assigned to an individual based on how far or close an individual is from the solution; greater the fitness value better the solution it contains.
- Fitness function: a function that assigns fitness value to the individual.
- Breeding: taking two fit individuals and then intermingling their chromosome to create new two individuals.
- Mutation: changing a random gene in an individual.
- Selection: selecting individuals for creating the next generation.

GA begins with a set of solutions (represented by chromosomes) called the population.
Solutions from one population are taken and used to form a new population. This is motivated by the possibility that the new population will be better than the old one. Solutions are selected according to their fitness to form new solutions (offspring); more suitable they are, more chances they have to reproduce. This is repeated until some condition (e.g. number of populations or improvement of the best solution) is satisfied.

4.2.2. Outline of the basic Genetic Algorithm
1. [Start] Generate random population of n chromosomes (i.e. suitable solutions for the problem).
2. [Fitness] Evaluate the fitness $f(x)$ of each chromosome x in the population.
3. [New population] Create a new population by repeating the following steps until the new population is complete.
(a) [Selection] Select two parent chromosomes from a population according to their fitness (better the fitness, bigger the chance to be selected)
(b) [Crossover] with a crossover probability, cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.
(c) [Mutation] with a mutation probability, mutate new offspring at each locus (position in chromosome).
(d) [Accepting] place new offspring in the new population
4. [Replace] Use new generated population for a further run of the algorithm
5. [Test]. If the end condition is satisfied, stop, and returns the best solution in current population.
6. [Loop] Go to step 2.
Note: The genetic algorithms performance is largely influenced by two operators called cross over and mutation. These two operators are the most important parts of GA.

4.2.3. Encoding
Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form so that a computer can process. One common approach is to encode solutions as binary strings: sequences of 1's and 0's, where the digit at each position represents the value of some aspect of the solution. For Example, a Gene represents some data (eye colour, hair colour, sight, etc.).
A chromosome is an array of genes. In binary form a Gene looks like: (11100010)
A Chromosome looks like: Gene 1 Gene 2 Gene 3 Gene 4
(11000010, 00001110, 001111010, 10100011)
A Chromosome should in some way contain information about solution which it represents; it thus requires encoding. The most popular way of encoding is a binary string like:
Chromosome 1: 1101100100110110
Chromosome 2: 1101111000011110
Each bit in the string represents some characteristics of the solution. There are many other ways of encoding, e.g., encoding values as integer or real numbers or some permutations and so on. The virtue of these encoding methods depends on the problem to work on.

4.2.3.1. Binary Encoding
Binary Encoding is the most common to represent information contained. In GAs, it was first used because of its relative simplicity. In binary encoding, every chromosomes is a string of bits: 0 or 1, like:
Chromosome 1:1 0 1 1 0 0 1 0 11 00101011100101
Chromosome 2:111111100000110000011111

Binary encoding gives many possible chromosomes even with a small number of allele's i.e. possible settings for a trait (features).This encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation.

Example 4.3.1
One variable function, say 0 to 15 numbers, numeric values, represented by 4 bit binary string:

Table 1.One variable function, represented by 4 bit binary string.

Numeric value	4-bit string	Numeric value	4-bit string	Numeric value	4-bit string
0	0000	6	0110	12	1100
1	0001	7	0111	13	1101
2	0010	8	1000	14	1110
3	0011	9	1001	15	1111
4	0100	10	1010		
5	0101	11	1011		

Example 4.3.2
Two variable functions represented by 4 bit string for each variable:
Let two variables x_1,x_2 (1011, 0110).
Every variable will have both upper and lower limit as $X_1^L \quad X_1 \quad X_1^U$
Because 4-bit string can represent integers from 0 to 15, so, (0000 0000) and (1111 1111) represent the points for x_1,x_2 as (X_1^L, X_2^L) and (X_1^U , X_2^U) respectively.
Thus, an n- bit string can represent integer from 0 to$2^n - 1$, i.e. 2^n integers.
Binary Coding Equivalent integer Decoded binary substring

2	10
2	5
2	2
	1

Remainder 1 0 1 0

Of length n_i.Then, decoded binary
0
1
0

Let X_I be coded as a substring S_i

substring S_i as K=$n_i - 1$
$2^K S_K$, K=0 whereS_i can be 0 or 1
and the string S is represented as
$S_{n-1} \ldots S_3 S_2 S_1 S_0$

4.3.3. Decoding Value
Consider a 4-bit string (0111) the decoded value is equal to $2^3.0 + 2^2.1 + 2^1.1 + 2^0.1$=7 knowing X_i^L and X_i^U corresponding to (0000) and (1111), the equivalent value for any 4-bit string can be obtained as

$$X_i = X_i^L + \frac{(X_i^U - X_i^L)}{(2^{ni} - 1)} \times \text{(decoded value of string)}$$

For example, a variable X_i; let $X_i^L = 17$, find what value the 4-bit string $X_i = (1010)$ would represent. First get decoded value for $S_i = 1010 = 2^3.0 + 2^2.0 + 2^1.1 + 2^0.0 = 10$ then,

$$X_i = 2 + \frac{(17-2)}{(2^4-1)} \times 10 = 12 \text{ v}$$

The accuracy obtained with a 4-bit code is 1/16 of search space. By increasing the string length by 1-bit, accuracy increases to 1/32.

