Chapter 8
Genetic Algorithm Implementation Using Matlab

8.1 Introduction

MATLAB (**Mat**rix **Lab**oratory), a product of Mathworks, is a scientific software package designed to provide integrated numeric computation and graphics visualization in high-level programming language. Dr Cleve Moler, Chief scientist at MathWorks, Inc., originally wrote MATLAB, to provide easy access to matrix software developed in the LINPACK and EISPACK projects. The very first version was written in the late 1970s for use in courses in matrix theory, linear algebra, and numerical analysis. MATLAB is therefore built upon a foundation of sophisticated matrix software, in which the basic data element is a matrix that does not require predimensioning.

MATLAB has a wide variety of functions useful to the genetic algorithm practitioner and those wishing to experiment with the genetic algorithm for the first time. Given the versatility of MATLAB's high-level language, problems can be coded in m-files in a fraction of the time that it would take to create C or Fortran programs for the same purpose. Couple this with MATLAB's advanced data analysis, visualisation tools and special purpose application domain toolboxes and the user is presented with a uniform environment with which to explore the potential of genetic algorithms.

The Genetic Algorithm Toolbox uses MATLAB matrix functions to build a set of versatile tools for implementing a wide range of genetic algorithm methods. The Genetic Algorithm Toolbox is a collection of routines, written mostly in m-files, which implement the most important functions in genetic algorithms.

8.2 Data Structures

MATLAB essentially supports only one data type, a rectangular matrix of real or complex numeric elements. The main data structures in the Genetic Algorithm toolbox are:

- chromosomes
- objective function values
- fitness values

These data structures are discussed in the following subsections.

8.2.1 Chromosomes

The chromosome data structure stores an entire population in a single matrix of size Nind × Lind, where Nind is the number of individuals in the population and Lind is the length of the genotypic representation of those individuals. Each row corresponds to an individual's genotype, consisting of base-n, typically binary, values.

$$\text{Chrom} = \begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} & \cdots & g_{1,\text{Lind}} \\ g_{2,1} & g_{2,2} & g_{2,3} & \cdots & g_{2,\text{Lind}} \\ g_{3,1} & g_{3,2} & g_{3,3} & \cdots & g_{3,\text{Lind}} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ g_{\text{Nind},1} & g_{\text{Nind},2} & g_{\text{Nind},3} & \cdots & g_{\text{Nind},\text{Lind}} \end{bmatrix} \begin{matrix} \text{individual 1} \\ \text{individual 2} \\ \text{individual 3} \\ \cdot \\ \text{individual Nind} \end{matrix}$$

This data representation does not force a structure on the chromosome structure, only requiring that all chromosomes are of equal length. Thus, structured populations or populations with varying genotypic bases may be used in the Genetic Algorithm Toolbox provided that a suitable decoding function, mapping chromosomes onto phenotypes, is employed.

8.2.2 Phenotypes

The decision variables, or phenotypes, in the genetic algorithm are obtained by applying some mapping from the chromosome representation into the decision variable space. Here, each string contained in the chromosome structure decodes to a row vector of order Nvar, according to the number of dimensions in the search space and corresponding to the decision variable vector value. The decision variables are stored in a numerical matrix of size Nind×Nvar. Again, each row corresponds to a particular individual's phenotype. An example of the phenotype data structure is given below, where bin2real is used to represent an arbitrary function, possibly from the GA Toolbox, mapping the genotypes onto the phenotypes.

```
Phen = bin2real(Chrom) % map genotype to phenotype
```

$$= \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,\text{Nvar}} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,\text{Nvar}} \\ x_{3,1} & x_{3,2} & x_{3,3} & \cdots & x_{3,\text{Nvar}} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ x_{\text{Nind},1} & x_{\text{Nind},2} & x_{\text{Nind},3} & \cdots & x_{\text{Nind},\text{Nvar}} \end{bmatrix} \begin{matrix} \text{individual 1} \\ \text{individual 2} \\ \text{individual 3} \\ \cdot \\ \text{individual Nind} \end{matrix}$$

The actual mapping between the chromosome representation and their phenotypic values depends upon the bin2real function used. It is perfectly feasible using this

representation to have vectors of decision variables of different types. For example, it is possible to mix integer, real-valued, and binary decision variables in the same Phen data structure.

8.2.3 Objective Function Values

An objective function is used to evaluate the performance of the phenotypes in the problem domain. Objective function values can be scalar or, in the case of multiobjective problems, vectorial. Note that objective function values are not necessarily the same as the fitness values. Objective function values are stored in a numerical matrix of size Nind $\times$ Nobj, where Nobj is the number of objectives. Each row corresponds to a particular individual's objective vector.

```
Objv = OBJFUN(Phen) % Objective Function
```

$$= \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & \cdots & y_{1,\mathrm{Nvar}} \\ y_{2,1} & y_{2,2} & y_{2,3} & \cdots & y_{2,\mathrm{Nvar}} \\ y_{3,1} & y_{3,2} & y_{3,3} & \cdots & y_{3,\mathrm{Nvar}} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ y_{\mathrm{Nind},1} & y_{\mathrm{Nind},2} & y_{\mathrm{Nind},3} & \cdots & y_{\mathrm{Nind},\mathrm{Nvar}} \end{bmatrix} \begin{matrix} \text{individual 1} \\ \text{individual 2} \\ \text{individual 3} \\ \cdot \\ \text{individual Nind} \end{matrix}$$

8.2.4 Fitness Values

Fitness values are derived from objective function values through a scaling or ranking function. Fitnesses are non-negative scalars and are stored in column vectors of length Nind, an example of which is shown below. Again, Ranking is an arbitrary fitness function.

```
Fitn = ranking(ObjV) % fitness function
```

$$\mathrm{Fitn} = \begin{matrix} f_1 & \text{individual 1} \\ f_2 & \text{individual 2} \\ f_3 & \text{individual 3} \\ \cdots & \\ f_{\mathrm{Nind}} & \text{individual Nind} \end{matrix}$$

8.2.5 Multiple Subpopulations

This toolbox supports the use of a single population divided into a number of subpopulations or demes by modifying the use of data structures so that subpopulations

are stored in contiguous blocks within a single matrix. For example, the chromosome data structure, Chrom, composed of Subpop subpopulations, each of length N individuals Ind, is stored as:

$$
\begin{aligned}
\text{Chrom} = \quad & \ldots \\
& \text{Ind}_1\ \text{Subpop}_1 \\
& \text{Ind}_2\ \text{Subpop}_1 \\
& \quad \ldots \\
& \text{Ind}_N\ \text{Subpop}_1 \\
& \text{Ind}_1\ \text{Subpop}_2 \\
& \text{Ind}_2\ \text{Subpop}_2 \\
& \quad \ldots \\
& \text{Ind}_N\ \text{Subpop}_2 \\
& \quad \ldots \\
& \text{Ind}_1\ \text{Subpop}_{\text{subpop}} \\
& \text{Ind}_2\ \text{Subpop}_{\text{subpop}} \\
& \quad \ldots \\
& \text{Ind}_N\ \text{Subpop}_{\text{subpop}}
\end{aligned}
$$

This is known as the *regional model*, also called *migration* or *island model*.

8.3 Toolbox Functions

The Genetic Algorithm and Direct Search Toolbox is a collection of functions that extend the capabilities of the Optimization Toolbox and the MATLAB numeric computing environment. The Genetic Algorithm and Direct Search Toolbox includes routines for solving optimization problems using

- Genetic algorithm
- Direct search

These algorithms enable you to solve a variety of optimization problems that lie outside the scope of the Optimization Toolbox.

The genetic algorithm uses three main types of rules at each step to create the next generation from the current population:

- *Selection rules* select the individuals, called *parents*, that contribute to the population at the next generation.
- *Crossover rules* combine two parents to form children for the next generation.
- *Mutation rules* apply random changes to individual parents to form children.

The genetic algorithm at the command line, call the genetic algorithm function ga with the syntax

```
[x fval] = ga(@fitnessfun, nvars, options)
```

where

- @fitnessfun is a handle to the fitness function.
- nvars is the number of independent variables for the fitness function.
- options is a structure containing options for the genetic algorithm. If you do not pass in this argument, 'ga' uses its default options.

The results are given by

- x — Point at which the final value is attained
- fval — Final value of the fitness function

Toolboxes are set of standard library functions, which consists of predefined algorithms. The genetic algorithm and direct search toolbox of MATLAB consists of the following functions:

Solvers

ga	- Genetic algorithm solver.
gatool	- Genetic algorithm GUI.
patternsearch	- Pattern search solver.
psearchtool	- Pattern search GUI

Accessing options

gaoptimset	- Create/modify a genetic algorithm options structure.
gaoptimget	- Get options for genetic algorithm.
psoptimset	- Create/modify a pattern search options structure.
psoptimget	- Get options for pattern search.

Fitness scaling for genetic algorithm

fitscalingshiftlinear	- Offset and scale fitness to desired range.
fitscalingprop	- Proportional fitness scaling.
fitscalingrank	- Rank based fitness scaling.
fitscalingtop	- Top individuals reproduce equally.

Selection for genetic algorithm

selectionremainder	- Remainder stochastic sampling without replacement.
selectionroulette	- Choose parents using roulette wheel.
selectionstochunif	- Choose parents using stochastic universal sampling (SUS).
selectiontournament	- Each parent is the best of a random set.
selectionuniform	- Choose parents at random.

Crossover (recombination) functions for genetic algorithm.

crossoverheuristic	- Move from worst parent to slightly past best parent.
crossoverintermediate	- Weighted average of the parents.

crossoverscattered	- Position independent crossover function.
crossoversinglepoint	- Single point crossover.
crossovertwopoint	- Two point crossover.

Mutation functions for genetic algorithm

mutationgaussian	- Gaussian mutation.
mutationuniform	- Uniform multi-point mutation.

Plot Functions for genetic algorithm

gaplotbestf	- Plots the best score and the mean score.
gaplotbestindiv	- Plots the best individual in every generation as a bar plot.
gaplotdistance	- Averages several samples of distances between individuals.
gaplotexpectation	- Plots raw scores vs the expected number of offspring.
gaplotgenealogy	- Plot the ancestors of every individual.
gaplotrange	- Plots the min, mean, and max of the scores.
gaplotscordiversity	- Plots a histogram of this generations scores.
gaplotscores	- Plots the scores of every member of the population.
gaplotselection	- A histogram of parents.
gaplotstopping	- Display stopping criteria levels.

Output Functions for genetic algorithm

gaoutputgen	- Displays generation number and best function value in a separate window.
gaoutputoptions	- Prints all of the non-default options settings.

Custom search functions for pattern search

searchlhs	- Implements latin hypercube sampling as a search method.
searchneldermead	- Implements nelder-mead simplex method (FMINSEARCH) to use as a search method.
searchga	- Implements genetic algorithm (GA) to use as a search method.
searchfcntemplate	- Template file for a custom search method.

Plot Functions for pattern search

psplotbestf	- Plots best function value.
psplotbestx	- Plots current point in every iteration as a bar plot.
psplotfuncount	- Plots the number of function evaluation in every iteration.
psplotmeshsize	- Plots mesh size used in every iteration.

Output functions for pattern search

psoutputhistory	- Displays iteration number, number of function evaluations, function value, mesh size and method used in every iteration in a separate window.
psoutputfcntemplate	- Template file for a custom output function.

Utility functions

allfeasible	- Filter infeasible points.
gacreationuniform	- Create the initial population for genetic algorithm.
gray2int	- Convert a gray code array to an integer.
lhspoint	- Generates latin hypercube design point.
nextpoint	- Return the best iterate assuming feasibility.

Help files for genetic algorithm

fitnessfunction	- Help on fitness functions.
fitnessscaling	- Help on fitness scaling

These are the toolbox functions present in the MATLAB. There also exist Genetic and Evolutionary Algorithm Toolbox for use with MATLAB that contains a broad range of tools for solving real-world optimization problems. They not only cover pure optimization, but also the preparation of the problem to be solved, the visualization of the optimization process, the reporting and saving of results, and as well as some other special tools. The list of various functions using Genetic and Evolutionary Algorithm Toolbox for use with MATLAB is as follows:

Objective functions

initdopi	- INITialization function for DOuble Integrator objdopi
initfun1	- INITialization function for de jong's FUNction 1
mopfonseca1	- MultiObjective Problem: FONSECA's function 1
mopfonseca2	- MultiObjective Problem: FONSECA's function 1
moptest	- MultiObjective function TESTing
obj4wings	- OBJective function FOUR-WINGS.
objbran	- OBJective function for BRANin rcos function
objdopi	- OBJective function for DOuble Integrator
objeaso	- OBJective function for EASom function
objfletwell	- OBJective function after FLETcher and PoWELL
objfractal	- OBJective function Fractal Mandelbrot
objfun1	- OBJective function for de jong's FUNction 1
objfun10	- OBJective function for ackley's path FUNction 10
objfun11	- OBJective function for langermann's function 11
objfun12	- OBJective function for michalewicz's function 12
objfun1a	- OBJective function for axis parallel hyper-ellipsoid
objfun1b	- OBJective function for rotated hyper-ellipsoid
objfun1c	- OBJective function for moved axis parallel hyper ellipsoid 1c
objfun2	- OBJective function for rosenbrock's FUNction
objfun6	- OBJective function for rastrigins FUNction 6
objfun7	- OBJective function for schwefel's FUNction
objfun8	- OBJective function for griewangk's FUNction
objfun9	- OBJective function for sum of different power FUNction 9
objgold	- OBJective function for GOLDstein-price function
objharv	- OBJective function for HARVest problem
objint1	- OBJective function for INT function 1
objint2	- OBJective function for INT function 1
objint3	- OBJective function for INT function 3
objint4	- OBJective function for INT function 4
objlinq	- OBJective function for discrete LINear Quadratic problem
objlinq2	- OBJective function for LINear Quadratic problem 2
objone1	- OBJective function for ONEmax function 1
objpush	- OBJective function for PUSH-cart problem
objridge	- OBJective function RIDGE
objsixh	- OBJective function for SIX Hump camelback function
objsoland	- OBJective function for SOLAND function
objtsp1	- OBJective function for the traveling salesman example
objtsplib	- OBJective function for the traveling salesman library
plotdopi	- PLOTing of DO(Ppel)uble Integration results
plottsplib	- PLOTing of results of TSP optimization (TSPLIB examples)
simdopi1	- M-file description of the SIMULINK system named SIMDOPI1
simdopiv	- SIMulation Modell of DOPpelIntegrator, s-function, Vectorized

simlinq1	- M-file description of the SIMULINK system named SIMLINQ1
simlinq2	- Modell of Linear Quadratic Problem, s-function
tsp_readlib	- TSP utility function, reads TSPLIB data files
tsp_uscity	- TSP utility function, reads US City definitions

Conversion functions

bin2int	- BINary string to INTeger string conversion
bin2real	- BINary string to REAL vector conversion
bindecod	- BINary DECODing to binary, integer or real numbers

Initialization functions

initbp	- CReaTe an initial Binary Population
initip	- CReaTe an initial (Integer value) Population
initpop	- INITialization of POPulation (including innoculation)
initpp	- Create an INITial Permutation Population
initrp	- INITialize an Real value Population

Selection functions

selection	- high level SELECTion function
sellocal	- SELection in a LOCAL neighbourhood
selrws	- SELection by Roulette Wheel Selection
selsus	- SELection by Stochastic Universal Sampling
seltour	- SELection by TOURnament
seltrunc	- SELection by TRUNCation
rankgoal	- perform goal preference calculation between multiple objective values
ranking	- RANK-based fitness assignment, single and multi objective, linear and nonlinear
rankplt	- RANK two multi objective values Partially Less Than
rankshare	- SHARing between individuals

Crossover functions

recdis	- RECombination DIScrete
recdp	- RECombination Double Point
recdprs	- RECombination Double Point with Reduced Surrogate
recgp	- RECombination Generalized Position
recint	- RECombination extended INTermediate
reclin	- RECombination extended LINe
reclinex	- EXtended LINe RECombination
recmp	- RECombination Multi-Point, low level function
recombin	- high level RECOMBINation function
recpm	- RECombination Partial Matching
recsh	- RECombination SHuffle
recshrs	- RECombination SHuffle with Reduced Surrogate
recsp	- RECombination Single Point
recsprs	- RECombination Single Point with Reduced Surrogate
reins	- high-level RE-INSertion function
reinsloc	- RE-INSertion of offspring in population replacing parents LOCal
reinsreg	- REINSertion of offspring in REGional population model replac

Mutation functions

mutate	- high level MUTATion function
mutbin	- MUTation for BINary representation
mutbmd	- real value Mutation like Discrete Breeder genetic algorithm
mutcomb	- MUTation for combinatorial problems
mutes1	- MUTation by Evolutionary Strategies 1, derandomized Self Adaption
mutes2	- MUTation by Evolutionary Strategies 2, derandomized self adaption
mutexch	- MUTation by eXCHange

mutint	- MUTation for INTeger representation
mutinvert	- MUTation by INVERTing variables
mutmove	- MUTation by MOVEing variables
mutrand	- MUTation RANDom
mutrandbin	- MUTation RANDom of binary variables
mutrandint	- MUTation RANDom of integer variables
mutrandperm	- MUTation RANDom of binary variables
mutrandreal	- MUTation RANDom of real variables
mutreal	- real value Mutation like Discrete Breeder genetic algorithm
mutswap	- MUTation by SWAPping variables
mutswaptyp	- MUTation by SWAPping variables of identical typ

Other functions

compdiv	- COMPute DIVerse things of GEA Toolbox
compdiv2	- COMPute DIVerse things of GEA Toolbox
compete	- COMPETition between subpopulations
comploc	- COMPute LOCal model things of toolbox
compplot	- COMPute PLOT things of GEA Toolbox
geamain2	- MAIN function for Genetic and Evolutionary Algorithm toolbox for matlab

Plot Functions

fitdistc	- FITness DISTance Correlation computation
meshvar	- create grafics of objective functions with plotmesh.
plotmesh	- PLOT of objective functions as MESH Plot
plotmop	- PLOT properties of MultiObjective functions
reslook	- LOOK at saved RESults
resplot	- RESult PLOTing of GEA Toolbox optimization
samdata	- sammon mapping: data examples
samgrad	- Sammon mapping gradient calculation
sammon	- Multidimensional scaling (SAMMON mapping)
samobj	- Sammon mapping objective function
samplot	- Plot function for Multidimensional scaling (SAMMON mapping)

8.4 Genetic Algorithm Graphical User Interface Toolbox

The Genetic Algorithm Tool is a graphical user interface that enables you to use the genetic algorithm without working at the command line. To open the Genetic Algorithm Tool, enter

```
gatool
```

at the MATLAB command prompt.

