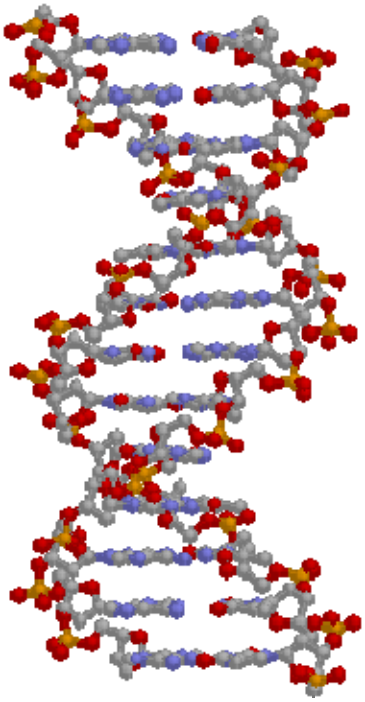
School of Electrical, Electronic and Computer Engineering

EEE 8005 – Student Directed Learning (SDL)

Industrial Automation – Genetic Algorithms

Written by: Shady Gadoue



[1]

Module Leader: Dr. Damian Giaouris

Damian.Giaouris@ncl.ac.uk

Genetic Algorithms

Introduction

Genetic Algorithm (GA) is a search technique that mimics the mechanisms of natural selection. Recently, GA has been recognized as an effective and powerful technique to solve optimization and search problems which represents an intelligent utilization of a random search within a defined search space to solve a problem. During GA computing, a population of artificial individuals is modified repeatedly based on biological evolution rules that converge towards better solution of the problem being solved. At each step individuals are selected in random from the current population to be parents. These individuals are used to produce children for the next generation. Based on biological basics, the fittest individuals survive and the least fit die. Through successful generations, the population evolves towards an optimal solution. Compared with other optimization techniques, particularly gradient search methods, GA is superior in avoiding local minima which is a common aspect of nonlinear systems. The difference between local and global minimum is shown in Fig.1. Furthermore, GA is a derivative-free optimization technique that eliminates the need to the existence of a function derivative to solve the optimization problem. This makes it more attractive for applications that involve nonsmooth or noisy signals. It can be also used to solve problems that are ill defined. GA is considered as a part of evolutionary computing technique which is a rapidly emergent area of artificial intelligence.

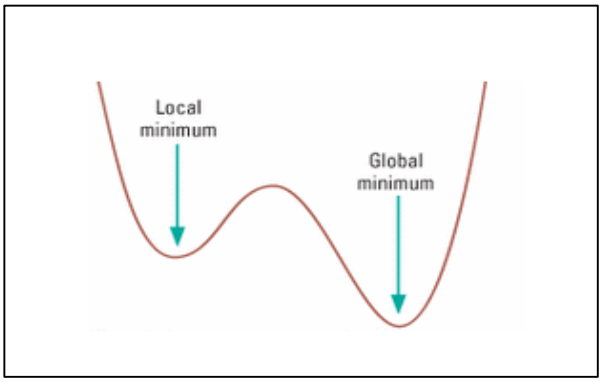


Fig.1 Global and local minimum

Biological Background

Cell is the building unit of all living organisms. In each cell there is a set of **chromosomes** which are strings of **DNA**. Every chromosome consists of **genes** which encode a particular protein. During reproduction, first occurs **crossover**. Genes from parents form in some way the whole new chromosome. However, the new created offspring can be mutated. **Mutation** occurs when the elements of DNA are a bit changed. These changes are mainly caused by errors in copying genes from parents. The **fitness** of an organism is measured by success of the organism in its life. With generations, the good characteristics remain and the bad ones died which represents "The survival of the fittest".

Steps of GA

GA starts with an initial population containing a number of *chromosomes* where each one represents a solution of the problem (*for example a possible pair of optimum PI gains or a value of x that minimises f(x)*). The most common way of representing a chromosome is performed using fixed-length binary bit string which is combinations of 0s and 1s representing a number in binary form (*there are many ways to achieve that but this is outside the scope of this module, we will simply assume that using specific Matlab commands we can represent a chromosome in a binary form*). Each bit of the binary string is called *gene*. For example an individual in the initial population can be represented by the binary string 11010 where as another individual is represented by another binary string 010110. Hence, the initial population is a collection of randomly generated individuals encoded to binary strings and representing different solutions to the optimization problem. Once the initial population has been chosen (*possible pair of PI gains*…) a fitness function is used to evaluate the performance of each chromosome. The fitness function is a numerical value proportional to the nearness of the individual to the solution. So if the task is to minimize a function *f* the individual giving the minimum value of *f* is considered to be the most fit and therefore assigned highest fitness value.

