New Genetic Algorithms for Exploring Design Parameters for MEMS

By

Ehimwenma Valerie Obaseki

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES OF
THE UNIVERSITY OF MANITOBA
IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

This work is dedicated to my mum and dad (Dr & Mrs Chris Obaseki) for their endless love and support and for starting me on this journey and supporting me throughout the journey.

ABSTRACT

The design of complex MEMS systems can be time consuming when many variables and geometric parameters are required. Proper exploration of the design space, which is required for finding good solutions have been a major challenge. This thesis applies novel genetic algorithm methods (AVGM, RBAM and MORBAM) to the design of micro-electromechanical systems (MEMS). The main objectives of the algorithms, which are introduced in this thesis is better identification of high performance region for MEMS geometry design and faster computational time. The Average-Mixture (AVGM) and the Random Based Average Mixture (RBAM) Genetic Algorithm methods are applied to the single objective problems while the Multi-Objective Random Based Method is applied to the multi-objective problems. The main advantages of the methods over the traditional genetic algorithm methods are their ability to identify high performance regions while maintaining diversity by exploring the search space efficiently. These algorithms provide many good results, which are diverse in nature.

ACKNOWLEDGEMENT

I will like to thank God Almighty for seeing me through my program.

My sincere appreciation goes to my advisor Dr. Cyrus Shafai for his guidance, mentorship and endless support throughout my graduate program.

I will also like to thank Dr. Ken Ferens and Dr. Dean McNeil for their assistance and the knowledge they instilled in me while offering their courses in class.

I also appreciate my family for their support and encouragement throughout my program. My mum and dad (Dr and Mrs Chris Obaseki) sponsored and encouraged me through the process from the beginning to the end and I am very grateful to them.

I appreciate my husband (Emmanuel Briggs) whom I got married to while completing my program for his love and support.

I will also like to thank Amy Dario for all her administrative assistance throughout the program. She's been nice and really supportive from the moment I started my program to when I ended the program.

My appreciation also goes to Mount-First Ng, Guy Jonatschick and other Faculty of Engineering members who in one way or the other made it possible for me to complete my program by maintaining the computer systems and keeping the servers working.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 MEMS SYSTEM DESIGN AND GENETIC ALGORITHM

Micro-electromechanical systems (MEMS) refer to systems with at least one dimension being small in the order of micrometers, and which combine electrical and mechanical properties. They are fabricated with the use of integrated circuit batch- processing technologies [2]. These systems are often more reliable than macro systems, are light-weighted with low cost, can produce high frequency operation. They are applied in the automotive industry, for example in the use accelerometer for airbag system. MEMS applications also extend to the industrial, military and consumer market. MEMS are used in life science and technology for tissue engineering, smart bandages, implantable devices, drugs delivery etc. They are also used in personal care and technology for sensor (used for safety and health monitoring), built-in cooling system. They are used in electronics for high capacity batteries, fuels cells, printers etc.

MEMS are made up of four different components; Microelectronics, Microactuators, Microstructures and Microsensors. They are applicable in various industrial, and medical fields and there is need to improve their design by optimizing the modeling techniques and creating strategies to aid in their design (some analysis needs to be done before the fabrication process). Searching for the right design parameters can be a complicated process without the knowledge of how to start. Traditional methods of designing complex MEMS have been improved with the use of computational modeling simulation software. However, this presents challenges because a lot of experiments and simulations for various design

options still have to be performed with software.

Optimization algorithms that are robust need to be introduced for better exploration of the search space. MEMS designs usually involve systems with numerous design variables and conflicting objectives. For example, consider a simple MEMS cantilever beam in Fig. 1.1, obtaining a high resonance frequency would mean lower deflection and vice versa. These objectives are dependent on the length, width, thickness and the material property of the cantilever beam. These numerous design variables can be quite time consuming to find and \would require a robust search algorithm for the exploration of the search space. Optimizing the design the comb actuator of Fig. 1.2 poses a greater challenges, and again there are conflicting objectives that need to be managed.

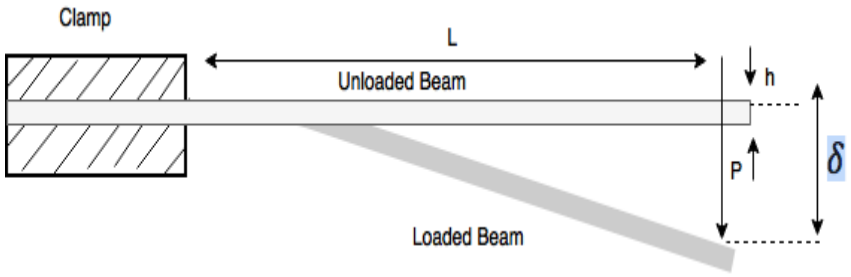


Figure 1.1 MEMS Cantilever Beam

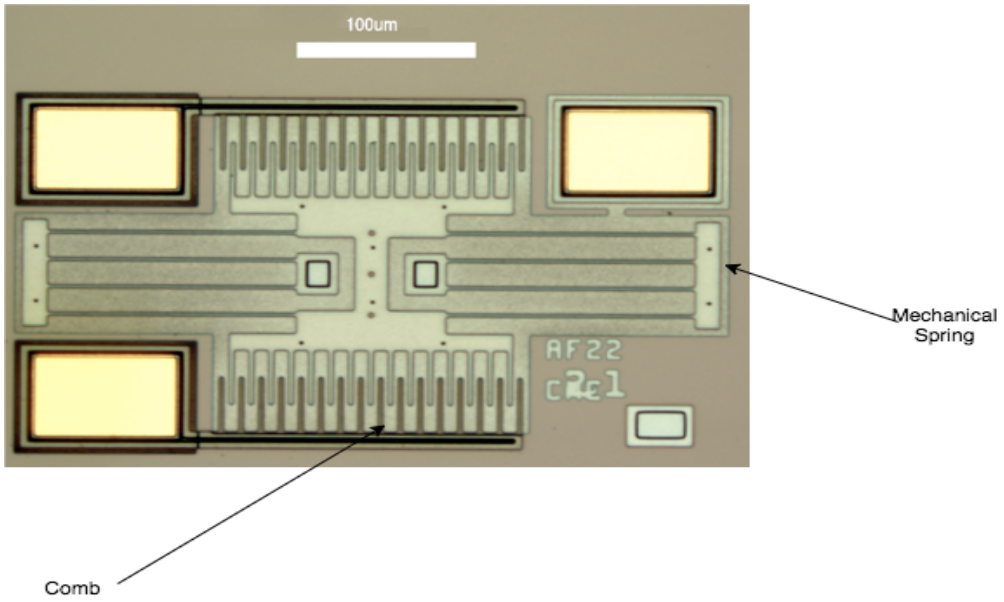


Figure 1.2 MEMS Comb Actuator

There are different methods of searching for suitable and optimized solutions that can be used for MEMS design. Some examples are brute force (exhaustive search), hill climbing, simulated annealing, and genetic algorithm.

The exhaustive search involves exploring the entire search space. This could require a lot of computational time.

Hill Climbing is quite different from simulated annealing and genetic algorithm in the way it accepts solutions, it has a way of accepting better solutions. It starts with an initial solution, generates some neighboring solutions, selects the best and continues until better neighboring solutions are founds. It searches by walking its way up until it can't find any neighboring solution better than the present solution, then it stops. One major drawback to this approach is that it might end up in the local optimum. When the algorithm stops, there's a possibility that it can get to a low point before climbing back up and so accepting weaker solutions to attain the global optimum is sometimes needed. This makes the algorithm quite a greedy search for always going with the better solution.

Simulated annealing idea borrows its name from the industrial process of annealing

where the material is heated above critical point to soften it and it is gradually cooled in order to erase defects in its crystalline structure producing more stable and regular lattice arrangement of atoms [3]. Similar to genetic algorithm, the simulated annealing process also involves the use of a fitness function. The simulated annealing process only involves one candidate solution as opposed to that of GAs, which consists of a population of candidate solutions. Although hill climbing has a means of generating good solutions, simulated annealing has a better way of avoiding getting stuck in the local optimum. Simulated annealing escapes this local optimum by the way it accepts its solutions by accepting weaker solutions occasionally. If the neighboring solution is better than the current solution, it accepts it unconditionally. If the solution is worse than the current solution it considers certain factors like how bad the solution is and how high the temperature (annealing schedule) is. The algorithm is likely to accept worse solutions with higher temperature.

A genetic algorithm (GA) is an evolutionary algorithm that can be used for optimizing engineering design parameters. GAs are optimization techniques that apply natural selection and are from the family of evolutional algorithms. Genetic algorithms basically use "survival of the fittest", in which the fittest individuals are likely to make it to the next generation for crossover. Genetic algorithms (GAs) make use of exploration and exploitation by searching the search space and exploiting the good solutions.

Genetic algorithms don't necessarily find the best solution (global optimum) but they deal with solutions that are good enough. GAs possess several traits, which makes them distinct from other Evolutionary Algorithms; they make use of the crossover and mutation operators that help maintain certain amount of diversity in the population and prevents it from being stuck in the local optimum. This diversity helps the algorithm achieve a global search faster because it gives the GA the opportunity to explore the space faster for

possible solutions. At the beginning high diversity is maintained but towards the end, the diversity should be kept minimum to avoid losing the good accumulated results.

1.2 GENETIC ALGORITHM

There are two categories of genetic algorithms solution problems namely;

- Single-objective genetic algorithm
- Multi-objective genetic algorithm

A single objective problem would be designing a physical system that has only one objective function. The task of finding a single optimum solution is called *single- objective optimization* and the problem is called *single-objective problem [1]*. The single-objective problem is a useful tool, which provides insights to the nature of the problem [*1*].

A multi-objective problem involves one or more competing or conflicting problems, the task of finding one or more optimum solutions is known as *multi-objective optimization* and the problem is called *multi-objective problem [1]*.

1.2 STRENGTHS OF GENETIC ALGORITHM

There are several strengths a GA has namely;

- The most noticeable advantage of genetic algorithm is its parallelism. Most other algorithms can find one solution at a time and if something happens the work is abandoned and restarted. In a genetic algorithm, there's an option to explore multiple spaces at once due to the number of individuals or offspring. Weaker regions can be ignored to aid the exploitation of promising regions. These factors help the algorithm have a better chance at reaching the global optimum.
- As a result of this parallelism, many solutions can be evaluated at once. It is particularly useful in scenarios where there are a huge number of potential solutions. As a result of this, optimal or very good results can be found in a short

period of time.

- Another strength of genetic algorithm is its potential ability to escape the local search space. In a scenario where there is a vast solution space (similar solutions), it can be easy to settle in the local search space. Genetic algorithm operators like population size, crossover and mutation are significant in aiding this escape and leading the algorithm to the global search.
- The genetic algorithm has the ability to manipulate multiple parameters simultaneously and this helps in finding multiple solutions especially in cases where there are multiple objectives. If there is a chance that improving one objective doesn't necessarily decrease the other objective, the algorithm can be pretty good at discovering it.

1.4 FACTORS TO CONSIDER WITH GENETIC ALGORITHM

There have been some problems with traditional genetic algorithm methods and previous research works have been done to make the algorithm more efficient. Some of the limitations observed over the years are as follows:

- The representation of the problem.
- Construction of the fitness function so that the fitness function can be found.
- Deceptive fitness function providing misleading information about the global search space.
- The computational time.
- Proper optimization of the crossover and mutation technique. The crossover and mutation techniques can be found that are capable of exploiting the solution space in an efficient manner.
- Maintaining diversity, which helps prevent the loss of good solutions.

- Starting with a suitable population size to ensure proper exploration of the search space.
- Preventing early convergence as a result of loss of diversity in the population.

1.5 THESIS OBJECTIVE

The main objective of this thesis is to study MEMS design with the application of new genetic algorithm techniques. In the process of applying a genetic algorithm to the specific MEMS design problem, methods for applying crossover, population number, and mutation over any number of genes would be evaluated. Three novel methods of genetic algorithm are being proposed here.

1.5.1 ***Average Mixture (AVGM)*** is applied to a problem with a single objective. It was the first method used for the design of the simple cantilever beam. The advantage of using this method is the low computational time required when applied to a single objective problem like the cantilever. However, one drawback to this method is its lack of flexibility when applied to problems with many variables (genes). It was therefore necessary to introduce a better method with more flexibility for problems with multiple variables.

1.5.2 ***Random-based Average Mixture (RBAM)*** is also used for a problem with a single-objective. This was the second method tested for the design of MEMS as a result of the lack of flexibility with the AVGM method. The RBAM method can be used for problems with multiple genes. It was tested on problems with different genes such as the cantilever beam and the comb drive actuator. This method is faster at generating results than the AVGM and can be used for problems with many variables without the need to readjust important factors for different problems.

1.5.3 ***Multi-Objective Random-based Average Mixture (MORBAM)*** is used for problems with two or more objectives. This method applies RBAM to different objectives and selects the elite from each to make up a population for uniform crossover. The method used RBAM and not AVGM for each objective because RBAM was a more flexible method.

The notable differences between these algorithms are seen in chapter six and seven when the genetic algorithm methods are analyzed, evaluated and compared to see their efficiency in maintaining diversity, speed and so producing suitable design results while ensuring the designs can be manufactured. They apply the basic GA techniques with more intelligent ways to crossover and mutate the genetic algorithm. The reasons for the proposed methods are as follows;

1. Quick combination of materials, dimensions and material properties.
2. Using this analysis to determine a novel genetic algorithm method required for devices from small to large feature sizes.
3. Maintaining diversity to facilitate MEMS feature options to select from. A MEMS device can have multiple variable combinations that produce the same goal (objective). This makes it possible to have many design options.
4. A genetic algorithm technique that can be applied to any MEMS problem with many variables. Two basic methods with details on when, and when not, to use them based on basic requirements such as the number of starting individuals in a population, number of genes, crossover and mutation technique.
5. An evaluation method for multi-objective genetic algorithm and finding a balance or middle point.
6. Generating a code that is reusable and easy to modify for other types of design

problems.

1.6 THESIS OUTLINE

This work is made up of eight chapters. Chapter two provides the basic background for genetic algorithm by providing specific information on methods that have been used in the past and explaining how the basic (more general) genetic algorithm works in a clearer manner. Chapter three introduces the new methods proposed in this thesis, it also explains why these new methods were proposed and explains how the proposed algorithms function. Chapter four introduces the simple MEMS cantilever beam and explains how the proposed Genetic Algorithm methods can be applied to the design of the cantilever beam by assigning values to the variables created in the proposed algorithms introduced in chapter three. Chapter five describes the design of a complete electrostatic comb drive actuator; it also describes the flexural or spring design and how it can be applied to a basic comb drive and assigns values to the genetic algorithm variables introduced in chapter three. Chapter six evaluates how the single-objective (AVGM, RBAM) genetic algorithm methods perform, by evaluating their results based on different terms and it also states noticeable differences seen in the two genetic algorithm single objective problems. Chapter seven explains why the proposed multi-objective genetic algorithm (MORBAM) is necessary by using results obtained from the application of RBAM and AVGM GAs to the cantilever and also evaluates the proposed MORBAM GA. The work is concluded in chapter eight with details on how to proceed with further work in the future.

CHAPTER 2

GENETIC ALGORITHM BACKGROUND

2.1 THE ALGORITHM

Genetic algorithms deal with populations of individuals. Encoding these individuals depends on the type of problem being solved. There are different ways to encode these individuals; binary encoding, permutation encoding, value encoding and tree encoding.

Binary encoding is made up of strings of 0s and 1s. Permutation encoding is typically used for ordering problems such as the traveling salesman problem. Value encoding is used where complicated values such as real numbers are used. In tree encoding every value is a tree of some objects such as functions or commands in programming language [42]. This work uses value encoding for real values introduced in this thesis. There are different ways of doing a GA and the flowchart in fig 2.1 shows a more general way. See table 2.1 for descriptions of the terminologies used in the general genetic algorithm.

2.1.1 THE STEPS

The steps listed below (step 1 to step 5 are shown in the flowchart in Fig 2.1).

Step 1) Generate a random population of individuals (initialize population).

Step 2) Evaluate their fitness value (the fitness value or function is the function the algorithm is trying to optimize).

Step 3) Create a new population by selection based on fitness value, crossover, and mutation.

a) Select parents by fitness value.

b) Create children based on crossover and mutation.

Step 4) Replace the old population.

Step 5) Repeat from step two if still not satisfied with the results.

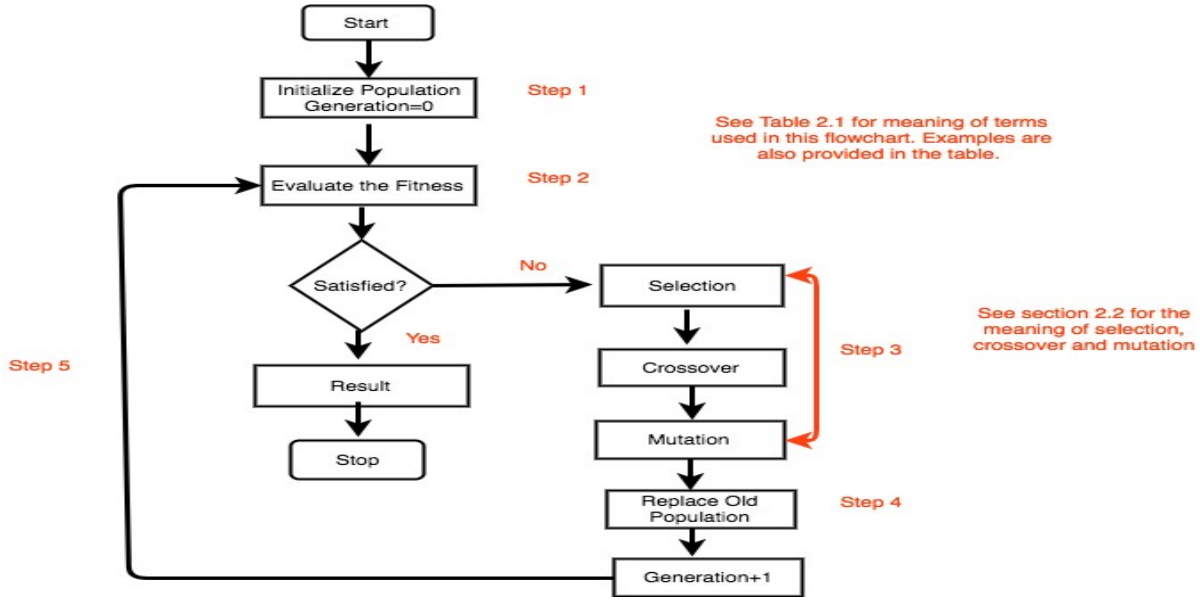


Figure 2.1 The Genetic Algorithm

Table 2.1: GA Terms and their Meaning

TERM	MEANING
Population	Group of individuals. Mathematically represented as an array of individuals. For example, given a population size of four and three variables in the fitness function. The population is represented as a 4 by 3 matrix. See Figure 2.2
Individual	Group of genes known as a candidate solution.
Genes	Genetic property of an individual, which are known as the number of variables in the fitness function.
Parent	Individual before crossover
Child	Individual after crossover
Generation	A series of computation is performed on the current population to produce a new population; each successive population is a new generation.
Fitness	The function to be optimized
Fitness Value	Value of the fitness function for an individual

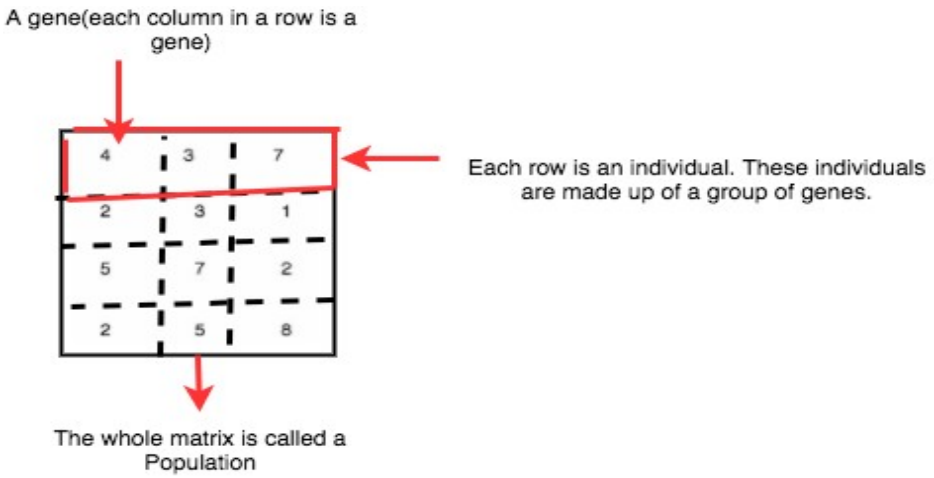


Figure 2.2 Genetic Algorithm Population

2.2 GA BASIC OPERATIONS

Genetic operators are responsible for changing the genetic information of an individual. There are two types of operators in genetic algorithms [43]:

1. Genetic operators: crossover and mutation
2. Evolution operation: selection

2.2.1 SELECTION

A. Roulette Wheel Selection

The roulette wheel, which is known as fitness proportionate selection, is one of the earliest methods of selection in genetic algorithm. In this case, the parents are selected according to their fitness value. The individuals with higher fitness values have a higher chance of being selected, which implies that the probability of selection is dependent on the fitness value. The fittest individual occupies the largest part of the roulette wheel and the weakest individual occupies the smallest part of the wheel.

B. Elitism

In this selection strategy, some of the candidate solutions are kept unchanged i.e. it involves keeping the best candidates and taking them over to the next generation. This method helps to prevent losing really good solutions. The method has been known to improve the performance of a genetic algorithm by ensuring that the quality of the solution does not decrease from one generation to the other.

C. Tournament Selection

This selection strategy is one of the most popular. It involves taking a sub-part of the population and comparing their fitness values, thereby eliminating the loser or the ones with the lowest fitness values. The candidate with the best fitness value is known as the winner of the tournament.

2.2.2 CROSSOVER AND MUTATION

In genetic algorithm, crossover and mutation are basic operators and how these processes are performed can affect the speed and efficiency of the results. There are different ways to crossover and mutate in genetic algorithm.

A. Crossover

The crossover technique also known as recombination is one of the most important genetic algorithm operation, and it is performed right after selection. It involves the combination or exchange of genetic properties from different individuals in the population. The way the genetic algorithm is encoded can also determine the type of crossover technique used.

1. Single Point Crossover

This is the simplest crossover technique. It involves randomly choosing a crossover point, which divides an individual into two distinct parts. For example, given two parents (individual A and B), the first part of individual A could be merged with the second part of individual B, and the first part of individual B would be merged with the second part of individual A, forming two children (see figure 2.3).

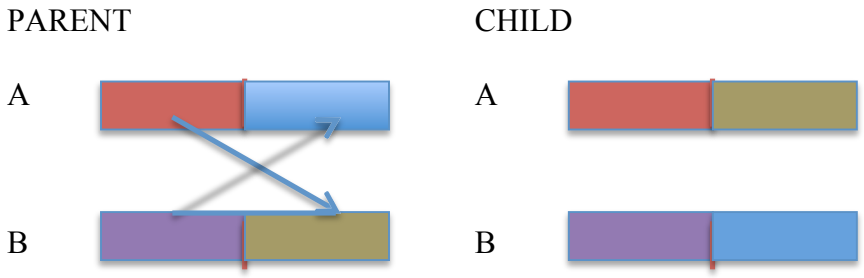


Figure 2.3 Single Point Crossover

2. Two point crossover

In this crossover technique, two points are randomly selected from an individual (individual A) and a crossover is performed with the second individual (individual B) as illustrated in Figure 2.4. From individual A, the first part to the first crossover point is copied. From individual B we copy from the first to the second crossover point. The rest is copied from individual A to form the new individual. For the second new individual the same is done, but with the genetic materials taken from the opposite parent.

=

A

B

Figure 2.4 Two Point Crossover

3. Uniform Crossover

In this crossover technique, some parts of Parent A are randomly copied to Parent B, and some parts of Parent B are randomly copied to Parent A to form two children.

PARENT CHILD

A

B

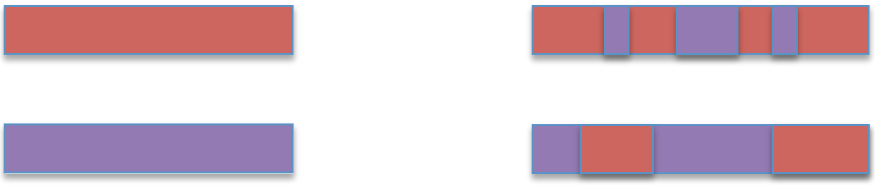


Figure 2.5 Uniform Crossover

4. Arithmetic Crossover

This crossover method involves some arithmetic operations. In the case of binary encoding bitwise AND would be used.