4.3.4. Value Encoding
The value encoding can be used in problems where values such as real numbers are used. Use of binary encoding for this type of problems would be difficult.
1. In value encoding, every chromosome is a sequence of some values.
2. The values can be anything connected to the problem, such as: real numbers, characters or objects.
For example: Chromosome A 1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B ABCDJEIFJDHDIERJFDLDFLFEGT
Chromosome C (back), (back),(right),(forward),(left)
3. Value encoding is often necessary to develop some new types of crossovers and mutations specific for the problem.

4. 3. 5. Permutation Encoding
Permutation encoding can be used in ordering problems, such as travelling salesman problem or task ordering problem.
1. In permutation encoding, every chromosome is a string of number that represent a position in a sequence.

Chromosome	A	1 5 3 2 6 4 7 9 8
Chromosome	B	8 5 6 7 2 3 1 4 9

2. Permutation encoding is useful for ordering problems. For some problems, crossover and mutation corrections must be made to leave the chromosome consistent.

4.3.6. Operator of G A
Genetic operators used in genetic algorithms maintain genetic diversity. Genetic diversity or variation is a necessity for the process of evolution. Genetic operators are analogous to those which occur in the natural world: Reproduction (or Selection), Crossover (or Recombination); and Mutation. In addition to these operators, there are some parameters of GA. One important parameter is population size. Population size says how many chromosomes are in population (in one generation).If there are only few chromosomes, then GA would have a few possibilities to perform crossover and only a small part of search space is explored. If there are many chromosomes, then GA slows down.

4.3.7. Reproduction or Selection
Reproduction is usually the first operator applied on population. From the population, the chromosomes are selected to be parents to crossover and produce offspring. According to Darwin's evolution theory'' survival of the fittest'', the best ones should survive and create new offspring. The reproduction operators are also called Selection operators. Selection means extract a subset of genes from an existing population, according to any definition of quality. Every gene has a meaning, so one can derive from the gene a kind of quality measurement called fitness function. Following (fitness value), selection can be performed. Fitness function quantifies the optimality of a solution (chromosome) so that a particular solution may be ranked against all the other solutions. The function depicts the closeness of a given 'solution' to the desired result. Many reproduction operators exists and they all essentially do same thing. They pick from current population the strings of above average and insert their multiple copies in the mating pool in a probabilistic manner. The most commonly used methods of selecting chromosomes for parents to crossover are: Roulette wheel selection, Boltzmann selection, tournament selection, Rank selection, Steady state selection. The Roulette wheel and Boltzmann selections methods are illustrated next.

4.3.7.1 Roulette Wheel Selection (Fitness- proportionate Selection)
Roulette-wheel selection, also known as Fitness proportionate Selection, is a genetic operator, used for selecting potentially useful solutions for recombination. In fitness-proportionate selection: the chance of an individual's being selected is proportional to its fitness, greater or less than its competitors' fitness. Conceptually, this can be thought as a game of Roulette.

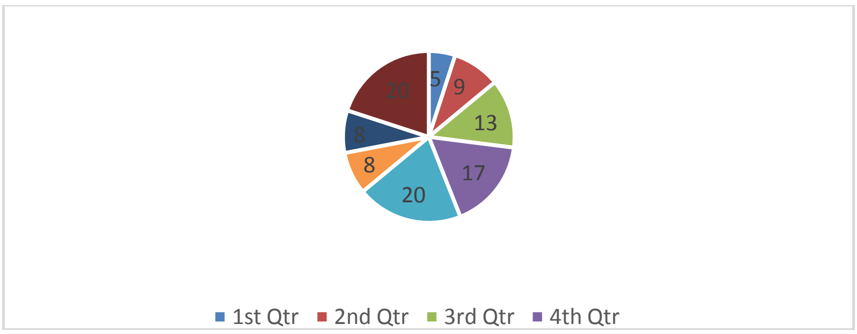


Figure 1.Roulette-wheel shows 8 individual with fitness

The Roulette-wheel simulates 8 individuals with fitness values F_i , marked at its circumference ; e.g., the 5^{th} individual has a higher fitness than others, so the wheel would choose the 5^{th} individual more than other individuals. The fitness of the individuals is calculated as the wheel is spun n=8 times, each time selecting an instance, of the string, chosen by the wheel pointer. Probability of i^{th} string is $p_i = \frac{F_i}{\sum_{j=1}^{n} F_j}$, where

$n =$ Number of individuals, called population size;
$P_i =$ probability of i^{th} string being selected

F_i= fitness for i^{th} string in the population. Because the circumference of the wheel is marked according to a string's fitness, the Roulette-wheel mechanism is expected to make $\frac{F}{\bar{F}}$ copies of the i^{th} string.
Average fitness= $\bar{F}\, F_j/_n$; Expected count = (n=8)xP_i ; Cumulative probability= ${}^{N=5}_{i=1}P_i$

Example 4.3
Evolutionary Algorithms is to maximize the function $f(x) = x^2$ with x in the integer interval
[0, 31], i.e., x =0, 1,….,30, 31.
1. The first step is encoding of chromosomes; use binary representation for integers; 5-bits are used to represent integers up to 31.
2. Assume that the population size is 4.
3. Generate initial population at random. They are chromosomes or genotypes;
E.g. 01101, 11000, 01000, 10011
4. Calculate fitness value for each individual.
(a) Decode the individual into an integer (called phenotypes),
01101 13; 11000 24; 01000 8; 10011 19;
(b) Evaluate the fitness according to $f(x) = x^2$, 13 169; 24 576; 8 64; 19 361.
5. Select parents (two individuals) for crossover based on their fitness in p_i.Out of many methods for selecting the best chromosomes, if roulette-wheel selection is used, then the probability of the i^{th} string in the population is $p_i = \frac{F_i}{{}^{n}_{(j=1}F_j)}$, $p_i = \frac{F_i}{\sum_{j=1}^{n} F_j}$, where
F_i is fitness for the string i in the population, expressed as $f(x)$
P_i is probability of the string i being selected,
n is number of individuals in the population, is population size, $n = 4$
$n * P_i$ is expected count