Example

Assume it is required to minimize the function:

$$f(x_1, x_2) = x_1^2 + x_2^2$$

And the number of individuals *n*=4.

Assume that the initial population consist of four individuals:

$x_1 = 10\,;\, x2 = 10 \quad f_1 = 200$
$x_1 = 5\,;\, x2 = 10 \quad f_2 = 125$
$x_1 = 1\,;\, x2 = 1 \quad f_3 = 2$
$x_1 = 2\,;\, x2 = 4 \quad f_4 = 20$

Assuming that these individuals are encoded by the following binary strings: 1110010, 0001010, 1100101, 0111010. The fitness value of each individual could be the inverse of the objective function value *f*, so the lowest the objective function value the highest the fitness value. The fitness off the four individuals can be $Fitness_1$=1/200, $Fitness_2$=1/125, $Fitness_3$=1/2, $Fitness_4$=1/20. It can be seen that the fittest is the third individual with least value of the function to be minimized, followed by the fourth then the third and finally the first.

Generally, and assuming an initial population, GA consists of three main stages: Selection, Crossover and Mutation:

- **Selection stage**:

As it has been seen in the initial population there are many chromosomes that are not fit. These are ignored and only the fittest are used for reproduction, i.e. to produce better chromosomes (*better PI gains…*). In order to keep the size of the initial population fixed some fit chromosomes will be used more than once, for example if the initial population had 10 chromosomes and we decided to use only 8 then 2 selected chromosomes will be used twice (or one three times) to keep the size fixed at 10. This new subset is called "*mating pool*". The selected individuals are called *parents* and the new chromosomes that are created in the next generation are called *children*. As it will be mentioned later a child is created by combing 2 parents. The purpose of this operation is to obtain a mating pool with fittest individuals according to a probabilistic rule that allows the fittest individuals to be mated into the new population. Among many parent selection rules, "Roulette

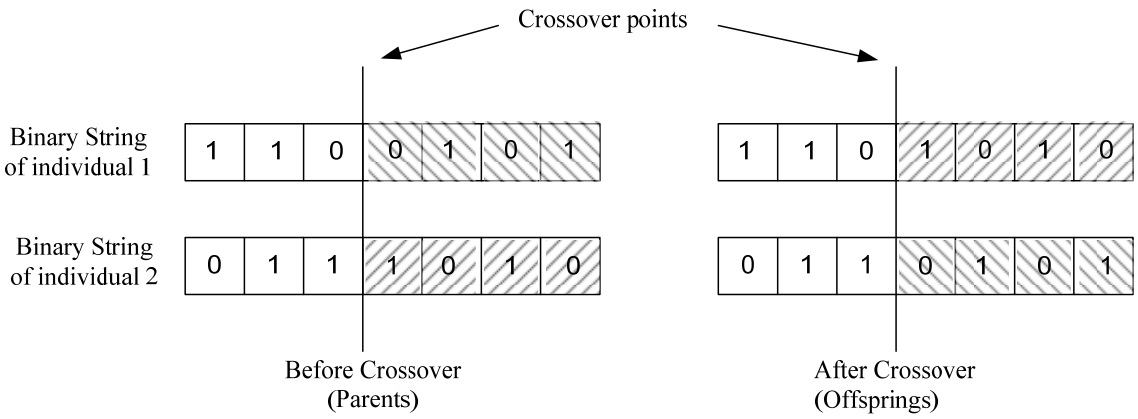
Wheel selection" is the most common technique used in the selection stage. According to this technique, the probability of an individual to be selected for reproduction is directly proportional to its fitness value and can be calculated from the ratio: $\frac{Individual\ Fitness}{Sum\ of\ \ Fitness}$.

In our case sum of fitness =1/200+1/125+1/2+1/20=563/1000, The third individual will have (1/2)/(563/1000)=500/563 selection probability, the fourth 50/563, the second 8/563 and the first individual 5/563.