B. Mutation

This is another basic operation in GA and it helps to randomly change the genetic property of an individual because at some point the genetic algorithm starts to converge and mutation is a method used to further randomize genetic properties, and so preventing the solution from converging too soon. Examples of mutation techniques are shown below.

1. Bit inversion

This is used in binary encoding and selected bits are inverted in this instance.

11**1**01001 => 11**0**01001

2. Order Changing

This method is used in permutation encoding, and involves the selection and exchange of some genes.

2 4 **5** 6 7 **8** 9 => 2 4 **8** 6 7 **5** 9

3. Adding or subtracting small values

This method is used when the genetic algorithm is value encoded. A small value is added to or subtracted from the selected value.

10.30 12.45 **2.34** 5.56 => **10.40** 12.45 **2.23** 5.56

2.4 MULTI-OBJECTIVE GENETIC ALGORITHM

The selection methods discussed earlier are based on a genetic algorithm with a single objective. In the real world, most engineering and MEMS problems have multiple objectives and evaluating all objectives based on a single objective could compromise the results. This could result in one objective dominating the others, because two or more objectives might

be in conflict with each other. For example, consider a simple cantilever beam, with a single objective like attaining a displacement of 10 micrometers.

Evolving the algorithm towards the displacement wouldn't be much of a problem, because it has just one goal, and as a result the genes can just evolve towards satisfying the genetic algorithm requirement. However, what if for this simple cantilever beam we are required to have two objectives, like displacement and resonance frequency? This is a conflicting problem because they are two opposite requirements. To achieve a high displacement, the structure would need to be less stiff. While to achieve a high resonance frequency the structure would need to be stiffer. This can easily be seen from the equations 2.1 and 2.2:

$$Displacement = \frac{Force}{k} \tag{2.1}$$

$$\omega = \sqrt{\frac{k}{m}} \tag{2.2}$$

where, k is the spring constant, and m is the mass of the structure.

Several methods have been proposed for multi-objective genetic algorithm problems. Including the weighted-sum method that assigns weight to each objective, and the Pereto method in which no objective dominates the other and creates a sort of balance by providing options to select from. A generic GA can easily be modified to find a set of multiple non-dominated results [1]. The crossover operation can be modified with multiple objectives to create new non-dominated results in unexplored parts, and most GAs do not require the user to prioritize, scale, or weigh objectives [1]. Different multiple objective GA have been explored such as the Niched Pereto Genetic Algorithm, Random Weighted Genetic Algorithm (RWGA), Non-dominated Sorting Genetic Algorithm (NSGA) etc.

CHAPTER 3

AVGM, RBAM AND MORBAM GENETIC ALGORITHM

After carefully reviewing and testing traditional genetic algorithm methods, I applied them to the design of MEMS and I wasn't satisfied with the performance of these algorithms in optimizing the MEMS design. It was therefore relevant to introduce new GA techniques (AVGM, RBAM and MORBAM) for better performance and reusability. The AVGM and RBAM are applied to the single objective problems while the MORBAM is applied to the multi-objective problems.

The significant factor in the AVGM method of genetic algorithm is the term "Average" which signifies the use of mean values for crossover and mutation. The algorithm creates children by recombining the generated population with the mean values of randomly picked individuals. The mutation is also a significant part of the AVGM method; randomly choosing individuals from the mean values to add to the children produced to make a population and then revaluating the fitness values to pick the best individuals that would make it to the next generation optimizes the mutation and crossover method.

The significant thing about the RBAM is the randomness in crossover and mutation with a bit of mean value added to it. The crossover is performed in a random way and the mutation is also based on a random method. This makes the RBAM method very efficient with little or no need for the mutation; it generates result in a timely manner and maintains diversity.

The MORBAM GA applies the RBAM genetic algorithm method to optimize the multiple objectives. The RBAM is used because of its robustness.

This chapter will first provide a brief overview of each genetic algorithm method introduced, then these methods will be explained in a clearer manner and the complete

algorithms will be introduced. In evaluating these new genetic algorithm methods, several factors have been taken into consideration. These factors include:

1. Speed or computational time of the algorithm
2. Efficiency of the algorithm (exploration and exploitation)
3. Diversity created by the algorithm
4. Effect of initial starting population and selected subset of population
5. Elitism and its advantage
6. Reason for chosen crossover technique
7. Reason for chosen mutation technique
8. Reason for adding mean values of some individual
9. Effect of genetic algorithm technique on number of genes
10. Analysis of multi-objective method and how it can be applied to any problem.
11. Reusability for other types of problems

Table 3.1: Variable and their Representation

Variables	**Representation**
Initial Population	Pop_{init}
Genes	v_{i}
Individuals	ind_{i}
Selected Population	Pop_{ns}
Fitness value	F_{v}
Generation	num_Gen
Elite Individuals	$Elite_{\text{inds}}$
Randomly Selected Individuals	Pop_{rs}
Mean values of some Individuals	Pop_{avg}
Parent	P
Children	C
Fitness Objective	Obj
$Pop_{\text{ns}} - Elite_{\text{inds}}$	new_Pop_{ns}

3.1 Average Mixture (AVGM)

The main characteristic of the AVGM genetic algorithm (see figure 3.1a) is the use of the mean values for important operations like crossover and mutation. The steps below provide an overview of how the algorithm works.

Steps

1) Initialization - Start at Gen=0
   1a) Generate an initial large population.

2) Fitness Evaluation
   2a) Evaluate the fitness.
   2b) Arrange in order of fitness value.

3) Selection.
   3a) Select the population.
   3b) Save the elite.
   3c) Save the new population (elite excluded).
   3d) Randomly select individuals from the new population.
   3e) Save the new population (randomly selected individuals excluded).
   3f) Create mean values from some randomly selected individuals in initial population.

4) Evolution
   4a) Crossover: Perform a crossover between the average values and randomly selected individuals from the rest of the population (elite excluded). Dividing the remaining population into two and performing a crossover.
   4b) Mutate: Perform hidden mutation by replacing some individuals (individuals that have lower fitness values) in the new population with some mean values if the mean values have a higher fitness value.

5) Next generation is created (Gen + 1).
6) Back to step two until satisfied.

The discussion below further details some of the process in the steps.

Step 3c Elitism in AVGM

Elitism is a three-step process in the AVGM. The selected population (Pop_{ns}) is known as the elite from Pop_{init}. The elite ($Elite_{\mathrm{inds}}$) are selected from (Pop_{ns}) to be added to the new population created after crossover. The elite are also selected to make it to the next generation.

Step 3e Mean Value Process in AVGM

The AVGM method basically involves the use of mean values in the evolutionary processes (crossover and mutation). Pop_{avg} is derived by selecting some individuals from Pop_{init} and obtaining the mean values of the genes to form new sub-population. The number selected for Pop_{avg} is what determines the number of individuals selected for crossover. These mean values are further used for crossover and mutation.

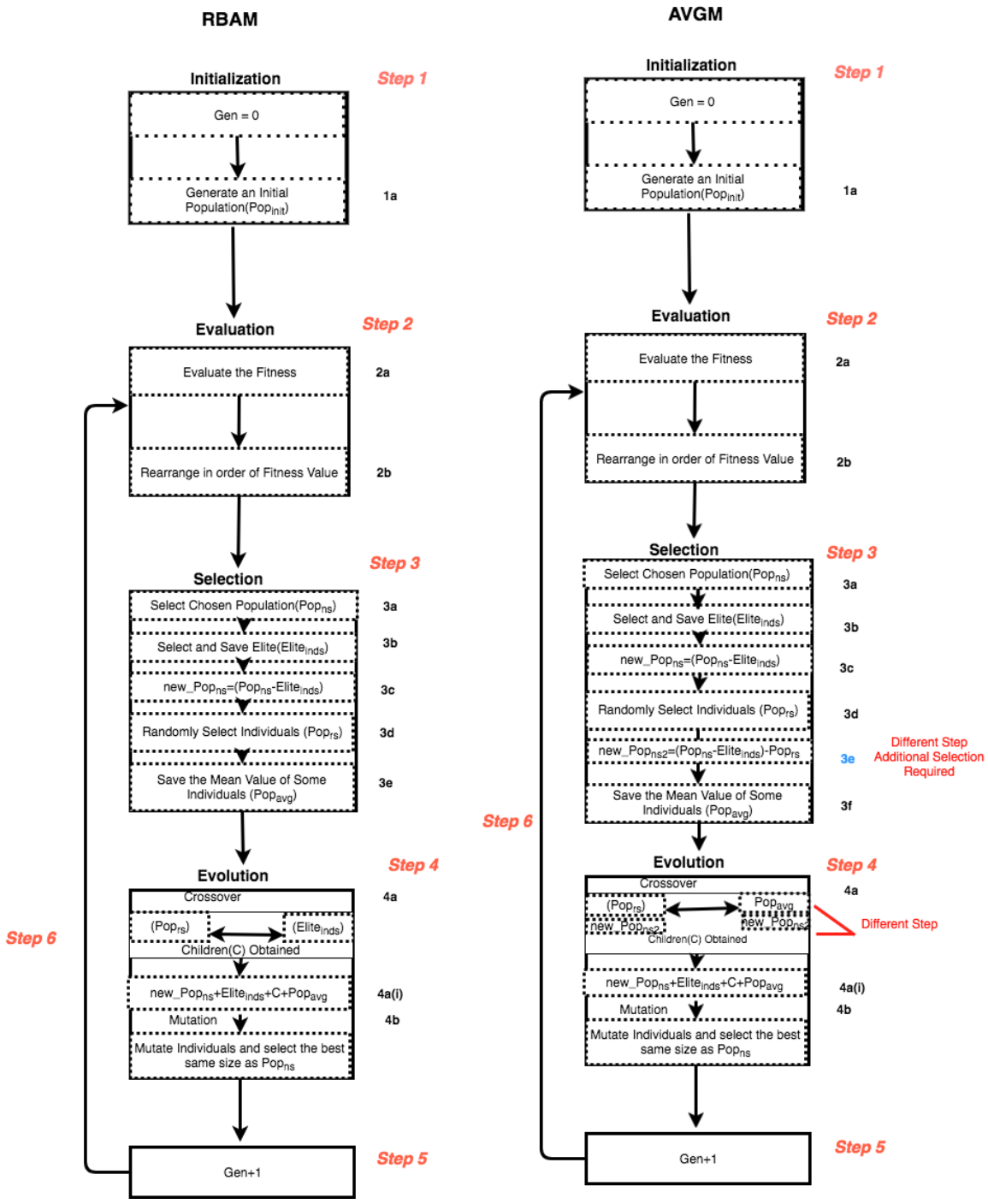
Step 4a The Crossover Process in AVGM

A uniform crossover method is used in AVGM GA. The crossover process is a two-step process. First step involves crossover between Pop_{rs} and Pop_{avg}. Second step involves crossover between individuals in new_Pop_{ns2}, this is done by dividing new_Pop_{ns2} into two groups and performing a crossover between both groups. One or more genes are randomly selected in each generation, this gene is chosen as the part that is copied from parent A and B. The randomly selected genes are then exchanged between parents. The Children (C) are then obtained after these steps have been followed.

Step 4b The Mutation Process in AVGM

The AVGM performs what is known as "hidden mutation". After the elitism, and population after crossover have been obtained. Some individuals are randomly selected

from Pop_{avg} (this is how the hidden mutation is performed) and added to the new population. The population ends up being slightly larger than the size of selected population. Selecting the best individuals from this new population to make it to the next generation then creates another form of elitism. If the value of the individuals selected from $Pop_{!"\#}$ have higher fitness values than some individuals in the new population, they will replace those individuals and this drops the extra individuals. This has a way of optimizing results obtained by the AVGMGA.



a) RBAM b) AVGM

Figure 3.1 RBAM and AVGM GA methods

AVGM Algorithm Definition

1. **Initialization**
   a) Generate initial starting population (Pop_{init}) of size $p\ by\ m$ where

$$p = number\ of\ individuals\ in\ the\ population$$
$$m = number\ of\ genes\ in\ the\ population$$

Each individual is made up of m genes

$$ind_{\text{i}} = [v_{\text{I},1}\ v_{\text{I},2} \ldots v_{\text{I,m}}]$$

The population is made up of p individuals

$$Pop_{\text{init}} = \begin{matrix} ind_1 \\ ind_2 \\ \vdots \\ ind_{\text{p-1}} \\ ind_{\text{p}} \end{matrix}$$

2. **Evaluation**
   a) Given a function y and a fitness objective, evaluate the fitness (F_{v}) of each individual in the population

$$F_{\text{v}}(Pop_{\text{init}}) = \begin{matrix} F_{\text{v}}(ind_1) \\ F_{\text{v}}(ind_2) \\ \vdots \\ F_{\text{v}}(ind_{\text{p-1}}) \\ F_{\text{v}}(ind_{\text{p}}) \end{matrix}$$

b) Organize the population in order of their fitness value (F_{v}) from individual with the highest F_{v} to individual lowest F_{v}, we would then have:

$$Pop_{\text{init}} = \begin{matrix} ind_1 \\ ind_2 \\ \vdots \\ ind_{\text{p-1}} \\ ind_{\text{p}} \end{matrix}$$

$$ind_1 = Individual\ with\ highest\ fitness\ value$$
$$ind_{\text{p}} = Individual\ with\ \text{low}est\ fitness\ value$$

3. Selection

a) Select a subset (Pop_{ns}) of size $n\ by\ m$ from Pop_{init} where
$n = number\ of\ selected\ individuals\ in\ the\ population$

b) From Pop_{ns}, select the first u individuals and save as Elite individuals ($Elite_{\text{inds}}$)
Note: First u individuals are the best individuals because the population has already been arranged in order of their fitness value
$$size(Elite_{\text{inds}}) = u\ by\ m$$

c) From Pop_{ns} randomly select some individuals in the population and get their mean values and save as (Pop_{avg})
$$size(Pop_{\text{avg}}) = z\ by\ m$$
$$z = number\ of\ individuals\ in\ Pop_{\text{avg}}$$

d) Randomly select individuals from Pop_{ns}, the same size as that of the Pop_{avg} $(z\ by\ m)$ and save as Pop_{rs}.
$$size(Pop_{\text{rs}}) = z\ by\ m$$

e) Delete Pop_{rs} and $Elite_{\text{inds}}$ from Pop_{ns} as save as new_Pop_{ns}
$$new\_Pop_{\text{ns}} = (Pop_{\text{ns}} - Pop_{\text{rs}}) - Elite_{\text{inds}}$$
$$size(new\_Pop_{\text{ns}}) = (n - (2z))\ by\ m$$

4. Evolution

a) Crossover

I. Perform a crossover between the selected Parents (Pop_{avg} and Pop_{rs}) by randomly picking one or more genes to crossover.
$$size(C_{\text{T}}) = (2z\ by\ m)$$

II. Randomly pick z individuals from C_{T} to make up C_{S}
$$size(C_{\text{S}}) = (z\ by\ m)$$

III. Divide new_Pop_{ns2} into two groups and perform a crossover between the two groups to create children (C_{np2}).

IV. A new selected population is created
$Pop_{\text{ns}} = new\_Pop_{\text{ns}} + Elite_{\text{inds}} + C_{\text{S}} + C_{\text{np2}} + Pop_{\text{avg}}$
Note: Size of Pop_{ns} $doesn't\ change$

b) Mutation

I. Random select u individuals ($named\ avr_{\text{sel}}$) from Pop_{avg}

II. Add avr_{sel} to Pop_{ns} and evaluate the fitness value

III. Select all individuals with the highest fitness values

as the new Pop_{ns} to make it to the next generation

Note: The number of individual selected as the new Pop_{ns} is dependent on the size of the initial Pop_{ns}

5. **New generation is created after mutation**
6. **Repeat from step 2b.**

3.2 Random-based Average Mixture (RBAM)

The RBAM works with a lot of randomness, which makes it quite flexible and applicable in many instances. The flowchart in Fig 3.1 illustrates how the algorithm works and the algorithm definition is also provided to help explain the steps in a clear manner. The algorithm picks a gene for crossover randomly and performs mutation in a random way too.

Steps

1) Initialization - Start at Gen=0.
   1a) Generate an initial large population.

2) Fitness Evaluation
   2a) Evaluate the fitness.
   2b) Arrange in order of fitness value.

3) Selection
   3a) Select the population.
   3b) Save the elite.
   3c) Save the new population (elite excluded).
   3d) Randomly select individuals from the new population.
   3e) Create average values by randomly selecting some individuals from initial population.

4) Evolution
   - Crossover: Perform a crossover between the elite and the rest of the population.
   - Mutate: Randomly pick some individuals from the population whose genes would be mutated.
5) Next generation is created (Gen + 1).
6) Back to step two until satisfied.

The following discussion expands on some of these steps.

Step 3e Mean Value Process in RBAM

The use of mean values in RBAM basically involves the addition of mean values in the to the newly created population after crossover and mutation. Pop_{avg} is derived by selecting some individuals from Pop_{init} and obtaining the mean values of the genes to form new sub-population. These mean values (Pop_{avg}) are added to the population have a way of making the algorithm efficient and faster in deriving results.

Step 4a The Crossover Process in RBAM

A uniform crossover method is used in RBAM GA. A gene is randomly selected in each generation, this gene is chosen as the part that is copied from parent A and B. The crossover process involves the randomly selected population (new_Pop_{ns}) and the elite ($Elite_{\text{inds}}$). The random genes are then exchanged between parents in new_Pop_{ns}, and parents in $Elite_{\text{inds}}$.

Step 4b The Mutation Process in RBAM

The mutation process involves generating random individuals just like the initialization step at the beginning of the algorithm. The size of the generated individuals is always equal to the size of the Pop_{ns}. One or more genes are randomly selected from these newly generated individuals to replace some genes in the generated population after crossover.

3.2.4 The Final Selection and New Generation of Individuals in RBAM

After the initial selection and evolution process, a final selection mechanism is used in the RBAM. The new population is obtained after elitism, mean value process, crossover and mutation process. The population is made up of the elite, the children generated and the remaining individuals left untouched in the population. The size of

this new generation ends up larger than the size of Pop_{ns}. Selecting only individuals with the highest finest values as the new generation and disposing of the rest individuals deploy a final selection mechanism. This is done to make this new generation equivalent to the size of Pop_{ns}.

RBAM Algorithm Definition

1. **Initialization**
   a) Generate initial starting population (Pop_{init}) of size $p\ by\ m$ where

$$p = number\ of\ individuals\ in\ the\ population$$
$$m = number\ of\ genes\ in\ the\ population$$

Each individual is made up of m genes

$$ind_{\text{i}} = [v_{\text{I},1}\ v_{\text{I},2} \dots v_{\text{I,m}}]$$

The population is made up of p individuals

$$Pop_{\text{init}} = \begin{matrix} ind_1 \\ ind_2 \\ \vdots \\ ind_{\text{p-1}} \\ ind_{\text{p}} \end{matrix}$$

2. **Evaluation**
   a) Given a function y and a fitness objective, evaluate the fitness (F_{v}) of each individual in the population

$$F_{\text{v}}(Pop_{\text{init}}) = \begin{matrix} F_{\text{v}}(ind_1) \\ F_{\text{v}}(ind_2) \\ \vdots \\ F_{\text{v}}(ind_{\text{p-1}}) \\ F_{\text{v}}(ind_{\text{p}}) \end{matrix}$$

   b) Organize the population in order of their fitness value (F_{v}) from individual with the highest F_{v} to individual lowest F_{v}, we would then have:

$$Pop_{\text{init}} = \begin{matrix} ind_1 \\ ind_2 \\ \vdots \\ ind_{\text{p-1}} \\ ind_{\text{p}} \end{matrix}$$

$$ind_1 = Individual\ with\ highest\ fitness\ value$$
$$ind_{\mathrm{p}} = Individual\ with\ \mathrm{low}est\ fitness\ value$$

3. Selection

a) Select a subset (Pop_{ns}) of size $n\ by\ m$ from Pop_{init} where
$n = number\ of\ selected\ individuals\ in\ the\ population$

b) From Pop_{ns}, select the first u individuals and save as Elite individuals ($Elite_{\mathrm{inds}}$)
Note: First u individuals are the best individuals because the population has already been arranged in order of their fitness value
$$size\ (\ Elite_{\mathrm{inds}}) = \ u\ by\ m$$

c) From Pop_{ns} randomly select some individuals in the population and get their mean values and save as (Pop_{avg})
$$size(\ Pop_{\mathrm{avg}}) = \ z\ by\ m$$
$$z\ = \ number\ of\ individuals\ in\ Pop_{\mathrm{avg}}$$

d) Randomly select individuals from Pop_{ns}, the same size as that of the Elite ($u\ by\ m$) and save as Pop_{rs}.
$$size(Pop_{\mathrm{rs}}) = z\ by\ m$$

e) Delete Pop_{rs} and $Elite_{\mathrm{inds}}$ from Pop_{ns} as save as new_Pop_{ns}
$$new\_Pop_{\mathrm{ns}} = (Pop_{\mathrm{ns}} - Pop_{\mathrm{rs}})\ \text{-}\ Elite_{\mathrm{inds}}$$
$$size(new\_Pop_{\mathrm{ns}}) = (\ n - u)\ by\ m$$
$$\mathrm{n}\ = \ number\ of\ individuals\ in\ new\_Pop_{\mathrm{ns}}$$

4. Evolution

a) Crossover

I. Perform a crossover between the selected Parents ($Elite_{\mathrm{inds}}$ and Pop_{rs}) by randomly picking one or more genes to crossover.
$$size\ (\ C) \ = (\ 2u\ by\ m)$$
$$2u\ = number\ of\ Children\ generate$$

II. A new selected population is created
$$Pop_{\mathrm{ns}} = new\_Pop_{\mathrm{ns}} + Elite_{\mathrm{inds}} + C + Pop_{\mathrm{avg}}$$
Note: Size of Pop_{ns} changes
$$size(Pop_{\mathrm{ns}}) = ((n - u)\ by\ m) + (2u\ by\ m) + (\mathrm{z}\ by\ m) = (n + u)\ by\ m$$
$$(n + k) = number\ of\ individuals\ in\ this\ new\ Pop_{\mathrm{ns}}$$

b) Mutation

I. Generate a population of size $n\ by\ m$ (similar to step one with a population size same (Pop_{ns})
II. Randomly select some individuals from the generated population
III. Randomly select specified genes
IV. Get the individual number and gene number and exchange it with exact individual and gene number in population created by previous step.

5. New generation is created after mutation

a) Select all individuals with the highest fitness values as the new Pop_{ns} to make it to the next generation

Note: The number of individual selected as the new Pop_{ns} is dependent on the size of the initial Pop_{ns}

6. Repeat from step 2.

3.3 Multi-objective Random-based Average Mixture (MORBAM)

This MORBAM genetic algorithm is illustrated in the flowchart in Fig 3.4 and just as the name implies, it is used when we have two or more objectives, it is quite close to the RBAM genetic algorithm with some addition to accommodate the multi-objective property. The algorithm applies RBAM GA to each objective and selects the elite from each to make up a population for crossover and mutation. The algorithm definition is also provided to help explain the process better. The crossover and mutation are performed in a way similar to that of the RBAM, the difference being the values that are recombined because of the multiple objectives. The RBAM is used instead of AVGM because of its robustness and flexibility.

Steps

1. For each objective perform RBAM (refer to section 3.2).
2. Select and save the elite from each objective.
3. Create a population by combining these elite individuals.
4. Divide the population into groups and perform a uniform crossover between these individuals.
5. A new population is created after this crossover.

6. Select the elite from the new population created after crossover (best in all objectives).
7. Selecting random individuals from the population created after crossover and adding them to the elite create new generation.
8. Repeat from step one until satisfied.