Table 2.Data for maximum chance of selection

String No	Initial population	x value	Fitness $F_i f(x) = x^2$	P_i	Expected count N* prob$i$
1	0 1 1 0 1	13	169	0.14	0.58
2	1 1 0 0 0	24	576	0.49	1.97
3	0 1 0 0 0	8	64	0.06	0.22
4	1 0 0 1 1	19	361	0.31	1.23
Sum(Total)			1170	1.00	4.00
Average			293	0.25	1.00
Max			576	0.49	1.97

The string number 2 has maximum chance of selection.

4.4. Boltzmann Selection

Simulated annealing is a method used to minimize or maximize a function.

- This method simulates the process of slow cooling of molten metal to achieve the minimum function value in a minimization problem.
- The cooling phenomenon is simulated by controlling a temperature like parameter introduced with the concept of Boltzmann probability distribution.
- The system in thermal equilibrium at a temperature T has its energy distribution based on the probability defined by $(E) = exp(\frac{-E}{KT})$, where K is the Boltzmann constant.
- This expression suggests that a system at a higher temperature has almost uniform probability at any energy state, but at lower temperature it has a small probability of being at a higher energy state.
- Thus, by controlling the temperature T and assuming that the search process follows Boltzmann probability distribution, the convergence of the algorithm is controlled.

4.5. Crossover

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user- definable crossover probability. Crossover selects genes from parent chromosomes and creates a new offspring. The crossover operators are of many types. One simple way is, One-point crossover. The others are Two point, Uniform, and Arithmetic crossovers. The operators are selected based on the way chromosomes are encoded.

One Point Crossover

One-point crossover operator randomly selects one crossover point and then copy everything before this point from the first parent and then everything after the crossover point copy from the second parent. The Crossover would then look as shown below.

Consider the two parents selected for crossover.

Parent 1 1 1 0 1 1   0 01 0 0 1 1 0 1 1 0

Parent 2:   1 1 0 1 1   1 1 0 0 0 0 1 1 1 1 0

Interchanging the parents chromosomes after the crossover points, the offspring produced are:

Offspring: 1 1 1 0 1 1| 1 1 0 0 0 0 1 1 1 1 0

Offspring: 2 1 1 0 1 1| 0 0 1 0 0 1 1 0 1 1

Note: The symbol, a vertical line,| is the chosen crossover point.

Two-point crossover

Two-point crossover operator randomly selects two crossover points within chromosomes then interchanges the two parent chromosomes between these points to produce two new offspring.

Consider the two parents chromosomes between the crossovers points. The Offspring produced are: Offspring 1   1 1 0 1 1 | 0 0 1 0 0 1 1 | 0 1 1 0 ,

Offspring 2   1 1 0 1 1 | 0 0 1 0 0 1 1| 0 1 1 0

Arithmetic Crossover

Arithmetic crossover operator linearly combines two parent chromosome vectors to produce two new offspring according to the equations:

Offspring 1 = a * parent 1+ (1- a) * parent 2

Offspring 2 = (1-a) * parent 1 + a* parent 2

Where a is a random weighting factor chosen before each crossover operation. Consider two parents (each of 4 float genes) selected for crossover:

Parent 1   (0.3)   (1.4) (0.2) (7.4)

Parent 2   (0.5)   (4.5)   (0.1)   (5.6)

Applying the above two equations and assuming the weighting factor a=0.7, applying above equation, we get two resulting offspring. The possible set of offspring after arithmetic crossover would be:

Offspring 1: (0.36)   (2.33)   (0.17)   (6.86)

Offspring 2: (0.44)   (3.57)   (0.13)   (6.14)

4.6. Mutation

After a crossover is performed, mutation takes place. Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next. Mutation occurs during evolution according to a user-definable mutation probability, usually set to fairly low value, say 0.01 a good first choice. Mutation alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With the new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search, helps to prevent the population from stagnating at any local optima. Mutation is intended to prevent these arches falling into local optimum of the state space. The mutation operators are of much type.

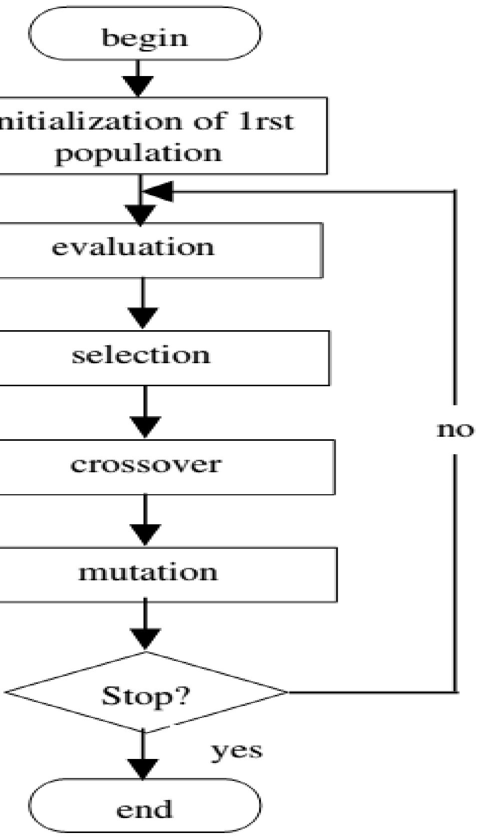


Figure 2: Flow chart of GA

4.7. Some examples and applications of GA

GA is applicable in different optimization problems. Here, we try to solve some optimization problems by using GA.