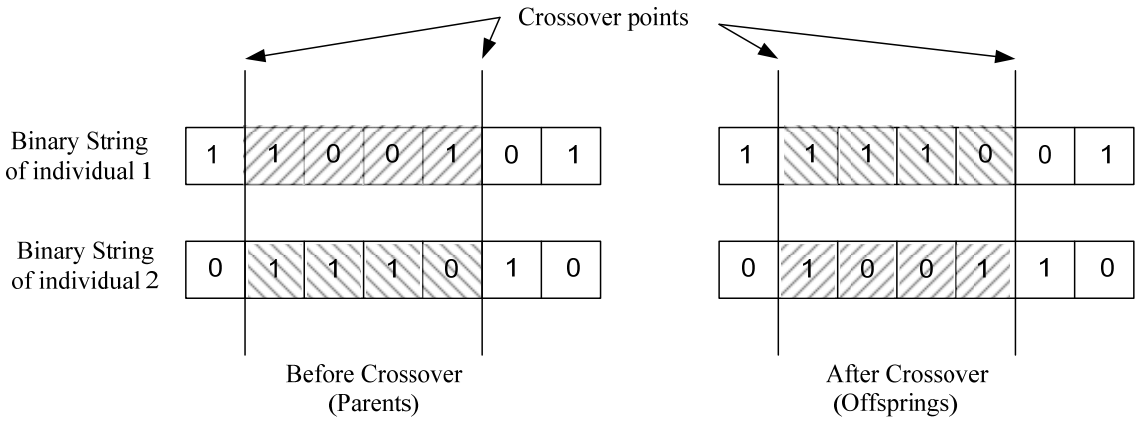
To follow the concept of the “survival of the fittest”, the mating pool may consist of two copies of the third individual (the fittest individual has higher probability to survive), one copy of the fourth individual and a copy of the second individual and no copy of the first individual (the least fit has lower probability to survive and is therefore eliminated). This is according to the "Roulette Wheel selection" where the fittest individuals have more chance to survive. Therefore the mating pool may consist of 1100101 1100101 0111010 0001010 where the first individual (the least fit) may probably “die”. The selected individuals will be used in the next GA step.

- **Crossover stage**:

After selection stage, genetic crossover operation is then applied between parent pairs from the mating pool. Each pair will produce 2 children (hence the size of the population remains fixed). The crossover operation forces the children (the new chromosomes) to have genes (properties/characteristics) from both parents, i.e. the genes of a child are a mixture of the two parents. This crossover operation does not take place always but is performed with a crossover probability (P_c). If this probability is set to 1, then crossover always occurs between parents. The crossover operation can be one-point or double point operation. In the case of one-point crossover, the bits of the two parents are interchanged at a single location while in the double point case, parent bits are swapped between two locations. Therefore a probable crossover for example 1 can be described as shown in Fig.2.



(a) Single point crossover



(b) Double point crossover

Fig.2 Example of Genetic crossover

- **Mutation stage**:

The last operation is the genetic mutation which takes place after the crossover operation. The application of genetic mutation introduces a change in the offspring bit string to produce new chromosomes which may represent a solution of the problem and in the same time to avoid the population falling in a local optimal point. The mutation operation is performed with a mutation probability (P_m) which is usually low to preserve good chromosomes and to mimic real life where the mutation is rarely happened. An example of the mutation process for the last example assuming single point crossover is given below:

Offspring Binary String 1101010 Before mutation
↓
1101000 After mutation

The application of these three basic operations allows the creation of new individuals which may be better than their parents. This algorithm is repeated for many generations and finally stops when reaching individuals providing optimum solution to the problem. The GA architecture is shown in Fig.3.

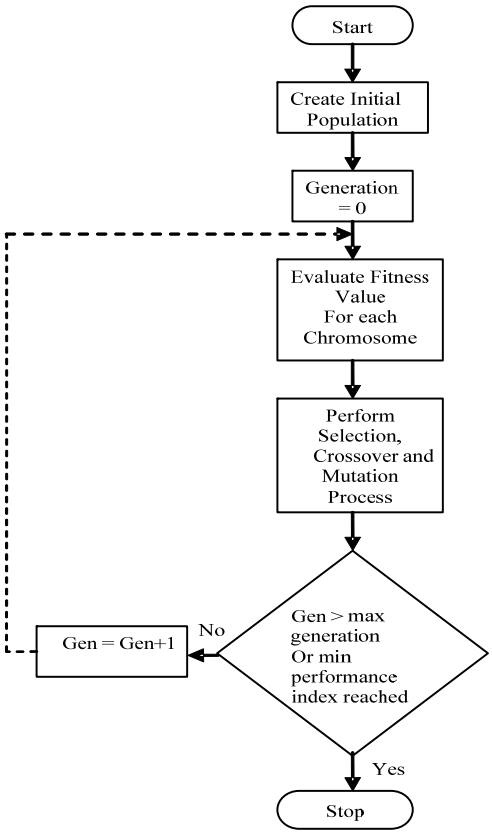


Fig.3 Genetic Algorithm Architecture

Useful definitions

Crossover probability says how often will be crossover performed.

Mutation probability says how often will be parts of chromosome mutated. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to random search.

Chromosome: The Chromosome contains a solution to the problem in form of genes.

Individual: Individual is another way to name the chromosome.

Gene: The Gene is a part of the chromosome. It contains a part of solution.

Population size says how many chromosomes are in population (in one generation).

Fitness function: is the function to be optimized. This represents how close an individual is close from the solution of the problem.

Mutation: Changing a random gene in an individual.

Selection: Selecting individuals for creating the next generation.