This opens the tool as shown in the following Fig. 8.1

To use the Genetic Algorithm Tool, you must first enter the following information:

Fitness function – The objective function you want to minimize. Enter the fitness function in the form @fitnessfun, where fitnessfun.m is an M-file that computes the fitness function.

Number of Variables – The number of variables in the given fitness function should be given.

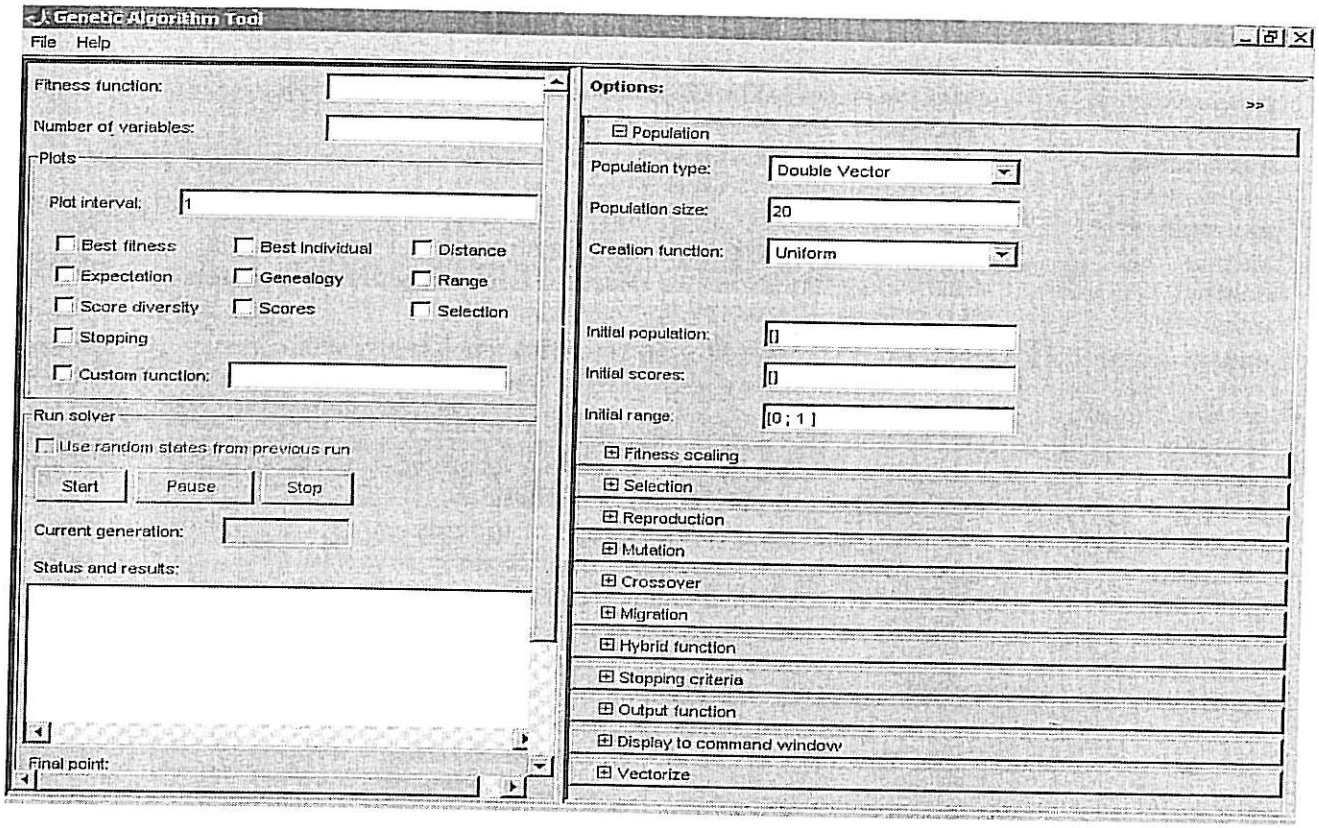

Fig. 8.1 Genetic Algorithm Tool

The **plot** options

1. Best fitness
2. Best individual
3. Distance
4. Expectation
5. Genealogy
6. Range
7. Score Diversity
8. Scores
9. Selection
10. Stopping

Based upon ones problem, custom function my also be built.

The various parameters essential for running Genetic algorithm tool should be specified appropriately. The parameters appear on the right hand side of the GA tool. The description is as follows:

1. **Population**
   In this case population type, population size and creation function may be selected. The initial population and initial score may be specified, if not, the "Ga tool" creates them. The initial range should be given.
2. **Fitness Scaling**
   The fitness scaling should be any of the following

a. Rank
b. Proportional
c. Top
d. Shift Linear
e. Custom

3. Selection

The selection is made on any one of the following mentioned methods

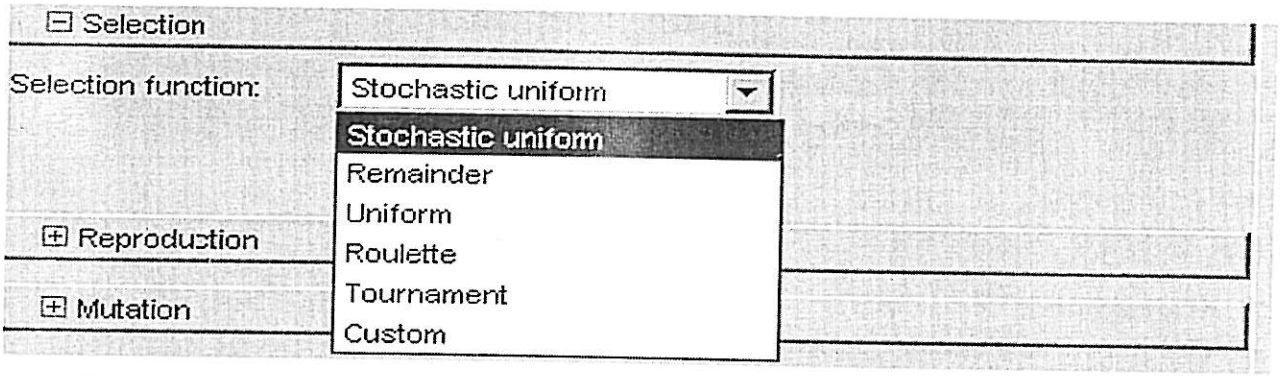

Fig. 8.2 Selection

4. Reproduction

In reproduction the elite count and cross over fraction should be given. If elite count not specified, it is taken as 2.

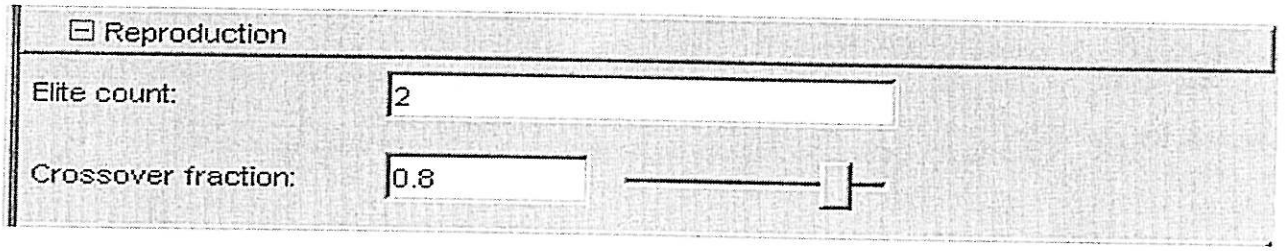

Fig. 8.3 Reproduction

5. Mutation

Generally Gaussian or Uniform Mutation is carried out. The user may define own customized mutation operation.

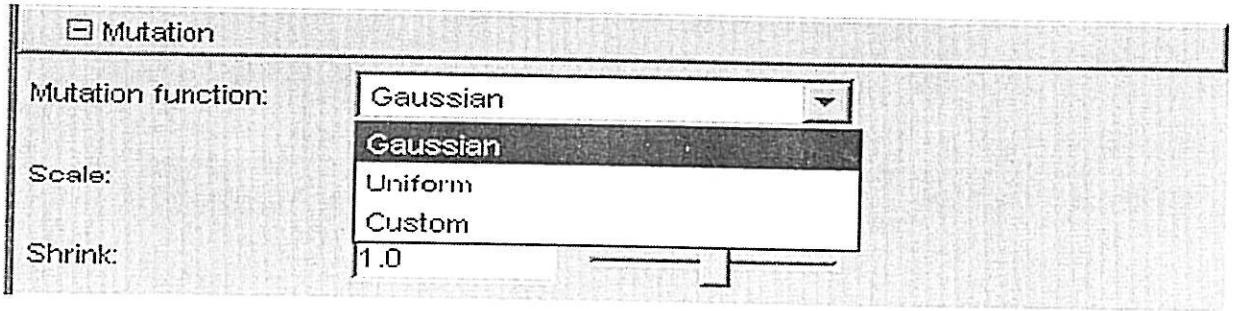

Fig. 8.4 Mutation

6. Crossover

The various crossover techniques are as follows:

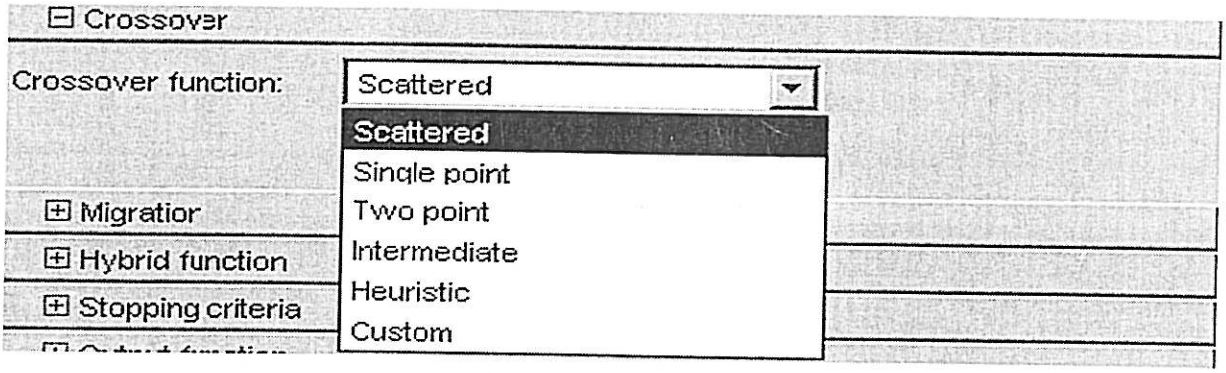

Fig. 8.5 Crossover

7. Migration

The parameter for migration should be defined as follows:

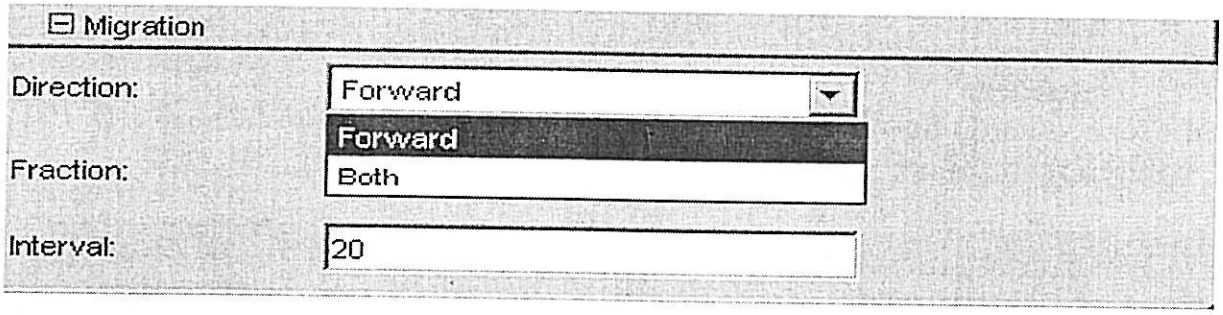

Fig. 8.6 Migration

8. Hybrid Function

Any one of the following hybrid functions may be selected,

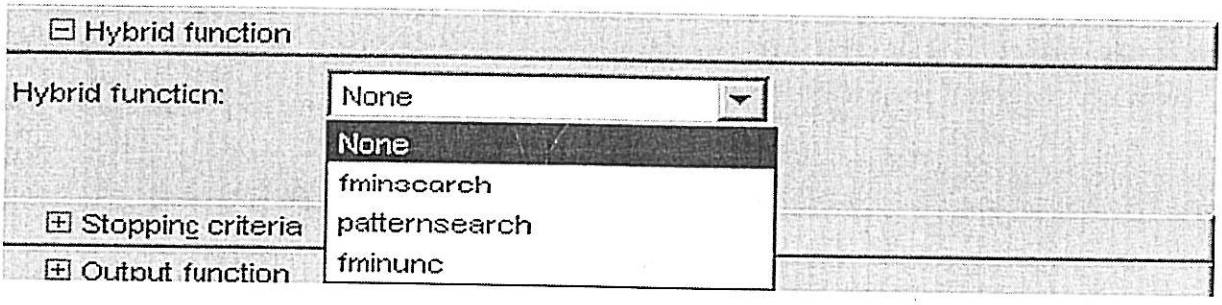

Fig. 8.7 Hybrid function

9. Stopping Criteria

The stopping criteria plays a major role in simulation. They are:

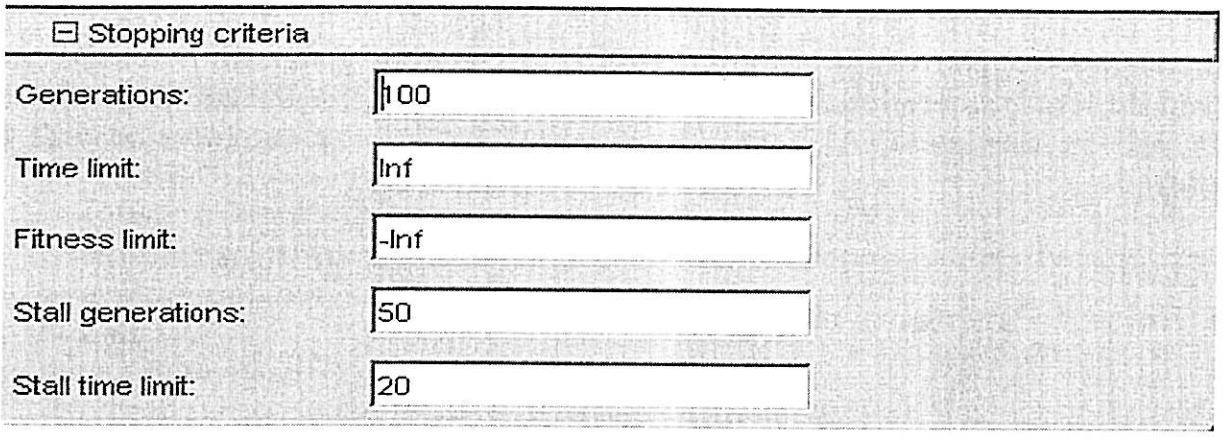

Fig. 8.8 Stopping criteria

The other parameters **Output Function, Display to command window and Vectorize** may be suitably defined by the user.

10. Running and Simulation

The menu shown below helps the user for running the GA tool.