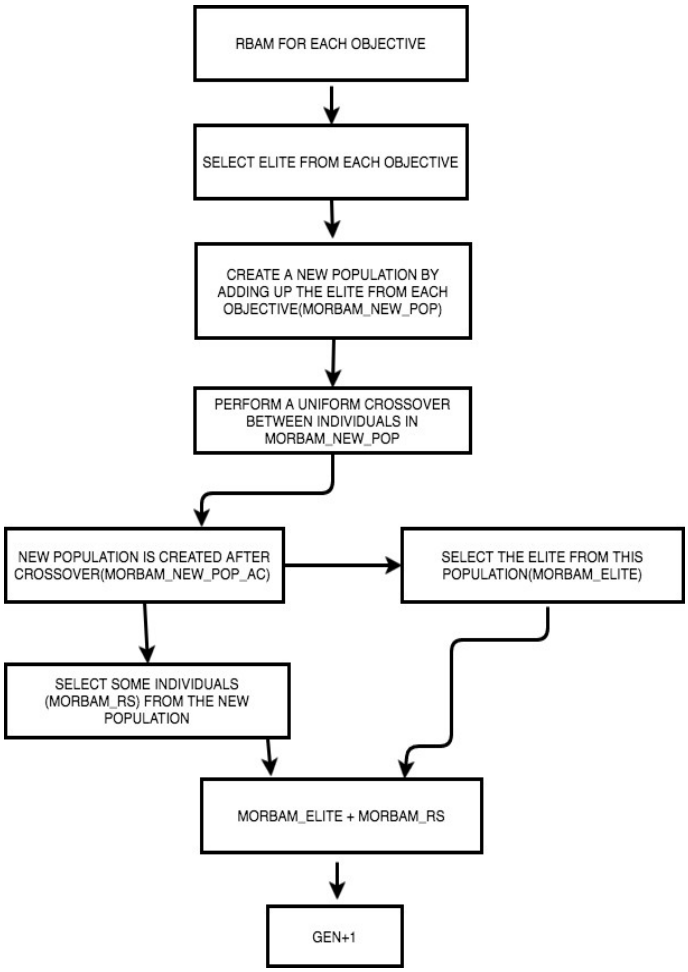


Figure 3.2 MORBAM GA

3.3.1 The Elite (MORBAM_ELITE)

The elite for multi-objective problem is selected by randomly selecting individuals from the population after crossover (MORBAM_NEW_POP_AC), comparing each objective in each individual, and saving the individuals with better fitness values in all objectives.

3.3.2 The Crossover in MORBAM

The main crossover is performed between the individuals derived after RBAM GA has been applied to individual objective. This crossover is performed in a similar way to the RBAM and AVGM. A random gene is selected and exchanged between individuals. Before arriving at the crossover point, we are certain that the population for crossover is trying to evolve towards all objectives. After the crossover, we are certain that the population meets a midpoint. It evolves towards all objectives and finds a balance.

3.3.3 The Mutation in MORBAM

It is not hard to notice that mutation isn't stated in the process. This is because each objective has a mutation factor. The mutation is more of an internal process.

3.4 The GAs Evaluation

Evaluating the RBAM, AVGM and MORBAM GAs, depend on some factors. These variables are listed in Table 3.2. The variables are used in each algorithm definition. The values assigned to the variables can affect how well the genetic algorithm can solve the problem.

Table 3.2: GA representation

Variables	Value
P	To be found
m	To be found
n	To be found
u	To be found
z	To be found
numGen	To be found
Fitness_obj (Deflection)	Given

CHAPTER 4

APPLICATION OF GENETIC ALGORITHM TO A SIMPLE END LOADED CANTILEVER BEAM WITH AND WITHOUT MASS

4.1 MEMS CANTILEVER BEAM

Cantilever beams are very popular and sensitive structures in the field of MEMS. They have a very simple mechanical structure, that's free at one end and fixed at the other end. There are various properties that contribute to the design of the cantilever beam such as the geometric shape and the material properties. A good example of the MEMS cantilever beam is the resonator. The length of the beam is usually significantly more than the thickness and the width. Some equations are used to understand the behavior of MEMS cantilevers; the deflection equation and spring constant equation (stiffness).

The deflection of the beam is dependent on the length (L), width (w), thickness (t), and on the material properties of the beam. The stiffness depends on the geometric structure and material property of the beam. Cantilevers are analyzed based of the type of load applied.

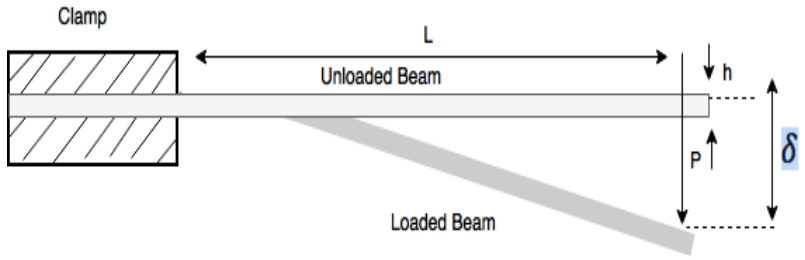


Figure 4.1 The Cantilever Beam

Beams are analyzed for a couple of reasons in MEMS, some of which include;

- *Deflection* of the beam
- S*tiffness* of the beam
- *Resonance Frequency*
- *Internal stress at any point*
- *Maximum stress and its location*

The focus here would be on a simple cantilever beam to illustrate how genetic algorithm can be used for a single objective genetic algorithm with the methods initiated in chapter three.

4.1.1 End loaded Cantilever (Concentrated load at free end)

The deflection in terms of x at any section is given as;

$$y(x) = \frac{L^3}{3EI}(3x^2L - x^3) \tag{4.1}$$

When we substitute L into x (i.e. x= L), the maximum deflection (y) at the free end is then given as:

$$y(x = L) = F\frac{L^3}{3EI} \tag{4.2}$$

where F is the force, L is the length, E is Young's modulus and I is the moment of inertia and for a rectangular beam I is given as:

$$I = \frac{wt^3}{12} \tag{4.3}$$

w is given as the width of the beam and t is the thickness of the beam.

The stiffness (spring constant) is derived from $F = ky$ and for an end loaded cantilever, we can derive stiffness as $k = \frac{3EI}{L^3}$

For an unloaded cantilever, the resonance frequency can be given as

$$\omega_n = 3.52\sqrt{\frac{EI}{m_l L^4}} \tag{4.4}$$

m_l is the mass per unit length of the cantilever.

4.2 DESIGN PARAMETERS USED FOR THE CANTILEVER BEAM

The techniques discussed in chapter three are applied to this simple end loaded cantilever beam and the results are evaluated just to have an idea of how the genetic algorithm technique is implemented. The parameters are provided in Tables 4.1 and 4.2. We will assume a design specification of a deflection of 10 micrometers. The algorithm discussed in chapter three can be applied and represented with Table 4.3. The results of the evaluation using AVGM and RBAM for the design parameters of the genetic algorithm are shown in chapter six.

Table 4.1: Material Properties of Poly-silicon

Properties	Values
Young's Modulus (Pa)	$170\times10^{!}$
Density (kg/m^3)	2320
Poisson's Ratio	0.22

Table 4.2: Genetic Algorithm Parameters for Cantilever

Dimensions	Range (μm)
Length (L)	10-1000
Width (w)	2-50
Thickness (t)	0.5-20
Other Parameters	**Desired Value**
Force (N)	50
Deflection (μm)	10
Resonance Frequency (Hz)	1000Hz

Table 4.3: GA representation

Variables	**Representation**
P (Initial Population size)	100, 200, 500
m (number of genes or variables)	3
n (selected Population size)	100
u (number of elite)	30
z (number of average individuals)	5
numGen	0 to 500
Fitness_obj (Deflection)	10

CHAPTER 5

COMPLETE DESIGN OF COMB DRIVE

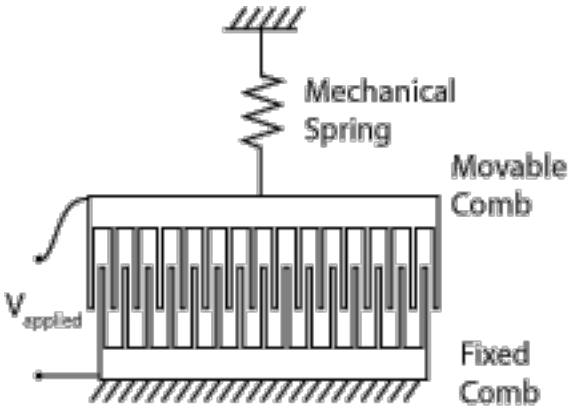


Figure 5.1. The Comb Drive

5.1 INTRODUCTION TO MEMS COMB DRIVE

In MEMS, the comb drive is a laterally driven mechanical actuator, which operates by the use of electrostatic force. It is made up of two inter-digitated fingers structures with one comb fixed and other connected to a suspension as shown in Fig 5.1. When a voltage is applied between the two parts of the comb drive, deflection of the moveable comb fingers occurs by means of electrostatic force. The force generated is small and requires many fingers to generate sufficient force [34] and this reduces the areal efficiency. The generated force is proportional to and in the direction of increasing capacitance. When the combs attract, overlap between the comb finger is increased and this in turn increases the capacitance. The change in capacitance is large when the gap between the fingers decreases and so in order to prevent side-snapping, springs usually restrict motion to the transversal direction. The force in an electrostatic comb drive (which is a non-linear device) is proportional to the applied voltage squared. The position of the moveable finger structure

is controlled by a balance between the electrostatic force and the mechanical restoring force of the compliant suspension.

Mechanical forces also play a vital role in the actuation of electrostatic comb drive. These mechanical forces are generated through spring structures, and they are directly dependent on the stiffness of the flexures, which implies that when these flexures are changed, the mechanical forces also change.

There are several applications of comb drive; they can be used to control micro grippers, resonators, optical shutters, etc. Comb drives are easy to fabricate and with increasing number of comb fingers and decreasing gap spacing, the electrostatic forces increases but due to limitations of fabrication process, the dimensions are limited by minimum feature size constraining. A comb drive with large deflection and minimum voltage is desired here and in order to achieve such, the right combination of materials and feature sizes are required.

5.2 LATERAL DEFLECTION

The capacitance between the stator fixed comb and the moveable comb is given as

$$F_{el} = \frac{1}{2}\frac{\partial C}{\partial y}V^2 = \frac{n\epsilon_0 bV^2}{g} \tag{5.1}$$

where n is the number of comb fingers, ϵ_o is the dielectric constant in air, b is the width of the comb fingers, y is the initial comb finger overlap and dy is the comb displacement and g is the gap between the comb fingers.

The lateral electrostatic force is given as

$$F_{el} = \frac{1}{2}\frac{\partial C}{\partial y}V^2 = \frac{n\epsilon_0 bV^2}{g} \tag{5.2}$$

where V is the applied voltage between the rotor and stator. The electrostatic force acts

on the spring connected to the moveable comb and this results in a deflection given as;

$$y = \frac{n\epsilon_0 bV^2}{K_y g} \qquad (5.3)$$

From the above there are certain factors to consider

- Large force, large displacement and low power consumption is required for the actuator.
- Decreased stiffness in the x-direction causes increased lateral deflection in the y- direction.
- When the spring flexure length is increased, the stiffness in actuation direction decreases.
- When the flexure length is increased, the displacement and capacitance increases.
- Increase in voltage produces an increase in displacement.
- From the stiffness, a direct relationship between applied voltage and displacement covered by the finger is found.
- Number of comb fingers
- Gap between comb fingers

5.3 SPRINGS AND FLEXURAL DESIGN FOR THE COMB DRIVE ACTUATOR

Comb drive actuators have made use of a variety of spring designs, such as clamped-clamped beam, folded beam flexure and crab-leg flexure. There has been a lot of work concentrating on what type of beam produces maximum actuating deflection, folded beam structure has been shown to produce large deflection with low voltage (which is desirable). It is usually desirable to have a structure that's stiff at one direction and compliant in the orthogonal direction.

The linear spring constant is expressed as:

$$k_i = \frac{F_i}{y_i} \tag{5.4}$$

where k_i in this case represents stiffness in the i direction, F_i and y_i represents the force and deflection in the respective direction. This work picks a particular spring type (folded beam) and this is used as the spring for the comb drive, as shown in Figure 5.2.

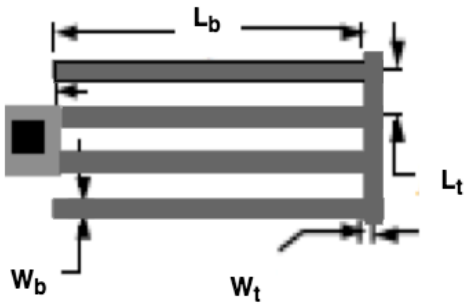


Figure 5.2. Folded Flexure

5.4 FOLDED FLEXURE

There are many benefits associated with the use of folded flexure beams which includes high y-axis to x-axis stiffness (added stiffness to the undesired x-axis) and stress relief by allowing the folding truss to move in the y-direction enabling beams to expand and contract.

The spring constant in the axial direction and lateral direction is found from;

$$k_x = \frac{2Etw}{L} \tag{5.5}$$

$$k_y = \frac{2Etw^3}{L^3} \tag{5.6}$$

The stiffness ratio is given as

$$\frac{k_x}{k_y} = (L/t)^2 \tag{5.7}$$

The folded beam flexure is used for large deflection and strongly reduces the development of axial forces [44]. These large deflections can be obtained by viewing the folded flexure as four parallel folded beams. The stiffness of the folded flexure beam in the x-direction decreases with increased displacement in the y-direction.

These folded flexure beam are quite popular with MEMS comb drive actuator that needs lateral movement in x-axis. The beams are anchored near the center and the trusses allow expansion and contraction of the beams along the x-axis. Each folded beam is a combination of two clamped guided beam connected in series [44]. The folded flexure can be made quite stiff in the y-axis by increasing the width of the truss and made stiff in the z-axis (while decreasing stiffness in x-axis) by increasing the beam thickness.

The beam must be flexible enough in the direction of actuation, increasing the stiffness of the beam would require large electrostatic force and high driving voltage to cause deflection [44].

The stiffness of the folded flexure in the x-direction is given as:

$$k_{x,flexure} = \frac{2Etw_b^3}{L_b^3} \frac{L_t^2+14\propto L_t L_b+36\propto^2 L_b^2}{4L_t^2+41\propto L_t L_b+36\propto^2 L_b^2} \qquad (5.8)$$

The stiffness of the folded flexure (without compression of spring taken into consideration) in the y-direction is given as:

$$k_{y,flexure} = \frac{2Etw_t^3}{L_t^3} \frac{8L_t^2+8\propto L_t L_b+\propto^2 L_b^2}{4L_t^2+10\propto L_t L_b+5\propto^2 L_b^2} \qquad (5.9)$$

$$\propto = \left({}^{w_t}/_{w_b}\right)^3 \qquad (5.10)$$

The spring stiffness due to compression, alone which is given by Hooke's law, is:

$$k_{y,beamcomp} = \frac{2Etw_b}{L_b} \quad (5.11)$$

The effective stiffness in the y-direction is given as a combination of the two springs:

$$k_{y,eff} = \frac{K_{y,flexure} * K_{y,beamcomp}}{K_{y,flexure} + K_{y,beamcomp}} \quad (5.12)$$

I will be solving equation 5.12 in the GA analysis of this structure.

5.5 APPLICATION OF GENETIC ALGORITHM TO COMB DRIVE

Table 6.1 provides the parameters used for the comb drive genetic algorithm with a low applied voltage of 50V and a single objective, which requires the displacement to be 10um. For the multi-objective case, there are three objectives (Deflection, Force and Capacitance) to be met. The deflection has an objective of 10um while, the force has an objective of 50μm and for the capacitance the higher the value, the better it is. As explained in MORBAM, there are three objectives in this case and in step 2 of the MORBAM, the fitness is evaluated based on each objective and are stored separately. In the case of the comb drive, there are three separate evaluations, making up three mini- populations that are crossed over in step 4a(II) of the MORBAM.

Table 5.1: Design Parameters for Comb Drive GA

DIMENSIONS/PARAMETERS	RANGE MIN-MAX
1) Number of moveable comb fingers (n)	3-20
2) Gap between moveable and fixed comb fingers (μm)	2-10
3) Spring width (μm)	2-50
4) Spring length (μm)	10-1000
5) Thickness of actuator (t) overall (μm)	0.5-20
6) Initial Overlap (μm)	5-20
7) Length of truss (μm)	5-30
8) Width of truss (μm)	2-50
OTHER DESIGN PARAMETERS	VALUES
9) Length of comb fingers (μm)	10
10) Width of comb fingers (μm)	2-50
OTHER PARAMETERS	DESIRED VALUES
11) Deflection (μm)	10
12) Force (N)	50
13) Capacitance (pF)	High
14) Voltage (V)	50

CHAPTER 6

EVALUATION OF THE AVGM AND RBAM GENETIC ALGORITHMS IN FULFILMENT OF OBJECTIVES

6.1 EXPLORATION AND EXPLOITATION

The terms in genetic algorithm are 'exploration' and 'exploitation'. The ability for the genetic algorithm to explore the search space is known as *exploration* and exploiting the good solutions to reach a good enough or satisfactory solution is known as *exploitation*. The algorithms move up to 500 generations for testing purpose, to have a good view of how the genetic algorithm works.

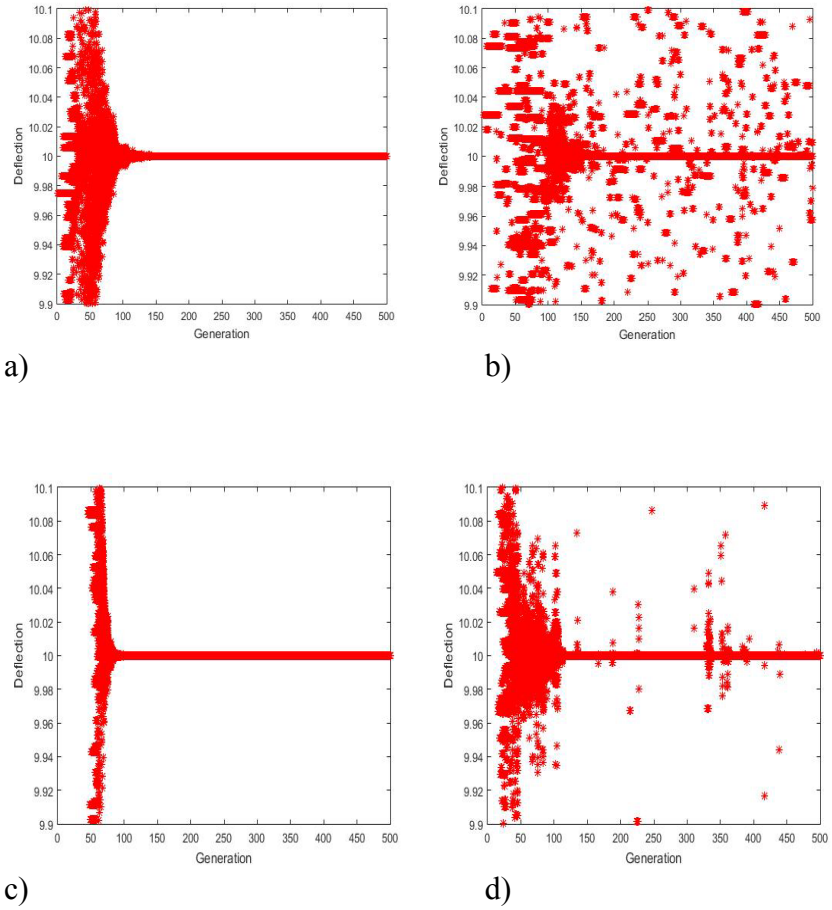


Figure 6.1(a) Shows results obtained using the AVGM for the cantilever beam (b) shows results obtained using RBAM for the cantilever beam (c) shows results obtained using the AVGM for the comb drive actuator (d) shows results obtained using RBAM for the comb drive actuator.

The plots in figure 6.1 and 6.2 illustrate how well the algorithm explores the search space in the early generations and exploits the good solutions by gradually converging and attaining the satisfactory solution. The fitness function is constructed to attain a deflection of 10microns. The AVGM genetic algorithm method works well for the cantilever, with results obtained as early as the first generation but doesn't get results in the early generations when applied to the comb drive actuator and this could be as a result of the different number of genes (variables) in each problem. The RBAM genetic algorithm obtains results as early as generation one when applied to the cantilever beam and comb drive actuator. This implies that there's a lot of flexibility when using the RBAM genetic algorithm for different problems and this makes it a lot more efficient. We also see that the RBAM method maintains a larger number of possible good solutions in the population.

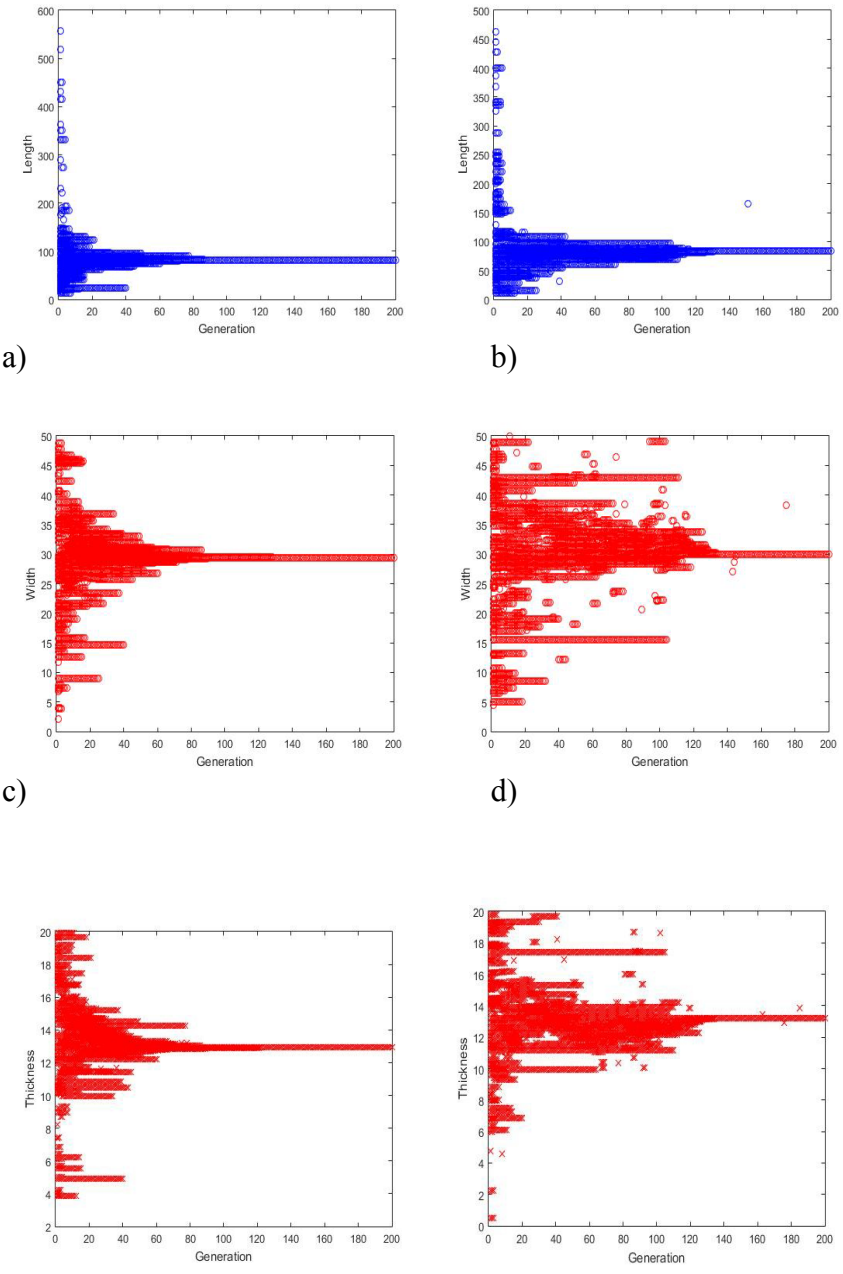


Figure 6.2(a,c,e) Shows the results obtained using the AVGM for the length, width and thickness of the cantilever beam respectively (b,d,f) shows number of solutions obtained using RBAM for the length, width and thickness of the cantilever beam

6.2 DIVERSITY CREATED BY THE ALGORITHM

The diversity, which is related to how well the algorithm can explore a solution without being stuck in the local optimum, plays an important role in the results generated by the genetic algorithm. We want to have good options to pick from and need to be sure the algorithm is constructed in a manner that explores the search space well enough to maintain good diversity. The genetic algorithm can avoid getting stuck in the local optimum by its ability to generate new potential values. The plots in fig 6.3 illustrate how well the algorithm maintains diversity by obtaining distinct values in each generation. The plots show the number of distinct values (see table 6.1) from the results obtained in each generation. These distinct values contribute to the ability for the algorithm to provide a diverse range of solutions. The conclusion is that the RBAM has a larger number of unique solutions.