Parents and children: To create the next generation, the genetic algorithm selects certain individuals in the current population, called *parents*, and uses them to create individuals in the next generation, called *children*. Typically, the algorithm is more likely to select parents that have better fitness values.

Disadvantages of GA

- Highly computational effort
- They can not find the exact solution but may find a near to optimal solution
- Very slow

GA Programming using Matlab

Solving optimization problems using GA can be performed using *the Genetic Algorithm and Direct Search Toolbox* in Matlab. This toolbox includes routines to solve optimization problems that lie outside the standard optimization toolbox. The algorithm works to minimize the objective function.

Writing M-files for objective function

To use *the Genetic Algorithm and Direct Search Toolbox* the objective function to be minimized is to be written in the form of M-file. This M-file should accept a row vector with length equal to the number of independent variables for the objective function and return a scalar.

Outline of the Algorithm

The following outline summarizes how the genetic algorithm works:

1- The algorithm begins by creating a random initial population.

2- The algorithm then creates a sequence of new populations, or generations.

At each step, the algorithm uses the individuals in the current generation, *parents*, to create the next generation, *children*. To create the new generation, the following steps are performed:

- Scores each member of the current population by computing its fitness value.
- Scales the raw fitness scores to convert them into a more usable range of values.
- Selects parents based on their fitness.
- Produces children from the parents. Children are produced either by making random changes to a single parent, *mutation*, or by combining the vector entries of a pair of parents, crossover.
- Replaces the current population with the children to form the next generation.

3- The algorithm stops when one of the stopping criteria is met. Stopping criteria could be:

•**Generations:** The algorithm stops when the number of generations reaches the value of **Generations**.

•**Time limit:** The algorithm stops after running for an amount of time in seconds equal to **Time limit**.

•**Fitness limit:** The algorithm stops when the value of the fitness function for the best point in the current population is less than or equal to **Fitness limit**.

•**Stall generations:** The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length **Stall generations**.

•**Stall time limit:** The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to **Stall time limit**.

Example 2:

Suppose it is required to optimize the following function:

$$f(x_1, x_2) = 3x_1^2 - 4x_1x_2 + 2x_2^2 - 7x_1 + 10x_2$$

The first step is to write an M-file describing this objective function. This must accept a row vector of $\boldsymbol{x}$ of length 2, representing the two variables x_1 and x_2 and return a scalar which represents the value of the function f at $\boldsymbol{x}$. Steps to write the M-file are:

1- From *File* menu select *New*

2- Select M-File from the drop menu. A new M-file is then opened in the editor.

3- Start to write the code for the objective function in the M-file:

```
function z = fun(x)
z = 3*x(1)^2 - 4*x(1)*x(2) + 2*x(2)^2  - 7*x(1) +10*x(2);
```

4- Save your file in a directory on the Matlab path with the name *fun*

5- To check that your M-file returns the right value, from the command window type:

```
fun( [2 2] )
```

This should return a value of 10 which is the value of the function f at $\boldsymbol{x}$ = [2 2]

Note that *the Genetic Algorithm and Direct Search Toolbox* **minimizes** the objective function so if it is required to **maximize** a function you should convert it to a minimization problem.

Example 3:

If it is required to maximize the function:

$$f(x_1, x_2) = x_1^2 - 3x_1x_2 + 6x_2^2$$

The problem should be converted to a minimization problem by writing the M-file to compute:

$$-f(x_1, x_2) = -x_1^2 + 3x_1x_2 - 6x_2^2$$

This function will be then minimized.

Using GA toolbox

- **From the command line (not studied here):**

To use GA Toolbox from the command line, the function ga can be used:

[x fval] = ga (@fitnessfunction, Nvars, options)

Where:

@fitnessfunction: is used to handle to the fitness function

Nvars: is the number of independent variables for the optimization problem

Options: This contains the options for the GA. Default options will be used if this is not written in the argument.

The results will be given as:

fval: Final value of the fitness function

X: the point at which the final value of the fitness function is obtained

- **Using GA Tool**

Another way to work with GA without working at the command line is to use GA tool based on Graphical User Interface GUI.

