Journal of the Nigerian Association of Mathematical Physics
Volume **11 (November 2007), 213 - 222**

A Genetic algorithm for evaluating the zeros (roots) of polynomial functions, optimizing and solving *n*-dimensional systems of equations

[1]J. A. Akpobi and [2]E. D. Akpobi
[1]Production Engineering Department, University of Benin, Nigeria.
[2]Petroleum Engineering Department, University of Benin, Nigeria.
[1]e-mail: alwaysjohnie@yahoo.com

Abstract

This paper presents a Genetic Algorithm software (which is a computational, search technique) for finding the zeros (roots) of any given polynomial function, and optimizing and solving N-dimensional systems of equations. The software is particularly useful since most of the classic schemes are not all embracing. For example; Newton-Raphson Scheme can only solve the zeros (roots) of polynomial, while and Gauss-Jordan scheme can only solve set of linear simultaneous equations. This characteristics of classical schemes, thus pose a limitation to the scope of problems they can they can used to solve. This limitation is effectively and accurately easily resolved using the genetic algorithm programme. It is demonstrated using a number of examples . Thus it solves a wider class of optimization problems, and also solves for the zeros or roots of polynomial. The program was designed and implemented using Microsoft Visual Basic object oriented programming Language.

Keywords: Genetic Algorithm, Optimization, Polynomials Functions, Roots or zeros, *N*-dimensional equations

1.0 Introduction

Genetic Algorithm (GA) is a probabilistic optimization method which is based on the principles of evolution. It was first discovered and introduced in 1975 by John Holland and his colleagues at the University of Michigan, (Goldberg, 1989) [5]. Evidently, John Holland is referred to as the father of genetic algorithms, but the basis of this optimization method on can be traced back to the father of evolution, Charles Darwin (Darwin, 1859) [3].

Genetic algorithms are typically implemented as a computer simulation in which a population of abstract representations called chromosomes of solutions (called individuals) to an optimization problem evolves towards better solutions. Practically, solutions are represented in binary as strings of 0s and 1s. Each of these binary strings represents a gene. Each string (chromosome) has its own fitness measure that reflects how well a creature can survive in its surrounding environment.

Since the inception of this computational, probabilistic search method, genetic-algorithms has been applied to so many optimization problems in various fields and professions with successful results. For example, Parks et al (2001) [4] used genetic algorithms to investigate the efficacy of various elitist selection strategies in a multi-objective genetic algorithm implementation, with parents being selected both from the current population and the archival record of non dominated solutions encountered during search. The work showed that it was possible to improve the search performance of the algorithm through the use of strongly elitist selection strategies.

Coit et al (1995) [2] researched on the inefficiencies of the application of GA when constrained optimization problems have feasible solutions that are difficult to find or achieve. Their method resulted in the adaptive penalty technique, which makes use of feedback obtained during the search along with a dynamic distance matrix.

Two main methods exist in resolving the difficulties of handling constraints in genetic algorithms. One is to allow only feasible solutions in the population, and the other is to apply a penalty to those solutions that violate constraints but neither of these two methods has proved to perform satisfactorily in general. Another researcher, Carlson and Shonkwiler (1995) [1] studied a class of variable fitness genetic algorithms as a technique for use on constrained optimization problems. This method has been successfully applied to a problem of engineering interest: the ground water treatment problem for unconfined aquifers.

Genetic algorithms has been used in so many applications but so far from the research that has been carried out to the best of the authors knowledge, no work has been done on how to use genetic algorithm for finding the roots of polynomial functions, optimizing and generate optimal (and also solve) solutions for set of N-dimensional linear and nonlinear systems. This work addresses this problem. The superiority of this method over the other classic methods is illustrated with a number of examples.

Genetic algorithm provides solutions by generating a set of chromosomes referred to as a generation. The new generation of strings is gotten through three major genetic operations or processes, namely:

(i) Selection
(ii) Crossover
(iii) Mutation

Selection: There are different methods of selection, namely:

(1) Roulette wheel method
(2) Rank selection
(3) Steady-State selection
(4) Elitist selection
(5) Fitness-proportionate selection
(6) Scaling selection
(7) Tournament selection
(8) Generational selection
(9) Hierarchical selection

Crossover: There are different methods of crossover namely:

(1) One-point crossover
(2) Two-point crossover
(3) Segmented crossover

The one-point crossover is when crossover or exchange of genes occurs at one point. It is a simple and often used method. For example

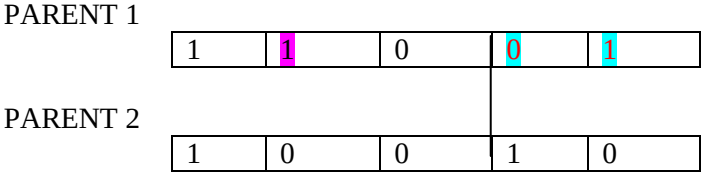

Figure 1: Parents and Cross over segment.