Example 4.4 Minimizing Rastrigin's function

Rastrigin's Function

This section presents an example that shows how to find the minimum of Rastrigin's function, a function that is often used ti test the genetic algorithm.

For two independent variables, Rastrigin's function is defined as

$Ras(x) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$

Global Optimization Tool box software contains the `rastriginsfcn.m file, which` computes the values of Rastrigin's function. The following figure shows a plot of Rastrigin's function.

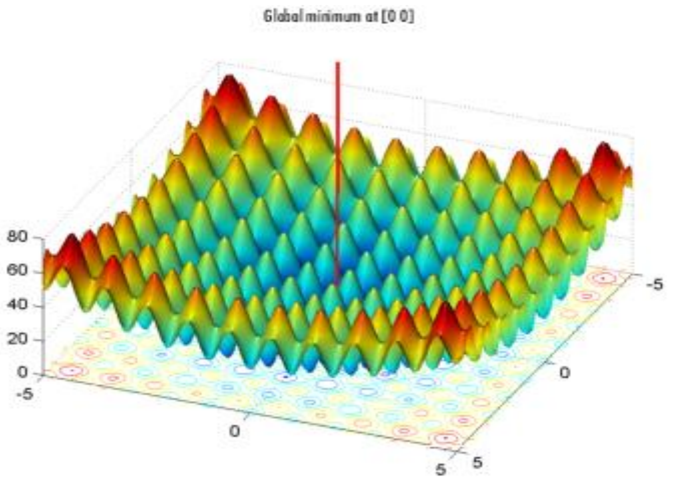


Figure 3. Graph of Rastrigin's function

As the plot shows, Rastrigin's function has many local minima-the "valleys" in the plot. However, the function has just one global minimum, which occurs at the point [0 0] in the *x-y* plane, as indicated by the vertical line in the plot, where the value of the function is 0. At any local minimum other than [0 0], the value of Rastrigin's function is greater than 0. The farther the local minimum is from the origin, the larger the value of the function is at that point.

Rastrigin's function is often used to test the genetic algorithm, because its many local minima make it difficult for standard, gradient-based methods to find the global minimum.

The following contour plot of Rastrigin's function shows the alternating maxima and minima.

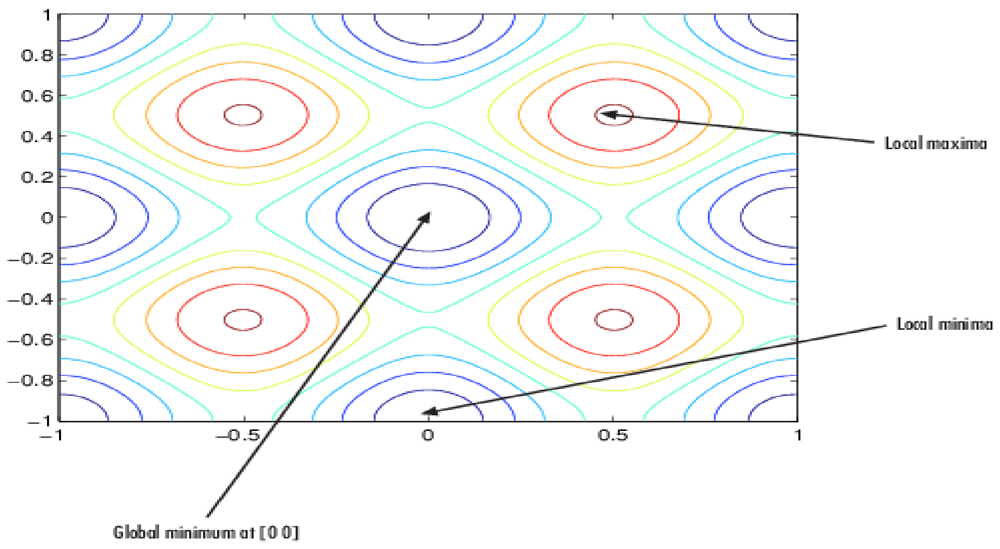


Figure 4. Alternating maxima and minima of Rastrigin's Function

4.8. Finding the Minimum of Rastrigin's Function

This section explains how to find the minimum of Rastrigin's function using the genetic algorithm.

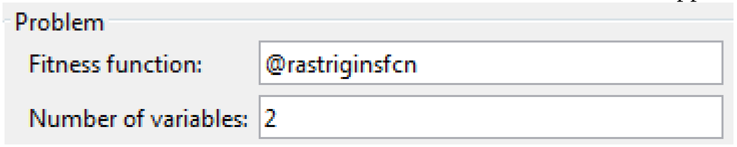
Note: Because the genetic algorithm uses random number generators, the algorithm returns slightly differently results each time you run it.

To find the minimum, do the following steps:

1. Enter `optimtool('ga')` at the command line to open the Optimization app.
2. Enter the following in the Optimization app:

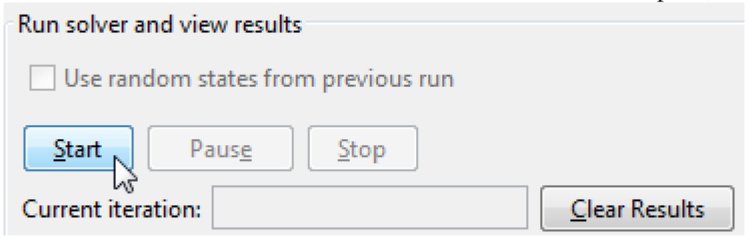
- In the **Fitness function** field, enter `@rastriginsfcn`.
- In the **Number of variables** field, enter `2`, the number of independent variables for Rastrigin's function.