Typing *gatool* at the command line will open the tool and the following figure, Fig.4, will appear:

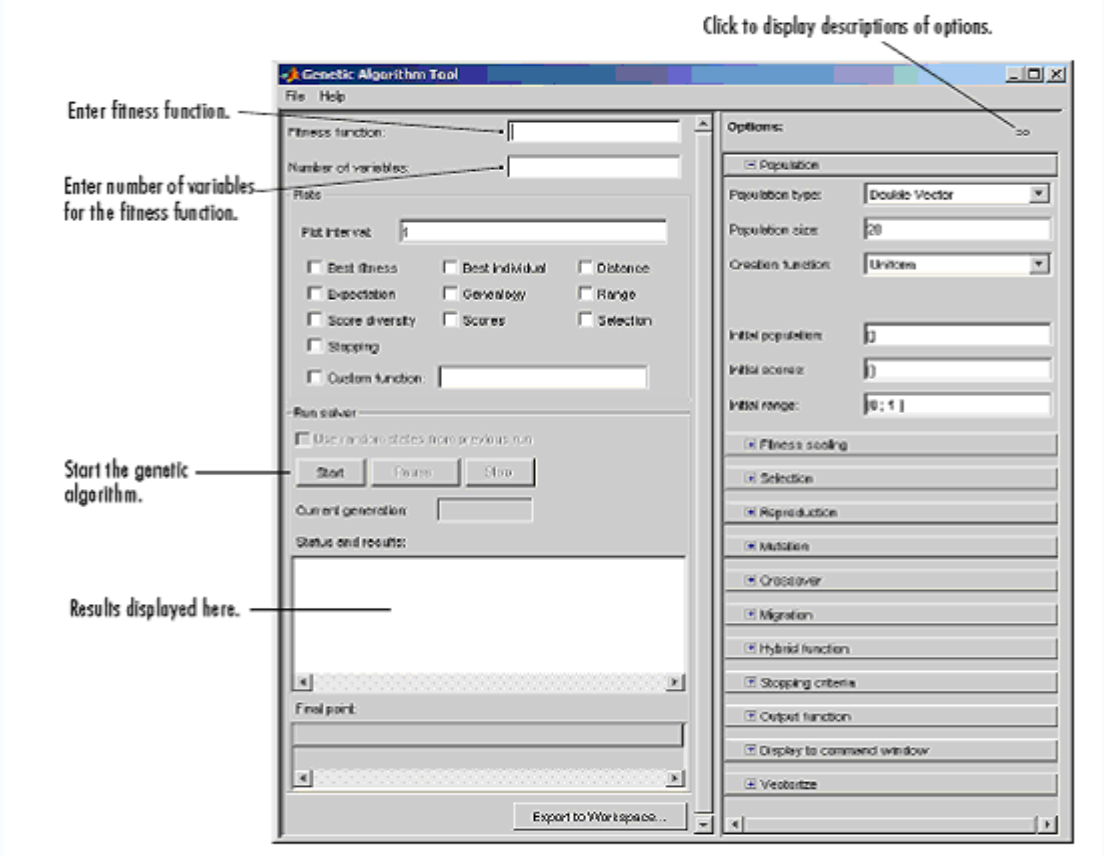


Fig.4 GA Tool

The following information should be entered:

Fitness function: This represents the objective function to be minimized. This will be entered on the form @fitnessfunction, where fitnessfunction is defined in a separate M-file as described earlier.

Number of variables: This represents the length of the vector ***x*** as described in the m-file. This is the number of independent variables in the optimization problem. In biological words, this represents the number of Genes per Chromosome.

Pressing **Start** button will run the algorithm and the results of the optimization will be displayed in **Status and Results** pan.

The options for the GA can be viewed and changed using the **Options** pane.

Example 4:

It is required to use GA to minimize the following function:

$$f(x_1, x_2) = 3x_1^2 + 6x_2^2$$

It is well known that the minimum of the function *f* is zero and occurs at ***x*** = [0 0]. To use GA for the optimization problem we will follow the next steps:

Step 1: Open a new m-file and enter your objective function:

```
function z = fun1(x)
z = 3*x(1)^2 +6*x(2);
```

Step 2: Save the M-file in a directory on the Matlab path with the name *fun1*

Step 3: From the Matlab command window type *gatool* to open the GA tool.

Step 4: in the Fitness function tab enter @your m-file name to define your fitness function to be minimized by GA

Step 5: in the number of variables tab enter 2

Step 6: From the **plot** pan choose the plots you want to see for the optimization problem.

Step 7: Browse the GA options to be used for the optimization.