Table 6.1: Plot Terms and Meaning

Term	Meaning
Total Number of Solutions	The number of times the goal is achieved in a generation
Number of Unique Solutions	The number of different values generated in the current generation
Number of Distinct Solutions in Each Generation	The number of values generated in the current generation that are different from the values generated in the previous generation

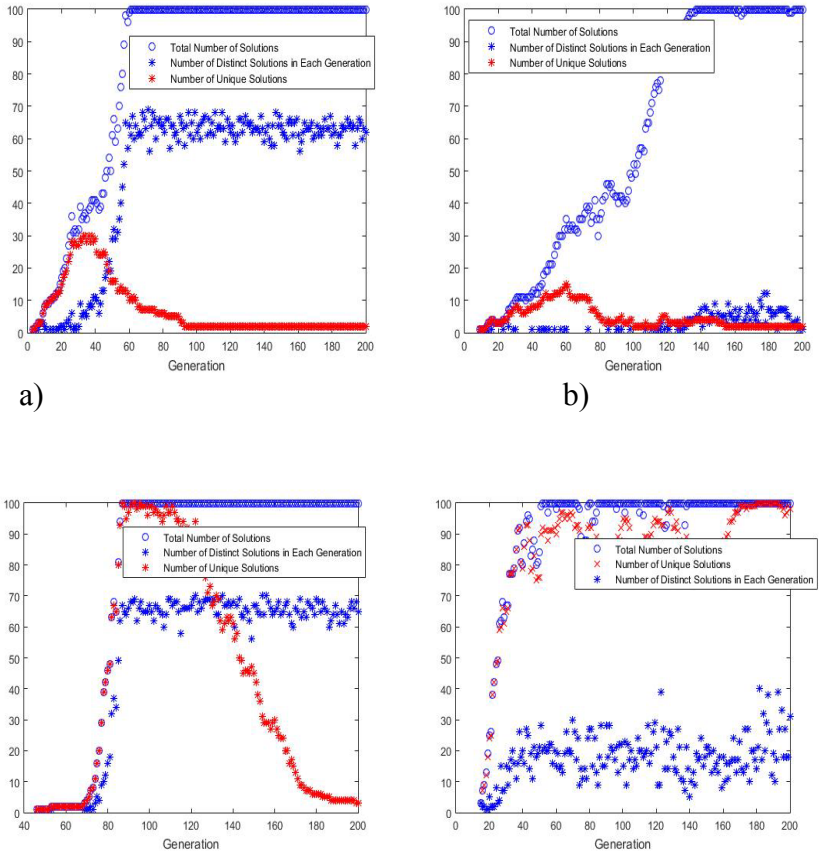


Figure 6.3(a) Shows the number of solutions obtained using the AVGM for the cantilever beam (b) shows number of solutions obtained using RBAM for the cantilever beam (c) shows number of solutions obtained using the AVGM for the comb drive actuator (d) shows number of solutions obtained using RBAM for the comb drive actuator.

6.3 INITIAL STARTING POPULATION AND SELECTED SUBSET

The idea of generating an initial starting population but working with a selected subset introduces a new method of making a genetic algorithm efficient. At the beginning of the algorithm, an initial starting population that's quite large helps with the exploration aspects, ensuring that the search space is optimized by giving as many options as possible to choose from. It's advisable to make this initial starting population a bit large but not necessarily too large i.e. anything from 400 to 1000 should be fine because the algorithm makes it possible

to explore all search space and even if something is missing, the algorithm makes it possible to find it later. In evaluating the results in this chapter population size (P) of 500 was chosen and a selected population size of 100 was chosen (refer to table 4.3). It is also important to note that the AVGM and RBAM algorithm don't suffer from traditional genetic algorithm problem where the size of the population has to increase to decrease computational time and find a solution. These algorithms find solutions in a timely manner regardless of the size of initial starting population.

The selected subset ends up being the size of the population the algorithm ends up working with. It is a percentage of the initial starting population selected to work with until the end. In this case it's always 100, the number increases at the end of the algorithm but 100 is always selected to work with eventually. This method introduced doesn't work with the initial size throughout but rather it uses it as a means to select the size to work with later on.

6.4 SELECTION TECHNIQUE

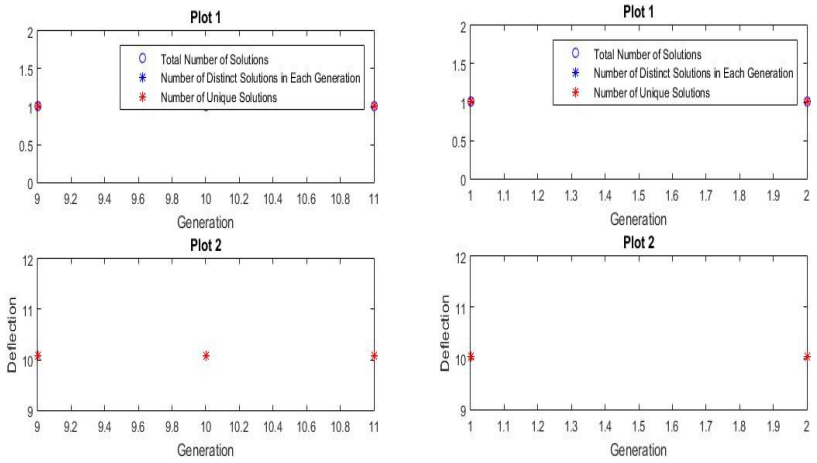
The selection technique has been known to have a huge impact in how the algorithm works. The algorithm saves a percentage of the population as the elite and selects the percentage that would be crossed over.

6.4.1 ELITISM SELECTION MECHANISM

Elitism has been widely used in this thesis to avoid loss of good solutions. The plots in fig 6.4 show results obtained without the use of elitism, we can see that the algorithm becomes less efficient in obtaining results. The AVGM and RBAM genetic algorithm makes good use of the elitism mechanism. In the AVGM GA, the top best solutions are saved while the rest of the population is recombined (crossed over). In the RBAM GA, the best

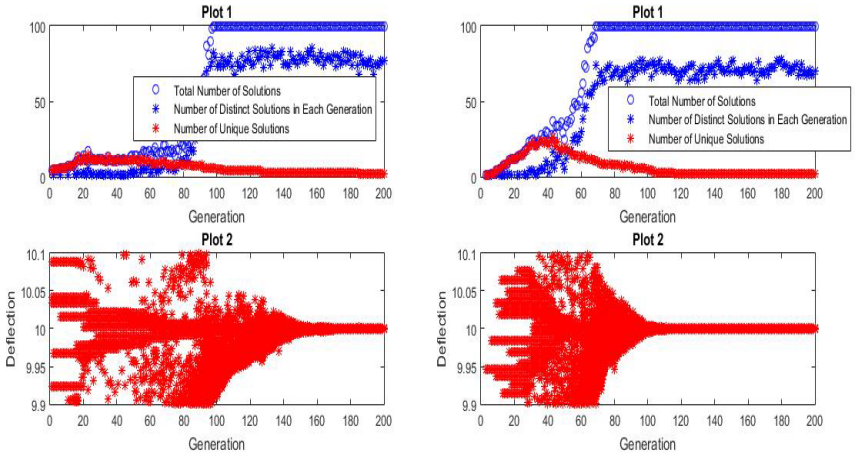
solutions are also saved to survive the next generation but the crossover is also performed with the elite. The question is what percentage of the population should be saved as the Elite?

The plots in Fig 6.5 show the results using the AVGM GA with different percentage selected as elite when applied to the design of a cantilever beam. The algorithm is run a hundred times to have an overall picture of how many results can be generated. More results are generated when the elitism percentage is increased.

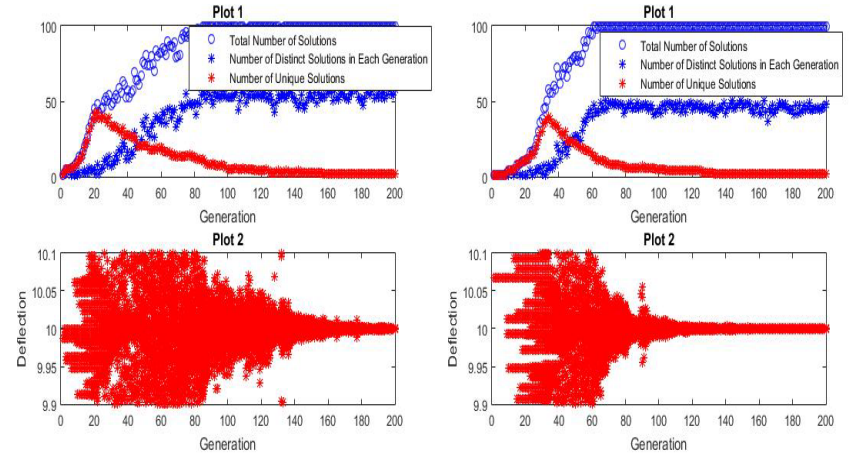


a) b)

Figure 6.4(a) Shows the number of solutions obtained using the RBAM for the cantilever beam without elitism, (b) Shows the number of solutions obtained using the RBAM for the comb drive without elitism.

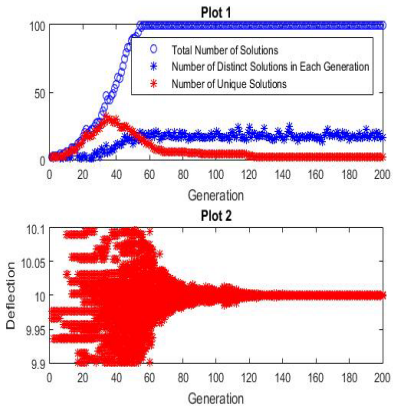


a) b)

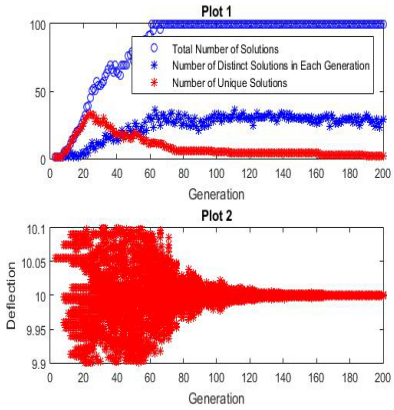


c) d)

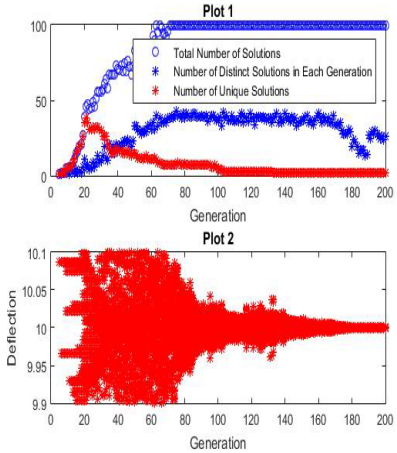
Figure 6.5 (a), (b), (c), (d) Shows the number of solutions obtained using the AVGM for the cantilever beam with different percentage selected as elite. (a) 10%, (b) 20%, (c) 40%, (d) 50%.



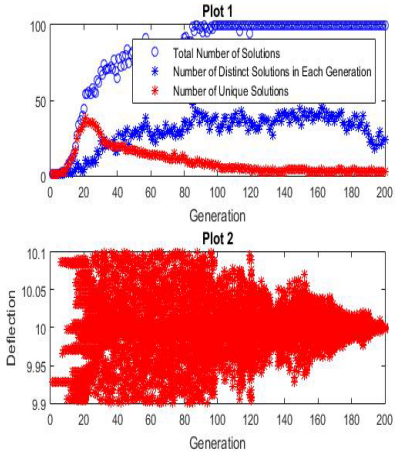
a)



b)



c)



d)

Figure 6.6 (a),(b),(c),(d) Shows the number of solutions obtained using the RBAM for the cantilever beam with different percentage selected as elite. (a) 15%, (b) 30%, (c) 39%, (d) 45%.

Fig 6.5 and 6.6 shows results obtained with different elitism percentage. The higher the percentage, the more the results and the slower it converges. In fig 6.6 the number of distinct solutions start to drop when 39% of the population are reserved as elite in the RBAM GA. In the AVGM GA, there is no significant drop in the number distinct solutions.

6.5 MEAN VALUE

As explained in the earlier sections, the mean value makes up the AVGM method. Fig 6.7 and 6.8 illustrate what happens in the case of the RBAM (for cantilever and comb drive), when the mean value percentage is increased. The mean value addition significantly aids in increment of the number of distinct values produced in each generation for the cantilever beam and comb drive actuator. The number of unique values also increases in each generation. A mean value of about 4% seems appropriate. Another significant use of this mean value is its ability to increase computational speed.

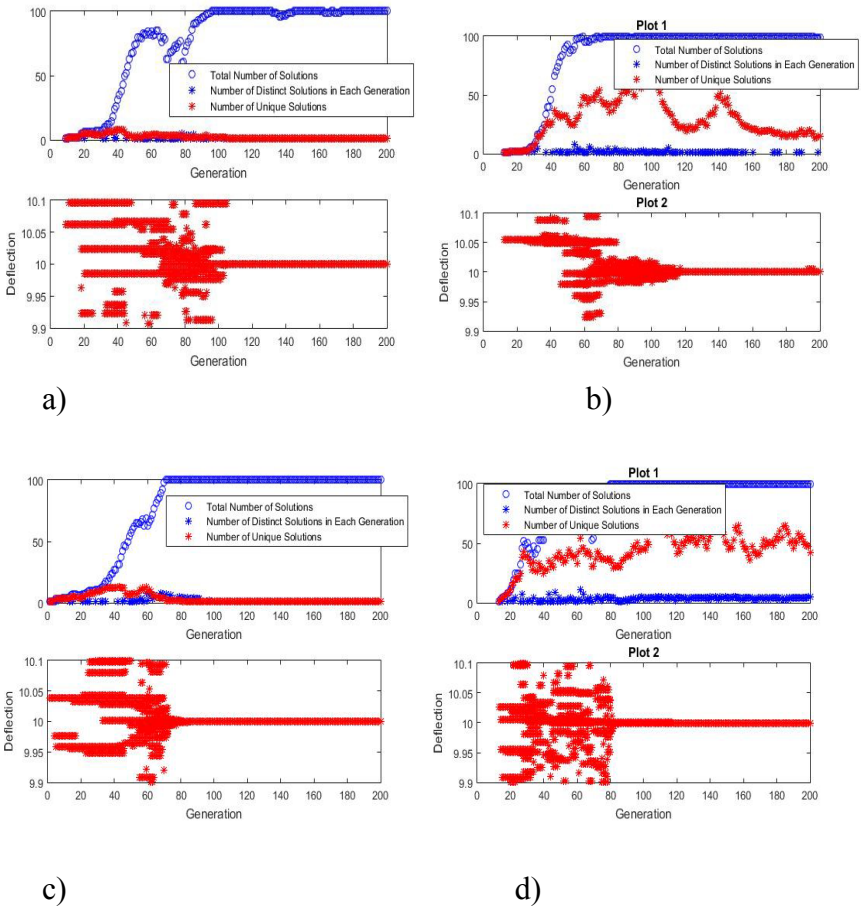
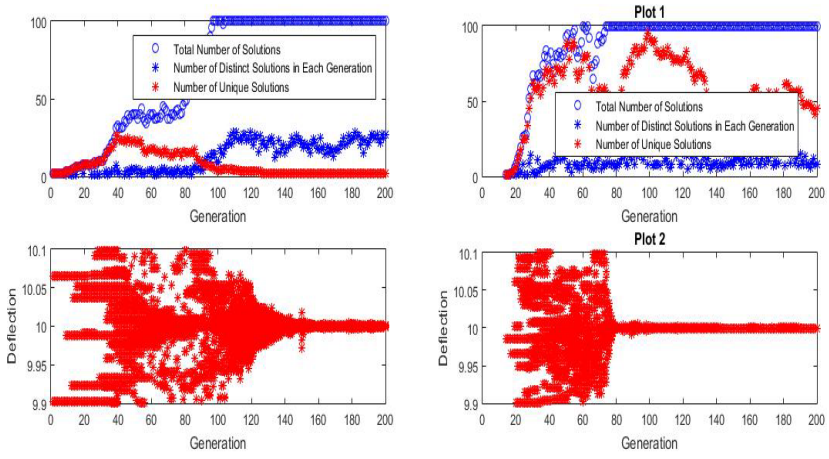
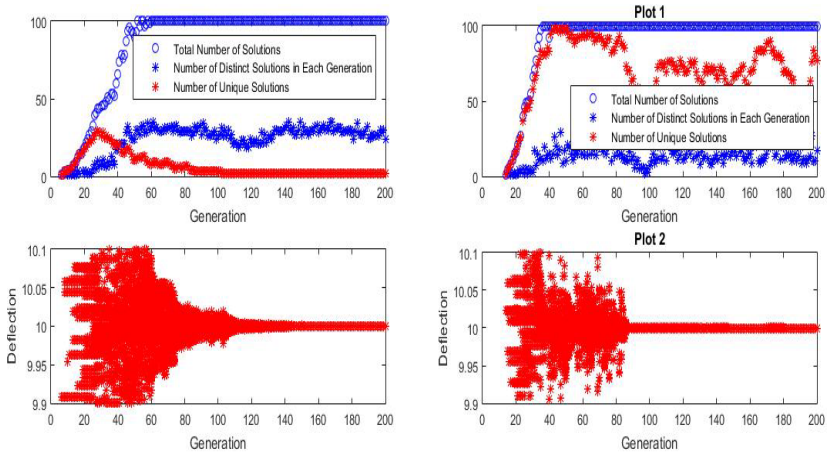


Figure 6.7(a),(c) Shows the number of solutions obtained using the RBAM for the cantilever beam with different percentage selected as mean value. (a) 1%, (c) 4%. (b),(d) Shows the number of solutions obtained using the RBAM for the comb drive actuator with different percentage selected to as mean value. (b) 1%, (d) 4%.



a) b)



c) d)

Figure 6.8(a),(c) Shows the number of solutions obtained using the RBAM for the cantilever beam with different percentage selected as mean value. (a) 8%, (c) 15%. (b),(d) Shows the number of solutions obtained using the RBAM for the comb drive actuator with different percentage selected to as mean value. (b) 8%, (d) 15%.

6.6 CROSSOVER EFFECT

The number percentage of genes in each individual selected for crossover can affect the number of unique solutions generation in each generation. Fig 6.9 and Fig 6.10 and 6.11 show how the crossover percentage affects the population. For the cantilever beam, the ideal crossover percentage is found to be 33.3%. For the Comb Drive, the ideal percentage is 50%(see plot 6.11d).

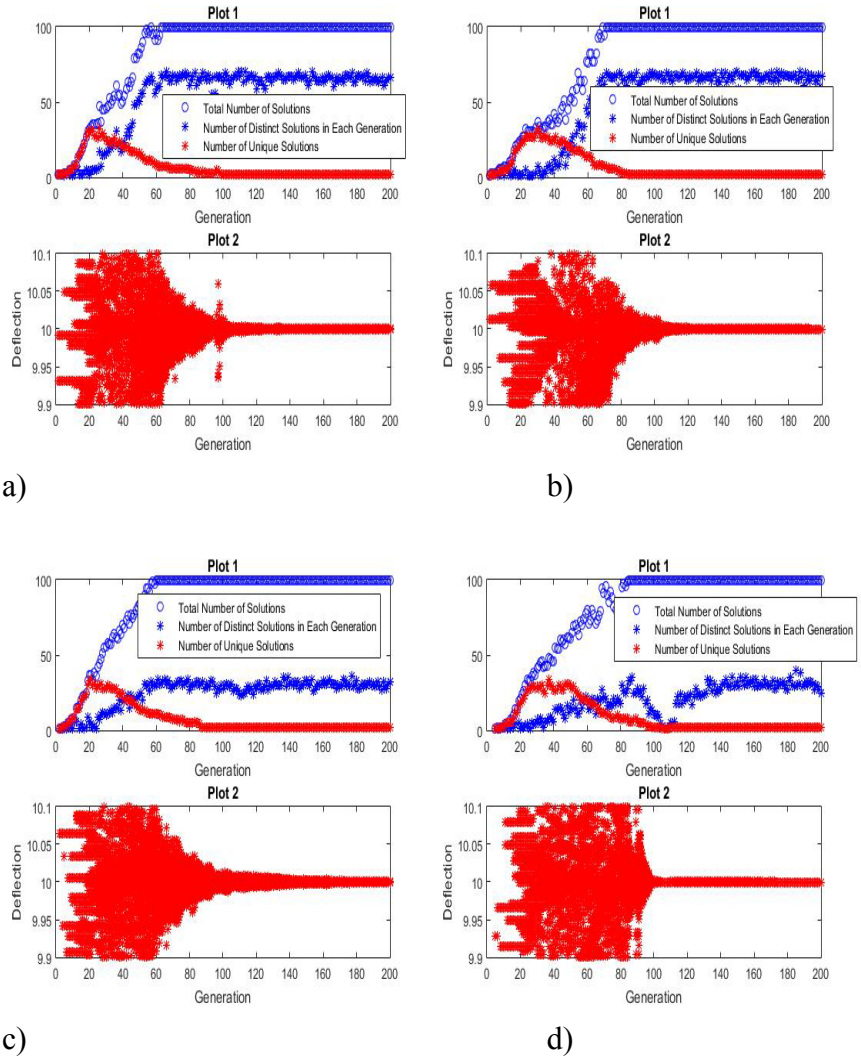
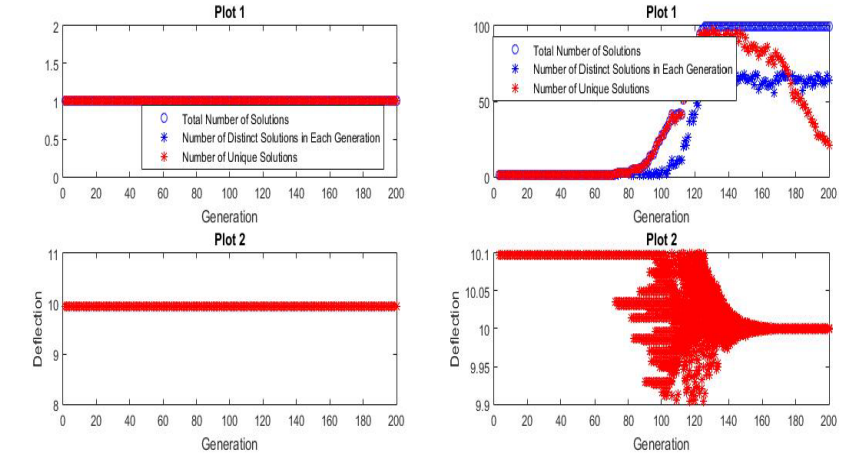
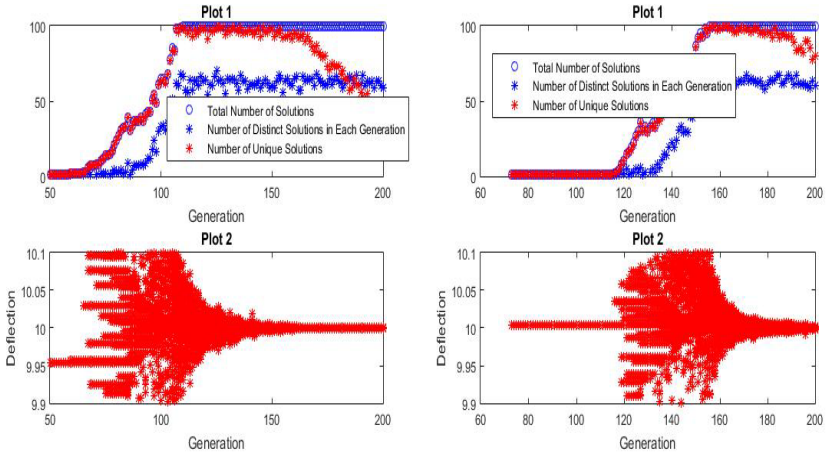


Figure 6.9 Shows the number of solutions obtained using the AVGM for the cantilever beam with different percentage of gene selected for crossover. (a) 33.3%, (b) 66.6%. Shows the number of solutions obtained using the RBAM for the cantilever beam with different percentage selected for crossover. (c) 33.3%, (d) 66.6%.