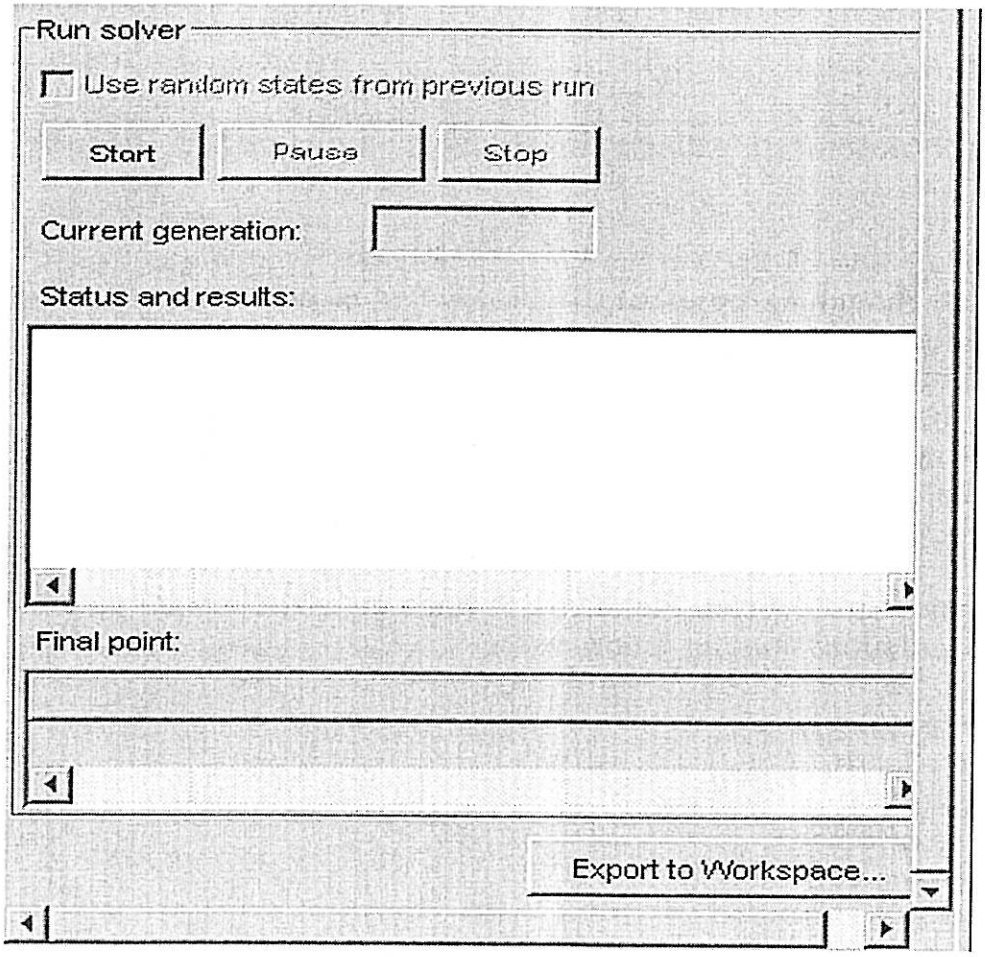

Fig. 8.9 Run solver

The running process may be temporarily stopped using "Pause" option and permanently stopped using "Stop" option. The "current generation" will be displayed during the iteration. Once the iterations are completed, the status and results will be displayed. Also the "final point" for the fitness function will be displayed.

8.5 Solved Problems using MATLAB

Problem 1

Write a MATLAB program for maximizing $f(x) = x^2$ using genetic algorithm, where x ranges from 0 to 31. Perform 4 iterations.

Note

In MATLAB % indicates comment statement.

Source Code

```
%program for Genetic algorithm to maximize the function f(x) =xsquare
clear all;
clc;
%x ranges from 0 to 31 2power5 = 32
%five bits are enough to represent x in binary representation
n=input('Enter no. of population in each iteration');
nit=input('Enter no. of iterations');
%Generate the initial population
[oldchrom]=initbp(n,5)
%The population in binary is converted to integer
FieldD=[5;0;31;0;0;1;1]
for i=1:nit
   phen=bindecod(oldchrom,FieldD,3); % phen gives the integer value of the
   binary population %obtain fitness value
   sqx=phen. ^2;
   sumsqx=sum(sqx);
   avsqx=sumsqx/n;
   hsqx=max(sqx);
   pselect=sqx./sumsqx;
   sumpselect=sum(pselect);
   avpselect=sumpselect/n;
   hpselect=max(pselect);
   %apply roulette wheel selection
```

```
FitnV=sqx;
Nsel=4;
newchrix=selrws(FitnV, Nsel);
newchrom=oldchrom(newchrix,:);
%Perform Crossover
crossoverrate=1;
newchromc=recsp(newchrom,crossoverrate); %new population after crossover
%Perform mutation
vlub=0:31;
mutationrate=0.001;
newchromm=mutrandbin(newchromc,vlub,mutationrate); %new population after
mutation disp('For iteration');
i
disp('Population');
oldchrom
disp('X');
phen
disp('f(X)');
sqx
oldchrom=newchromm;
end
```

Output

```
Enter no. of population in each iteration4
Enter no. of iterations4
oldchrom =
   1   0   0   1   0
   0   1   0   1   0
   0   0   1   1   0
   1   1   1   1   0
FieldD =
   5
   0
  31
   0
   0
   1
   1
For iteration
i =
   1
Population
oldchrom =
```

```
     1   0   0   1   0
     0   1   0   1   0
     0   0   1   1   0
     1   1   1   1   0
 X
 phen =
     18
     10
      6
     30
 f(X)
 sqx =
     324
     100
      36
     900
 For iteration
 i =
     2
 Population
 oldchrom =
     1   1   1   0   0
     0   1   1   0   1
     0   0   1   1   0
     1   0   1   0   1
 X
 phen =
     28
     13
      6
     21
 f(X)
 sqx =
     784
     169
      36
     441
 For iteration
 i =
     3
 Population
 oldchrom =
     0   0   0   0   1
     0   0   1   1   1
     0   0   0   0   1
     1   0   1   0   0
```

```
X
phen =
    1
    7
    1
   20
f(X)
sqx =
    1
   49
    1
  400
For iteration
i =
    4
Population
oldchrom =
    1   0   0   0   0
    1   1   0   1   1
    1   0   0   1   1
    0   1   1   1   1
X
phen =
   16
   27
   19
   15
f(X)
sqx =
  256
  729
  361
  225
```

Problem 2

Find a minimum of a non-smooth objective function using the Genetic Algorithm (GA) function in the Genetic Algorithm and Direct Search Toolbox.

Description

Traditional derivative-based optimization methods, like those found in the Optimization Toolbox, are fast and accurate for many types of optimization problems.

These methods are designed to solve "smooth", i.e., continuous and differentiable, minimization problems, as they use derivatives to determine the direction of descent. While using derivatives makes these methods fast and accurate, they often are not effective when problems lack smoothness, e.g., problems with discontinuous, non-differentiable, or stochastic objective functions. When faced with solving such non-smooth problems, methods like the genetic algorithm or the more recently developed pattern search methods, both found in the Genetic Algorithm and Direct Search Toolbox, are effective alternatives.

Source Code

```
clear all; close all;format compact
Objfcn = @nonSmoothFcn;          %Handle to the objective function
X0 = [2 -2];                     % Starting point
range = [-6 6;-6 6];             %Range used to plot the objective function
rand('state',0);                 %Reset the state of random number generators
randn('state',0);
type nonSmoothFcn.m              % Non-smooth Objective Function
showNonSmoothFcn(Objfcn,range);
set(gca,'CameraPosition',[-36.9991 62.6267 207.3622]);
set(gca,'CameraTarget',[0.1059 -1.8145 22.3668])
set(gca,'CameraViewAngle',6.0924)
%Plot of the starting point (used by the PATTERNSEARCH solver)
plot3(X0(1),X0(2),feval(Objfcn,X0),'or','MarkerSize',10,'MarkerFaceColor','r');
fig = gcf;
% Minimization Using The Genetic Algorithm
FitnessFcn = @nonSmoothFcn;
numberOfVariables = 2;
optionsGA = gaoptimset('PlotFcns',@gaplotbestfun,'PlotInterval',5, ...
'PopInitRange',[-5;5]);
% We run GA with the options 'optionsGA' as the third argument.
[Xga,Fga] = ga(FitnessFcn,numberOfVariables,optionsGA)
% Plot the final solution
figure(fig)
hold on;
plot3(Xga(1),Xga(2),Fga,'vm','MarkerSize',10,'MarkerFaceColor','m');
hold off;
fig = gcf;
```

% The optimum is at x* = (−4.7124, 0.0). GA found the point % (−4.7775,0.0481) near the optimum, but could not get closer with the default stopping criteria. By changing the stopping criteria, we might find a more accurate solution, but it may take many more function evaluations to reach x* = (−4.7124, 0.0). Instead, we can use a more efficient local search that starts where GA left off. The hybrid function field in GA provides this feature automatically.

```
% Minimization Using A Hybrid Function

% Our choices are FMINSEARCH, PATTERNSEARCH, or FMINUNC. Since this
optimization example is smooth near the optimizer, we can use the FMINUNC func-
tion from the Optimization toolbox as our hybrid function as this is likely to be the
most efficient. Since FMINUNC has its own options structure, we provide it as an
additional argument when specifying the hybrid function.

% Run GA-FMINUNC Hybrid
optHybrid = gaoptimset(optionsGA,'Generations',15, 'PlotInterval',1,...
                'HybridFcn',{ @fminunc,optimset('OutputFcn',@fminuncOut)});
[Xhybrid,Fhybrid] = ga(Objfcn,2,optHybrid);
% Plot the final solution
figure(fig);
hold on;
plot3(Xhybrid(1),Xhybrid(2),Fhybrid+1,' ^ c','MarkerSize',10,
'MarkerFaceColor','c');
hold off;
disp(['The norm of |Xga - Xhb| is ', num2str(norm(Xga-Xhybrid))]);
disp(['The difference in function values Fga and Fhb is ',
num2str(Fga - Fhybrid)]);


%% Minimization Using The Pattern Search Algorithm

% To minimize our objective function using the PATTERNSEARCH function, we
need to pass in a function handle to the objective function as well as specifying a
start point as the second argument.

ObjectiveFunction = @nonSmoothFcn;
X0 = [2 -2]; % Starting point
% Some plot functions are selected to monitor the performance of the solver.
optionsPS = psoptimset('PlotFcns',@psplotbestf);
% Run pattern search solver
[Xps,Fps] = patternsearch(Objfcn,X0,[],[],[],[],[],[],optionsPS)
% Plot the final solution
figure(fig)
hold on;
plot3(Xps(1),Xps(2),Fps+1,'*y','MarkerSize',14,'MarkerFaceColor','y');
hold off;
```

The various functions used in optimization of non-smooth function are a follows:

```
function [f, g] = nonSmoothFcn(x)
%NONSMOOTHFCN is a non-smooth objective function
```

```
 for i = 1:size(x,1)
    if x(i,1) < -7
       f(i) = (x(i,1))^2 + (x(i,2))^2 ;
    elseif x(i,1) < -3
       f(i) = -2*sin(x(i,1)) - (x(i,1)*x(i,2)^2)/10 + 15 ;
    elseif x(i,1) < 0
       f(i) = 0.5*x(i,1)^2 + 20 + abs(x(i,2))+ patho(x(i,:));
    elseif x(i,1) >= 0
       f(i) = .3*sqrt(x(i,1)) + 25 +abs(x(i,2)) + patho(x(i,:));
    end
 end
 %Calculate gradient
 g = NaN;
 if x(i,1) < -7
    g = 2*[x(i,1); x(i,2)];
 elseif x(i,1) < -3
    g = [-2*cos(x(i,1))-(x(i,2)^2)/10; -x(i,1)*x(i,2)/5];
 elseif x(i,1) < 0
    [fp,gp] = patho(x(i,:));
    if x(i,2) > 0
       g = [x(i,1)+gp(1); 1+gp(2)];
    elseif x(i,2) < 0
       g = [x(i,1)+gp(1); -1+gp(2)];
    end
 elseif x(i,1) >0
    [fp,gp] = patho(x(i,:));
    if x(i,2) > 0
       g = [.15/sqrt(x(i,1))+gp(1); 1+ gp(2)];
    elseif x(i,2) < 0
       g = [.15/sqrt(x(i,1))+gp(1); -1+ gp(2)];
    end
 end
 function [f,g] = patho(x)
 Max = 500;
 f = zeros(size(x,1),1);
 g = zeros(size(x));
 for k = 1:Max %k
    arg = sin(pi*k^2*x)/(pi*k^2);
    f = f + sum(arg,2);
    g = g + cos(pi*k^2*x);
 end

 function showNonSmoothFcn(fcn,range)
 if(nargin == 0)
    fcn = @rastriginsfcn;
    range = [-5,5;-5,5];
```

```
end
pts = 25;
span = diff(range')/(pts - 1);
x = range(1,1): span(1) : range(1,2);
y = range(2,1): span(2) : range(2,2);
pop = zeros(pts * pts,2);
k = 1;
for i = 1:pts
    for j = 1:pts
      pop(k,:) = [x(i),y(j)];
      k = k + 1;
    end
end
values = feval(fcn,pop);
values = reshape(values,pts,pts);
surf(x,y,values)
shading interp
light
lighting phong
hold on
contour(x,y,values)
rotate3d
view(37,60)
%Annotations
figure1 = gcf;
% Create arrow
annotation1 = annotation(figure1,'arrow',[0.5946 0.4196],[0.9024 0.6738]);
% Create textbox
annotation2 = annotation(...
  figure1,'textbox',...
  'Position',[0.575 0.9071 0.1571 0.07402],...
  'FitHeightToText','off',...
  'FontWeight','bold',...
  'String',{'Start point'});
% Create textarrow
annotation3 = annotation(...
  figure1,'textarrow',...
  [0.3679 0.4661],[0.1476 0.3214],...
  'String',{'Non-differentiable regions'},...
  'FontWeight','bold');
% Create arrow
annotation4 = annotation(figure1,'arrow',[0.1196 0.04107],[0.1381 0.5429]);
% Create textarrow
annotation5 = annotation(...
  figure1,'textarrow',...
  [0.7411 0.5321],[0.05476 0.1381],...
```

```
    'LineWidth',2,...
    'Color',[1 0 0],...
    'String',{'Smooth region'},...
    'FontWeight','bold',...
    'TextLineWidth',2,...
    'TextEdgeColor',[1 0 0]);
  % Create arrow
  annotation6 = annotation(...
    figure1,'arrow',...
    [0.8946 0.9179],[0.05714 0.531],...
    'Color',[1 0 0]);

  function stop = fminuncOut(x,optimvalues, state)
  persistent fig gaIter
  stop = false;
  switch state
      case 'init'
          fig = findobj(0,'type','figure','name','Genetic Algorithm');
          limits = get(gca,'XLim');
          gaIter = limits(2);
          hold on;
      case 'iter'
          set(gca,'Xlim', [1 optimvalues.iteration + gaIter]);
          fval = optimvalues.fval;
          iter = gaIter + optimvalues.iteration;
          plot(iter,fval,'dr')
          title(['Best function value: ',num2str(fval)],'interp','none')
      case 'done'
          fval = optimvalues.fval;
          iter = gaIter + optimvalues.iteration;
          title(['Best function value: ',num2str(fval)],'interp','none')
          % Create textarrow
          annotation1 = annotation(...
              gcf,'textarrow',...
              [0.6643 0.7286],[0.3833 0.119],...
              'String',{'Algorithm switch to FMINUNC'},...
              'FontWeight','bold');
          hold off
  end
```

Output

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

Xga =
   -4.7775 0.0481
```

```
Fga =
   13.0053


Optimization terminated: maximum number of generations exceeded.
Switching to the hybrid optimization algorithm (FMINUNC).
In fminunc at 241
In ga at 268
In nonSmoothOpt at 101
Optimization terminated: relative infinity-norm of gradient less than options.TolFun.
The norm of |Xga - Xhb| is 0.08092
The difference in function values Fga and Fhb is 0.0053385
Optimization terminated: current mesh size 9.5367e-007 is less than 'TolMesh'.