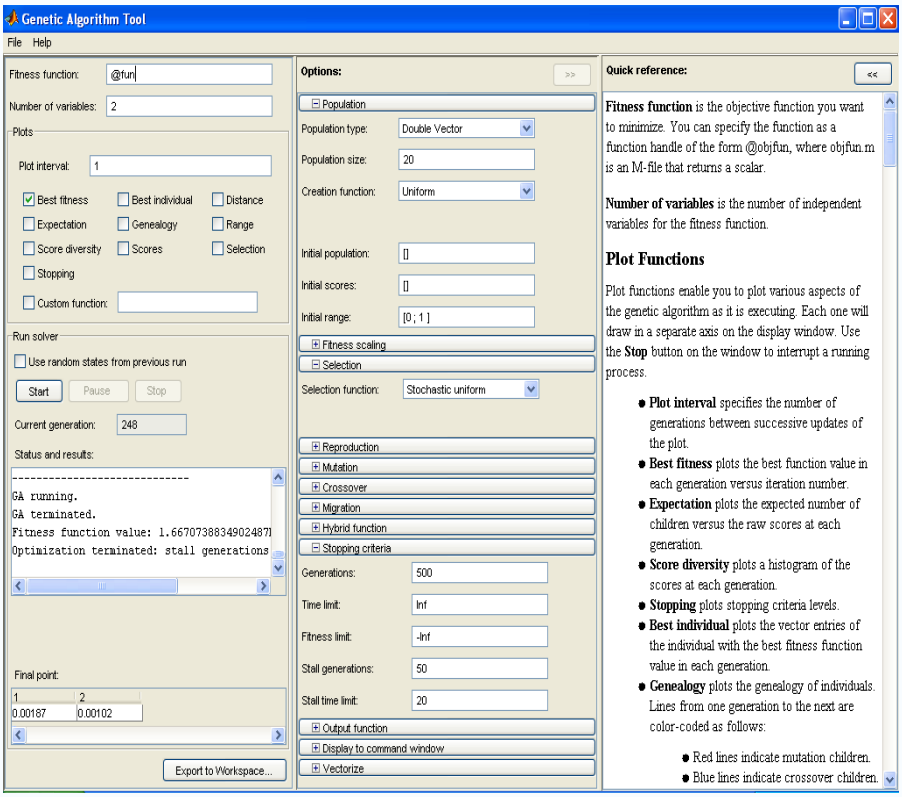
Step 8: Press the Start button to start the optimization

You can try it several times with different GA options.

Note: If you restart the algorithm again you may find different results. This is because that depends how far the initial population is from the solution and the stopping criteria.

In this example after running the GA toolbox with the default settings the solution was obtained at the point (0.00187, 0.00102) with a function value of 1.667e-5 as shown in Fig.5. Both mean and best fitness decay with no of generations until the algorithm stops.

Default settings: Selection: stochastic uniform, Crossover function: scattered, stopping criteria generations=500.