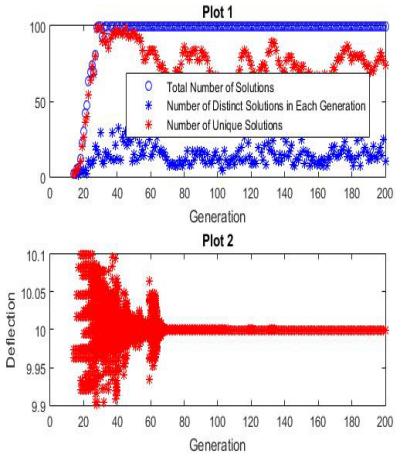


a) b)

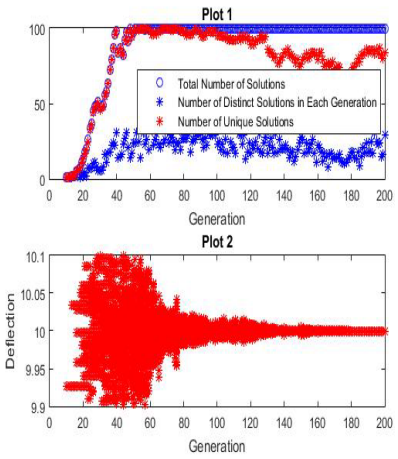


b) d)

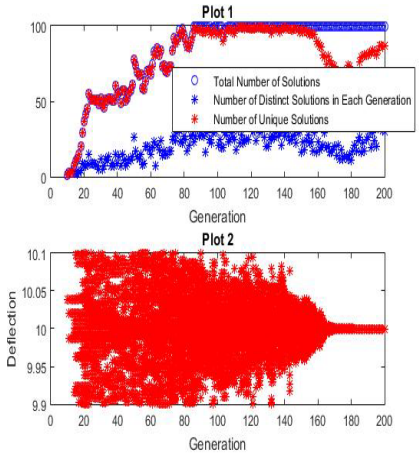
Figure 6.10 Shows the number of solutions obtained using the AVGM for the comb drive actuator with different percentage of gene selected for crossover. (a) 12.5%, (b) 25%, (c) 37.5%, (d) 50%.



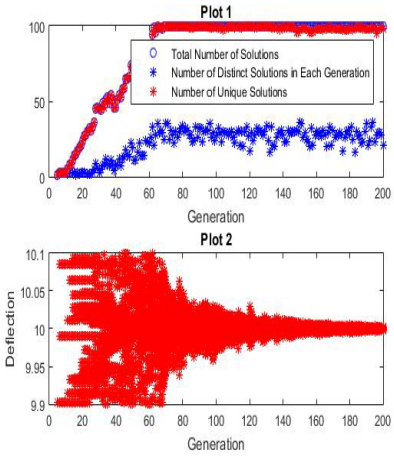
a)



b)



c)



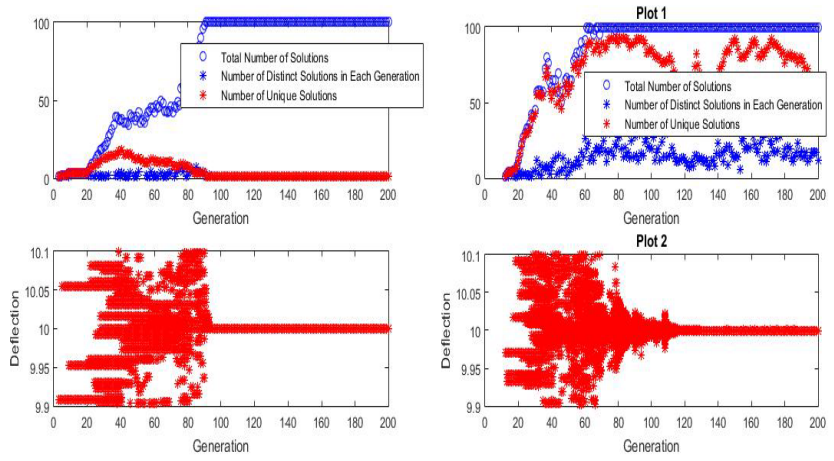
d)

Figure 6.11 Shows the number of solutions obtained using the RBAM for the comb drive actuator with different percentage selected for crossover. (a) 12.5%, (b) 25%, (c) 37.5%, (d) 50%.

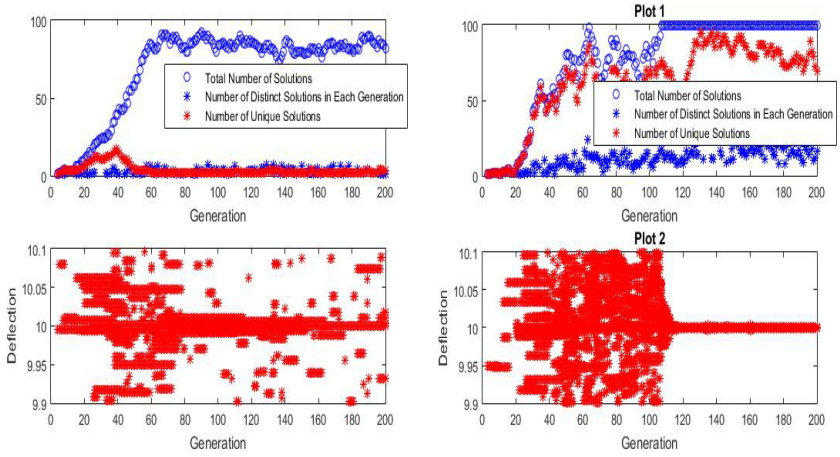
After carefully analyzing the results, the ideal crossover percentage for an even number of genes can be taken as 50% (i.e. the number of gene is simply $n/2$). The number of genes selected for odd numbers ($n/2 - 1$). The crossover technique plays a very important role in the genetic algorithm and its results. The RBAM makes the crossover quite random, ensuring that there's a possibility that different genes get crossed over in different generations. This can also help speed up the algorithm.

6.7 MUTATION EFFECT

A question to ask is: how do we mutate gene values? Given that each gene has different ranges to work with, unlike the binary scheme where a string can be exchanged or like the traditional method of encoding real-valued genetic algorithm were a value is added or subtracted to the gene. A value can't just be added or subtracted because we have to be sure they stay within the specified range and can't be exchanged either because it never converts the values to binary strings at any point. A novel method of mutation was introduced here, that preserves the range and introduces constant diversity while ensuring the algorithm doesn't get stuck in the local search space. The right mutation rate is needed to prevent too much randomness, which prevents the algorithm from converging. The plots in fig 6.12 and 6.13 illustrate results obtained with different mutation rates applied to the cantilever and comb drive designs. The higher the mutation rate the more random the algorithm gets. However good solutions can also be lost in the process which can result in fewer good solutions. In fig 6.13d we see that at the rate of 0.15, the algorithm becomes quite random compared to 6.12a, the number of results in the former is also higher than that of the latter.

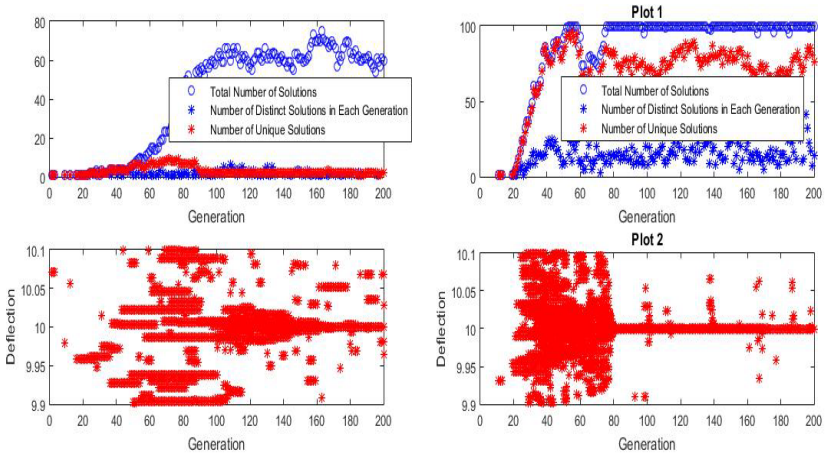


a) b)

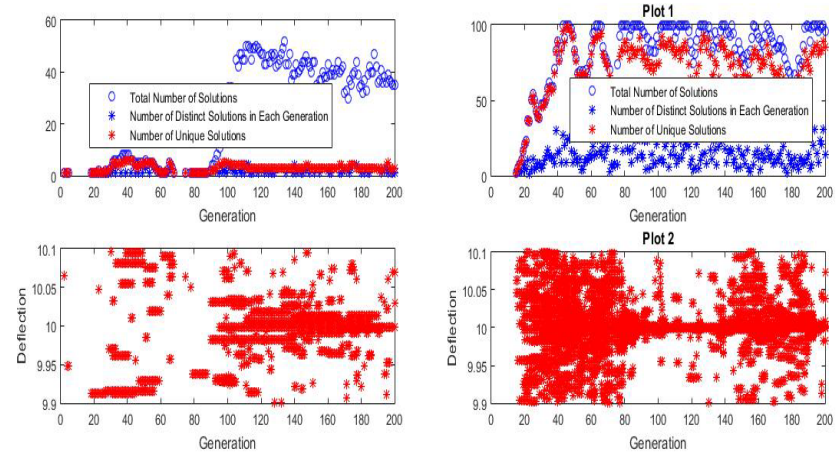


c) d)

Figure 6.12 (a),(c) Shows the number of solutions obtained using the RBAM for the cantilever beam with different percentage selected to mutate. (a) 1%, (c) 5%. (b),(d), Shows the number of solutions obtained using the RBAM for the comb drive actuator with different percentage selected to mutate. (b) 1%, (d) 5%.



a) b)



c) d)

Figure 6.13 (a),(c) Shows the number of solutions obtained using the RBAM for the cantilever beam with different percentage selected to mutate. (a) 10%, (c) 15%. (b),(d) Shows the number of solutions obtained using the RBAM for the comb drive actuator with different percentage selected to mutate. (b) 10%, (d) 15%.

CHAPTER 7

MULTI-OBJECTIVE GENETIC ALGORITHM

The equations provided for the micro-cantilever shows the relationship between length and thickness with resonance frequency and deflection. After simplifying Equation 7.1 to Equation 7.2, it is clear that increasing length and reducing the thickness of the cantilever could increase deflection but decrease resonance frequency and vice versa.

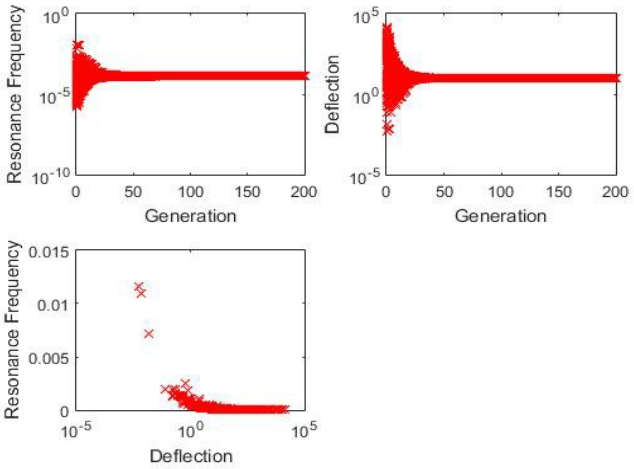
$$f_n = \frac{1}{2\pi}\omega_n = \frac{1}{2\pi}\sqrt{\frac{3EI}{ML^3}} \tag{7.1}$$

$$f_n = \frac{1}{2\pi}\sqrt{\frac{3E}{density}}\frac{t}{L^2} \tag{7.2}$$

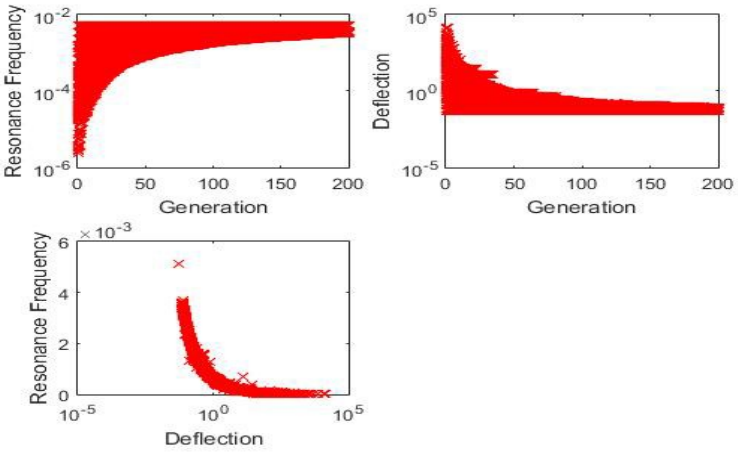
The problem could arise when there are dual objectives to fulfill. Assuming we are required to attain a high deflection and high resonance frequency, the conflicting interest would make it impossible to obtain both, so a balance has to be found somehow. Evolving the genetic algorithm towards deflection would result in a poor resonance frequency result and evolving the result towards resonance frequency would affect the result for deflection.

The plot in Figure 7.1 (b) shows how evolving the algorithm towards deflection alone can't provide the maximum resonance frequency possible. The plot works with the single objective of finding a deflection of $10\mu m$ and it can be seen that the resonance frequency doesn't get better but worse if it's works with the deflection objective. The algorithm was run multiple times to see the maximum attainable resonance frequency, which is visible in the plot. The algorithm was also tried with another single objective of finding a high resonance frequency and Figure 7.1(c) illustrates the maximum possible resonance frequency that can be found with the given specifications, it is clear that if it were evolving

towards resonance frequency alone, a better or higher resonance frequency result can be found. We are left with the question of what to do when we are given more than one objective and the answer would require optimizing the algorithm to create good options to pick from when provided with multiple objectives. We might not get the best of each objective but it is possible to have an overall picture of different options to select from rather than losing totally in one objective and winning in the other.



(a)



(b)

Figure 7.1. (a), shows how increase in deflection for the cantilever causes a decrease in resonance frequency (10^6), (b) shows the maximum result that can be found when evolving the algorithm towards a single objective (deflection).

7.1 EVALUATION OF MORBAM GENETIC ALGORITHMS APPLIED TO THE CANTILEVER AND COMB DRIVE ACTUATOR

There are two key factors in the main multi-objective RBAM (MORBAM) GA. These key factors are Elitism and crossover. Elite individuals are combined for each objective to make a population that is then recombined. The elite for the MORBAM applied to the cantilever are individuals who have better resonance frequency and deflection. The elite for the comb Drive are the individuals with better deflection, capacitance and force. Diversity is also maintained in the algorithm with the use of MORBAM.

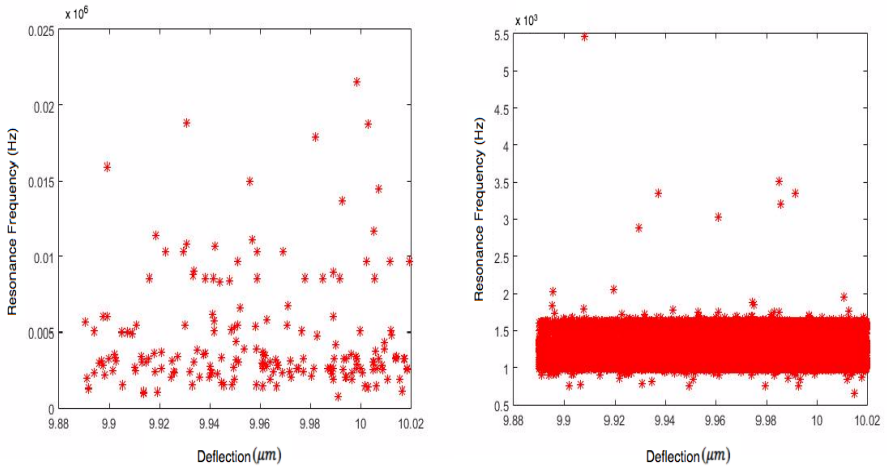
In solving the GA problems, the cantilever beam has two objectives and the comb drive actuator has four objectives (see table 7.1 and 7.2).

7.2 MULTIOBJECTIVE EVALUATION OF THE CANTILEVER

The MORBAM GA applied to the cantilever works in a very efficient way. Rather than working blindly in generations and creating excessive randomness, the algorithm utilizes RBAM GA for each objective and the combines the elite from each objective to make up a population to be recombined. The elite is then selected from this new population and reserved to make it to the next generation. The recombination of different objectives ensures that a mid-point is met between all objectives in a case where one objective doesn't dominate the other.

Table 7.1: Multi-Objective GA representation for the Cantilever Beam

Variables	Representation
P	500
m	3
n	100
k	20
z	10
numGen	0 to 200
(Number of Objectives) Obj	2
Obj_1 (Deflection(μm))	10
Obj_2 (Resonance Frequency(Hz))	As high as possible
r	10



a) b)

Figure 7.2) Shows result using MORBAM for resonance frequency (10^6) and deflection and b) shows result obtained using RBAM.

Fig 7.2a shows higher results can be obtained for resonance frequency for a cantilever beam when the algorithm does not work in the direction of one objective. The results do not settle much in one region like that of fig 7.1b because it is not aimed at one objective.

Fig 7.2b shows that for the cantilever to maintain a deflection of about $10\mu m$ the resonance frequency converges in a certain region. This means that for a deflection of $10\mu m$ in the RBAM the resonance frequency stays around the 1000Hz region and hardly rises above 5500Hz. Obtaining as high as 5500Hz is most likely possible due to the good method

of mutation for the RBAM.

The algorithm is only able to get a good mid-point between both objectives as a result of recombination between the elite in both objectives. Fig 7.3 and 7.4 shows the results obtained with different elite percentage. The higher the elite percentage, the more diverse the solutions. With 10% elitism, the resonance frequency settles more below 4000Hz in Fig 7.3a with a peak (maximum) of about 1700Hz for other diverse range. In the latter plots the concentration in the lower region is less and the resonance frequency reaches a peak of about 2500Hz.

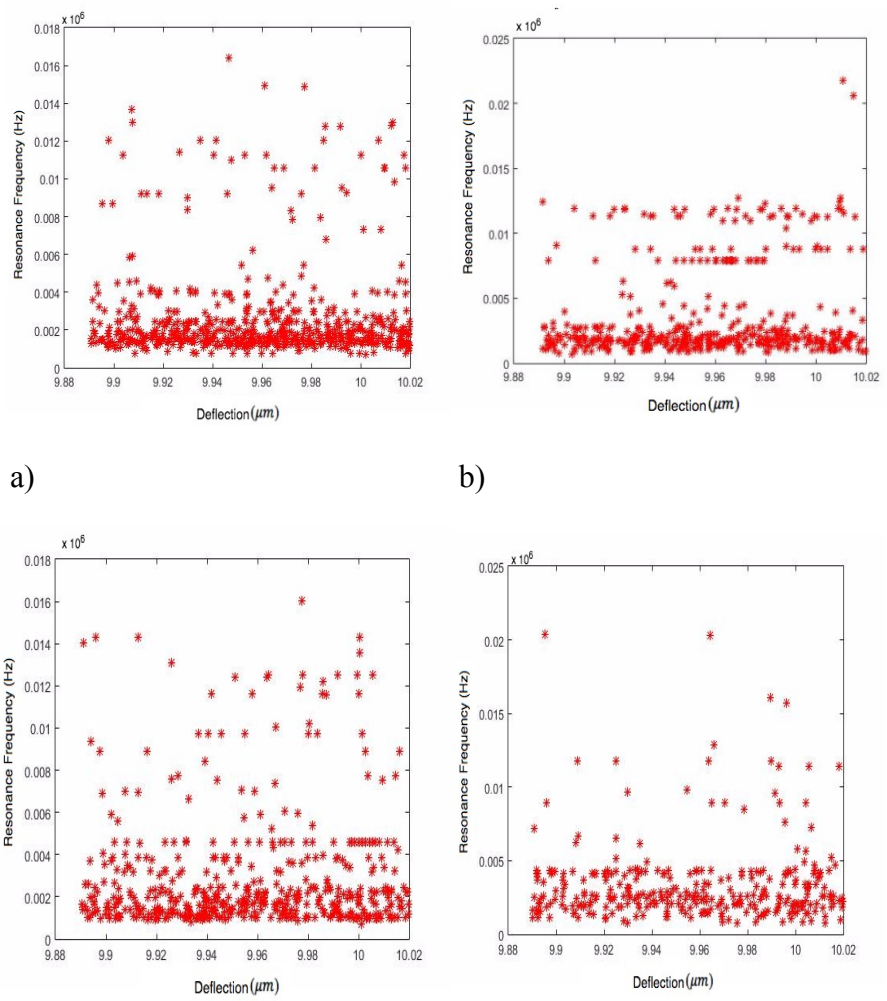
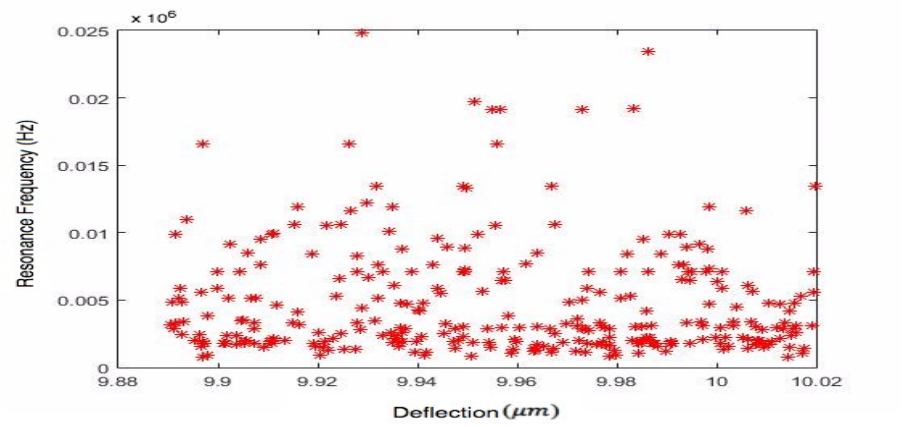
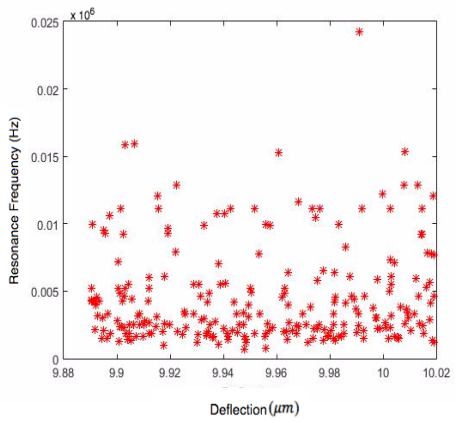


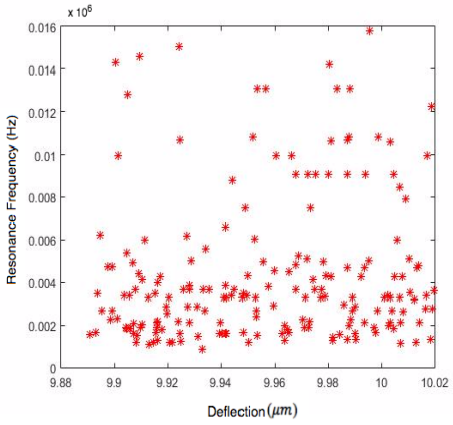
Figure 7.3 Results obtained using the MORBAM for the cantilever a) 10% elite, b)20% elite, c)30% elite, d) 40% elite.



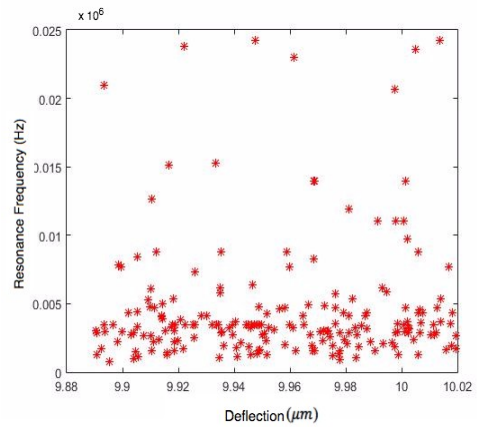
a)



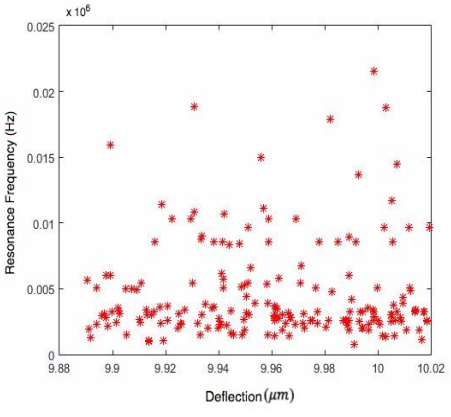
b)



c)



d)



e)

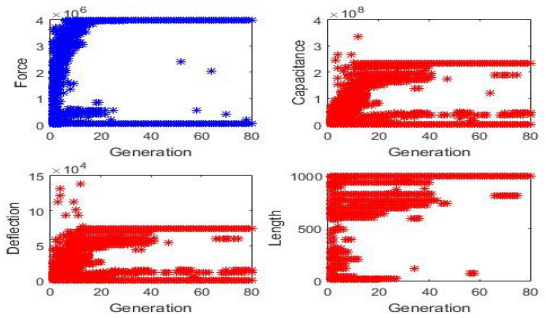
Figure 7.4 Results obtained using the MORBAM for the cantilever. a)50% elite, b)60% elite, c)70% elite, d)80% elite, e)90% elite.