After crossover at the point, the new generations created are:

OFFSPRING 1

1	1	0	1	0

OFFSPRING 2

1	0	0	0	1

Figure 2: Offsprings created after crossover

Mutation: This is the chance that a bit within a chromosome will be flipped (0 becomes 1, 1 becomes 0). It must be noted that erroneous reproduction or deformation of genes can occur. In genetic algorithms when these errors or deformations occur mutation is said to have taken place.

2.0 Software design

The Genetic algorithm software developed and reported in this work, provides solution to problems of: optimization, roots (zeros) and system's equations. It was developed, using Visual Basic (VB) 6.0 as the programming language. Visual Basic 6.0 is an object oriented programming language. This means that it makes use of objects which are a combination of data and codes. Data represents the relevant information that is used to programme the systems dynamics, while the codes represent the series of programming instructions written to process the systems data needed.

2.1 Programme description

The programme works in the following manner:

Step 1: When the program is initialized, a message box comes up asking you if you to specify your equation.

Step 2: An input box comes up requesting the user to input, the polynomial equation, or equation of the linear or nonlinear system to be solved.

Step 3: Values for Parent 1 and parent 2 can be chosen at random and then inputted into the text boxes associated with them.

Step 4: A crossover point or split point is also inputted. This is the point at which crossover will occur. The programme can use any crossover point within the range of 1-15.

Step 5: The roots of the equation or other optimization solutions are solved and displayed.

Step 6: Click 'Enter' and the programme undergo a maximum of 500 iterations to determine the roots of the equation.

Step 7: When one of the roots (or any other optimization parameter) is found, the programme automatically stops iterating. A blue indicator outputs that a root (or any other optimization parameter) has been found.

Step 8: The process continues until other solutions are found.

Step 9: If after 500 iterations the roots have not been found, a message comes up telling you to start with a new set of parents. Go back to 3.

2.2 Software's algorithm or pseudo code

The software's algorithm or pseudo code was developed using the follow sequence of programming:

2.2.1 Create a Random Initial State (Population)

An initial population is created from a random selection of solutions. These solutions are analogous to chromosomes.

2.2.2 Evaluate Fitness

A value for fitness is assigned to each solution (chromosome) depending on how close it actually is to solving the problem (thus arriving to the answer of the desired problem). We note that until the desired answer is obtained, each of these solutions is treated as possible characteristics that the system would employ in order to reach the final answer.

2.2.3 Reproduce (& Children Mutate)

Those chromosomes with a higher fitness value are more likely to reproduce offspring (which can mutate after reproduction). The offspring is a combination of the genes of the parents; this process is known as "crossing over".

2.2.4 Next Generation

If the new generation contains a result that produces an output that is close enough or equal to the desired answer (solution) then the problem has been solved. If the otherwise is the case, then the new generation will go through the same process as their parents did. This will continue until a solution is reached.

Using the technique just discussed the algorithm is given as follows:

Start

Initialize input box
Input equation
Read equation or set of equations
If equation not valid Then
Print “Equation invalid”,

Go to 1

Read inputs; parent 1, parent 2, crossover point
Initialize loop = 1
If parent 1 or parent 2 is decimal Then
Convert both parents by to whole integer numbers
Convert parent 1 and parent 2 to binary
If loop = 1 then cross parent 1 and parent 2 from inputted crossover point
Else cross parent 1 and parent 2 from crossover point downwards.
Crossover point chosen at random from 1-15
Evalutate the fitness of offsprings
If Either offspring = one of the roots of the equation or any relevant optimization parameter **Then**
Output “Perfect Offspring”

Else

Parent 1 = Offspring 1
Parent 2 = Offspring 2
Loop = Loop + 1
Repeat search process.

Stop

3.0 Examples illustrating the use of the software.

The following examples are provided to illustrate the use of the software:

***Example* 1**

Find the roots of the polynomial: $x^3 - 0.7x^2 - 0.14x + 0.048 = 0$

Begin the program

At the programme commencement, the user is requested to input the equation to be solved as shown in Figure 3.