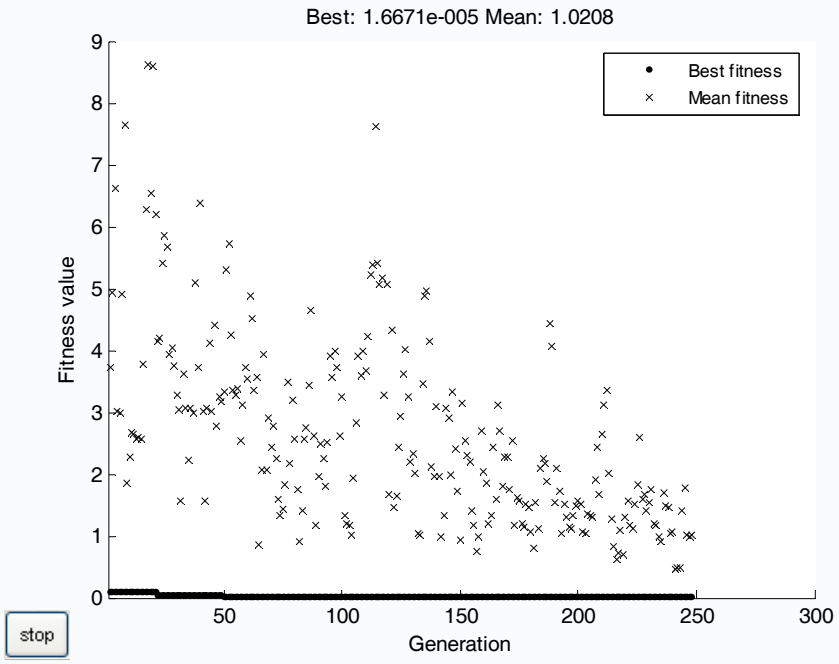
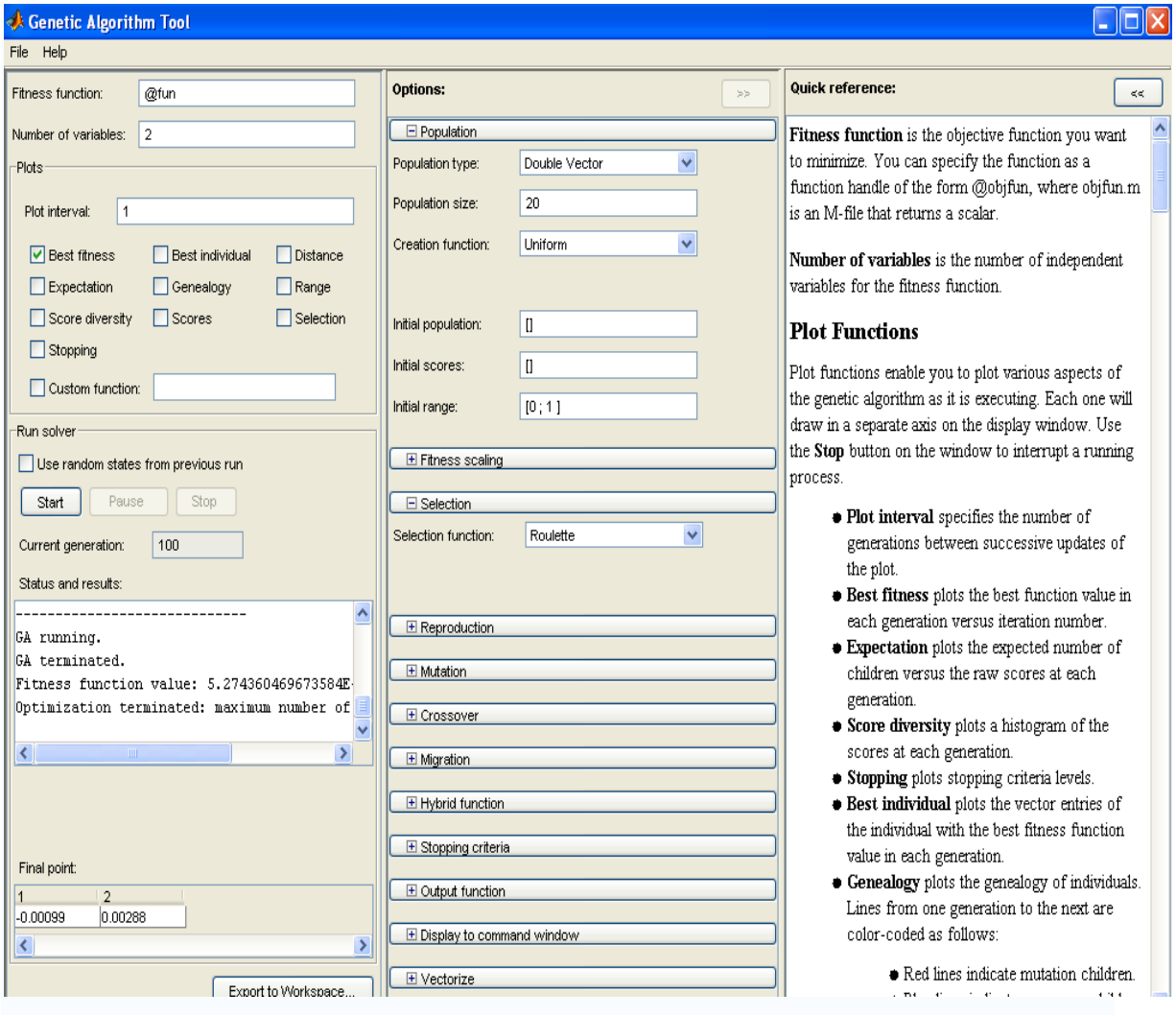


Fig. 5 optimization outputs for default settings

By changing the selection function to Roulette, stopping criteria generations=100: The solution is (-0.00099, 0.00288) giving a function value of 5.274e-5 as shown in Fig.6.


Genetic Algorithm Tool
File Help
Fitness function: @fun
Number of variables: 2
Plots
Plot interval: 1
Best fitness
Best individual
Distance
Expectation
Genealogy
Range
Score diversity
Scores
Selection
Stopping
Custom function:
Run solver
Use random states from previous run
Start
Pause
Stop
Current generation: 100
Status and results:
GA running.
GA terminated.
Fitness function value: 5.274360469673584E
Optimization terminated: maximum number of
Final point:
1
2
-0.00099
0.00288
Export to Workspace...
Options:
Population
Population type: Double Vector
Population size: 20
Creation function: Uniform
Initial population: []
Initial scores: []
Initial range: [0 ; 1]
Fitness scaling
Selection
Selection function: Roulette
Reproduction
Mutation
Crossover
Migration
Hybrid function
Stopping criteria
Output function
Display to command window
Vectorize
Quick reference:
Fitness function is the objective function you want to minimize. You can specify the function as a function handle of the form @objfun, where objfun.m is an M-file that returns a scalar.
Number of variables is the number of independent variables for the fitness function.
Plot Functions
Plot functions enable you to plot various aspects of the genetic algorithm as it is executing. Each one will draw in a separate axis on the display window. Use the Stop button on the window to interrupt a running process.
Plot interval specifies the number of generations between successive updates of the plot.
Best fitness plots the best function value in each generation versus iteration number.
Expectation plots the expected number of children versus the raw scores at each generation.
Score diversity plots a histogram of the scores at each generation.
Stopping plots stopping criteria levels.
Best individual plots the vector entries of the individual with the best fitness function value in each generation.
Genealogy plots the genealogy of individuals. Lines from one generation to the next are color-coded as follows:
Red lines indicate mutation children.