7.2 MULTI-OBJECTIVE RESULTS FOR THE COMB DRIVE ACTUATOR

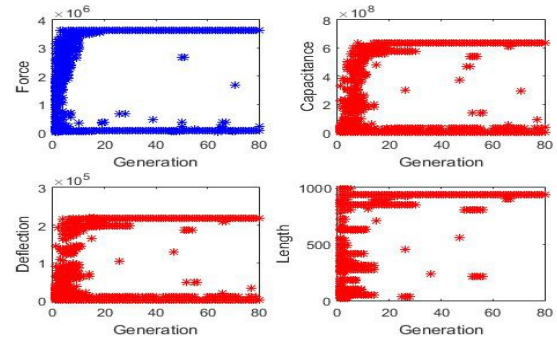
The MORBAM GA applied to the comb drive actuator also performs well and follows the MORBAM algorithm. It is important to note that the comb drive has more objectives than the deflection (refer to table 7.1 for cantilever multi-objective table). The results obtained when the MORBAM GA is applied to the cantilever shows that the algorithm has the capability of finding good solutions. Solutions appear to be more in certain regions as shown in Fig 7.5 and 7.6 but due to the flexibility of the algorithm, good solutions can also be found in unlikely regions. The results are also based of the elitism percentage. The results appear more in unlikely regions when 10% is selected for as the elite.

Table 7.2: Objectives for the Comb Drive Actuator

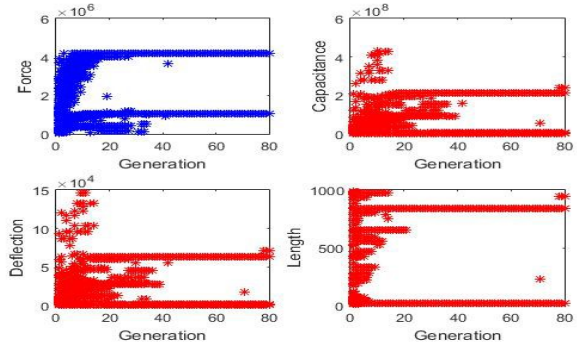
PARAMETER	DESIRED VALUES
Deflection (μm)	10
Force (N)	50
Capacitance (pF)	High
Voltage (V)	50



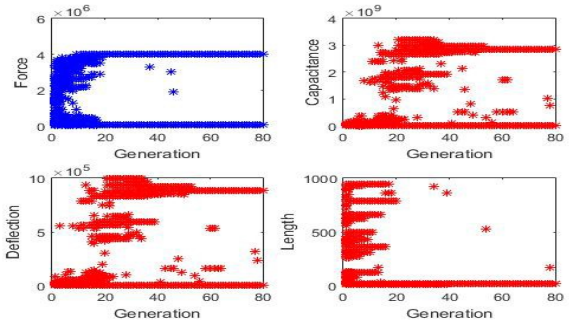
a)



b)

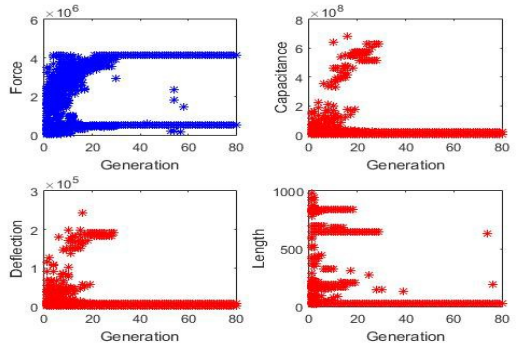


c)

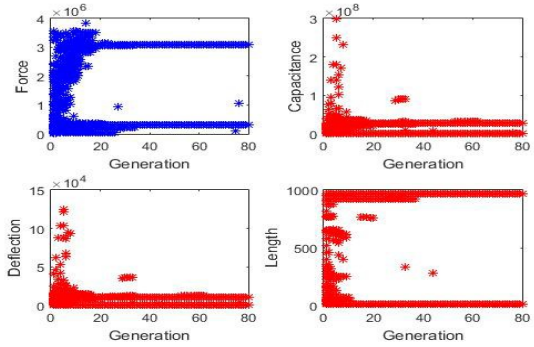


d)

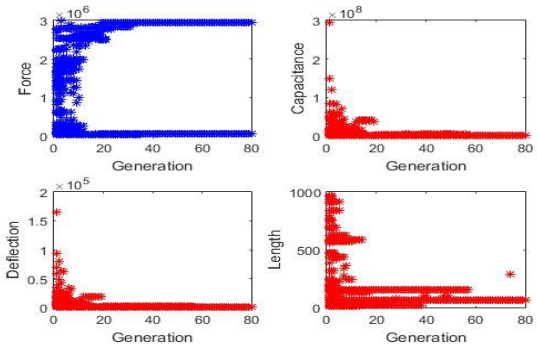
Figure 7.5 Results obtained using the MORBAM for comb drive. a) 10% elite, b) 20% elite, c) 30% elite,d) 40% elite.



a)

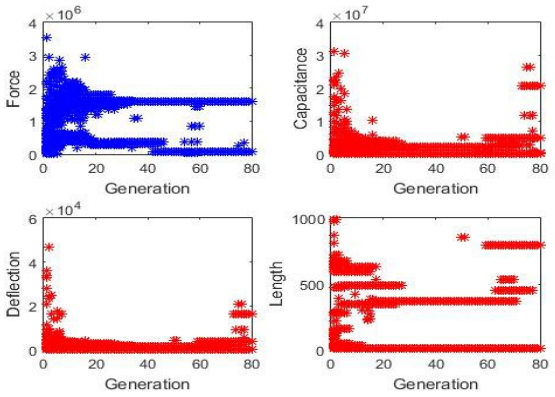


b)

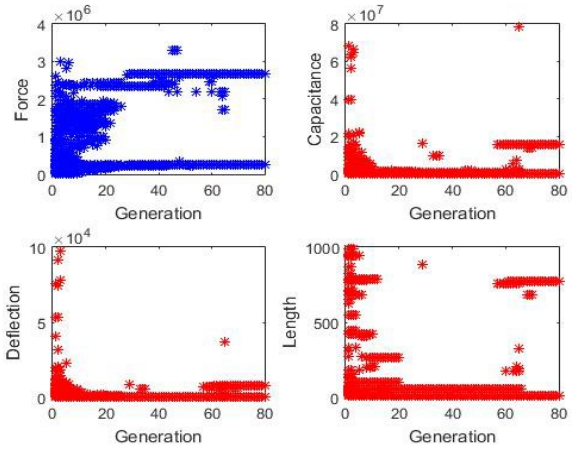


c)

Figure 7.6 Results obtained using the MORBAM for comb drive. a)50% elite, b)60% elite, c)70% elite.



a)



b)

Figure 7.7 Results obtained using the MORBAM for comb drive. a)80% elite, b)90% elite.

CHAPTER 8

CONCLUSION AND FUTURE WORK

Genetic algorithm has a significant use in MEMS. MEMS design parameter problems can be solved properly using genetic algorithm by finding significant and applicable ways of recombination (crossover), mutating and evaluating the genetic algorithm approach. This thesis explores different ideas and ways to use a genetic algorithm when there are different variables to work with. Novel genetic algorithm techniques are introduced by creating smart ways to crossover and trying out a different mutation technique. Three new techniques have been created and tested accordingly and have their different advantages or limitations. The results were also evaluated with the absence of an appropriate mutation technique and it was seen that that even without the new mutation technique, the algorithm found a way to produce good enough solutions making mutation just a plus in the algorithm. In the past, genetic algorithms have been encoded mostly in the binary way and so there's been not much to say about how to mutate except from the standard mutation techniques like (inversion, adding or subtracting a value etc). This work comes up with a new way to mutate the genetic algorithm. These genetic algorithm techniques can be applied to any problem and can produce good results in timely and efficient manner. It is therefore recommended that the AVGM, RBAM and the MORBAM methods are useful for the geometric design of MEMS. They provide quicker ways to optimize the dimensions and produce diverse design parameters that satisfy the end goal. The RBAM, which is one of the single-objective algorithm works better than the AVGM due to its flexibility and random nature while the MORBAM GA works well for problems with multiple objectives. A crossover percentage of 50% for the comb drive actuator produced unique results that were about the same size as the total number of results generated. For the cantilever beam, selecting one gene out of

three genes for crossover seemed appropriate. The mutation percentage responded differently for the different MEMS devices (cantilever and comb drive). It takes the algorithm more generations to converge as the elite percentage is increased. The mean value addition makes the algorithm generate results faster (takes as little as 3s), it also helps to improve the number of distinct solution generated, and aids in the increase of the number of unique solutions generated.

REFERENCES

[1] Er. Ashis Kumar Mishra, Er. Yogomaya Mohapatra, Er. Anil Kumar Mishra. 2013. *Multi-Objective Genetic Algorithm: A Comprehensive Survey*.

[2] Gad-el-Hak, Mohamed. 2006. *MEMS: introduction and fundamentals*. Boca Raton: CRC/Taylor & Francis.

[3] Haupt, Randy and Sue Ellen Haupt. *Practical Genetic Algorithms*. John Wiley & Sons, 1998.

[4] Suryansh Arora, Sumati, Arti Arora, P.J George, "Design of MEMS based Microcantilever using Comsol Multiphysics", Applied Engineering Research, Vol.7 No.11, 2012.

[5] Maziar Norouzi, Alireza K, "Design of Piezoelectric microcantilever Chemical Sensor in Comsol Multiphysics Area", Electrical and Electronics, Vol.2, issue 1,No.184,2009.

[6] Nitin S.Kale,V.Ramgopal Rao,"Design and Fabrication Issues in Affinity Cantilevers for bioMEMS Applications",Micro Electro Mechanical Systems,VOL.15,NO.6,2006.

[7] Sandeep Kumar Vashist,"A Review of Microcatilevers For Sensing Applications", 2007.

[8] Guanghua Wu, Ram H.Datar, Karolyn M.Hasen, Thomas Thundat, Richard J. Cote, Arjun Majumdar, "Bioassay of Prostate-Specific Antigen(PSA) Using Microcantilevers", Nature Biotechnology, VOL.19, No.856,2001.

[9] Pratt R I, Johnson G C, Howe R T and Chang J C 1991 Micromechanical structures for thin film characterization Proc. Int. Conf. on Solid-State Sens. Actuators, Transducers '91 (San Francisco, CA, 1991) pp 205–8.

[10] Groeneveld A W 1995 Electromechanical behaviour and study towards position control of electrostatic comb drive actuators Master Thesis University ofTwente, Enschede, The Netherlands.

[11] G. Leu, S. Simion, and A. Serbanescu. MEMS optimization using genetic algorithms," in CAS 2004 Proceedings, vol. 2, 2004, pp. 475{478.

[12] N. Zhou, A. Agogino, and K. S. J. Pister, \Automated design synthesis for micro-electro-mechanical systems," in Proceedings of DETC 2002: Design Automation, 2002.

[13] Comsol Multiphysics 4.0a, COMSOL Inc., 1997-2010. [Online]. Available: http://www.comsol.com/

[14] MATLAB R2009b, The MathWorks Inc., Natick, Massachusetts, 2009. [Online].

Available: http://www.mathworks.com/products/matlab/

[15] S. A. Campbell, The Science and Engineering of Microelectronic Fabrication, 2nd ed. Oxford University Press, 2001.

[16] K. Deep and M. Thakur. A new mutation operator for real coded genetic algorithms," Applied mathematics and Computation, vol. 193, pp. 211{230, 2007.

[17] F. Herrera and M. Lozano. Two-loop real-coded genetic algorithms with adaptive control of mutation step sizes," Applied Intelligence, vol. 13, no. 3, pp. 187{204, 2000.

[18] Teich, J., “Pareto-Front Exploration with Uncertain Objectives,” Evolutionary Multi-Criterion Optimization, First International Conference, pp314-328, 2001.

[19] Ursen, B.K, “Diversity-Guided Evoluationary Algorithm,” In: Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002), p. 462-471, 2002

[20] Venkataraman, P., "Applied Optimization with MATLAB Programming," John Wiley & Sons, Inc, 2001

[21] Weeks, R. W. and Moskwa, J.J., "Automotive Engine Modeling for Real-Time Control Using MATLAB/SIMULINK," SAE 950417, 1995

[22] ZIMMERMANN H.-J. AND H.-J. S., "Intelligent system design support by fuzzymulti-criteria decision making and/or evolutionary algorithms," in Proceedings of IEEE International Conference on Fuzzy Systems, Yokohama, Japan, 1995

[23] Zitzler, E., Deb, K. and Thiele, L. "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," Evolutionary Computation, 8(2), pp. 173-195, Summer 2000.

[24] Kokolo, I., Hajime, K., and Shigenobu, K., "Failure of pareto based MOEAs: Does non-dominated really mean near to optimal?," In Proceedings of the Congress on Evolutionary Computation 2001 (CEC 2001), vol. 2, pp. 957 – 962, Piscataway, New Jersey, IEEE Service Center, May 2001.

[25] Kirkpartrick, S., Gelatt, C. D., and Vecchi M.P. "Optimization by simulated annealing," Science, 220:671-680, 1983.

[26] Knowles, J. D., "Local Search and Hybrid Evolutionary Algorithms for Pareto Optimization," Doctoral Dissertation, The University of Reading, 2002.

[27] Knowles, J. D. and Corne, D. W., "Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy," Evolutionary Computation 149-172, 2000.

[28] Laumanns, M., Thiele, L., et al., "Combining convergence and diversity in evolutionary multi-objective optimization, "Evolutionary Computation, vol. 10, no. 3, pp. 263-282, Fall 2002.

[29] Lohn, J.D., Kraus, W.F., and Haith, G.L. "Comparing a Coevolutionary Genetic

Algorithm for Multiobjective Optimization," Proceeding of the 2002 IEEE Congress on Evolutionary Computation, pp. 1157-1162, May 2002.

[30] Lu, H. and Yen, G., "Rank-Density-Based Multiobjective Genetic Algorithm and Benchmark Test Function Study," IEEE Transactions on Evolutionary Computation, Vol. 5, No.4, P335-348, August 2001.

[31] Goldberg, D., 1989a, Genetic algorithms in search for optimization and machine learning:Addison-Wesley Pub. Co.

[32] Goldberg, D., 1989b, Sizing populations for serial and parallel genetic algorithms: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers.

[33] Haupt, R., and Haupt, S., 1998, Practical genetic algorithms: John Wyley and sons.

[34] Rob Legtenberg, A W Groeneveld and M Elwenspoek "Comb-Drive Actuators for large displacements" Micromech. Microeng. IOP published 96, pp 320-329 (1996).

[35] Swati Kapoor, Dinesh Kumar, B.Prasad, "MEMS Electrostatic comb actuators with different materials by using COMSOL 3.5a" Department of Electronic Science Kurukshetra University, Kurukshetra (2011).

[36] Rana I. Shakoor, Imran R. Chughtai, Shafaat A. Bazaz, Muhammad J. Hyder, Masood-ul-Hassan, "Numerical Simulations of MEMS Comb-Drive Using Coupled Mechanical and Electrostatic Analyses" PhD fellow, Department of Chemical & Materials Engineering, Pakistan Institute of Engineering & Applied Sciences, Nilore, Islamabad, Pakistan IEEE 7803-9262 July (2005).

[37] Tang W C, Nguyen T C and Howe R T 1989 Laterally driven polysilicon resonant microstructures Sensors Actuators 20 25–32

[38] Brenner R A, Pisano A P and Tang W C 1990 Multiplemode micromechanical

resonators Proc. IEEE Micro Electro Mech. Syst. (Napa Valley, CA 1990) pp 9–14.

[39] Kanchan Sharma, Isaac G. Macwan, Linfeng Zhang, Lawrence Hmurcik and Xingguo Xiong; "*Design Optimization of MEMS Comb Accelerometer" Sensors and Transducers*, Vol. 108, Issue 7, pp 15-30, 2006.

[40] Isabelle P.F. Harouche and C. Shafai "*Simulation of Shaped Comb Drive as a Stepped Actuator for Microtweezers Application*" Sensors and Actuators A Vol.123–124, pp 540–546, March 2005.

[41] Tai-Ran Hsu; "*Micro Assembly: A Technology on the Frontier of New Industrial Automation*" J. Microelectromech. Syst. Vol.10 Issue 2, pp1481–1490, 2005.

[42] Anit Kumar;Encoding Schemes in Genetic Algorithm. International Journal of Advanced Research in IT and Engineering.

[43] Mitsuo Gen, Runwei Cheng ; Genetic Algorithm and Engineering Design.

[44] Shefali Gupta, Tanu Pahwa , Rakesh Narwal , B.Prasad, Dinesh Kumar. Optimizing the Performance of MEMS Electrostatic Comb Drive Actuator with Different Flexure Springs.

APPENDIX A: MATLAB CANTILEVER AVGM CODE

```
%initializing the constants
Force = 50; %given value for force
E=170;  %given value for Young's_Modulus(Poly-Silicon)


ep = 0.30*100; %elite percentage
rap= ep./3;
 ep2= 100-ep;

genecount = 1:3;
selgene = datasample(genecount,1,2, 'Replace', false);%select a gene
   %add 5 averages
avr= 5;

Testvalue = 10.0; %termination condition(goal)
Testvalue2 = 50.0;
hh=15;    %mean value factor
hhh=5;    %number of random rows used to get the mean value(average)

unn = zeros(0,7);
unnn = zeros(0,7);

n=200; % number of generations the algorithm will run for.


%initialization step in the algorithm

numGen=0;  %current generation

Lmin=10.0; %minimum value for length
Lmax=100.0; %maximum value for length

wmin=2.0; %minimum value for width
wmax=50.0;  %maximum value for width

tmin=0.5; %minimum value for thickness
tmax=20.0;  %minimum value for thickness


Pop_init_size = 500;

v1= Lmin+rand(Pop_init_size,1)*(Lmax-Lmin); %All 500 values for length(Length
genes)

v2= wmin+rand(Pop_init_size,1)*(wmax-wmin);%All 500 values for width(width genes)
```

```
v3= tmin+rand(Pop_init_size,1)*(tmax-tmin); %All 500 values for tickness(tickness
genes)


Pop_init=[v1 v2 v3]; %the population
Chrom= Pop_init;


 %evaluate the parent population P(t)
  L=(Chrom(:, 1)); %Length is the first column of the matrix
   w=(Chrom(:, 2)); %width is the second column of the matrix
   t=(Chrom(:, 3));  %thickness is the third column of the matrix

%constructing the fitness objective

    Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
    I = (w.*t.*t.*t)/12;
    stiff=(3.*E.*I)./(L.*L.*L);
       Fitness1 = (0.07586.*t)./ (L.*L);%resonance frequency
   %while it doesn't meet termination condition which is testvalue
   stopFlag = 0;
   %starting the algorithm for n generations

   while stopFlag == 0
    for iii = 1:n
        hh=20;
        hhh=5;
        hep=hh+ep;
       FF = abs(Fitness - Testvalue);

       F = [Chrom FF];
       p =
       sortrows(F,4); h
       = p(1:100, 1:3);
       qw = h(1:ep, 1:3);
        h(1:ep,:)=[];

   avg = zeros(0,1);
         if iii<2
             unnn = zeros(0,7);
         end

      %create average values based on crossover percentage
         for iv = 1:hh

             ab = datasample(Chrom,hhh,1, 'Replace', false);

             ac = mean(ab);
```

```
        avg = [avg; ac];
     end
    avg;
    % select individuals to crossover same size as average
     [ad, add] = datasample(h,hh,1, 'Replace', false);
     h(add,:)= [];% remove selected individual to crossover


     avr_sel =datasample(avg,avr,1, 'Replace', false);%randomly select average
individual to add
     first = ad(:,selgene);
     ad(:,selgene) = avg(:,selgene);
     avg(:,selgene)= first;
     Ch = [ad;avg];
     Ch1= datasample(Ch,hh,1, 'Replace', false);
     l=length(h);
     l2=l/2;
     af=datasample(h,l2,1, 'Replace', false);
     ag=datasample(h,l2,1, 'Replace', false);
     second = af(:,selgene);
     af(:,selgene)=ag(:,selgene);
     ag(:,selgene)= second;
     Child11=[qw; af; ag; Ch1];
    Child1=[qw; af; ag; Ch1; avr_sel];


    Child = Child1;


    Chrom = Child;
    L=(Chrom(:,
    1));
    w=(Chrom(:, 2));
    t=(Chrom(:, 3));


    Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
    I = (w.*t.*t.*t)/12;
     stiff=(3.*E.*I)./(L.*L.*L);
    Fitness1 = (0.07586.*t)./ (L.*L);
    solChrom = Chrom;
     tes = abs(Fitness - Testvalue);
     sorti = sortrows(tes);
      solly = [solChrom stiff Fitness1 Fitness tes];
     sollyreal = sortrows(solly,7);
    avr1=(length(Child11))+avr
    sollyreal(101:avr1,:)=[];
```

```
      size(sollyreal)
      sollyreal2 = sollyreal(:,6);
       sollyreal3 = sollyreal(:,1:5);
     %now count the number of rows in column 5 that's less than 0.1
     thenumber = sum(tes < 0.1)
      nmm= sollyreal(:,7);
     tes1 = sum(nmm < 0.1);
     numGen = numGen + 1;
    %for i =1:158
    cv = sollyreal2(:,:);
     csv = sollyreal3(:,:);
     bn = size(cv);
   un = repmat(numGen, bn);
   unn = [csv cv un];
   unnn=[unnn;unn];

     stopFlag = 1;


end
end
   disp(unnn);% includes all results starting from generation one
```