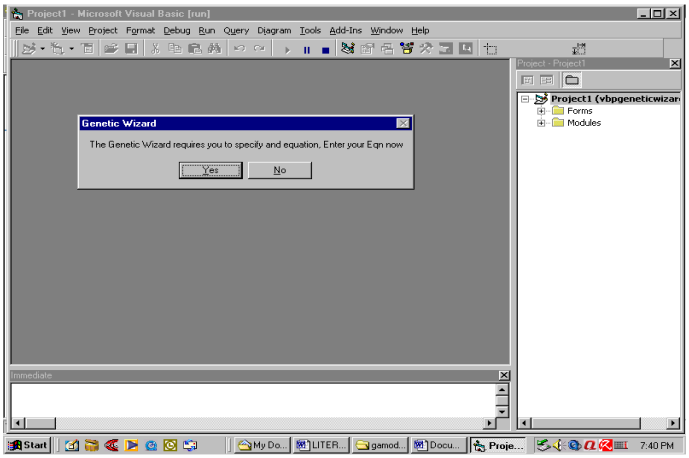

Figure 3: Shows a message box appear, requesting if you want to specify your equation.

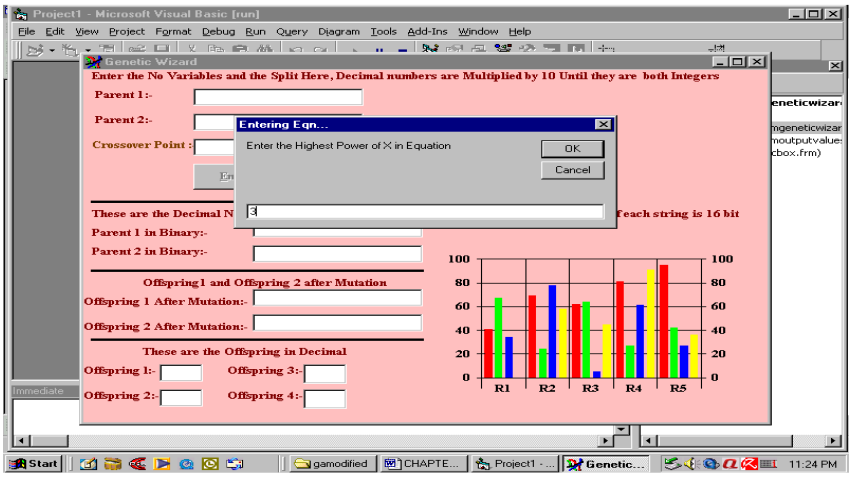

Figure 4: Shows an input box titled 'entering equation' along with the 'Genetic wizard form'. At this stage the equation is inputted

For each value you enter, click 'ok' to move to the next section until the last value is inputted. The highest power (3) is inputted as can be seen above and 'ok' is clicked.

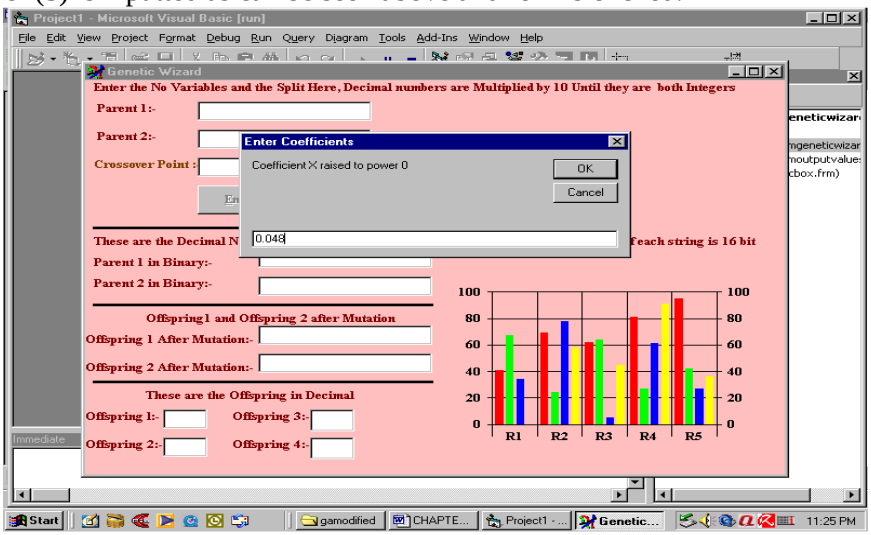

Figure 5: Shows how each coefficient of x is inputted one after the other in a descending other with respect to *x*.