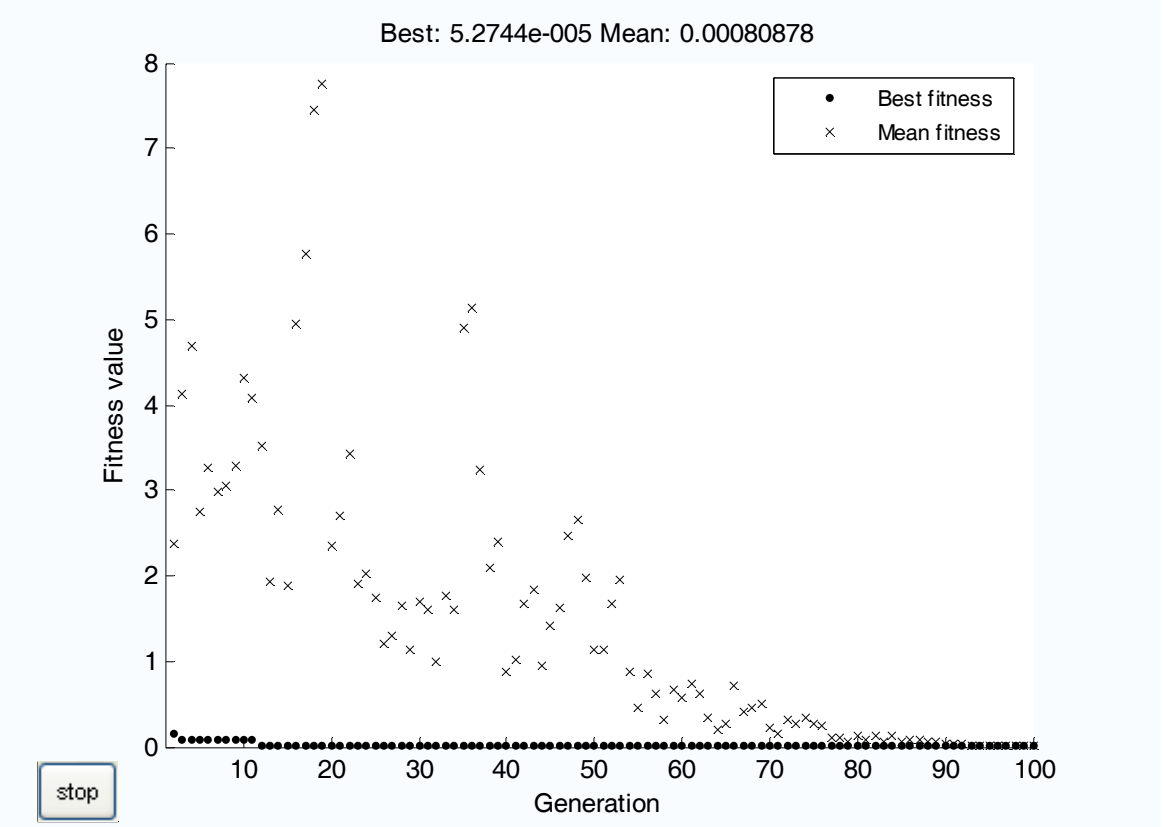


Fig.6 optimization output for new settings

GA application in control

Due to its effectiveness in searching nonlinear, multi- dimensional search spaces, GA can be applied in various applications in control such as tuning the gains of the PI controller.

During the search process, GA looks for the optimal setting of the PI speed controller gains which minimize a fitness function. This function is considered as the evolution criteria for the GA and is usually chosen based on an error function. Each chromosome represents a solution of the problem and hence it consists of two genes: the first one is the K_p value and the other one is the K_i value: Chromosome vector = [K_p K_i].

GA starts by generating an initial population containing a number of chromosomes then the fitness value of each chromosome in the initial population is calculated. The three main parts of GA: Selection, Crossover and Mutation take place and then a new

generation is produced. This procedure continues for a number of generations and then a convergence to the optimal solution represented by a given chromosome is reached.

References

[1] http://cs.felk.cvut.cz/~xobitko/ga/ accessed on 28/11/07.

[2] Matlab Genetic Algorithm and Direct Search Toolbox user guide retrieved from: http://www.mathworks.com/access/helpdesk_r13/help/pdf_doc/gads/gads_tb.pdf

[3] K.F. Man et al (1997) *Genetic algorithms for control and signal processing.* Berlin: Springer, c1997.

[4] David E. Goldberg (1989) *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Pub. Co., c1989.