APPENDIX B: MATLAB CANTILEVER RBAM CODE

```
%initializing the constants
Force = 50; %given value for force
E=170; %given value for Young's_Modulus(Poly-Silicon)
ep = 0.30*100; %elite percentage
rap= ep./3;
Testvalue = 10.0; %termination condition(goal)
Testvalue2 = 50.0;
hh=15;    %mean value factor
hhh=5;    %number of random rows used to get the mean value(average)
unn = zeros(0,7);
unnn = zeros(0,7);
n=200; % number of generations the algorithm will run for.



%initialization step in the algorithm
numGen=0; %current generation
Lmin=10.0; %minimum value for length
Lmax=100.0; %maximum value for length
wmin=2.0; %minimum value for width
wmax=50.0; %maximum value for width
tmin=0.5; %minimum value for thickness
tmax=20.0; %minimum value for thickness
Pop_init_size = 500;
v1= Lmin+rand(Pop_init_size,1)*(Lmax-Lmin); %All 500 values for length(Length
genes)
v2= wmin+rand(Pop_init_size,1)*(wmax-wmin);%All 500 values for width(width genes)
v3= tmin+rand(Pop_init_size,1)*(tmax-tmin); %All 500 values for tickness(tickness
genes)
Pop_init=[v1 v2 v3]; %the population
Chrom= Pop_init;


 %evaluate the parent population P(t)
  L=(Chrom(:, 1)); %Length is the first column of the matrix
   w=(Chrom(:, 2)); %width is the second column of the matrix
   t=(Chrom(:, 3));  %thickness is the third column of the matrix

%constructing the fitness objective
    Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
    I = (w.*t.*t.*t)/12;
    stiff=(3.*E.*I)./(L.*L.*L);
       Fitness1 = (0.07586.*t)./ (L.*L);%resonance frequency
   %while it doesn't meet termination condition which is testvalue
   stopFlag = 0;
```

```
%starting the algorithm for n generations
  while stopFlag == 0
 for iii = 1:n
    FF = abs(Fitness - Testvalue);

    F = [Chrom FF];
    p = sortrows(F,4);
    h = p(1:100, 1:3);
    qw = h(1:ep, 1:3);
    [z, qw1] = datasample(qw,ep,1, 'Replace', false);
     h(qw1,:) = [];


%constructing the average value
     avg = zeros(0,1);
    for iv = 1:hh

        ab = datasample(Chrom,hhh,1, 'Replace', false);

        ac = mean(ab);

         avg = [avg; ac];
     end
   favg=avg;%assigning favg to avg


   %crossover begins here

   a = datasample(z,ep,1, 'Replace', false);%selecting elite individual

   %dividing elite into three and randomly seecting a column(a gene) from each division
   [s b] = datasample(a,rap,1, 'Replace', false);
   [s1 b2] = datasample(a,1,2, 'Replace', false);
   a(b,:) = [];
   [ss bb] = datasample(a,rap,1, 'Replace', false);
   [ss1 bb2] = datasample(a,1,2, 'Replace', false);
   a(bb,:) = [];
   [ss2 bb3] = datasample(a,1,2, 'Replace', false);

   sss = s;
   sa=ss;
   aaa=a;


    [aa l] = datasample(h,ep,1, 'Replace', false);% select 30 unique from the whole
population
```

```
h(l,:) = [];%delete the 30 unique ones from h
mm = h; % save the rest of values in h in mm

%dividing the selected population into three so they can be recombined with the elite
[uu q] = datasample(aa,rap,1, 'Replace', false);
aa(q,:) = [];
[uuk qq] = datasample(aa,rap,1, 'Replace', false);
aa(bb,:) = [];
uuu = uu;
saa=uuk;
aak=datasample(aa,rap,1, 'Replace', false);
aakk=aak;

%recombination(crossover) happens here
uu(:,b2)=s(:, b2);
sss(:,b2)=uuu(:,b2);

uuk(:,bb2)=ss(:,bb2);
sa(:,bb2)=saa(:,bb2);

aak(:,bb3)=a(:,bb3);
aaa(:,bb3)=aakk(:,bb3);
%crossover ends here

childy = [uu; sss; uuk; sa; aak];%saving the children
childy2 = datasample(childy,ep,1, 'Replace', false);%randomly selected some children
Child1 = [childy2; z; mm; favg];
  %children are made up of the elite(z), the averege(favg),the randomly
selected(Childy2),and the rest of the population not recombined(mm)

%mutation begins here
Lmin=10.0; %minimum value for length
Lmax=100.0; %maximum value for length

wmin=2.0; %minimum value for width
wmax=50.0;  %maximum value for width

tmin=0.5; %minimum value for thickness
tmax=20.0; %minimum value for thickness
Pop_init_size = 100;

v1= Lmin+rand(Pop_init_size,1)*(Lmax-Lmin); %All 100 values for length(Length
genes)
v2= wmin+rand(Pop_init_size,1)*(wmax-wmin);%All 100 values for width(width
genes)
v3= tmin+rand(Pop_init_size,1)*(tmax-tmin); %All 100 values for tickness(tickness
genes)
Pop_init=[v1 v2 v3]; %the population
```

```
    Chrommutate = Pop_init;
    [D G] = datasample(Chrommutate,1,1, 'Replace', false);
    [HH CC] = datasample(D,1,2, 'Replace', false);
    Child1(G,CC) = HH;
    %mutation ends here
    Child = Child1; %the new population after mutation is assigned to variable called
child
      Chrom = Child;
      L=(Chrom(:,
      1));
      w=(Chrom(:, 2));
      t=(Chrom(:, 3));
      Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
      I = (w.*t.*t.*t)/12;
      stiff=(3.*E.*I)./(L.*L.*L);%stiffness
      Fitness1 = (0.07586.*t)./ (L.*L);
      solChrom = Chrom;
       tes = abs(Fitness - Testvalue);
       sorti = sortrows(tes);
       solly = [solChrom stiff Fitness1 Fitness tes];
       sollyreal = sortrows(solly,7);
       sollyreal(101:end,:)=[];%population is selected in order of fitness(100 individuals
selected)
       sollyreal2 = sollyreal(:,6);
       sollyreal3 = sollyreal(:,1:5);
       %now count the number of rows in column 5 that's less than 0.1
       thenumber = sum(tes < 0.1)
       nmm= sollyreal(:,7);
       tes1 = sum(nmm < 0.1);
       numGen = numGen + 1;
       cv = sollyreal2(1:tes1,:);
       csv = sollyreal3(1:tes1,:);
       bn = size(cv);
       un = repmat(numGen, bn);
       unn = [csv cv un];
       unnn = [unnn; unn];
       stopFlag = 1
  end
  end
    disp(unnn);% includes all results starting at numGen=1,ending at numGen=n
```

APPENDIX C: MATLAB MORBAM CANTILEVER CODE

```
for iiii=1:100;
Force = 50;
E=170;


    Nind = 500; %population of 30 individual individuals
    FieldDR = [10 2.0 0.5; 1000.0 50.0 20.0]; %Genes = 3; genes upper and lower bounds
    ep = 0.30*100;
    rap= ep./3;
    Testvalue = 10.0; %termination condition
    Testvalue2 = 50.0;
    hh=5;
    hhh=5;
    F2 = 0;
    Fitness= 0;
    unn = zeros(0,7);
    unnn = zeros(0,7);
    if iiii < 2
     unnnrun1 = zeros(0,7);

    end
    unnnrun1 = unnnrun1;
    genecount1 = 1:3;
    selgene = datasample(genecount1,1,2, 'Replace', false);%select a gene
   sel1=50;
    sel11=50;
   sel2=100;
   sel22=60;
   ell=60;
   elt=40;
% %
% %
    Chrom = crtrp(Nind, FieldDR);


   %evaluate the parent population P(t)
   L=(Chrom(:, 1));
   w=(Chrom(:, 2));
   t=(Chrom(:, 3));



    Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
    I = (w.*t.*t.*t)/12;
```

```
 stiff=(3.*E.*I)./(L.*L.*L);

    Fitness1 = (0.68037.*t)./ (L.*L);
%while it doesn't meet termination condition which is testvalue
stopFlag = 0;
numGen=0;
while stopFlag == 0
 for iii = 1:200
    FF = abs(Fitness - Testvalue);
     FFr = abs(Fitness1 -
     Testvalue2); FFF = [Chrom FF
     FFr];
    F = [Chrom FF];
    p =
    sortrows(F,4); h
    = p(1:100, 1:3);
     qw = h(1:ep, 1:3);
     [z, qw1] = datasample(qw,ep,1, 'Replace', false);
      h(qw1,:) = [];

%second fitness



    Fr = [Chrom FFr];
    pr = sortrows(Fr,4);
    hr = pr(1:100, 1:3);
     qwr = hr(1:ep, 1:3);
     [zr, qw1r] = datasample(qwr,ep,1, 'Replace', false);
      hr(qw1r,:) = [];

%         elitism selection



% %       if iii>1<5
% %          zh = unique(h, 'rows');
% %          z=zh(1:30,1:3);
% %
% %       end


      avg = zeros(0,1);
     for iv = 1:hh

         ab = datasample(Chrom,hhh,1, 'Replace', false);

         ac = mean(ab);
```

```
        avg = [avg; ac]; end

    favg=avg;



    a = datasample(z,ep,1, 'Replace', false);

%       [s i ] = datasample(m,2,1,  'Replace', false)
    [s b] = datasample(a,rap,1, 'Replace', false);
    [s1 b2] = datasample(a,1,2, 'Replace', false);
    a(b,:) = [];
    [ss bb] = datasample(a,rap,1, 'Replace', false);
    [ss1 bb2] = datasample(a,1,2, 'Replace', false);
    a(bb,:) = [];
    [ss2 bb3] = datasample(a,1,2, 'Replace', false);

    sss = s;
    sa=ss;
    aaa=a;



    [aa l] = datasample(h,ep,1, 'Replace', false);% select 30 unique from the 100
    h(l,:) = [];%delete the 30 unique ones from h
    mm = h; % save the rest of values in h in mm


    [uu q] = datasample(aa,rap,1, 'Replace', false);
    aa(q,:) = [];
    [uuk qq] = datasample(aa,rap,1, 'Replace', false);
    aa(bb,:) = [];
    uuu = uu;
    saa=uuk;
    aak=datasample(aa,rap,1, 'Replace', false);
    aakk=aak;


    uu(:,b2)=s(:, b2);
    sss(:,b2)=uuu(:,b2);

    uuk(:,bb2)=ss(:,bb2);
    sa(:,bb2)=saa(:,bb2);

    aak(:,bb3)=a(:,bb3);
    aaa(:,bb3)=aakk(:,bb3);
```

```
%         Child = [z ; a; j; m];
        childy = [uu; sss; uuk; sa; aak];
        childy2 = datasample(childy,ep,1, 'Replace', false);
        Child1 = [childy2; z; mm; favg];
%          Child = [z ; a; j; kk; m; uk];
      %mutate

     Nind = 100; %population of 30 individual individuals
    FieldDR = [10 2.0 0.5; 1000.0 50.0 20.0]; %Genes = 3; genes upper and lower bounds
     Chrommutate = crtrp(Nind, FieldDR);
     [D G] = datasample(Chrommutate,10,1, 'Replace', false);
    [HH CC] = datasample(D,1,2, 'Replace', false);
    Child1(G,CC) = HH;
       %after mutate
       Childd = Child1;
    %mutation didn't seem to have a problem on the space; probably doesnt
    %get stuck in local optimum
   L=(Childd (:, 1));
    w=(Childd (:, 2));
    t=(Childd (:, 3));



    Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
    I = (w.*t.*t.*t)/12;


       eva1 = abs(Fitness - Testvalue);
     Fit1 = [Childd eva1];
    Childd = sortrows(Fit1,4);
    Childd=Childd(:,1:3);

% % %     second objective

ar = datasample(zr,ep,1, 'Replace', false);

%       [s i ] = datasample(m,2,1,  'Replace', false)
      [s b] = datasample(ar,rap,1, 'Replace', false);
      [s1 b2] = datasample(ar,1,2, 'Replace', false);
      ar(b,:) = [];
      [ss bb] = datasample(ar,rap,1, 'Replace', false);
      [ss1 bb2] = datasample(ar,1,2, 'Replace', false); ar(bb,:) = [];
      [ss2 bb3] = datasample(ar,1,2, 'Replace', false);

      sss = s;
      sa=ss;
      aaa=ar;
```

```
[aa l] = datasample(hr,ep,1, 'Replace', false);% select 30 unique from the 100
hr(l,:) = [];%delete the 30 unique ones from h
mm = hr; % save the rest of values in h in mm


[uu q] = datasample(aa,rap,1, 'Replace', false);
aa(q,:) = [];
[uuk qq] = datasample(aa,rap,1, 'Replace', false);
aa(bb,:) = [];
uuu = uu;
saa=uuk;
aak=datasample(aa,rap,1, 'Replace', false);
aakk=aak;


uu(:,b2)=s(:, b2);
sss(:,b2)=uuu(:,b2);

uuk(:,bb2)=ss(:,bb2);
sa(:,bb2)=saa(:,bb2);

aak(:,bb3)=a(:,bb3);
aaa(:,bb3)=aakk(:,bb3);



%       Child = [z ; a; j; m];
      childyr = [uu; sss; uuk; sa; aak];
      childy2r = datasample(childyr,ep,1, 'Replace', false);
      Child1r = [childy2r; zr; mm];
%        Child = [z ; a; j; kk; m; uk];
    %mutate

   Nind = 100; %population of 30 individual individuals
  FieldDR = [10 2.0 0.5; 1000.0 50.0 20.0]; %Genes = 3; genes upper and lower bounds
   Chrommutater = crtrp(Nind, FieldDR);


   [D G] = datasample(Chrommutater,10,1, 'Replace', false);
  [HH CC] = datasample(D,1,2, 'Replace', false);
  Child1r(G,CC) = HH;
     %after mutate
     Childr = Child1r;
  %mutation didn't seem to have a problem on the space; probably doesnt
  %get stuck in local optimum

  L=(Childr (:, 1));
```

```
  w=(Childr (:, 2));
  t=(Childr (:, 3));



   I = (w.*t.*t.*t)/12;

     Fitness1 = (0.68037.*t)./ (L.*L);

    eva2 = abs(Fitness1 - Testvalue2);

    Fit2 = [Childr eva2];
    Childr = sortrows(Fit2,4);
    Childr=Childr(:,1:3);

% %    end of second objective


   %%select from the general population elite based on two objectives


    %%end of elite
    Childr=Childr(1:sel1,:);
    Childd=Childd(1:sel11,:);

    Children = [Childd; Childr];
    [crossing1, num]= datasample(Children,50,1, 'Replace', false);
    Children(num,:) = [];
    crossing2= datasample(Children,50,1, 'Replace', false);
    xcr = crossing1;
     crossing1(:,selgene)= crossing2(:,selgene);
    crossing2(:,selgene) = xcr(:,selgene);

    Childdy = [crossing1; crossing2];

% %          Children = Childdy;


% %      [crossing1, num]= datasample(Children,50,1, 'Replace', false);

     L=(Childdy(:, 1));
      w=(Childdy(:, 2));
      t=(Childdy(:, 3));

       Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
   I = (w.*t.*t.*t)/12;
   stiff=(3.*E.*I)./(L.*L.*L);
Fitness1 = (0.68037.*t)./ (L.*L);
```

```
 Childddy = [Childdy Fitness1 Fitness];
  el = zeros(0,5);
for u=1:sel2
[FA, FAA] = datasample(Childddy,1,1, 'Replace', false);

      [FB, FBB] = datasample(Childddy,1,1, 'Replace', false);

      selA = all((FA(:,4)<FB(:,4)),2);
      selB = all((FB(:,4)<FA(:,4)),2);
      selA1 = all((FA(:,5)<FB(:,5)),2);
      selB1 = all((FB(:,5)<FA(:,5)),2);

       sum1 = selA + selA1;
       sum2 = selB + selB1;

       if sum1<2
       ansS=FA;
       elseif
       sum2<2
       ansS=FB;
       else
        ansS=zeros(0,5);
       end


 el=[el;ansS];

end
 ell1=el(1:ell,1:3);%%picking better

 Child11 = datasample(Childdy,elt,1, 'Replace', false);%picking based on each obj
 Child11=Child11(:,1:3);


 Child = [Child11; ell1];
 Chrom = Child;
   L=(Chrom(:, 1));
   w=(Chrom(:, 2));
   t=(Chrom(:, 3));


Fitness =
     (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t)); I =
     (w.*t.*t.*t)/12; stiff=(3.*E.*I)./(L.*L.*L);
Fitness1 = (0.68037.*t)./ (L.*L);
     solChrom = Chrom;
      tes = abs(Fitness - Testvalue);
```

```
        sorti = sortrows(tes);
         solly = [solChrom stiff Fitness1 Fitness tes];
         sollyreal = sortrows(solly,7);

         sollyreal2 = sollyreal(:,6);
          sollyreal3 = sollyreal(:,1:5);
        %now count the number of rows in column 5 that's less than 0.1
        thenumber = sum(tes < 0.1)
         nmm= sollyreal(:,7);
        tes1 = sum(nmm < 0.1);
        numGen = numGen + 1;
       %for i =1:158
       cv = sollyreal2(:,:);
        csv = sollyreal3(:,:);
        bn = size(cv);
      un = repmat(numGen, bn);
       unn = [csv cv un];

       unnn = [unnn; unn];


Gensol(iii) = thenumber;

   end
   end

   unnn=sortrows(unnn,6);
   unnn=unnn(unnn(:,6)<10.02,:);
   unnn=unnn(unnn(:,6)>9.89,:);
solGen = unnn(:,7);
x = unique(solGen);
y=histc(solGen,x);
w1 = unique(unnn, 'rows');
w11 =w1(:,7);
w12= unique(w11);
w2 =histc(w11,w12);
%for unique results overall
[w4,ia] =unique(unnn(:,1:3),'rows');%unique rows based of first 6 columns
uA = unnn(ia,:);
w21 =uA(:,7);
w22= unique(w21);
w3 =histc(w21,w22);

unnnreal1 = unnn(unnn(:,6)> 2,:);
 unnnreal = unnnreal1(unnnreal1(:,6)<10,:);


unnnrun1 = [unnnrun1; unnn];
```

```
end


xx= unnnrun1(:,1);
xy= unnnrun1(:,2);
xz= unnnrun1(:,3);
xzz= unnnrun1(:,6);
xzzz= unnnrun1(:,5);
xzzzz= unnnrun1(:,7);


% % ax1 = subplot(2,1,1);
plot(xzzz,xzz,'r*');
xlabel('Resonance Frequency')
ylabel('Deflection')
```

APPENDIX D: PYTHON RBAM CANTILEVER CODE

```
from numpy import *
import numpy as np
import numpy.matlib

Pop_size=50
Force=50;
E=170;
ep = 0.30*100;
ep = 30
rap= ep/3;
rap=10;
Testvalue = 10.0;
hh=15;
hhh=5
;
F2 =
0;

unn = np.zeros((0, 7))
unnn = np.zeros((0,7));
cs=2;
L = np.random.uniform(low=10.0, high=100.0, size=(500,));##for population size of 500
w = np.random.uniform(low=5.0, high=50.0, size=(500,));
t = np.random.uniform(low=0.5, high=20.0, size=(500,));
Chrom=np.vstack((L, w, t)).T
L=Chrom[:,0]##assign first column to L
w=Chrom[:,1]##ASSIGN SECOND COLUMN TO W
t=Chrom[:,2]
Fitness_function=(4*Force)*(L*L*L)/(E*(w*t*t*t));
Popfit=np.vstack((L, w, t,Fitness_function)).T
numGen=0
stopFlag = 0
##fit=argsort(Popfit[:,3]);
##Pop=Popfit[fit,:];##new pop in order of fitness
while stopFlag == 0:
   for iii in range(20):
      FF = np.absolute (Fitness_function - Testvalue);
      F= np.column_stack((Chrom,FF))
      fit=argsort(Popfit[:,3]);
      p = Popfit[fit,:];##sorted rows in order of fitness
      h = p[0:100, 0:3];
##first 100
      qw = h[0:ep, 0:3];
##elite
      qw1=np.random.choice(qw.shape[0], ep, replace=False)
      z=qw[qw1,:]
      h = np.delete(h, qw1, 0)

      avg = np.zeros((0,3));
      for iv in range(hh):
      ##for a in range(10): or for a in range(0, 10):
         ab = Chrom[np.random.choice(Chrom.shape[0], hhh, replace=False), :]
         ac = ab.mean(axis=0)
         avg = np.vstack((avg,ac))

      favg=avg;

      a = z[np.random.choice(z.shape[0], ep, replace=False), :]
      b = np.random.choice(a.shape[0], rap, replace=False)
```

```
s=a[b,:]
b2 = np.random.choice(a.shape[1], cs, replace=False)
a = np.delete(a, b, 0)
bb = np.random.choice(a.shape[0], rap, replace=False)
ss=a[bb,:]
bb2 = np.random.choice(a.shape[1], cs, replace=False)
a = np.delete(a, bb, 0)
bb3 = np.random.choice(a.shape[1], cs, replace=False)

sss = s;
sa=ss;
aaa=a;

l= np.random.choice(h.shape[0], ep, replace=False)
aa=h[l,:]
h = np.delete(h, l, 0)##delete the 30 unique ones from h
mm = h ## save the rest of values in h in mm
q = np.random.choice(aa.shape[0], rap, replace=False);
uu=aa[q,:]
aa = np.delete(aa, q, 0)
qq = np.random.choice(aa.shape[0], rap, replace=False);
uuk=aa[qq,:]
a = np.delete(aa, qq, 0)
uuu = uu
saa=uuk
aak = aa[np.random.choice(aa.shape[0], rap, replace=False), :]
aakk=aak
uu[:,b2]=s[:, b2]
sss[:,b2]=uuu[:,b2]
uuk[:,bb2]=ss[:,bb2]
sa[:,bb2]=saa[:,bb2]
aak[:,bb3]=a[:,bb3]
aaa[:,bb3]=aakk[:,bb3]


childy = np.vstack((uu, sss, uuk, sa, aak))#stacking rows on top of each other
childy2 =childy[np.random.choice(childy.shape[0], ep, replace=False), :]
Child1 = np.vstack((childy2, z, mm, favg));



v1 = np.random.uniform(low=10.0, high=100.0, size=(100,));
v2 = np.random.uniform(low=5.0, high=50.0, size=(100,));
v3 = np.random.uniform(low=0.5, high=20.0, size=(100,));
Chrommutate=np.vstack((v1, v2, v3)).T
G= np.random.choice(Chrommutate.shape[0], 1, replace=False);
D=Chrommutate[[G],:]
CC= np.random.choice(D.shape[1], 1, replace=False);
HH=D[[CC],:]
Child = Child1
Chrom = Child
L=Chrom[:,0]##assign first column to L
w=Chrom[:,1]##ASSIGN SECOND COLUMN TO W
t=Chrom[:,2]
##Fitness = (Force)*(L*L*L)/(2*E*(w*t*t*t));
Fitness_function = (4*Force)*(L*L*L)/(E*(w*t*t*t));
I = (w*t*t*t)/12;
stiff=(3*E*I)/(L*L*L);
##Resfreq = sqrt((3*E*I)/(M*L*L*L));
Fitness1 = (0.68037*t)/ (L*L);
solChrom = Chrom;
```

```
    tes = np.absolute (Fitness_function - Testvalue);
    sorti = np.sort(tes);
    solly = np.column_stack((solChrom,stiff,Fitness1,Fitness_function,tes))

    nmf1=argsort(solly[:,6]);
    sollyreall=solly[nmf1,:]
    sollyreal=sollyreall[0:100,:]
    sollyreal2 = sollyreal[:,5];
    sollyreal3 = sollyreal[:,0:5];
    thenumber = sum(tes < 0.1)## or (tes> 0.1).sum()
    nmm= sollyreal[:,6];
##    nmf=argsort(sollyreal[:,6]);
####    nmm=sollyreal[nmf,:]
####    a.sum(axis=0)
    bj=size(sollyreal2)
    forsolly=np.zeros((bj,0));
    sollyreal2=np.column_stack((forsolly,sollyreal2))
    tes1 = sum(nmm < 0.1)
    numGen = numGen + 1;
    cv = sollyreal2[0:tes1,:];
    csv = sollyreal3[0:tes1,:];
    bn = len(cv)
    un = np.matlib.repmat(numGen, bn,1);
    unn = np.column_stack ((csv, cv, un))
    unnn = np.vstack((unnn, unn));
    continue
```

APPENDIX F: RBAM COMB DRIVE

```
Force = 50;
E=170;


   Nind = 500; %population of 30 individual individuals
   FieldDR = [10 2.0 0.5; 1000.0 50.0 20.0]; %Genes = 3; genes upper and lower bounds
   ep = 0.30*100;
   rap= ep./3;
   Testvalue = 10.0; %termination condition
   Testvalue2 = 50.0;
   hh=5;
   hhh=5;
   F2 = 0;
   Fitness= 0;
   unn = zeros(0,7);
   unnn = zeros(0,7);


% %
% %
   Chrom = crtrp(Nind, FieldDR);


  %evaluate the parent population P(t)
  L=(Chrom(:, 1));
  w=(Chrom(:, 2));
  t=(Chrom(:, 3));


% Fitness = Force.*(L.*L.*L)./((2*E).*(w.*t.*t.*t));
   Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
   I = (w.*t.*t.*t)/12;
   stiff=(3.*E.*I)./(L.*L.*L);
%     Resfreq = sqrt((3.*E.*I)./(M.*L.*L.*L));
%       Resfreq = sqrt((3.*E.*I)./(M.*L.*L.*L));
      Fitness1 = (0.07586.*t)./ (L.*L);
  %while it doesn't meet termination condition which is testvalue
  stopFlag = 0;
  numGen=0;
  while stopFlag == 0
   for iii = 1:500
      FF = abs(Fitness - Testvalue);

      F = [Chrom FF];
      p =
      sortrows(F,4);
       h = p(1:100, 1:3);
       qw = h(1:ep, 1:3);
       [z, qw1] = datasample(qw,ep,1, 'Replace', false);
        h(qw1,:) = [];
% %       if iii>1<5
% %           zh = unique(h, 'rows');
% %           z=zh(1:30,1:3);
% %
% %       end
      avg = zeros(0,1);
     for iv = 1:hh
```

```
      ab = datasample(Chrom,hhh,1, 'Replace', false);

      ac = mean(ab);

       avg = [avg; ac];
  end
favg=avg;



a = datasample(z,ep,1, 'Replace', false);
      [s b] = datasample(a,rap,1, 'Replace', false);
[s1 b2] = datasample(a,1,2, 'Replace', false);
a(b,:) = [];
[ss bb] = datasample(a,rap,1, 'Replace', false);
[ss1 bb2] = datasample(a,1,2, 'Replace', false);
a(bb,:) = [];
[ss2 bb3] = datasample(a,1,2, 'Replace', false);

sss = s;
sa=ss;
aaa=a;



[aa l] = datasample(h,ep,1, 'Replace', false);% select 30 unique from the 100
h(l,:) = [];%delete the 30 unique ones from h
mm = h; % save the rest of values in h in mm


[uu q] = datasample(aa,rap,1, 'Replace', false);
aa(q,:) = [];
[uuk qq] = datasample(aa,rap,1, 'Replace', false);
aa(bb,:) = [];
uuu = uu;
saa=uuk;
aak=datasample(aa,rap,1, 'Replace', false);
aakk=aak;


uu(:,b2)=s(:, b2);
sss(:,b2)=uuu(:,b2);

uuk(:,bb2)=ss(:,bb2);
sa(:,bb2)=saa(:,bb2);

aak(:,bb3)=a(:,bb3);
aaa(:,bb3)=aakk(:,bb3);


   childy = [uu; sss; uuk; sa; aak];
   childy2 = datasample(childy,ep,1, 'Replace', false);
   Child1 = [childy2; z; mm; favg];

 %mutate

 Nind = 100; %population of 30 individual individuals
FieldDR = [10 2.0 0.5; 1000.0 50.0 20.0]; %Genes = 3; genes upper and lower bounds
 Chrommutate = crtrp(Nind, FieldDR);
[D G] = datasample(Chrommutate,10,1, 'Replace', false);
```

```
    [HH CC] = datasample(D,1,2, 'Replace', false);
    Child1(G,CC) = HH;
      %after mutate
      Child = Child1;
    %mutation didn't seem to have a problem on the space; probably doesnt
    %get stuck in local optimum



      Chrom = Child;
      L=(Chrom(:, 1));
      w=(Chrom(:, 2));
      t=(Chrom(:, 3));

      Fitness = (4*Force).*(L.*L.*L)./(E.*(w.*t.*t.*t));
      I = (w.*t.*t.*t)/12;
       stiff=(3.*E.*I)./(L.*L.*L);
      Resfreq = sqrt((3.*E.*I)./(M.*L.*L.*L));
       Fitness1 = (0.07586.*t)./ (L.*L);
       solChrom = Chrom;
        tes = abs(Fitness - Testvalue);
        sorti = sortrows(tes);
         solly = [solChrom stiff Fitness1 Fitness tes];
         sollyreal = sortrows(solly,7);
         sollyreal(101:end,:)=[];
         sollyreal2 = sollyreal(:,6);
          sollyreal3 = sollyreal(:,1:5);
        %now count the number of rows in column 5 that's less than 0.1
        thenumber = sum(tes < 0.1)
         nmm= sollyreal(:,7);
        tes1 = sum(nmm < 0.1);
        numGen = numGen + 1;
       %for i =1:158
       cv = sollyreal2(1: tes1 ,:);
        csv = sollyreal3(1: tes1 ,:);
        bn = size(cv);
      un = repmat(numGen, bn);
       unn = [csv cv un];

       unnn = [unnn; unn];


         stopFlag = 1; Gensol(iii) = thenumber;

   end
   end
solGen = unnn(:,7);
x = unique(solGen);
y=histc(solGen,x);
w1 = unique(unnn, 'rows');
w11 =w1(:,7);
w12=   unique(w11);
w2 =histc(w11,w12);
%for unique results overall
[w4,ia] =unique(unnn(:,1:3),'rows');%unique rows based of first 6 columns uA = unnn(ia,:);
w21 =uA(:,7);
w22= unique(w21);
w3 =histc(w21,w22);

xx= unnn(:,1);
xy= unnn(:,2);
```

```
xz= unnn(:,3);
xzz= unnn(:,6);
plot(solGen,xzz,'r*');
```