Xps =
   -4.7124 0
Fps =
   13.0000
```

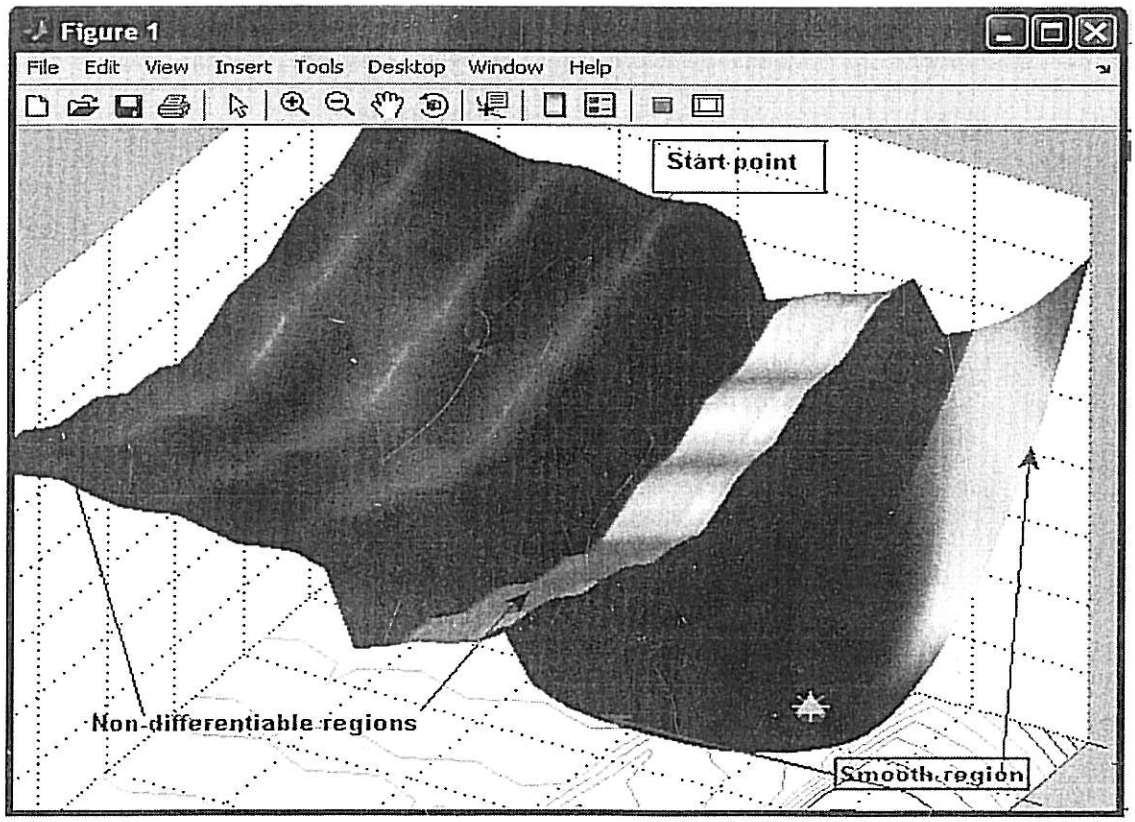

Fig. 8.10 Optimization of objective function using GA

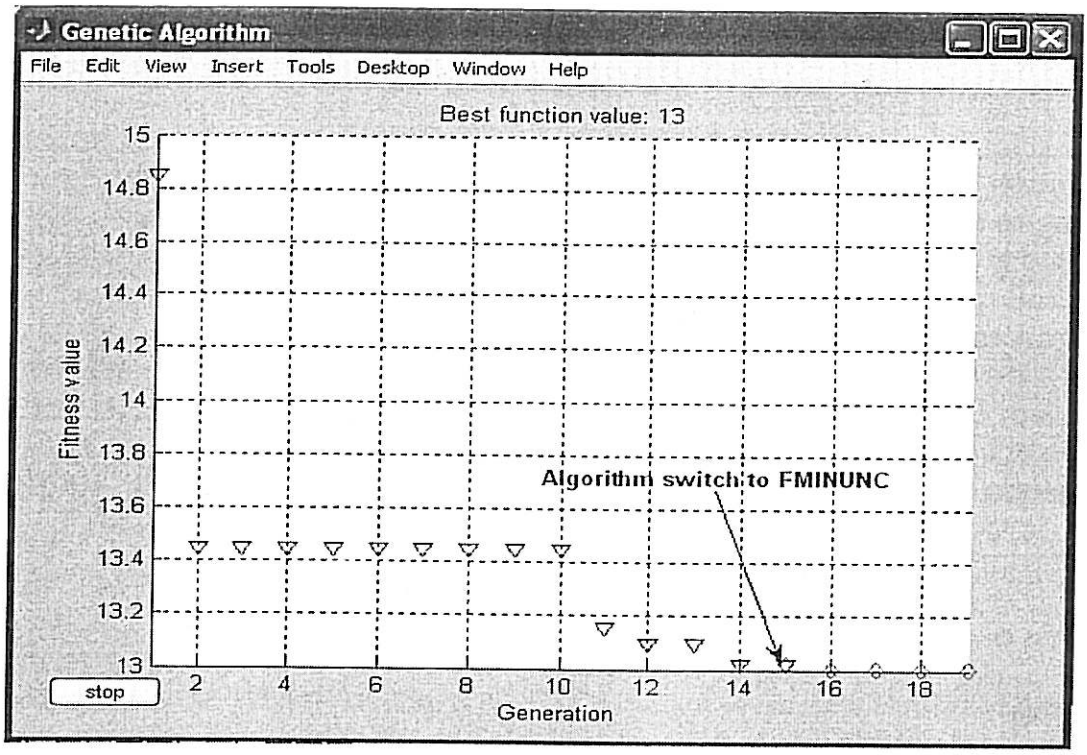

Fig. 8.11 Optimization using hybrid GA – FMINUNC

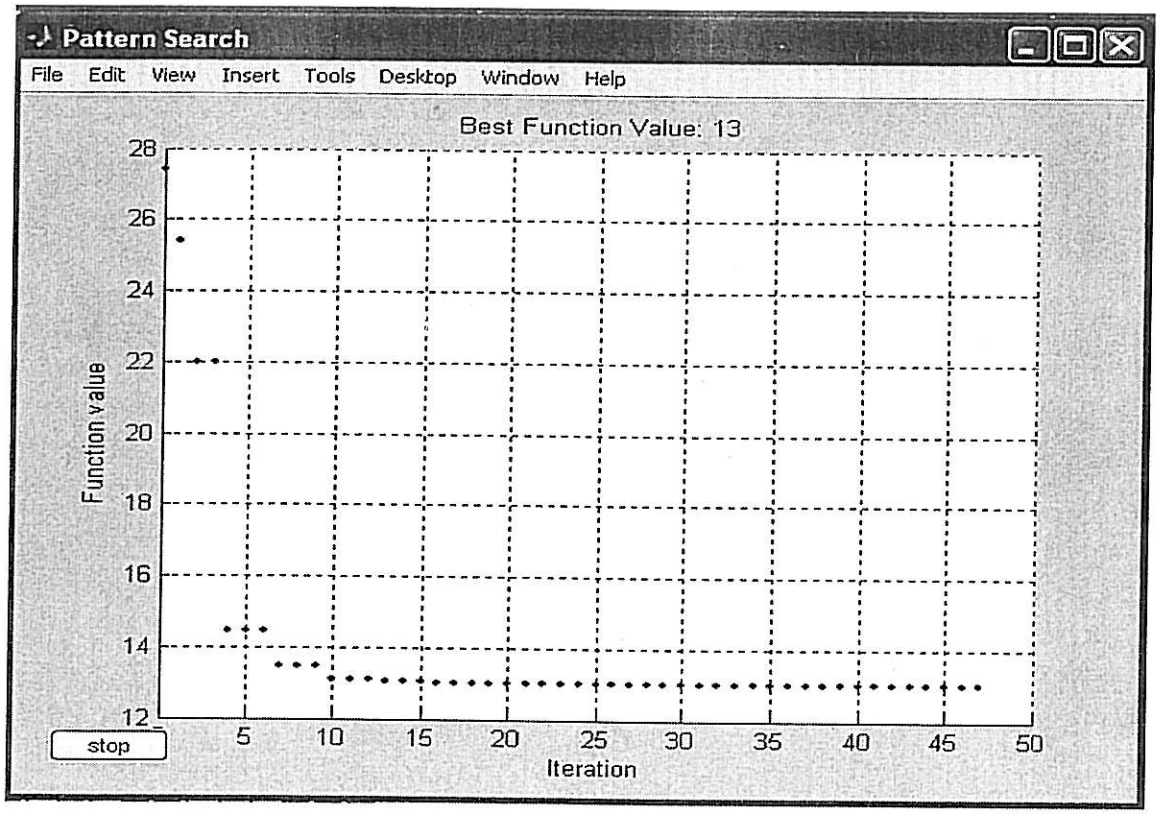

Fig. 8.12 Optimization using pattern search

Problem 3

Find a minimum of a stochastic objective function using PATTERNSEARCH function in the Genetic Algorithm and Direct Search Toolbox.

Source Code

```
% Pattern search optimization solver
format compact
X0 = [2.5 -2.5];                          %Starting point.
LB = [-5 -5];                             %Lower bound
UB = [5 5];                               %Upper bound
range = [LB(1) UB(1); LB(2) UB(2)];
Objfcn = @smoothFcn;                      %Handle to the objective function.
% Plot the smooth objective function
clf;showSmoothFcn(Objfcn,range); hold on;
title('Smooth objective function')
plot3(X0(1),X0(2),feval(Objfcn,X0)+30,'om','MarkerSize',12, ...
'MarkerFaceColor','r'); hold off;
set(gca,'CameraPosition',[-31.0391 -85.2792 -281.4265]);
set(gca,'CameraTarget',[0 0 -50])
set(gca,'CameraViewAngle',6.7937)
fig = gcf;
```

%% Run FMINCON on smooth objective function

% The objective function is smooth (twice continuously differentiable). Solving the optimization problem using FMINCON function from the Optimization Toolbox. FMINCON finds a constrained minimum of a function of several variables. This function has a unique minimum at the point $x^* = (-5.0, -5)$ where it has a function value $f(x^*) = -250$.

```
% Set options to display iterative results.
options = optimset('Display','iter','OutputFcn',@fminuncOut1);
[Xop,Fop] = fmincon(Objfcn,X0,[],[],[],[],LB,UB,[],options)
figure(fig);
hold on;
%Plot the final point
plot3(Xop(1),Xop(2),Fop,'dm','MarkerSize',12,'MarkerFaceColor','m');
hold off;
% Stochastic objective function The objective function is same as the previous
function and some noise
```

```
% added to it.
%Reset the state of random number generators
randn('state',0);
noise = 8.5;
Objfcn = @(x) smoothFcn(x,noise); %Handle to the objective function.
%Plot the objective function (non-smooth)
figure;
for i = 1:6
   showSmoothFcn(Objfcn,range);
   title('Stochastic objective function')
   set(gca,'CameraPosition',[-31.0391 -85.2792 -281.4265]);
   set(gca,'CameraTarget',[0 0 -50])
   set(gca,'CameraViewAngle',6.7937)
   drawnow; pause(0.2)
end
fig = gcf;
%% Run FMINCON on stochastic objective function
options = optimset('Display','iter');
[Xop,Fop] = fmincon(Objfcn,X0,[],[],[],[],LB,UB,[],options)
figure(fig);
hold on;
plot3(X0(1),X0(2),feval(Objfcn,X0)+30,'om','MarkerSize',16,
'MarkerFaceColor','r');
plot3(Xop(1),Xop(2),Fop,'dm','MarkerSize',12,'MarkerFaceColor','m');
%% Run PATTERNSEARCH. A pattern search algorithm does not require any
derivative information of the objective function to find an optimal point.
PSoptions = psoptimset('Display','iter','OutputFcn',@psOut);
[Xps,Fps] = patternsearch(Objfcn,X0,[],[],[],[],LB,UB,PSoptions)
figure(fig);
hold on;
plot3(Xps(1),Xps(2),Fps,'pr','MarkerSize',18,'MarkerFaceColor','r');
hold off
```

The various functions used in the above program are as follows:

```
function y = smoothFcn(z,noise)
% Objective function
if nargin < 2
   noise = 0;
end
LB = [-5 -5];          %Lower bound
UB = [5 5];            %Upper bound
y = zeros(1,size(z,1));
for i = 1:size(z,1)
```

```
    x = z(i,:);
    if any(x<LB) || any(x>UB)
        y(i) = Inf;
    else
    y(i) = x(1)^3 - x(2)^2 + ...
        100*x(2)/(10+x(1)) + noise*randn;
    end
end


function showSmoothFcn(fcn,range)
pts = 100;
span = diff(range')/(pts - 1);
x = range(1,1): span(1) : range(1,2);
y = range(2,1): span(2) : range(2,2);
pop = zeros(pts * pts,2);
k = 1;
for i = 1:pts
    for j = 1:pts
        pop(k,:) = [x(i),y(j)];
        k = k + 1;
    end
end
values = feval(fcn,pop);
values = reshape(values,pts,pts);
clf;
surf(x,y,values)
shading interp
light
lighting phong
hold on
rotate3d
view(37,60)
set(gcf,'Renderer','opengl');
set(gca,'ZLimMode','manual');
function stop = fminuncOut1(X, optimvalues, state)
stop = false;
figure1 = gcf;
if strcmpi(state,'done')
annotation1 = annotation(figure1,'arrow',[0.7506 0.7014],[0.3797 0.625]);
    % Create textbox
    annotation2 = annotation(...
        figure1,'textbox',...
        'Position',[0.6857 0.2968 0.1482 0.0746],...
        'FontWeight','bold',...
```

```
        'String',{'start point'},...
        'FitHeightToText','on');
    % Create arrow
annotation3 = annotation(figure1,'arrow',[0.4738 0.3489],[0.1774 0.2358]);
    % Create textbox
    annotation4 = annotation(...
        figure1,'textbox',...
        'Position',[0.4732 0.1444 0.2411 0.06032],...
        'FitHeightToText','off',...
        'FontWeight','bold',...
        'String',{'FMINCON solution'});
end


function [stop options changed] = psOut(optimvalues,options,flag)
stop = false;
changed = false;
figure1 = gcf;
if strcmpi(flag,'done')
    % Create textbox
    annotation1 = annotation(...
        figure1,'textbox',...
        'Position',[0.4679 0.1357 0.3321 0.06667],...
        'FitHeightToText','off',...
        'FontWeight','bold',...
        'String',{'Pattern Search solution'});
    % Create textbox
    annotation2 = annotation(...
        figure1,'textbox',...
        'Position',[0.5625 0.2786 0.2321 0.06667],...
        'FitHeightToText','off',...
        'FontWeight','bold',...
        'String',{'FMINCON solution'});
    % Create textbox
    annotation3 = annotation(...
        figure1,'textbox',...
        'Position',[0.3714 0.6905 0.1571 0.06449],...
        'FitHeightToText','off',...
        'FontWeight','bold',...
        'String',{'Start point'});
    % Create arrow
    annotation4 = annotation(figure1,'arrow',[0.7161 0.6768],[0.3452 0.4732]);
    % Create arrow
    annotation5 = annotation(figure1,'arrow',[0.4697 0.35],[0.1673 0.2119]);
    % Create arrow
```

```
    annotation6 = annotation(figure1,'arrow',[0.4523 0.6893],[0.6929 0.6]);
end
```

Output