The **Fitness function** and **Number of variables** fields should appear as shown in the following figure.



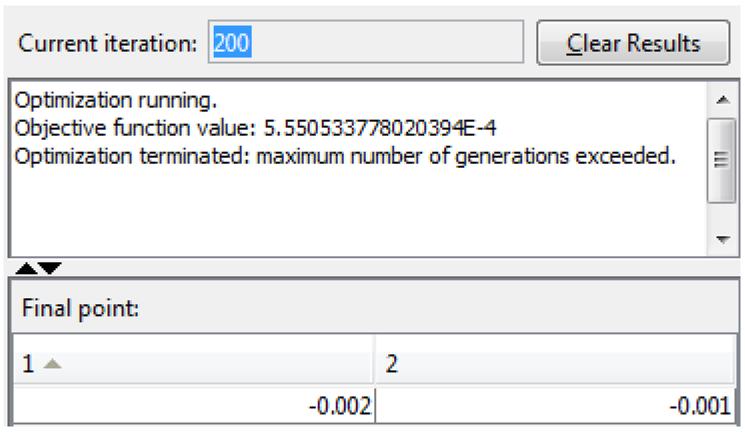
Fitness function:	@rastriginsfcn
Number of variables:	2

3. Click the **Start** button in the **Run solver and view results** pane, as shown in the following figure.


Run solver and view results
Use random states from previous run
Start
Pause
Stop
Current iteration:
Clear Results

While the algorithm is running, the **Current iteration** field displays the number of the current generation. You can temporarily pause the algorithm by clicking the **Pause** button. When you do so, the button name changes to **Resume**. To resume the algorithm from the point at which you paused it, click **Resume**.

When the algorithm is finished, the **Run solver and view results** pane appears as shown in the following figure. Your numerical results might differ from those in the figure, since ga is stochastic.


Current iteration: 200

Clear Results

```
Optimization running.
Objective function value: 5.550533778020394E-4
Optimization terminated: maximum number of generations exceeded.
```

Final point:

1	2
-0.002	-0.001

The display shows:

- The final value of the fitness function when the algorithm terminated:

```
Objective function value: 5.550533778020394E-4
```

Note that the value shown is very close to the actual minimum value of Rastrigin's function, which is `0`. Setting the Initial Range, Setting the Amount of Mutation, and Set Maximum Number of Generations describe some ways to get a result that is closer to the actual minimum.

- The reason the algorithm terminated.

```
Optimization terminated: maximum number of generations exceeded.
```

- The final point, which in this example is `[-0.002 -0.001]`.

Finding the Minimum from the Command Line

To find the minimum of Rastrigin's function from the command line, enter

```
rng(1,'twister') % for reproducibility
[xfvalexitflag] = ga(@rastriginsfcn, 2)
```

This returns

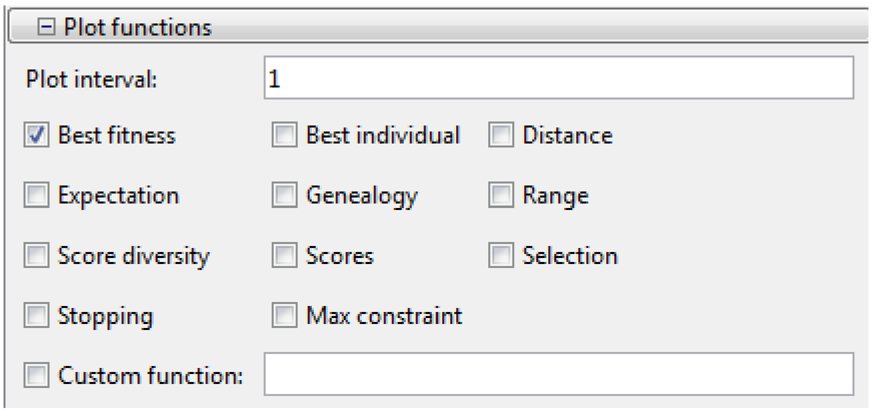
```
Optimization terminated:
average change in the fitness value less than options.FunctionTolerance.

x =
   -1.0421   -1.0018

fval =
    2.4385

exitflag = 1
```

- `x` is the final point returned by the algorithm.
- `fval` is the fitness function value at the final point.
- `exitflag` is integer value corresponding to the reason that the algorithm terminated.

Note: Because the genetic algorithm uses random number generators, the algorithm returns slightly different results each time you run it.

Displaying Plots

The Optimization app **Plot functions** pane enables you to display various plots that provide information about the genetic algorithm while it is running. This information can help you change options to improve the performance of the algorithm. For example, to plot the best and mean values of the fitness function at each generation, select the box next to **Best fitness**, as shown in the following figure.



⊟ Plot functions

Plot interval: 1

[✓ Best fitness] ☐ Best individual ☐ Distance

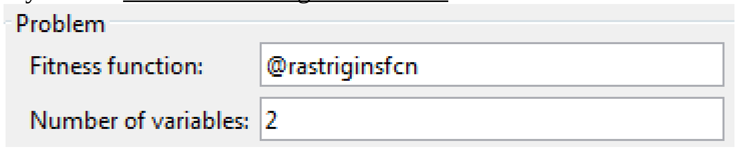
☐ Expectation ☐ Genealogy ☐ Range

☐ Score diversity ☐ Scores ☐ Selection

☐ Stopping ☐ Max constraint

☐ Custom function:

When you click **Start**, the Optimization app displays a plot of the best and mean values of the fitness function at each generation.
Try this on Minimize Rastrigin's Function:



Problem

Fitness function: @rastriginsfcn

Number of variables: 2

When the algorithm stops, the plot appears as shown in the following figure.