APPENDIX E: MATLAB MORBAM COMB DRIVE CODE

```
c = 8.85;
%n=4;
%g=7;
E=170;%young's modulus
den= 2.328;
Nind = 500;
r = randi([2 20],Nind,1);
%spring length, width, thickness,truss length, width
FieldDR = [10.0 2.0 0.5 5.0 2.0 5.0 2.0; 1000.0 50.0 20.0 40.0 50.0 20.0 18.0];
cr1=3;
cr2=4;
 ep = 0.30*100;
 rap= ep./3;
 hh= 15;%%setting hh to 25 works fine
 hhh=5;
Testvalue = 10.0;
Fitness = 0;
Force = 0;
unn = zeros(0,12);
unnn = zeros(0,12);
cs=4;
Chrompart1 = crtrp(Nind, FieldDR);
Chrom = [Chrompart1 r];
lb=(Chrom(:, 1));
wb=(Chrom(:, 2));
t=(Chrom(:, 3));
lt=(Chrom(:, 4));
wt=(Chrom(:, 5));
io=(Chrom(:, 6));
g=(Chrom(:, 7));
n=(Chrom(:, 8));

alpha = (wt.*wt.*wt)./(wb.*wb.*wb);
ft= 8.*lt.*lt;
sd = 8.*alpha.*lt.*lb;
td = alpha.*alpha.*lb.*lb;
fut= 4.*lt.*lt;
fif = 10.*alpha.*lt.*lb;
st = 5.*alpha.*alpha.*lb.*lb;
kflex1 = (2.*E.*(t.*wt.*wt.*wt))./(lt.*lt.*lt);
kflex2 = (ft + sd + td) ./ fut + fif+ st;
kflex = kflex1 .* kflex2;
kbcomp = (2.*E.*(t.*wb))./(lb);
keff = (kflex .* kbcomp)./(kflex + kbcomp);
V = 50;
```

```
Force = (n.*c.*t.*V.*V)./g;
Fitness = (n.*c.*t.*V.*V)./(keff.*g);
y = io + Fitness;
Capacitance = (2.*n.*c.*t.*y)./g;
genecount= 8;
 genecount1 = 1:8;

    selgene = datasample(genecount1,4,2, 'Replace', false);
sel1=50;
    sel11=50;
   sel2=100;
   sel22=60;
   ell=90;
   elt=10;
   %while it doesn't meet termination condition which is testvalue
   stopFlag = 0;
   numGen=0;
   while stopFlag == 0
    for iii = 1:1000
       FF = abs(Fitness - Testvalue);
       FF1 =[Chrom Force Capacitance Fitness];



       F = [Chrom FF];
       p =
       sortrows(F,9);

       qw = p(1:ep, 1:genecount);
       h = p(1:100, 1:genecount);
       [z, qw1] = datasample(qw,ep,1, 'Replace', false);
         h(qw1,:) = [];

      avg = zeros(0,1);
       for iv = 1:hh

           ab = datasample(Chrom,hhh,1, 'Replace', false);

           ac = mean(ab);

            avg = [avg; ac];
        end
      favg=avg;

      a = datasample(z,30,1, 'Replace', false);
%       [s i ] = datasample(m,2,1,  'Replace', false)
      [s b] = datasample(a,rap,1, 'Replace', false);
      [s1 b2] = datasample(a,cs,2, 'Replace', false);
```

```
a(b,:) = [];
[ss bb] = datasample(a,rap,1, 'Replace', false);
[ss1 bb2] = datasample(a,cs,2, 'Replace', false);
a(bb,:) = [];
[ss2 bb3] = datasample(a,cs,2, 'Replace', false);

sss = s;
sa=ss;
aaa=a;



[aa l] = datasample(h,ep,1, 'Replace', false);% select 30 unique from the 100
h(l,:) = [];%delete the 30 unique ones from h
mm = h; % save the rest of values in h in mm


[uu q] = datasample(aa,rap,1, 'Replace', false);
aa(q,:) = [];
[uuk qq] = datasample(aa,rap,1, 'Replace', false);
aa(bb,:) = [];
uuu = uu;
saa=uuk;
aak=datasample(aa,rap,1, 'Replace', false);
aakk=aak;


uu(:,b2)=s(:, b2);
sss(:,b2)=uuu(:,b2);

uuk(:,bb2)=ss(:,bb2);
sa(:,bb2)=saa(:,bb2);

aak(:,bb3)=a(:,bb3);
aaa(:,bb3)=aakk(:,bb3);



   childy = [uu; sss; uuk; sa; aak];
   childy2 = datasample(childy,ep,1, 'Replace', false);
   Childd = [childy2; z; mm; favg];
%      Child = [z ; a; j; kk; m; uk];
  %mutate

  Nind = 100; %population of 30 individual individuals
 r = randi([2 20],Nind,1); %with really little problems,there can be
FieldDR = [10.0 2.0 0.5 5.0 2.0 5.0 2.0; 1000.0 50.0 20.0 40.0 50.0 20.0 18.0];
 Chrommutate1 = crtrp(Nind, FieldDR);
```

```
   Chrommutate = [Chrommutate1 r];
   %randomly pick which 10 row to mutate
   [D G] = datasample(Chrommutate,1,1, 'Replace', false);
%select 3 columns from D
   [HH CC] = datasample(D,1,2, 'Replace', false);
   Childd(G,CC) = HH;
%     end of 1st
obj lb=(Childd(:,
1));
wb=(Childd(:, 2));
t=(Childd(:, 3));
lt=(Childd(:, 4));
wt=(Childd(:, 5));
io=(Childd(:, 6));
g=(Childd(:, 7));
n=(Childd(:, 8));



 alpha = (wt.*wt.*wt)./(wb.*wb.*wb);
ft= 8.*lt.*lt;
sd = 8.*alpha.*lt.*lb;
td = alpha.*alpha.*lb.*lb;
fut= 4.*lt.*lt;
fif = 10.*alpha.*lt.*lb;
st = 5.*alpha.*alpha.*lb.*lb;
kflex1 = (2.*E.*(t.*wt.*wt.*wt))./(lt.*lt.*lt);
kflex2 = (ft + sd + td) ./ fut + fif+ st;
kflex = kflex1 .* kflex2;
kbcomp = (2.*E.*(t.*wb))./(lb);
keff = (kflex .* kbcomp)./(kflex + kbcomp);
V = 50;
Force = (n.*c.*t.*V.*V)./g;
Fitness = (n.*c.*t.*V.*V)./(keff.*g);
y = io + Fitness;

eva1 = abs(Fitness - Testvalue);

 Fit1 = [Childd eva1];
    Childd = sortrows(Fit1,9);
    Childd=Childd(:,1:8);
%real end of first obj


  %second objective
     pf = sortrows(FF1,-9);

      qwf = pf(1:ep, 1:genecount);
```

```
 hf = pf(1:100, 1:genecount);
 [zf, qw1f] = datasample(qwf,ep,1, 'Replace', false);
   hf(qw1f,:) = [];


af = datasample(zf,30,1, 'Replace', false);

%      [s i ] = datasample(m,2,1,  'Replace', false)
[s b] = datasample(af,rap,1, 'Replace', false);
[s1 b2] = datasample(af,cs,2, 'Replace',
false); af(b,:) = [];
[ss bb] = datasample(af,rap,1, 'Replace', false);
[ss1 bb2] = datasample(af,cs,2, 'Replace', false);
af(bb,:) = [];
[ss2 bb3] = datasample(af,cs,2, 'Replace', false);

sss = s;
sa=ss;
aaa=af;



[aa l] = datasample(hf,ep,1, 'Replace', false);
hf(l,:) = [];
mm = hf;


[uu q] = datasample(aa,rap,1, 'Replace', false);
aa(q,:) = [];
[uuk qq] = datasample(aa,rap,1, 'Replace', false);
aa(bb,:) = [];
uuu = uu;
saa=uuk;
aak=datasample(aa,rap,1, 'Replace', false);
aakk=aak;


uu(:,b2)=s(:, b2);
sss(:,b2)=uuu(:,b2);
uuk(:,bb2)=ss(:,bb2);
sa(:,bb2)=saa(:,bb2);

aak(:,bb3)=a(:,bb3);
aaa(:,bb3)=aakk(:,bb3);



  childyf = [uu; sss; uuk; sa; aak];
```

```
        childy2f = datasample(childyf,ep,1, 'Replace', false);
        Child1f = [childy2f; zf; mm; favg];
%          Child = [z ; a; j; kk; m; uk];
      %mutate

     Nind = 100; %population of 30 individual individuals
   r = randi([2 20],Nind,1); %with really little problems,there can be
FieldDR = [10.0 2.0 0.5 5.0 2.0 5.0 2.0; 1000.0 50.0 20.0 40.0 50.0 20.0 18.0];
   Chrommutate1 = crtrp(Nind, FieldDR);
   Chrommutate = [Chrommutate1 r];
   %randomly pick which 10 row to mutate
   [D G] = datasample(Chrommutate,1,1, 'Replace', false);
%select 3 columns from D
   [HH CC] = datasample(D,1,2, 'Replace', false);
   Child1f(G,CC) = HH;
   Childf= Child1f;




     lb=(Childf(:, 1));
wb=(Childf(:, 2));
t=(Childf(:, 3));
lt=(Childf(:, 4));
wt=(Childf(:, 5));
io=(Childf(:, 6));
g=(Childf(:, 7));
n=(Childf(:, 8));

   alpha = (wt.*wt.*wt)./(wb.*wb.*wb);
   ft= 8.*lt.*lt;
   sd = 8.*alpha.*lt.*lb;
   td = alpha.*alpha.*lb.*lb;
   fut= 4.*lt.*lt;
   fif = 10.*alpha.*lt.*lb;
   st = 5.*alpha.*alpha.*lb.*lb;
   kflex1 = (2.*E.*(t.*wt.*wt.*wt))./(lt.*lt.*lt);
   kflex2 = (ft + sd + td) ./ fut + fif+ st;
   kflex = kflex1 .* kflex2;
   kbcomp = (2.*E.*(t.*wb))./(lb);
   keff = (kflex .* kbcomp)./(kflex + kbcomp);
   V = 50;
   Force = (n.*c.*t.*V.*V)./g;
   Fitness = (n.*c.*t.*V.*V)./(keff.*g);
   y = io + Fitness;


 Fit1 = [Childf Force];
```

```
    Childf = sortrows(Fit1,-9);
    Childf=Childf(:,1:8);

%end of second objective



   %third objective
    pc = sortrows(FF1,-10);

     qwc = pc(1:ep, 1:genecount);
     hc = pc(1:100, 1:genecount);
     [zc, qw1c] = datasample(qwc,ep,1, 'Replace', false);
       hc(qw1c,:) = [];


    ac = datasample(zc,30,1, 'Replace', false);

%       [s i ] = datasample(m,2,1,  'Replace', false)
    [s b] = datasample(ac,rap,1, 'Replace', false);
    [s1 b2] = datasample(ac,cs,2, 'Replace',
    false); ac(b,:) = [];
    [ss bb] = datasample(ac,rap,1, 'Replace', false);
    [ss1 bb2] = datasample(ac,cs,2, 'Replace', false);
    ac(bb,:) = [];
    [ss2 bb3] = datasample(ac,cs,2, 'Replace', false);

    sss = s;
    sa=ss;
    aaa=ac;



    [aa l] = datasample(hc,ep,1, 'Replace', false);
    hc(l,:) = [];
    mm = hc;


    [uu q] = datasample(aa,rap,1, 'Replace', false);
    aa(q,:) = [];
    [uuk qq] = datasample(aa,rap,1, 'Replace', false);
    aa(bb,:) = [];
    uuu = uu;
    saa=uuk;
    aak=datasample(aa,rap,1, 'Replace', false);
    aakk=aak;
```

```
    uu(:,b2)=s(:, b2);
    sss(:,b2)=uuu(:,b2);

    uuk(:,bb2)=ss(:,bb2);
    sa(:,bb2)=saa(:,bb2);

    aak(:,bb3)=a(:,bb3);
    aaa(:,bb3)=aakk(:,bb3);



      childyc = [uu; sss; uuk; sa; aak];
      childy2c = datasample(childyc,ep,1, 'Replace', false);
      Child1c = [childy2c; zc; mm; favg];
%         Child = [z ; a; j; kk; m; uk];
     %mutate

    Nind = 100; %population of 30 individual individuals
  r = randi([2 20],Nind,1); %with really little problems,there can be
FieldDR = [10.0 2.0 0.5 5.0 2.0 5.0 2.0; 1000.0 50.0 20.0 40.0 50.0 20.0 18.0];
  Chrommutate1 = crtrp(Nind, FieldDR);
  Chrommutate = [Chrommutate1 r];
  %randomly pick which 10 row to mutate
  [D G] = datasample(Chrommutate,1,1, 'Replace', false);
%select 3 columns from D
  [HH CC] = datasample(D,1,2, 'Replace', false);
  Child1c(G,CC) = HH;
  Childc = Child1c;

  lb=(Childc(:, 1));
wb=(Childc(:, 2));
t=(Childc(:, 3));
lt=(Childc(:, 4));
wt=(Childc(:, 5));
io=(Childc(:, 6));
g=(Childc(:, 7));
n=(Childc(:, 8));

alpha = (wt.*wt.*wt)./(wb.*wb.*wb);
ft= 8.*lt.*lt;
sd = 8.*alpha.*lt.*lb;
td = alpha.*alpha.*lb.*lb;
fut= 4.*lt.*lt;
fif = 10.*alpha.*lt.*lb;
st = 5.*alpha.*alpha.*lb.*lb;
kflex1 = (2.*E.*(t.*wt.*wt.*wt))./(lt.*lt.*lt);
kflex2 = (ft + sd + td) ./ fut + fif+ st;
kflex = kflex1 .* kflex2;
```

```
kbcomp = (2.*E.*(t.*wb))./(lb);
keff = (kflex .* kbcomp)./(kflex + kbcomp);
V = 50;
Fitness = (n.*c.*t.*V.*V)./(keff.*g);
y = io + Fitness;
Capacitance = (2.*n.*c.*t.*y)./g;

 Fit2 = [Childc Capacitance];
    Childc = sortrows(Fit2,-9);
    Childc=Childc(:,1:8);

  %end of third obj

 Childd=Childd(1:33,:);
    Childf=Childf(1:34,:);
    Childc=Childc(1:33,:);

    Children = [Childd; Childf; Childc;];
    [crossing1, num]= datasample(Children,50,1, 'Replace', false);
    Children(num,:) = [];
    crossing2= datasample(Children,50,1, 'Replace', false);
    xcr = crossing1;
     crossing1(:,selgene)= crossing2(:,selgene);
    crossing2(:,selgene) = xcr(:,selgene);

    Childdy = [crossing1; crossing2];

    Children = Childdy;
    [crossing1, num]= datasample(Children,50,1, 'Replace', false);
    Children(num,:) = [];
    crossing2= datasample(Children,50,1, 'Replace', false);
    xcr = crossing1;
     crossing1(:,selgene)= crossing2(:,selgene);
    crossing2(:,selgene) = xcr(:,selgene);
    Childdy = [crossing1; crossing2];

     lb=(Childdy(:, 1));
wb=(Childdy(:, 2));
t=(Childdy(:, 3));
lt=(Childdy(:, 4));
wt=(Childdy(:, 5));
io=(Childdy(:, 6));
g=(Childdy(:, 7));
n=(Childdy(:, 8));



alpha = (wt.*wt.*wt)./(wb.*wb.*wb);
```

```
    ft= 8.*lt.*lt;
    sd = 8.*alpha.*lt.*lb;
    td = alpha.*alpha.*lb.*lb;
    fut= 4.*lt.*lt;
    fif = 10.*alpha.*lt.*lb;
    st = 5.*alpha.*alpha.*lb.*lb;
    kflex1 = (2.*E.*(t.*wt.*wt.*wt))./(lt.*lt.*lt);
    kflex2 = (ft + sd + td) ./ fut + fif+ st;
    kflex = kflex1 .* kflex2;
    kbcomp = (2.*E.*(t.*wb))./(lb);
    keff = (kflex .* kbcomp)./(kflex + kbcomp);
    V = 50;
    Force = (n.*c.*t.*V.*V)./g;
    Fitness = (n.*c.*t.*V.*V)./(keff.*g);
    y = io + Fitness;
 Capacitance = (2.*n.*c.*t.*y)./g;
fe= abs(Fitness - Testvalue);
Childddy = [Childdy Force Capacitance Fitness];
      el = zeros(0,11);
    for u=1:sel2
    [FA, FAA] = datasample(Childddy,1,1, 'Replace', false);

          [FB, FBB] = datasample(Childddy,1,1, 'Replace', false);

          selA = all((FA(:,9)>FB(:,9)),2);
          selB = all((FB(:,9)>FA(:,9)),2);
          selA1 = all((FA(:,10)>FB(:,10)),2);
          selB1 = all((FB(:,10)>FA(:,10)),2);
          selA2 = all((FA(:,11)>FB(:,11)),2);
          selB2 = all((FB(:,11)>FA(:,11)),2);

           sum1 = selA + selA1 + selA2;
           sum2 = selB + selB1 + selB2;

           if sum1<3
           ansS=FA;
           elseif
           sum2<3
           ansS=FB;
           else
            ansS=zeros(0,11);
           end


     el=[el;ansS];

    end
     ell1=el(1:ell,1:8);
```

```
    Child11 = datasample(Childdy,elt,1, 'Replace', false);
    Child11=Child11(:,1:8);


    Child = [Child11; ell1];

      Chrom = Child;


lb=(Chrom(:, 1));
wb=(Chrom(:, 2));
t=(Chrom(:, 3));
lt=(Chrom(:, 4));
wt=(Chrom(:, 5));
io=(Chrom(:, 6));
g=(Chrom(:, 7));
n=(Chrom(:, 8));

alpha = (wt.*wt.*wt)./(wb.*wb.*wb);
ft= 8.*lt.*lt;
sd = 8.*alpha.*lt.*lb;
td = alpha.*alpha.*lb.*lb;
fut= 4.*lt.*lt;
fif = 10.*alpha.*lt.*lb;
st = 5.*alpha.*alpha.*lb.*lb;
kflex1 = (2.*E.*(t.*wt.*wt.*wt))./(lt.*lt.*lt);
kflex2 = (ft + sd + td) ./ fut + fif+ st;
kflex = kflex1 .* kflex2;
kbcomp = (2.*E.*(t.*wb))./(lb);
keff = (kflex .* kbcomp)./(kflex + kbcomp);
V = 50;
Force = (n.*c.*t.*V.*V)./g;
Fitness = (n.*c.*t.*V.*V)./(keff.*g);
y = io + Fitness;
Capacitance = (2.*n.*c.*t.*y)./g;

      solChrom = Chrom;
       tes = abs(Fitness - Testvalue);

       sorti = sortrows(tes);
        solly = [solChrom Force Capacitance Fitness tes];
        sollyreal = sortrows(solly,12);
        sollyreal(101:end,:)=[];
        sollyreal2 = sollyreal(:,11);
         sollyreal3 = sollyreal(:,1:10);
       %now count the number of rows in column 5 that's less than 0.1
       thenumber = sum(tes < 0.1)
```

```
        nmm= sollyreal(:,12);
       tes1 = sum(nmm < 0.1);
       numGen = numGen + 1;

      %for i =1:158
      cv = sollyreal2(:,:);
       csv = sollyreal3(:,:);
       bn = size(cv);
     un = repmat(numGen, bn);
      unn = [csv cv un];

      unnn = [unnn; unn];
      stopFlag = 1;

  Gensol(iii) = thenumber;

  end




  end
  solGen = unnn(:,12);
x = unique(solGen);
y=histc(solGen,x);
w1 = unique(unnn, 'rows');
w11 =w1(:,12);
w12= unique(w11);
w2 =histc(w11,w12);
%for unique results overall
[w4,ia] =unique(unnn(:,1:3),'rows');%unique rows based of first 6 columns
uA = unnn(ia,:);
w21 =uA(:,12);
w22= unique(w21);
w3 =histc(w21,w22);

xx= unnn(:,1);
xx3= unnn(:,2);
xx4= unnn(:,3);
xx5= unnn(:,4);
xx1= unnn(:,7);
xx2= unnn(:,8);
xy= unnn(:,9);
xz= unnn(:,10);
xzz= unnn(:,11);
% plot(solGen,xzz,'r*');
ax1 = subplot(3,3,1);
plot(solGen,xy,'b*');
```

```
xlabel(ax1,'Generation')
ylabel(ax1,'Force')


ax2 = subplot(3,3,2);
plot(solGen,xz,'r*');
xlabel(ax2,'Generation')
ylabel(ax2,'Capacitance')


ax3 = subplot(3,3,3);

plot(solGen,xzz,'r*');
% % xlabel(ax1,'Generation')
% % ylabel(ax1,'Deflection')
xlabel(ax3,'Generation')
ylabel(ax3,'Deflection')


ax4 = subplot(3,3,4);

plot(solGen,xx,'r*');
xlabel(ax4,'Generation')
ylabel(ax4,'Length')

ax5 = subplot(3,3,5);

plot(solGen,xx1,'r*');
xlabel(ax5,'Generation')
ylabel(ax5,'G')


ax6 = subplot(3,3,6);

plot(solGen,xx2,'r*');
xlabel(ax6,'Generation')
ylabel(ax6,'N')


ax7 = subplot(3,3,7);

plot(solGen,xx3,'r*');
xlabel(ax7,'Generation')
ylabel(ax7,'Beam Width')


ax8 = subplot(3,3,8);
```

```
plot(solGen,xx4,'r*');
xlabel(ax8,'Generation')
ylabel(ax8,'Acuator Thickness')


% % ax9 = subplot(3,3,9);
% %
% % plot(solGen,xx5,'r*');
% % xlabel(ax9,'Generation')
% % ylabel(ax9,'Truss Length')



ax9 = subplot(3,3,9);

plot(xzz,xz,'r*');
xlabel(ax9,'Deflection')
ylabel(ax9,'Capacitance')
```