```
    In fmincon at 260
    In PS at 33
Xop =
    -5 -5
Fop =
    -250
In fmincon at 260
    In PS at 70
Xop =
    1.2861 -4.8242
Fop =
    -86.0221
Xps =
    -5 -5
Fps =
    -247.3159
```

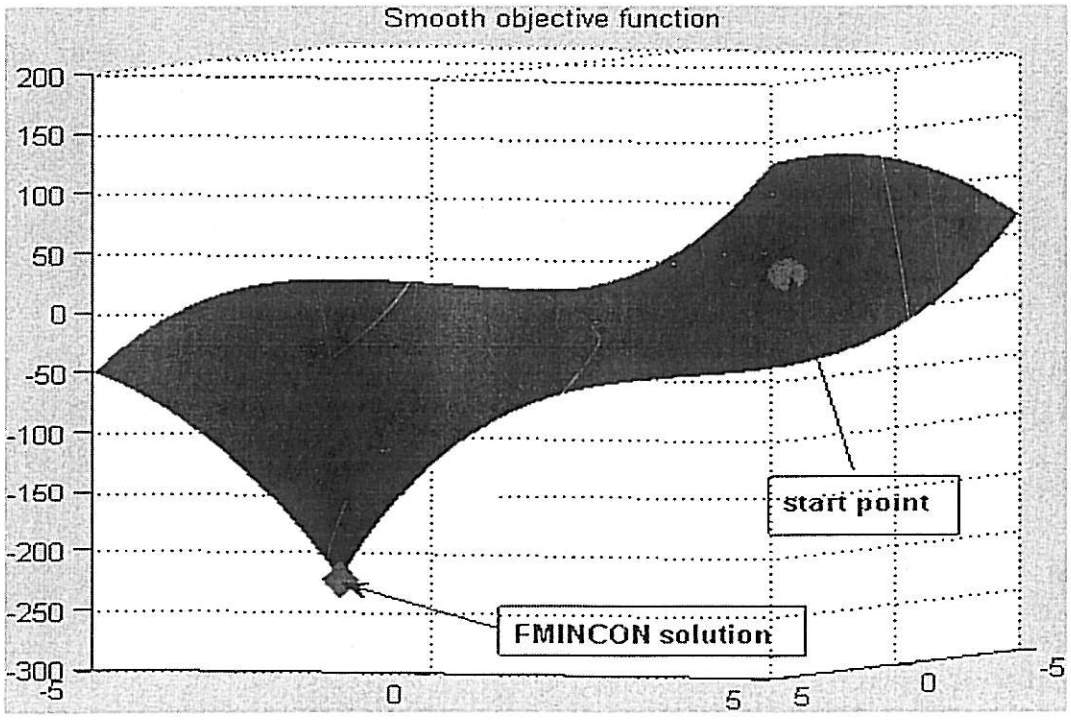

Fig. 8.13 Smooth objective function

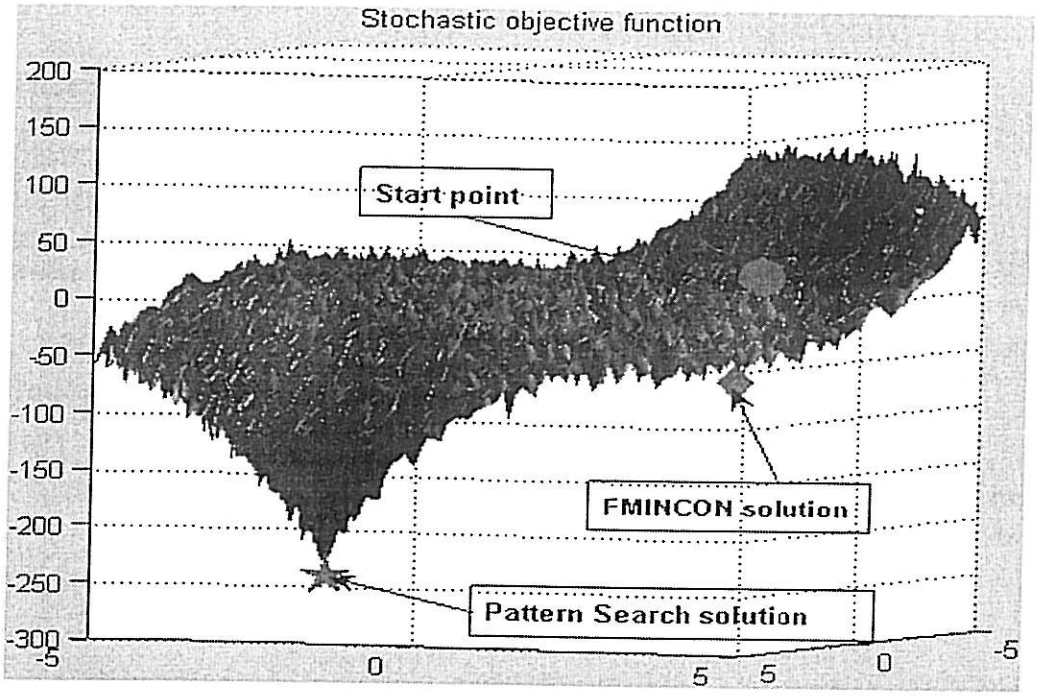

Fig. 8.14 Stochastic objective function

Pattern search algorithm is not affected by random noise in the objective functions. Pattern search requires only function value and not the derivatives, hence a noise (of some uniform kind) may not affect it.
Pattern search requires a lot more function evaluation to find the minima, a cost for not using the derivatives.

Problem 4

Write a Program to maximize sin(x) within the range $0<x<3.14$

Source Code

```
%program for Genetic algorithm to maximize the function f(x) =sin(x)
clear all;
clc;
%x ranges from 0 to 3.14
%five bits are enough to represent x in binary representation
n=input('Enter no. of population in each iteration');
nit=input('Enter no. of iterations');
%Generate the initial population
[oldchrom]=initbp(n,5)
%The population in binary is converted to integer
FieldD=[5;0;3.14;0;0;1;1]
```

```
for i=1:nit
    phen=bindecod(oldchrom,FieldD,3); % phen gives the integer value of the
    %binary population
    %obtain fitness value
    FitnV=sin(phen);
    %apply roulette wheel selection
    Nsel=4;
    newchrix=selrws(FitnV, Nsel);
    newchrom=oldchrom(newchrix,:);
    %Perform Crossover
    crossoverrate=1;
    newchromc=recsp(newchrom,crossoverrate); %new population after crossover
    %Perform mutation
    vlub=0:31;
    mutationrate=0.001;
    newchromm=mutrandbin(newchromc,vlub,mutationrate); %new population
    %after mutation
    disp('For iteration');
    i
    disp('Population');
    oldchrom
    disp('X');
    phen
    disp('f(X)');
    FitnV
    oldchrom=newchromm;
end
```

Output

```
Enter no. of population in each iteration5
Enter no. of iterations5
oldchrom =
   0   1   0   0   0
   0   1   0   0   1
   1   0   1   0   1
   1   0   0   1   1
   0   0   0   1   0
FieldD =
   5.0000
        0
   3.1400
        0
        0
```

```
    1.0000
    1.0000
For iteration
i =
    1
Population
oldchrom =
    0  1  0  0  0
    0  1  0  0  1
    1  0  1  0  1
    1  0  0  1  1
    0  0  0  1  0
X
phen =
    1
    1
    2
    2
    0
f(X)
FitnV =
    0.8415
    0.8415
    0.9093
    0.9093
         0
For iteration
i =
    2
Population
oldchrom =
    0  0  1  0  1
    1  0  1  1  0
    0  0  0  0  0
    1  0  0  0  1
X
phen =
    1
    2
    0
    2
f(X)
FitnV =
    0.8415
    0.9093
         0
```

```
    0.9093
For iteration
i =
    3
Population
oldchrom =
    0   0   0   0   1
    0   1   1   0   1
    0   1   1   1   0
    1   0   0   0   0
X
phen =
    0
    1
    1
    2
f(X)
FitnV =
         0
    0.8415
    0.8415
    0.9093
For iteration
i =
    4
Population
oldchrm =
    1   1   0   1   1
    0   0   1   0   0
    0   0   1   1   1
    0   1   0   1   1
X
phen =
    3
    0
    1
    1
f(X)
FitnV =
    0.1411
         0
    0.8415
    0.8415
For iteration
i =
    5
```

```
Population
oldchrom =
     0     1     1     0     0
     1     0     0     0     1
     0     0     0     0     0
     1     0     0     1     1
X
phen =
     1
     2
     0
     2
f(X)
FitnV =
    0.8415
    0.9093
         0
    0.9093
```

Problem 5

Find the minimum of the quadratic equation $f(x)=x^2+5x+2$

Source Code

The minimizes of the given quadratic equation is done within a single command line function. The function used is "ga"

1. Define the given function $f(x) = x^2+5x+2$ in a separate m-file as shown below.

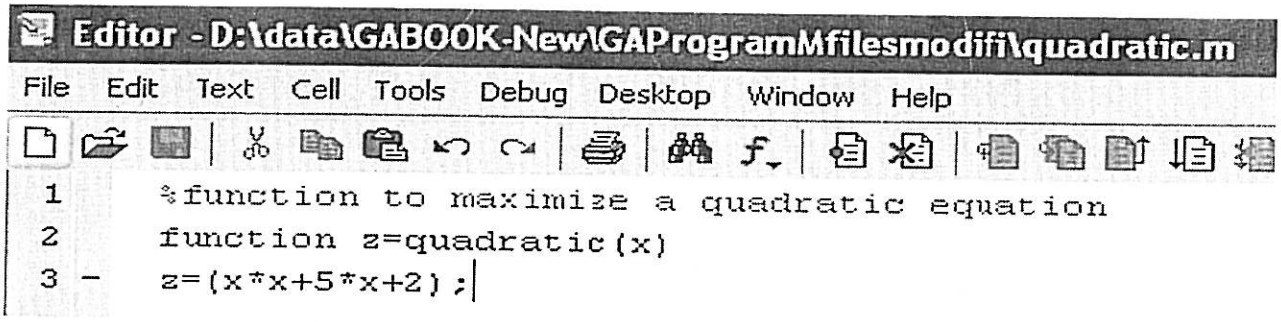

Fig. 8.15 Declaration of the quadratic function

2. Then use the command "ga" to obtained the minimized value of f(x). The format of command "ga" is,

X = GA(FITNESSFCN,NVARS) finds the minimum of FITNESSFCN using GA. NVARS is the dimension (number of design variables) of the FITNESSFCN.

3. Thus the command for the given problem is given by,

```
x=ga(@quadratic,1)
```

On running the above command, the output is obtained as given below.

Output

Optimization terminated: maximum number of generations exceeded.

x =
-2.5002

In this case, all the operations are performed with the default setting of the command "ga".
In the above case, if the command is specified as,

```
[x, fval, reason, output, population, scores] = ga(@quadratic,1)
```

then the output obtained is,
Optimization terminated: maximum number of generations exceeded.

x =
-2.5002
fval =
-4.2500
reason =
Optimization terminated: maximum number of generations exceeded.
output =
randstate: [35x1 double]
randnstate: [2x1 double]
generations: 100
funccount: 2000
message: 'Optimization terminated: maximum number of generations exceeded.'
population =
-2.5002
-2.5002
-2.5002
-2.4933
-2.5002
-2.5002
-2.5002

```
    -2.5002
    -2.5002
    -2.5002
    -2.5002
    -2.5097
    -2.5002
    -2.5002
    -2.5173
    -2.5002
    -2.5002
    -2.5002
    -2.4855
    -2.5002
scores =
    -4.2500
    -4.2500
    -4.2500
    -4.2500
    -4.2500
    -4.2500
    -4.2500
    -4.2500
    -4.2500
    -4.2500
    -4.2500
    -4.2499
    -4.2500
    -4.2500
    -4.2497
    -4.2500
    -4.2500
    -4.2500
    -4.2498
    -4.2500
```

Problem 6

Write a program to minimize Rastrigin's function. Also plot the best fitness value.

Description

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

$$\boldsymbol{R(x)} = 20 + \boldsymbol{x}_1^2 + \boldsymbol{x}_2^2 - 10(\boldsymbol{cos}\, 2\pi\boldsymbol{x}_1 + \boldsymbol{cos}\, 2\pi\boldsymbol{x}_2).$$

The MATLABtoolbox contains an M-file, rastriginsfcn.m, that computes the values of Rastrigin's function.

Source Code

The function is defined as follows:

```
function Rasfun = rastriginsfcn(pop) %RASTRIGINSFCN Compute the
    %"Rastrigin" function.
    Rasfun = 10.0 * size(pop,2) + sum(pop .^2 - 10.0 * cos(2 * pi .* pop),2);
```

The program for minimizing this function is given as,

```
%Program to minimize Rastrigins Function
%Depending upon user's need Options can be specified using the command
    'gaoptimset'. %If
%Options not specified default options are chosen.
options=gaoptimset('CrossoverFcn',@crossoversinglepoint,...
    'MutationFcn',@mutationuniform,'Plotfcns',@gaplotbestf)
%Generating the genetic algorithm for 10 variables with the options specified
    %above
[x,fval,reason] = ga(@rastriginsFcn,10,options)
```

Output

```
options =
    PopulationType: 'doubleVector'
     PopInitRange: [2x1 double]
    PopulationSize: 20
     EliteCount: 2
    CrossoverFraction: 0.8000
    MigrationDirection: 'forward'
    MigrationInterval: 20
    MigrationFraction: 0.2000
      Generations: 100
        TimeLimit: Inf
     FitnessLimit: -Inf
     StallGenLimit: 50
     StallTimeLimit: 20
    InitialPopulation: [ ]
     InitialScores: [ ]
     PlotInterval: 1
       CreationFcn: @gacreationuniform
    FitnessScalingFcn: @fitscalingrank
```

```
        SelectionFcn: @selectionstochunif
        CrossoverFcn: @crossoversinglepoint
        MutationFcn: @mutationuniform
         HybridFcn: [ ]
          Display: 'final'
          PlotFcns: @gaplotbestf
         OutputFcns: [ ]
         Vectorized: 'off'
Optimization terminated: maximum number of generations exceeded.
x =
    Columns 1 through 9
      0.9840  0.8061  0.9698  0.0493  0.0070  0.0408  0.0671  0.0970  0.9412
    Column 10
      0.9848
fval =
    15.4207
reason =

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

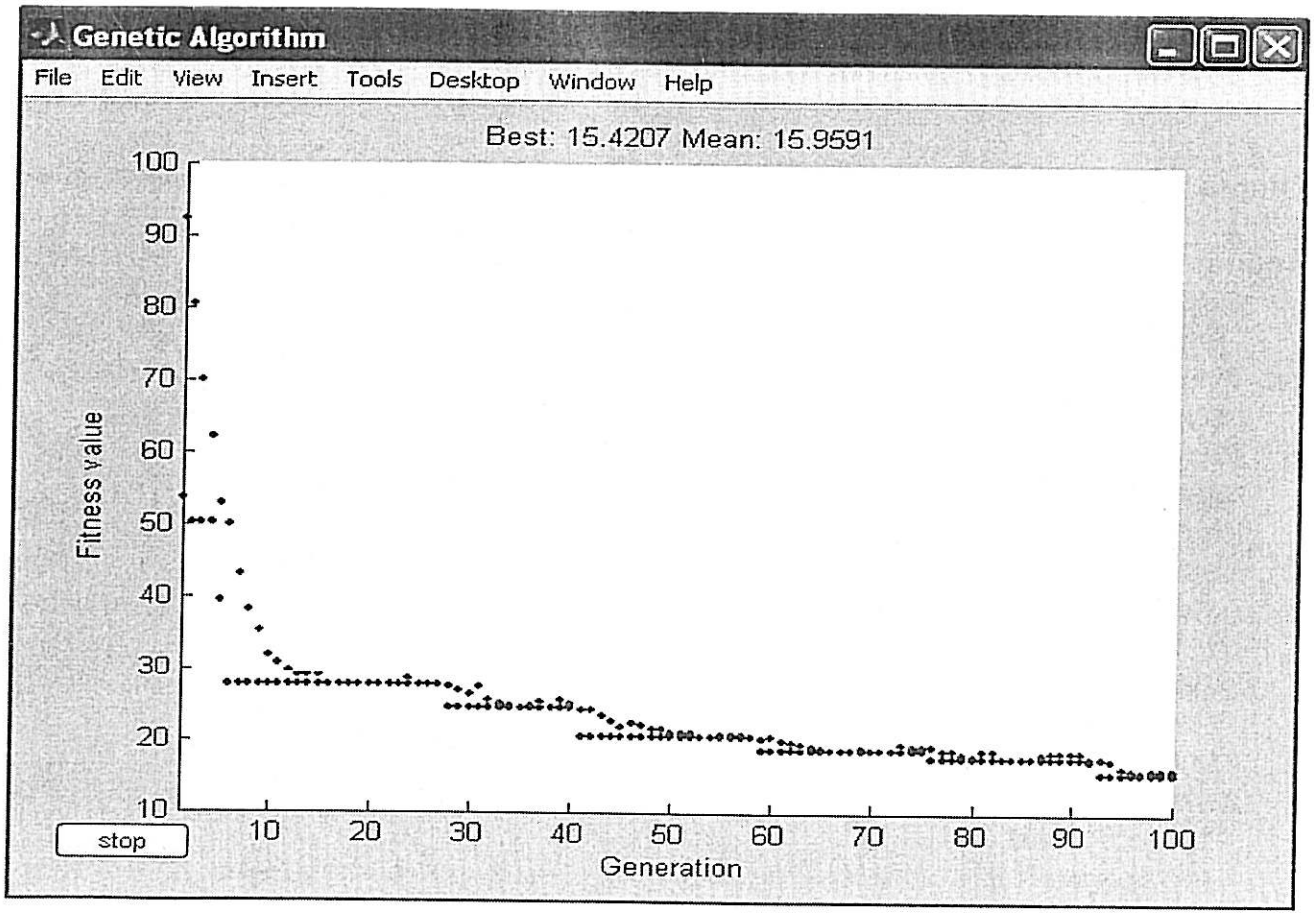

Fig. 8.16 Best fitness plot for minimization of Rastrigins function

Problem 7

Write a program to optimize a function $f(x_1,x_2)=3x_1+9x_2$. Set suitable plot options.

Source Code

The function is defined as follows:

```
%function to be optimized
function z=twofunc(x)
z=(3*x(1)+9*x(2));
```

The program code for optimization process is as follows:

```
%Program to optimize a function with two variables
%Options are set using the command 'gaoptimset'.
clc;
clear all;
options = gaoptimset('PlotFcns',...
          {@gaplotbestf,@gaplotbestindiv,@gaplotexpectation,
          @gaplotstopping});
%Generating the genetic algorithm for 2 variables
[x,fval,reason] = ga(@twofunc,2,options)
```

Output

```
x =
   -14.2395 -24.4720
fval =
   -262.9662
reason =
```

Optimization terminated: maximum number of generations exceeded.