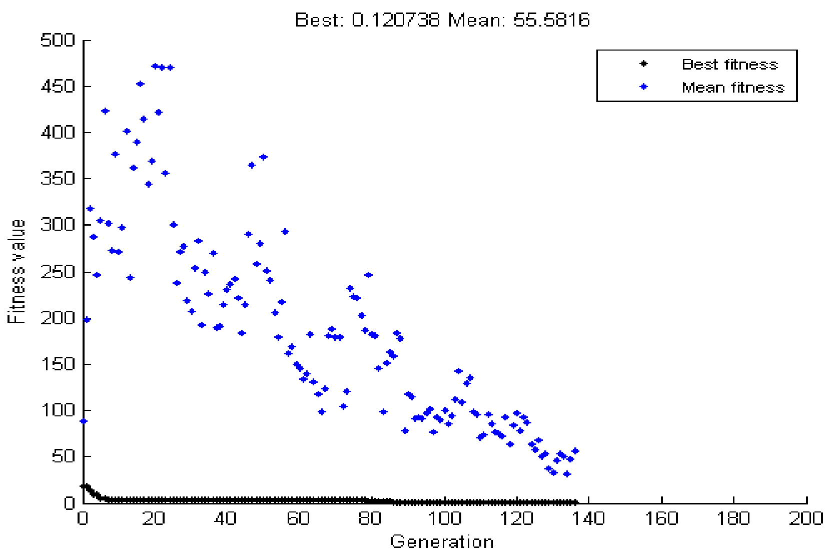
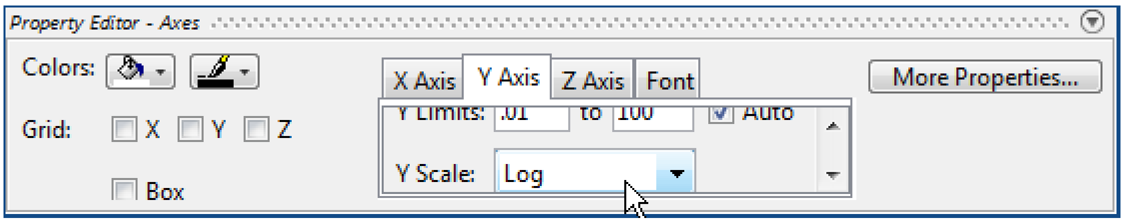


Figure 5.Best and Mean fitness of Rastrigin's Function

The points at the bottom of the plot denote the best fitness values, while the points above them denote the averages of the fitness values in each generation. The plot also displays the best and mean values in the current generation numerically at the top.
To get a better picture of how much the best fitness values are decreasing, you can change the scaling of the *y*-axis in the plot to logarithmic scaling. To do so,
1. Select **Axes Properties** from the **Edit** menu in the plot window to open the Property Editor attached to your figure window as shown below.



Y Limits: .01 to 100 [✓ Auto]

Y Scale: Log

2. Click the **Y Axis** tab.
3. In the **Y Scale** pane, select **Log**.
The plot now appears as shown in the following figure.

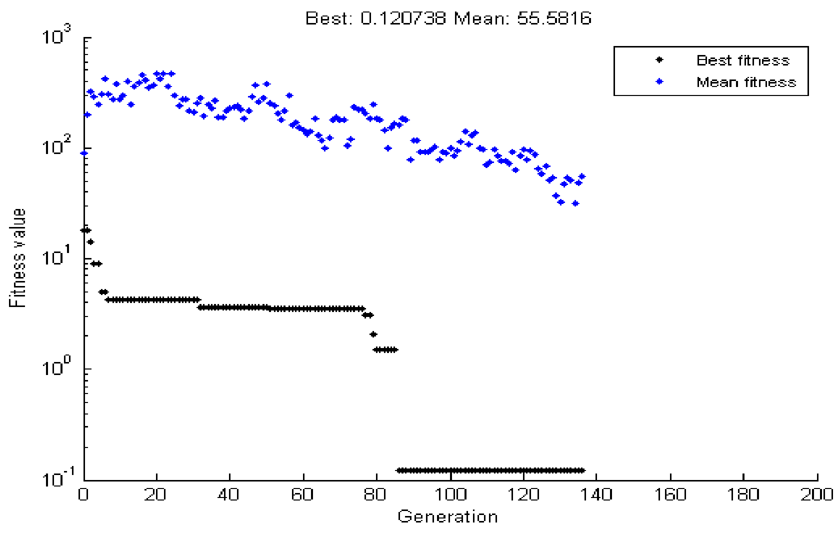


Figure 6: optional graph of Best and Mean fitness of Rastrigin's Function

Typically, the best fitness value improves rapidly in the early generations, when the individuals are farther from the optimum. The best fitness value improves more slowly in later generations, whose populations are closer to the optimal point.