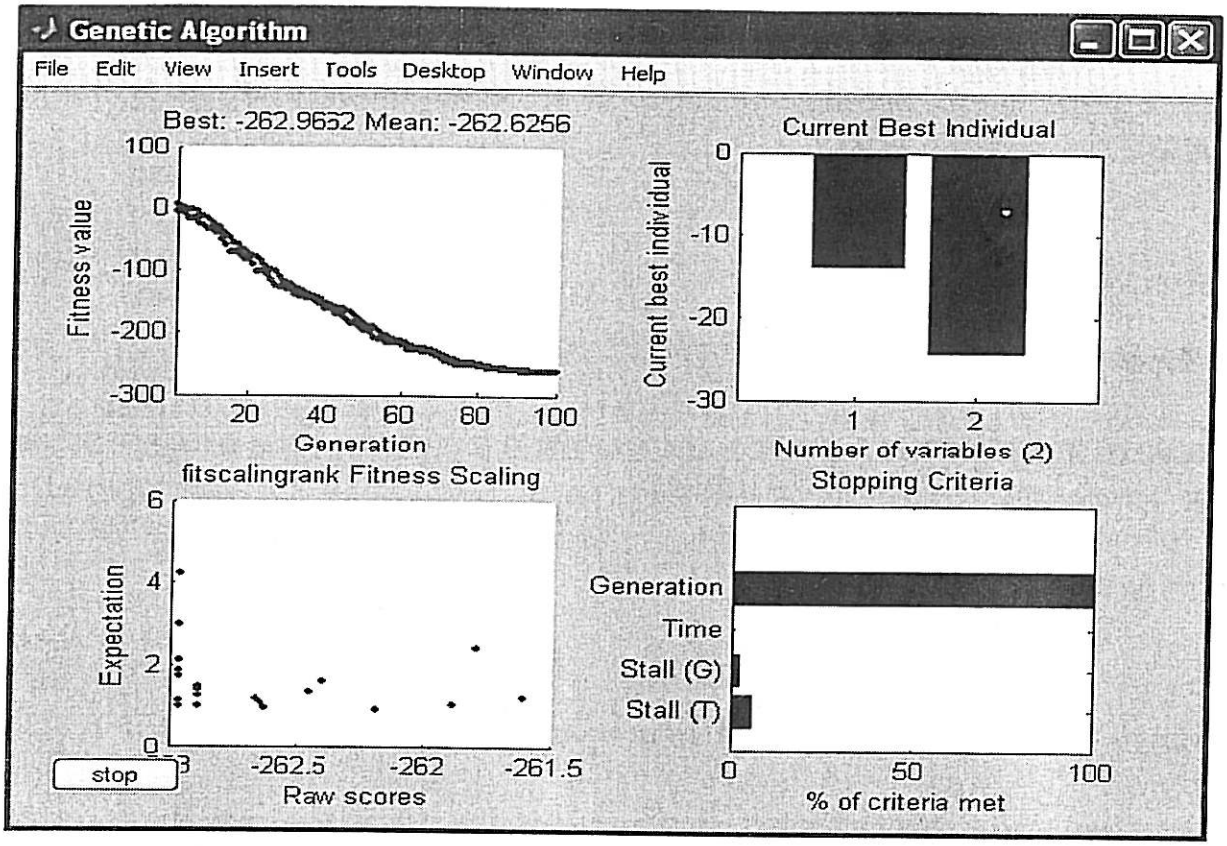

Fig. 8.17 Different Plots during optimization of the function $f(x)=3x_1+9x_2$

Problem 8

Obtain the best fitness value when the given linear function $f(x_1,x_2,x_3)=-(3x_1+7x_2+6x_3)$ is minimized.

Source Code

The function is defined as follows:

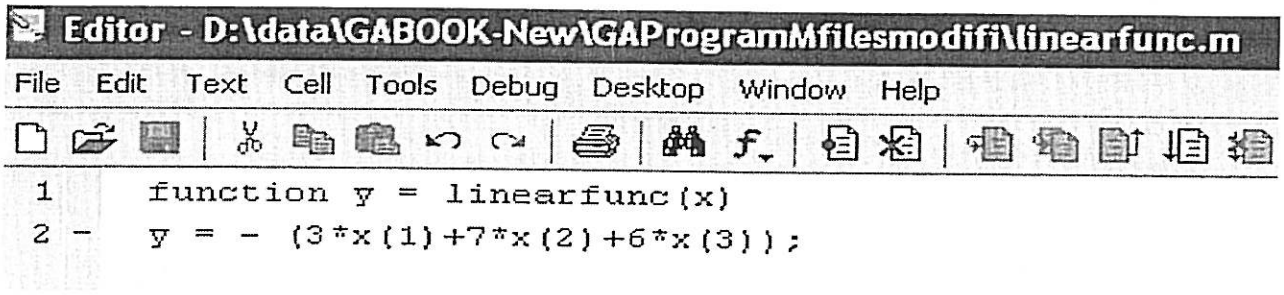

Fig. 8.18 Definition of the linear function

The program code is given by,

```
%Program to optimize the linear function with 3 variables
clc;
clear all;
%Setting the required options
options = gaoptimset('PlotFcns',...
          {@gaplotbestf,@gaplotbestindiv});
%Generating the genetic algorithm
[x,fval]=ga(@linearfunc,3,options)
```

Output

Optimization terminated: maximum number of generations exceeded.

```
x =
   12.3074  23.0895  19.5800
fval =
   -316.0282
```

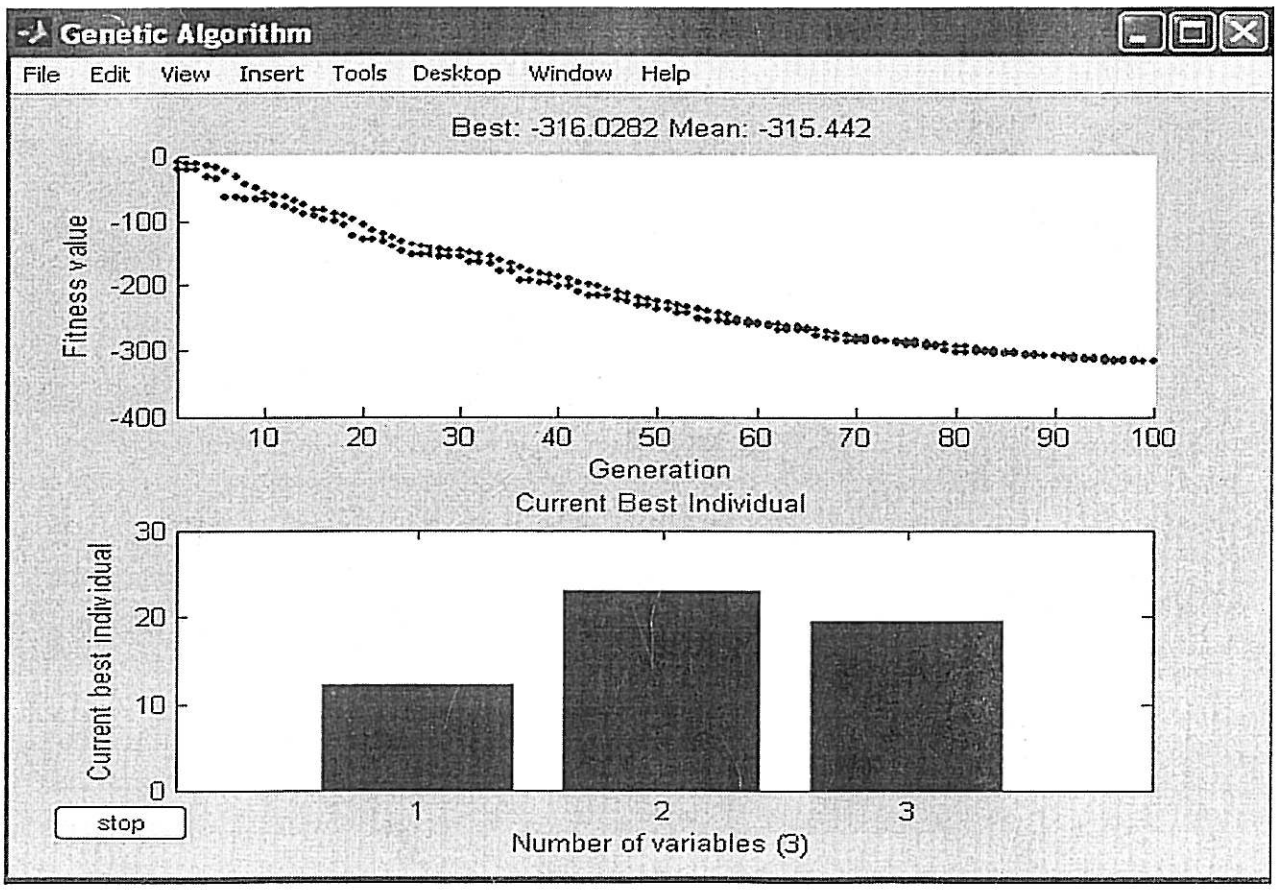

Fig. 8.19 Plot of the best fitness and best individual of the given linear function

Problem 9

Use **Gatool** and maximize the quadratic equation $f(x) = x^2+3x+2$ within the range $-6 \leq x \leq 0$.

Function Definition

Define the given function $f(x) = x^2+3x+2$ in a separate m-file as shown in Fig 8.20

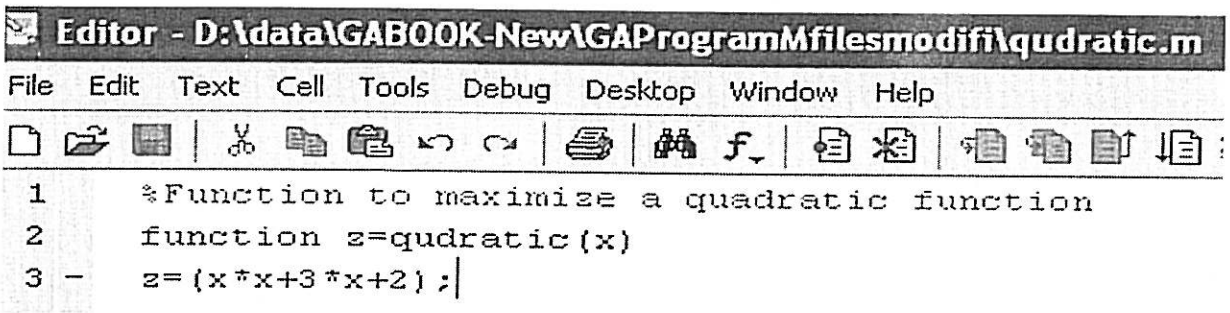

Fig. 8.20 M-file showing defined quadratic function

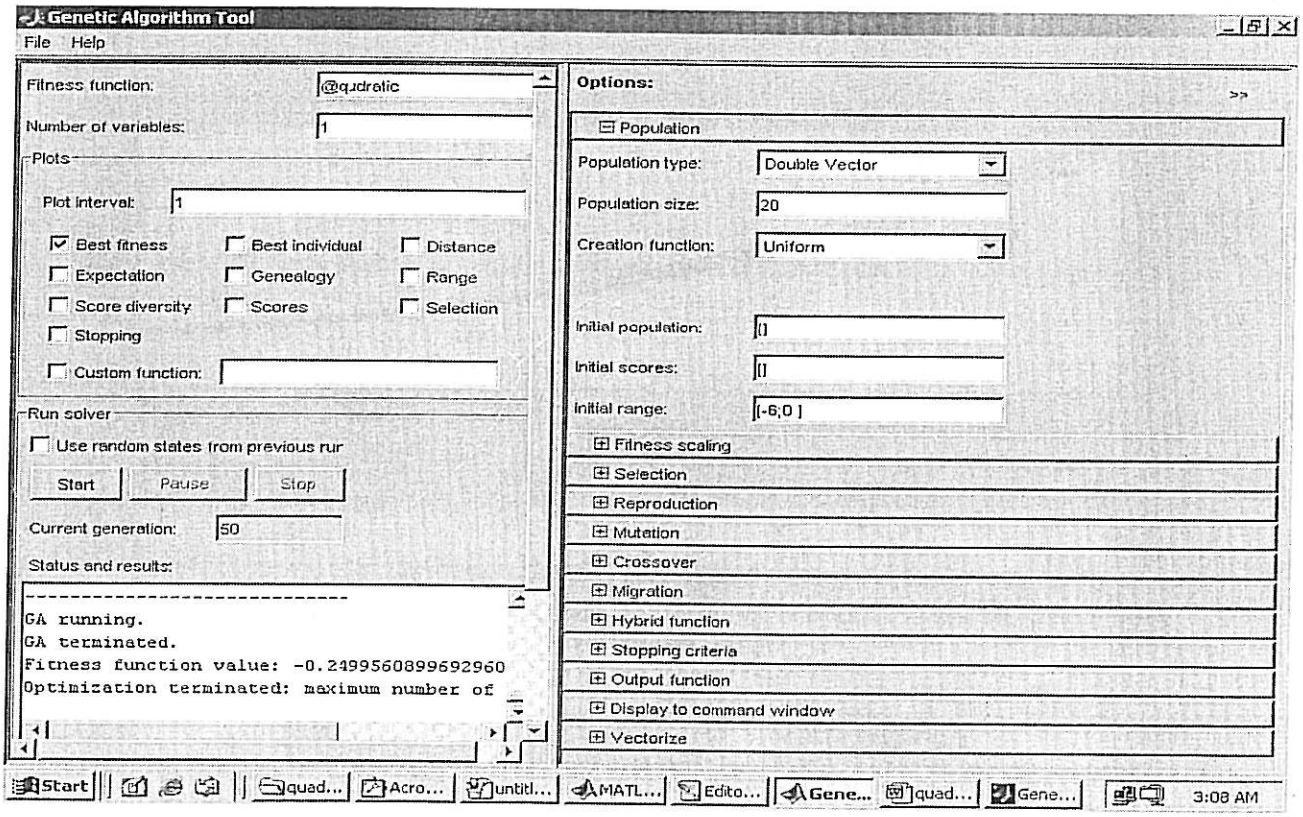

Fig. 8.21 Genetic algorithm tool for quadratic equation

Creation of Gatool

On typing "gatool" in the command prompt, the GA tool box opens. In tool, for fitness value type **@qudratic** and mention the number of variables defined in the function. Select **best fitness** in plot and specify the other parameters as shown in Fig. 8.21

Output

The output showing the best fitness for 50 generations is shown in Fig, 8.22

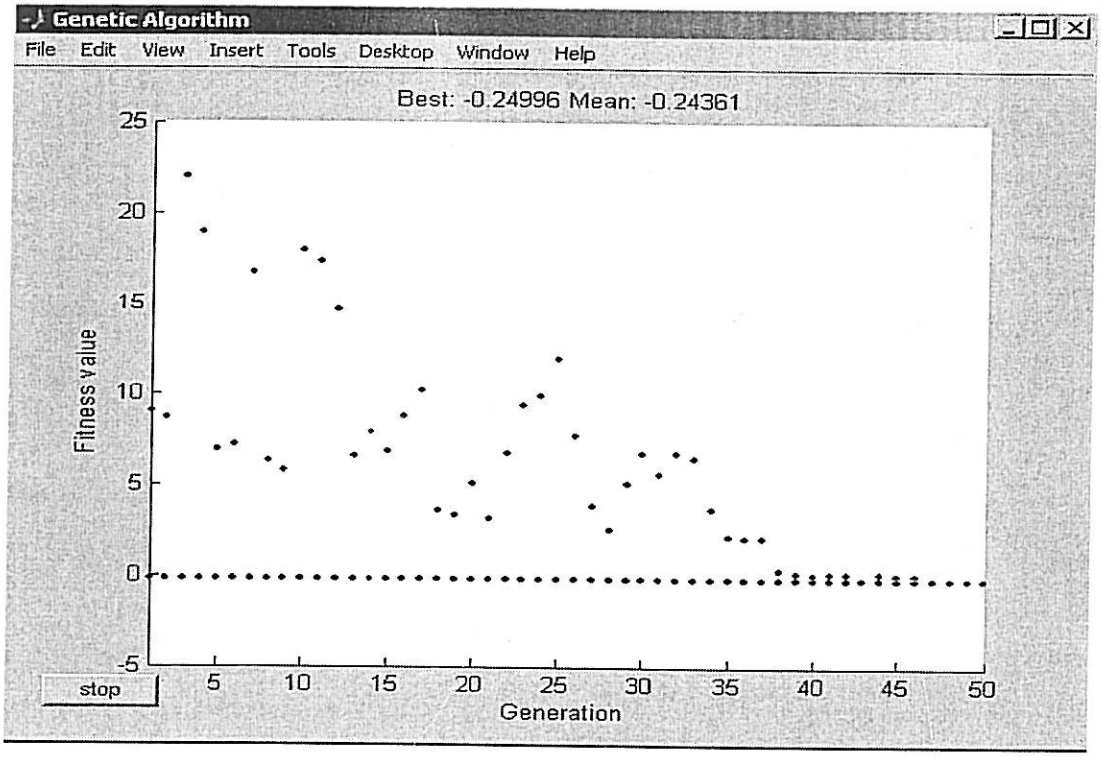

Fig. 8.22 Output response (Best fitness) for the function $f(x) = x^2+3x+2$

The status and results for this functions for 50 generations is as shown in Fig. 8.23

Problem 10

Create a **Gatool** to maximize the function $f(x1,x2)=4x_1+5x_2$ within the range 1 to 1.1