Example 4.5
Constrained Minimization Using GA
Suppose you want to minimize the simple fitness function of two variables x_1 andx_2.
$\min f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$
subject to the following nonlinear inequality constraints and bounds
$x_1.x_2 + x_1 - x_2 + 1.5 \leq 0$ (nonlinear constraint)
$10 - x_1.x_2 \leq 0$ (nonlinear constraint)
$0 \leq x_1 \leq 1$ (bound)
$0 \leq x_2 \leq 13$ (bound)
Begin by creating the fitness and constraint functions. First, create a file named simple_fitness.m as follows:
function y = simple_fitness(x)
y = 100*(x(1)^2 - x(2))^2 + (1 - x(1))^2;
(simple_fitness.m ships with Global Optimization Toolbox software.)
The genetic algorithm function,ga, assumes the fitness function will take one input **x**, where **x**has as many elements as the numberof variables in the problem. The fitness function computes the value of the function and returns that scalar value in its one return argument,**y.** Begin by creating the fitness and constraint functions. First, create a file named **simple_constraint.m,** as follows:
function [c, ceq] = simple_constraint(x)
c = [1.5 + x(1)*x(2) + x(1) - x(2);...-x(1)*x(2) + 10];
ceq = [];
(simple_constraint.mships with Global Optimization Toolbox software.)
The ga function assumes the constraint function will take one input x, where x has as many elements as the number of variables in the problem. The constraint function computes the values of all the inequality and equality constraint and returns two c and ceq respectively.

To minimize the fitness function, you need to pass a function handle to the fitness function as the first argument to the ga function, as well as specifying the number of variables as the second argument. Lower and upper bounds are provided as LB and UB respectively. In addition, you also need to pass a function handle to the nonlinear constraint function.

```
ObjectiveFunction = @simple_fitness;
nvars = 2; % Number of variables
LB = [0 0]; % Lower bound
UB = [1 13]; % Upper bound
ConstraintFunction = @simple_constraint;
rng(1,'twister') % for reproducibility
[x,fval] = ga(ObjectiveFunction,nvars,...
[],[],[],[],LB,UB,ConstraintFunction)
Optimization terminated: average change in the fitness value less than options.TolFun
and constraint violation is less than options.TolCon.
x =
0.8123 12.3137
fval =
1.3581e+04
```

The genetic algorithm solver handles linear constraints and bounds differently from nonlinear constraints. All the linear constraints and bounds are satisfied throughout the optimization. However, ga may not satisfy all the nonlinear constraints at every generation. If ga converges to a solution, the nonlinear constraints will be satisfied at that solution.

ga uses the mutation and crossover functions to produce new individual at every generation. ga satisfies linear and bound constraints by using mutation and crossover functions that only generate feasible points. For example, in the previous call to ga, the mutation function mutationguassiandoes not necessarily obey the bound constraints. So, when there are bound or linear constraints, the default ga mutation function is mutationadaptfeasible. If you provide a custom mutation must only generate points that are feasible with respect to the linear and bound constraints. All the included crossover functions generate points that satisfy the linear constraints and bounds except the crossoverheuristic function. To see the progress of the optimization, use the gaoptimset function to create an options structure that selects two plot functions. The first plot function gaplotbestf, which plots the best and mean score of the population function is gaplotmaxconstr, which plots the maximum constraint violation of nonlinear constraints at every generation. You can also visualize the progress of the algorithm by displaying information to the command window using the 'Display' option.

```
options = gaoptimset('PlotFcns',{@gaplotbestf,@gaplotmaxconstr},'Display','iter');
```

Rerun the ga solver.

```
Optimization running.
andOptimization terminated: average change in the fitness value less than options.TolFun
and constraint violation is less than options.TolCon.
x =
0.8123 12.3103
fval =
1.3574e+04
```

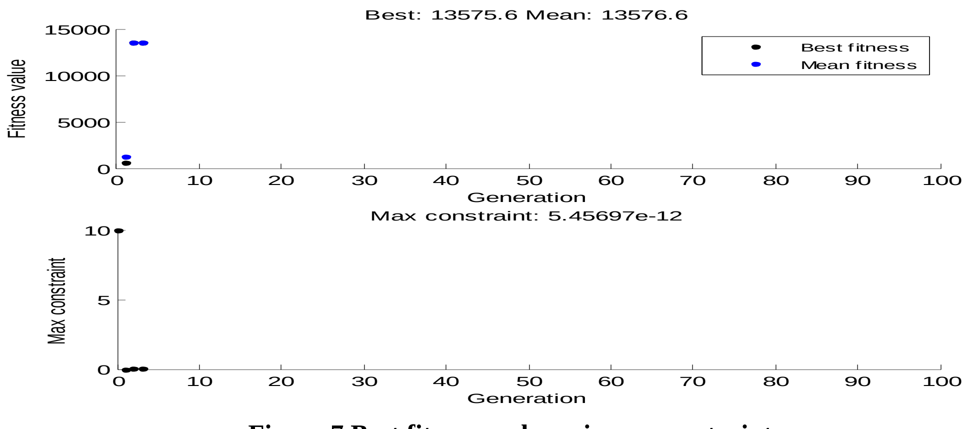


Figure 7.Best fitness and maximum constraint

You can provide a start point for the minimization to the ga function by specifying the Initial Population option. The ga function will use the first individual from InitialPopulation as a start point for a constrained minimization.

```
X0 = [0.5 0.5]; % Start point (row vector)
options = gaoptimset(options,'InitialPopulation',X0);
```

Now rerun the ga solver,

```
 [x,fval] = ga (ObjectiveFunction,nvars,[],[],[],[],...
LB,UB,ConstraintFunction,options)
Optimization terminated: average change in the fitness value less than options. TolFun
and constraint violation is less than options. TolCon.
x =
0.8122 12.3103
fval =
1.3574e+04
```

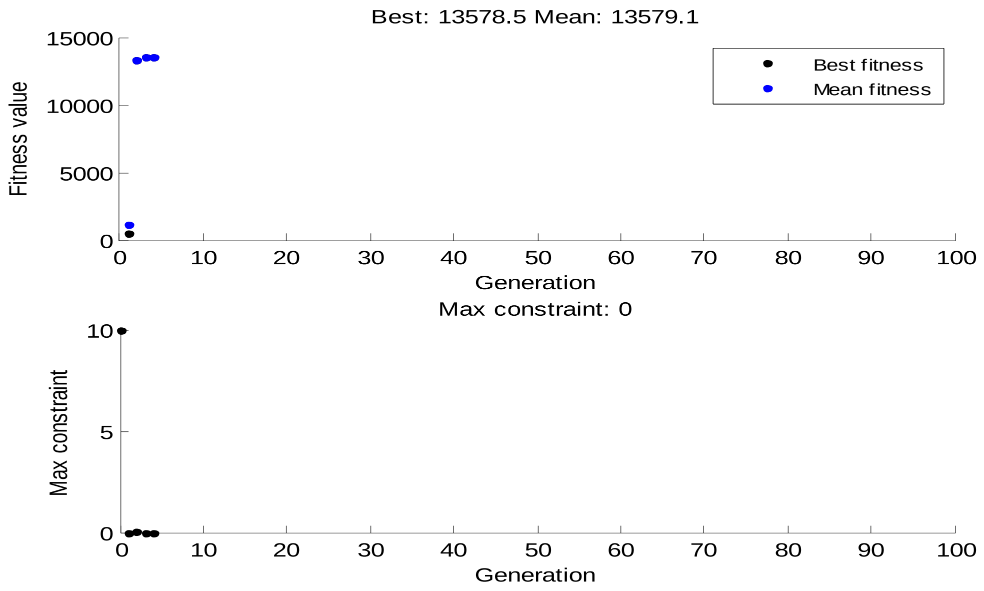


Figure 8.Optional best and Mean fitness of Rastrigin's Function

V. Summary

Mathematically, an optimization problem involves a fitness function describing the problem, under a set of constraints representing the solution space for the problem. Unfortunately, most of the traditional optimization techniques are centered around evaluating the first derivatives to locate the optima on a given constrained surface. Because of the difficulties in evaluating the first derivatives, to locate the optima for many rough and discontinuous optimization surfaces, in recent times, several derivative free optimization algorithms have been emerged. The optimization problem, nowadays, is represented as an intelligent search problem, where one or more agents are employed to determine the optima on a search landscape, representing the constrained surface for the optimization problem. In the later quarter of the twentieth century, Holland.J.H.1973 (Genetic algorithms and the optimal allocation of trials), pioneered a new concept on evolutionary search algorithms, and came up with a solution to the so far opened problem to nonlinear optimization problems. Derivative free optimization method helps to solve bound constrained optimization problems using algorithms that require only of objective the availability objective function values but not the derivative information. It also helps to evaluate a deterministic function f:$R^n \to R$ over a domain of interest that possibly includes lower and upper bounds on the problem variables. Genetic Algorithm is well suited for solving optimum design problems characterized by mixed continuous-discrete variables and discontinuous and non-convex design spaces. Genetic Algorithm can also help to find the global minimum solution with a high probability. Genetic Algorithm are based on Darwin's theory of survival of the fittest and also based on the principles of natural genetics and natural selection. Because, Genetic algorithms are based on the survival of the fittest principle of nature, they try to maximize a function called the fitness function. Thus, Genetic Algorithms are naturally suitable for solving unconstrained maximization problems. The fitness function$F(x)$,can be taken to be same as the objective function $f(x)$ of an unconstrained maximization problems so that $F(X) = f(x)$. A minimization problem can be transformed into a maximization problem before applying the Genetic Algorithms. Usually, the fitness function is chosen to be non-negative. The commonly used transformation to convert an unconstrained minimization problem to a fitness function is given by $F(X) = \frac{1}{1+f(x)}$ which does not alter the location of minimum of $f(x)$ but, converts the minimization problem into an equivalent maximization problems.

References

[1]. Argaez M. 2001. *Optimization Theory Application*, On the global convergence of a modified augmented Lagrangian line search interior-point Newton method for nonlinear programming, 144:1-25

[2]. Elizabeth D. Dolan and Jorge J.More. 2007. Benchmarking optimization software with performance profiles. *Mathematical programing*, 49(4):673-692.

[3]. Goldberg D.E. and Holland J.H. 1988.Genetic algorithms and machine learning. Machine Learning3: 95-99.

[4]. Hollad.J.H. 1975.*Adaption in Natural and Artificial Systems*.University of Michigan.Press. Ann Arbor. Michigan.

[5]. Ingber L. and Rosen B. 1992.Genetic Algorithms and very fast reannealing. A Comparison Mathl, Comput, Modelleing. 16(11), 87-100.

[6]. Kirkpatrick, S. Gelatt, C. D. and Vecchi M. P. 1983.*Optimization by simulated annealing*, vol. 220, pp. 671–680,

[7]. Marazzi,M, and Nodeal,J. 2002. Wedge trust region method for derivative free optimization.Mathprogram., 91, pp.289-300.

[8]. Nelder JA and Mead R. (1965). *A simplex method for function minimization*. Computer Journal 7: 308–313

Firaol Asfaw Wodajo. "Derivative Free Optimization by Using Genetic Algorithm Method." *IOSR Journal of Mathematics (IOSR-JM)*, 16(1), (2020): pp. 04-21.