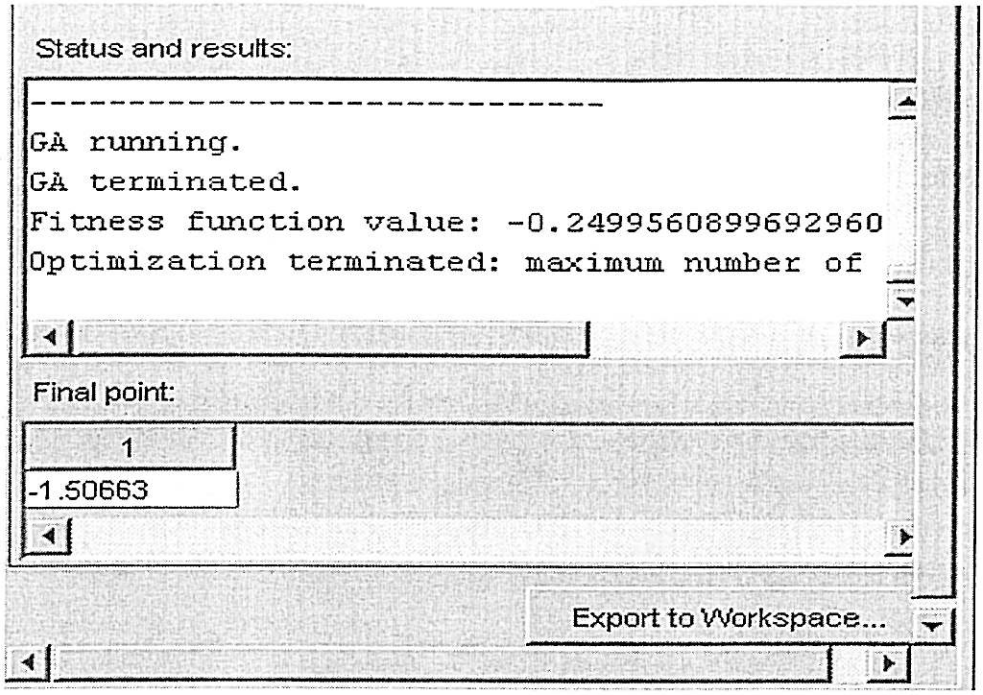

Fig. 8.23 Status and results for the function $f(x) = x^2+3x+2$

Function Definition

Define the given function $f(x1,x2)=4x_1+5x_2$ in a separate m-file as shown in Fig. 8.24

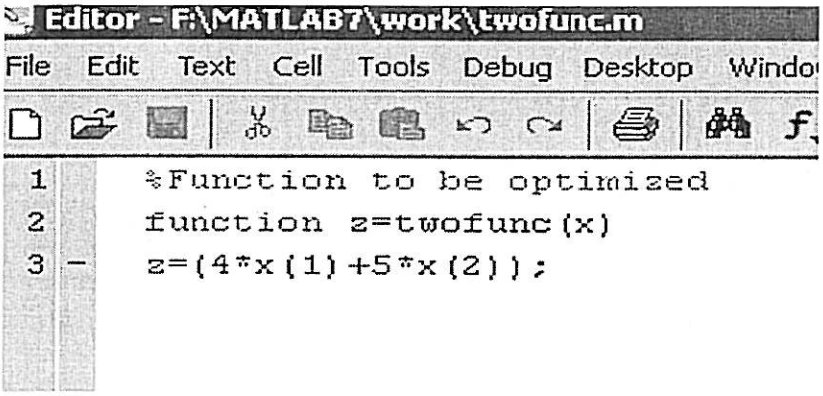

Fig. 8.24 M-file showing defined function

Creation of Gatool

On typing "gatool" in the command prompt, the GA toolbox opens. In tool, for fitness value type **@twofunc** and mention the number of variables defined in the

function. Select **best fitness** and **best individual** in plot and specify the other parameters as shown in Fig. 8.25

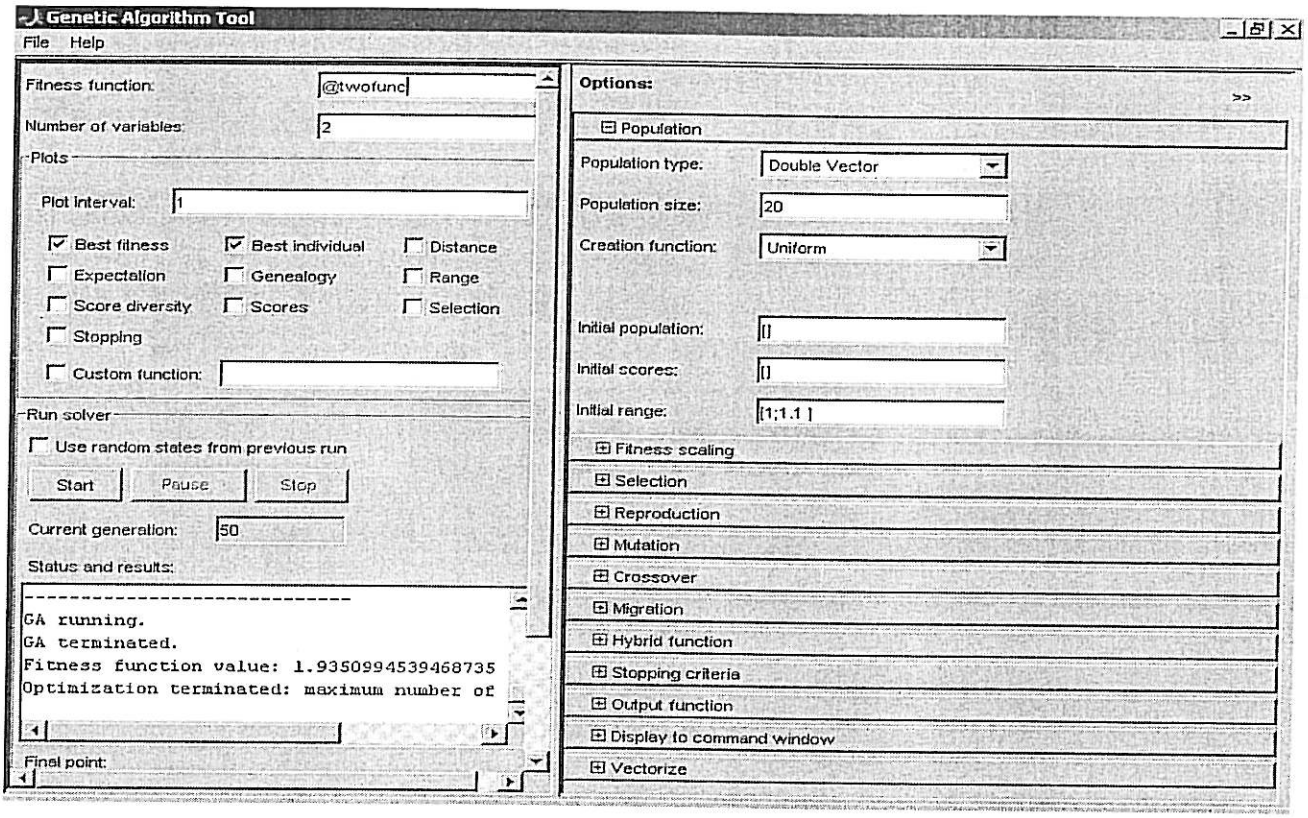

Fig. 8.25 Genetic algorithm tool for given function

Output

The output for 50 generations is as shown in Fig. 8.26 The output also shows the best inidvidual.
The status and results for this function is as shown in Fig. 8.27

Problem 11

Use **Gatool** and maximize the function

$$f(x1, x2, x3) = -5\sin(x1)\sin(x2)\sin(x3) + -\sin(5x1)\sin(5x2)\sin(x3)$$

where $0 <= xi <= pi$, for $1 <= i <= 3$.

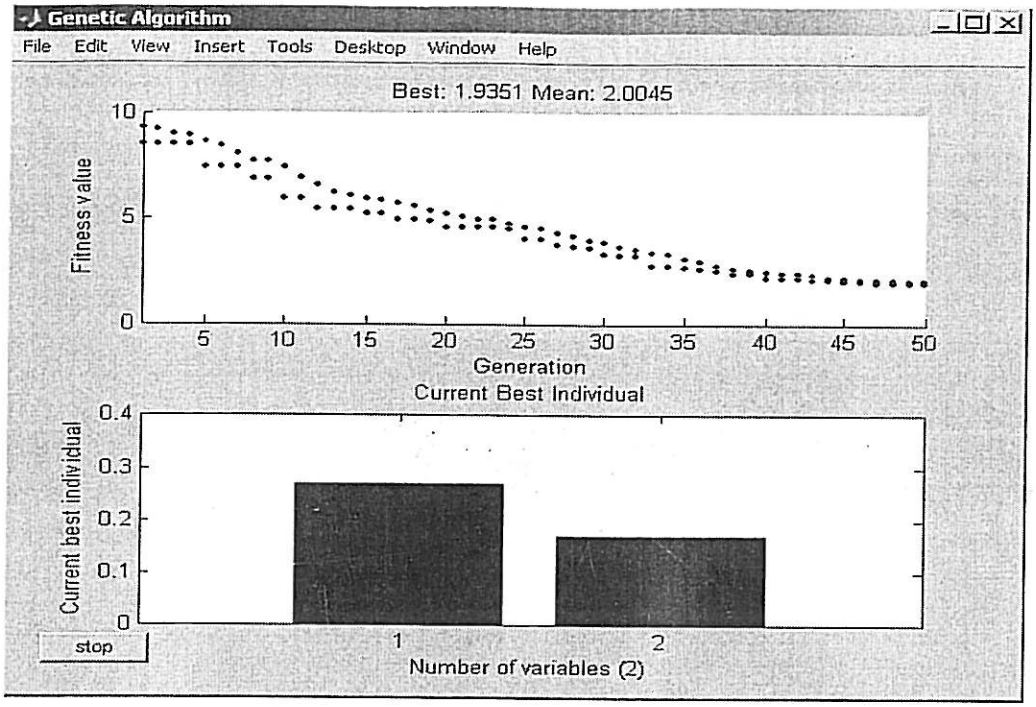

Fig. 8.26 Output response (Best fitness and best individual)

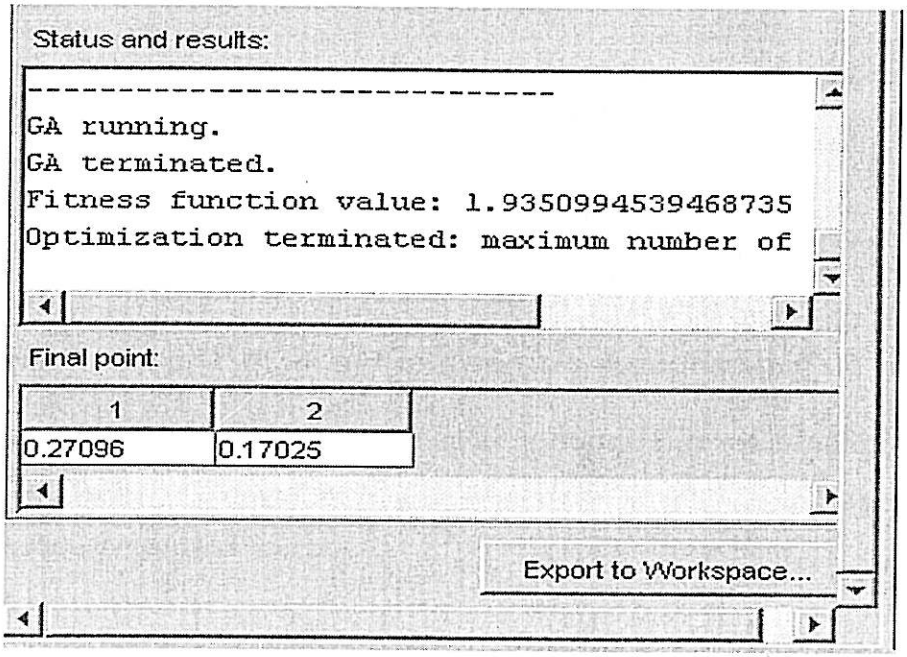

Fig. 8.27 Status and results for the function $f(x1,x2)=4x_1+5x_2$

Function Definition

Define the given function

$$f(x1, x2, x3) = -5 \sin(x1) \sin(x2) \sin(x3) + -\sin(5x1) \sin(5x2) \sin(x3)$$

in a separate m-file as shown in Fig. 8.28

Editor - D:\data\GABOOK-New\GAProgramMfilesmodifi\sinefn.m

File Edit Text Cell Tools Debug Desktop Window Help

```
1    %Function to maximize a quadratic function
2    function z=sinefn(x)
3    z=(-(5*sin(x(1))*sin(x(2))*sin(x(3)))+
4      (-(sin(5*(x(1)))*sin(5*(x(2)))*sin(x(3)))));
```

Fig. 8.28 M-file showing defined sine function

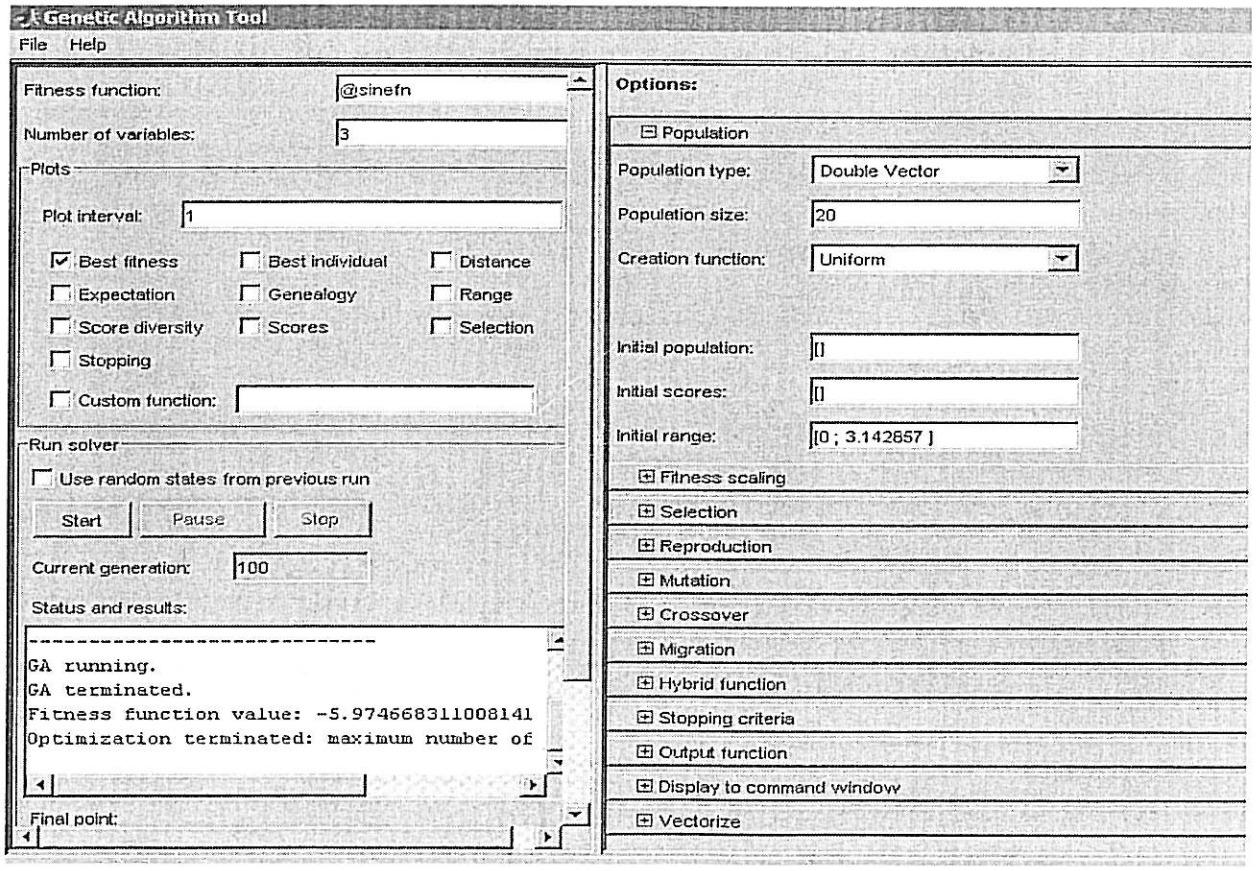

Fig. 8.29 Genetic algorithm tool for sine equation

Creation of Gatool

On typing "gatool" in the command prompt, the GA tool box opens. In tool, for fitness value type **@sinefn** and mention the number of variables defined in the function. Select **best fitness** in plot and specify the other parameters as shown in Fig. 8.29

Output

The output for 100 generations is as shown in Fig. 8.30.
The status and results for this function is as shown in Fig. 8.31

Problem 12

Create a "gatool" to minimize the function f(x) = cosx within the range $0 \leq x \leq 3.14$

Function Definition

Define the given function f(x) = cosx in a separate m-file as shown below:

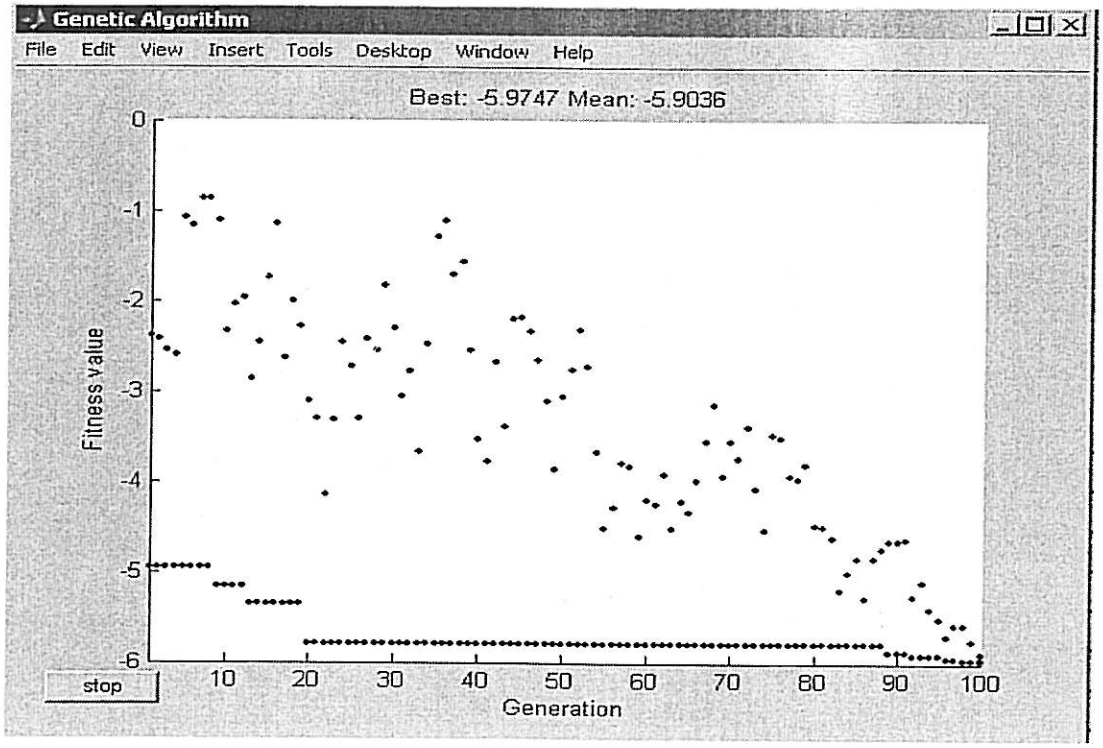

Fig. 8.30 Output response (Best fitness) for the function f(x1, x2, x3) = -5sin(x1)sin(x2)sin(x3)+–sin(5x1)sin(5x2)sin(x3)

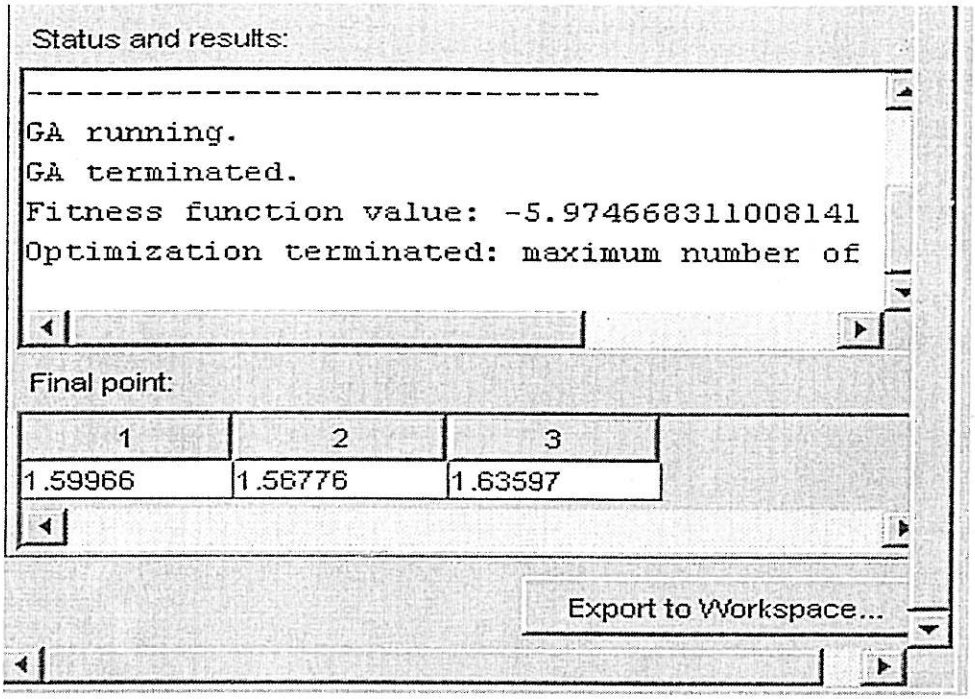

Fig. 8.31 Status and results for the function f(x1,x2,x3) = -5sin(x1)sin(x2)sin(x3)+-sin(5x1) sin(5x2)sin(x3)

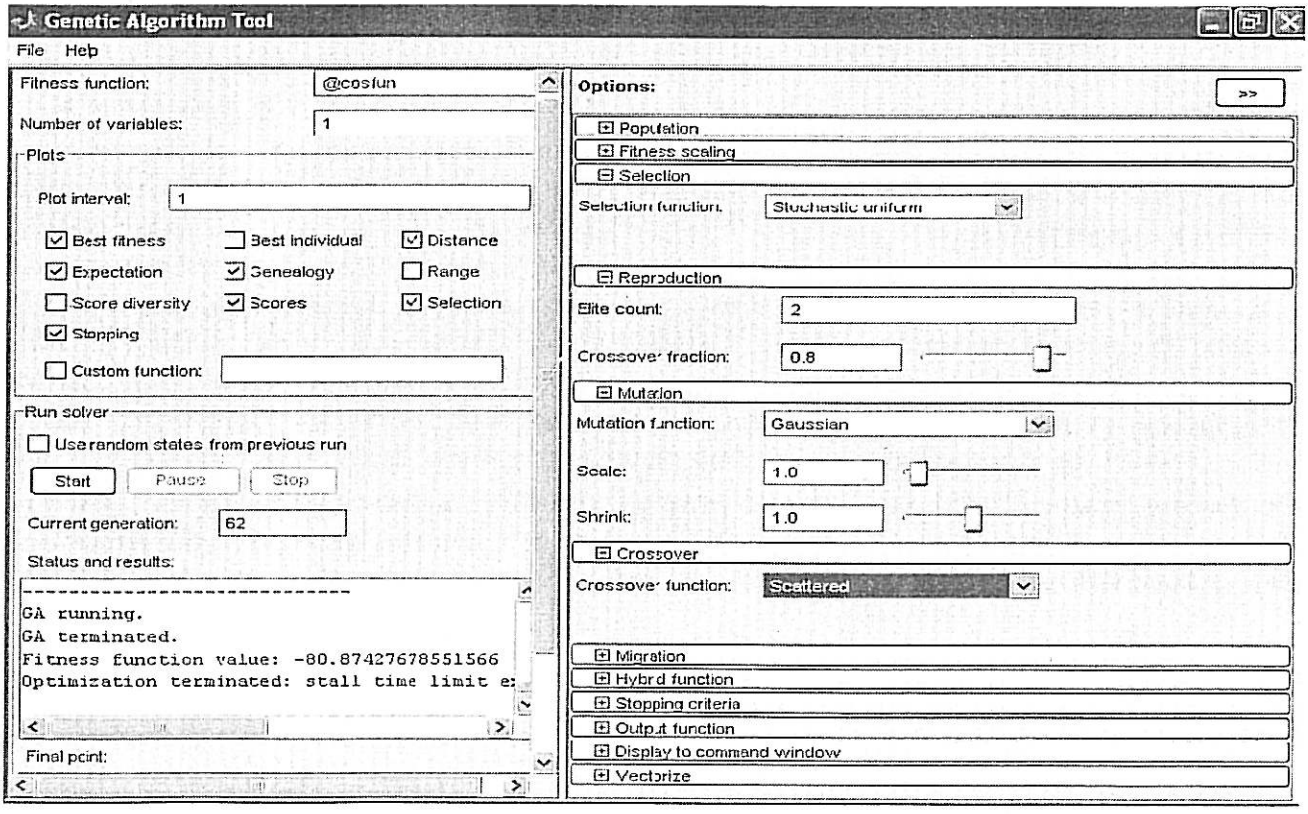

Fig. 8.32 Genetic algorithm tool for cosine function

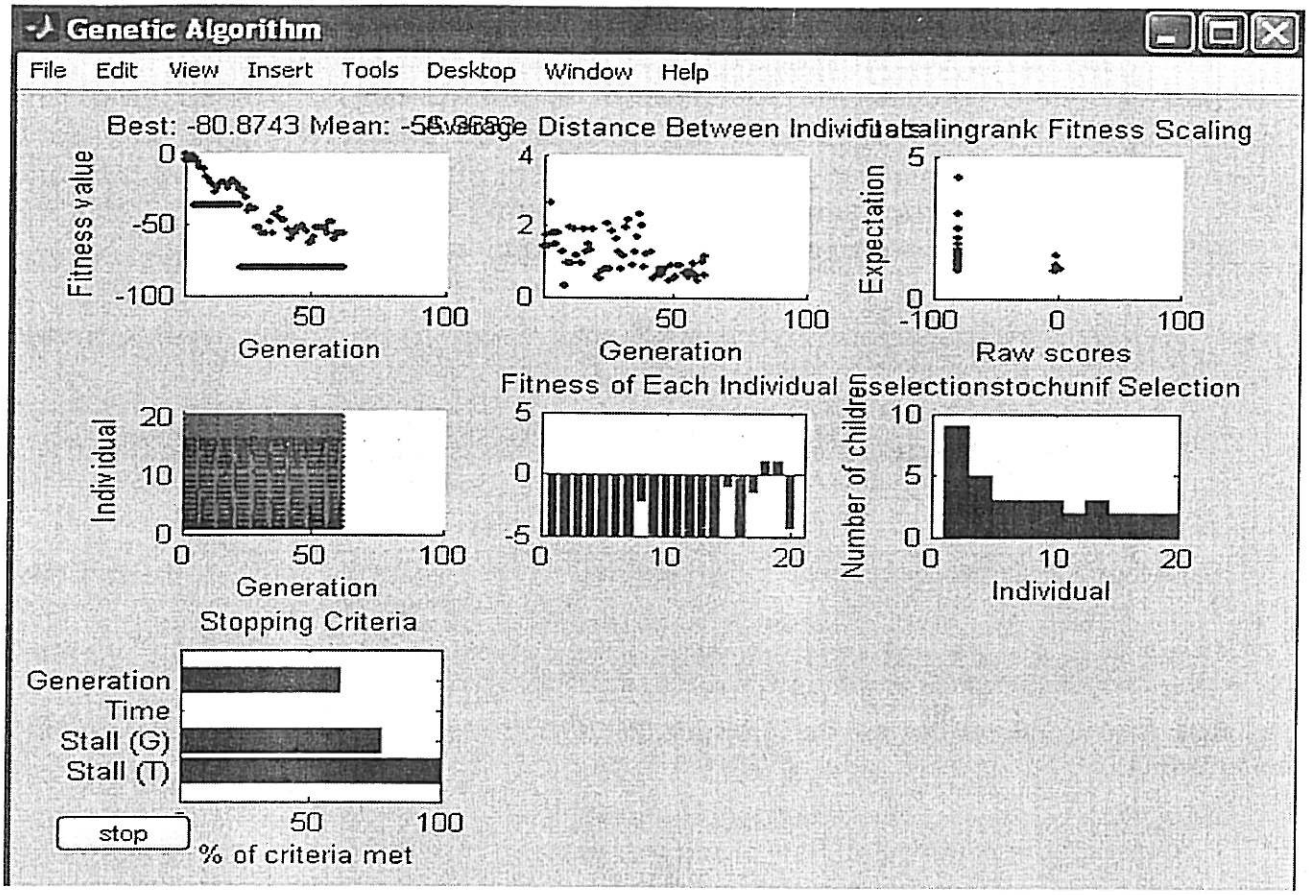

Fig. 8.33 Output response (Best fitness) for the function f(x) = cosx

```
%Function to minimize cosine function
function z=cosfun(x)
z=1/cos(x);
```

Creation of Gatool

On typing "gatool" in the command prompt, the GA toolbox opens. In tool, for fitness value type **@cosfn** and mention the number of variables defined in the function. Select **best fitness** in plot and specify the other parameters as shown in Fig. 8.32

Output

The output for 62 generations is as shown in Fig. 8.33

The status and results for this function is shown in Fig. 8.34

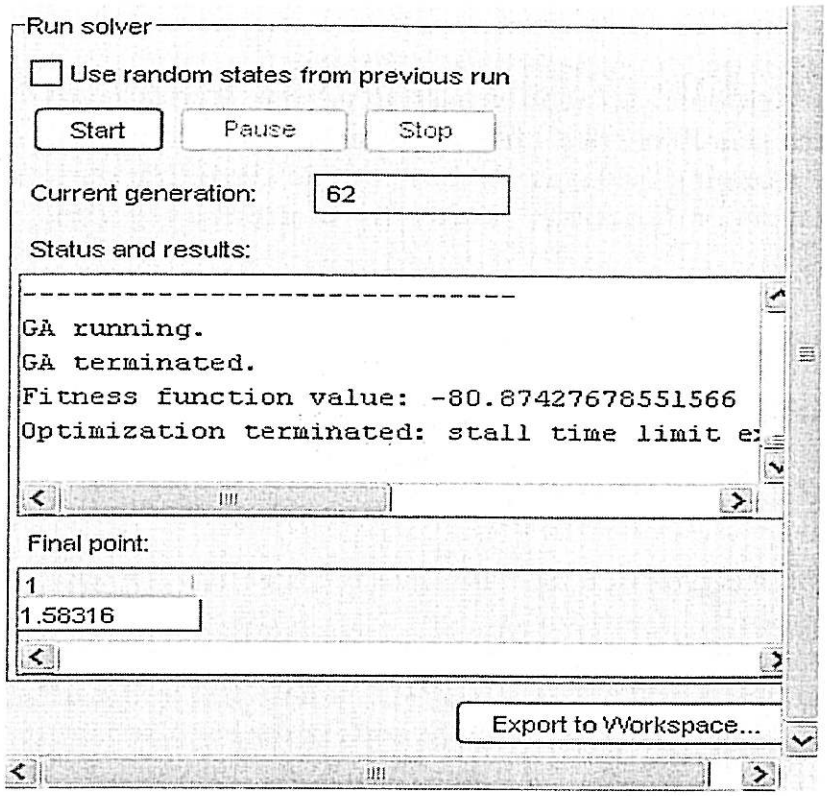

Fig. 8.34 Status and results for the function f(x) = cosx

8.6 Summary

In this chapter, the implementation of genetic algorithm using MATLAB software has been dealt. The various functions, which includes objective functions, crossover operations, mutation operations, plot functions, insertion operators, fitness scaling, utility functions and so on are listed for implementing the optimization process. One of the best tool of MATLAB is its Graphical user Interface (GUI) toolbox. The Genetic Algorithm GUI Toolbox plays a major role for obtaining an optimized solution and to find the best fitness value. This GUI tool gives us different plot related to best individual, best scores, distance, range, scorediversity, genealogy, stopping condition, best fitness value and generations. Few examples have been dealt using the MATLAB functions that are simulated and the outputs for easy reference.

Review Questions

1. Write note on the importance of MATLAB Software.
2. List the various functions used in MATLAB for performing crossover nd mutation operations.
3. Mention the different objective functions present in MATLAB Toolbox.
4. State the advantages of Graphicl User Interface toolbox.

5. Differentiate between forward and backward migration. What is migration rate?
6. Discuss in detail on the various plot functions.
7. What are the major data structures used in genetic algorithm MATLAB toolbox?
8. How is pattern search carried out using the toolboxes?
9. Mention few stopping criterias used in genetic algorithm optimization process.
10. What are conversion functions neccessary in genetic algorithm simulation process?

Exercise Problems

1. Implement a travelling saleman problem covering about 20 cities using MATLAB.
2. Write a MATLAB program to maximize the function $f(x)=4x^4+3x^3+2x^2+x+1$
3. Consider a binomial function of your own. Optimize the function to obtain a minimized solution.
4. Minimize the function $f(x,y,z)=x^2+y^2+z^2$, where x,y, and z are permitted to vary between –512 and +512. Use suitable coding for each substring.
5. Implement a MATLAB routine to perform mutation using an exponential distribution
6. Consider a Dejong's function, using MATLAB tool compare and contrast the alternative selection methods
7. Choose an application of your own, compare and contrast the output performance obtained using various crossover and mutation schemes
8. Create a "gatool" to minimize the function $f(x) = secx$ within the range $0 \leq x \leq 3.14$
9. Maximize the function f (x1, x2, x3) = -10sin(x1)sin(x2)sin(x3)+20sin(5x1) sin(5x2)sin(x3) where 0 <= xi <= pi, for 1 <= i <= 3.
10. Obtain the best fitness value when the given linear function $f(x_1,x_2,x_3)=-(9x_1+7x_2+6x_3)$ is minimized