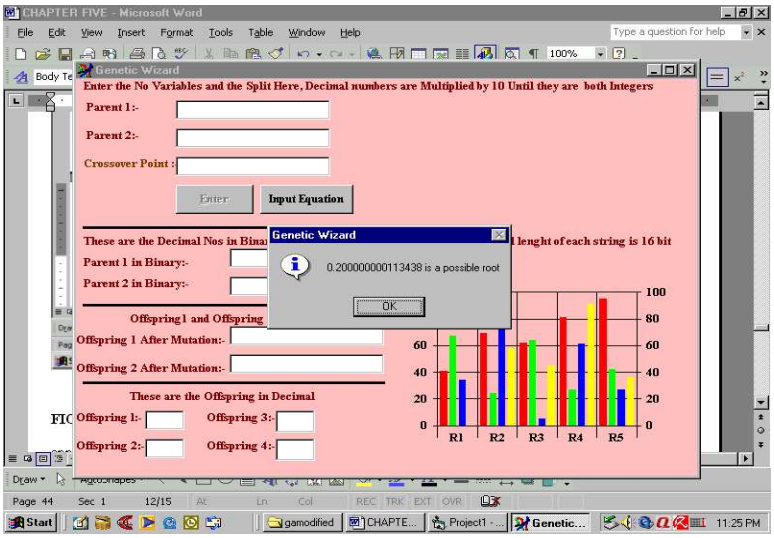

Figure 6: Shows one of the possible roots of the equation found.

As shown in Fig. 6, 0.20 is one of the roots of the equation and the other roots obtained using the software are: 0.30, and 0.80**.**

***Example* 2**

Find the roots of the polynomial**:** $x^5 - 18x^4 + 51x^3 + 286x^2 - 360x - 576 = 0$

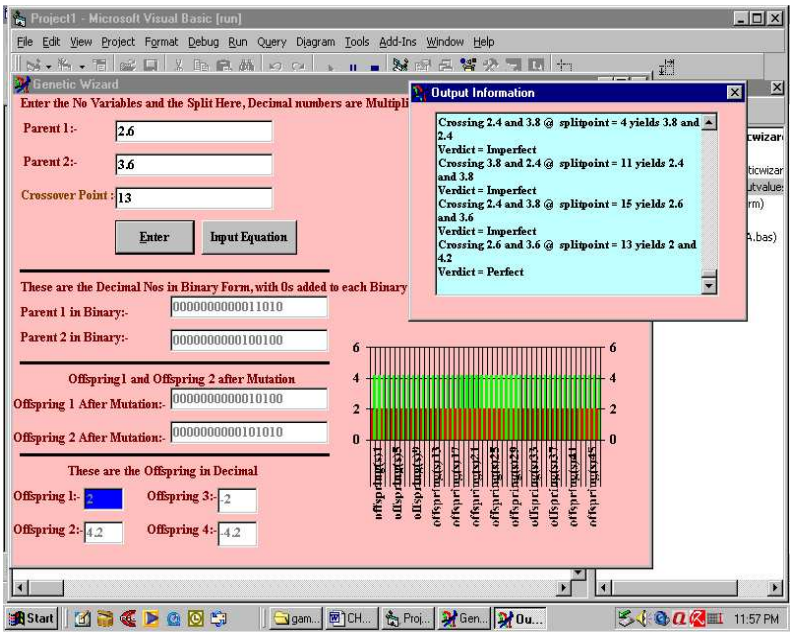

Figure 8: Shows the final form, one of the possible roots has been found, in this case after 48 iterations. A graphical display shows the number of generations produced and the offspring generated.

Evaluating example 2, using the genetic algorithm software, we have the following results shown in Table 1

Table 1: Comparison of Exact Solution and This model's Genetic Algorithm

Roots (Exact using Newton-Raphson)	Roots (This model's Genetic Algorithm)	Percentage difference
12.000	12.00	0.0000%
8.000	8.00	0.0000%
2.00	2.000	0.0000%
-3.00	-3.000	0.0000%
-1.00	-1.000	0.0000%

***Example* 3**

Evaluate the roots of the following polynomial:

$$x^4 + 8.5921x^3 + 23.7768x^2 + 23.5302x + 7.5857$$

The using the software the polynomial's roots was obtained in less the 3 seconds the values are presented in Table 2.

Table 2: Comparison of Exact Solution and This model's Genetic Algorithm

Roots (Exact using Newton-Raphson)	Roots (This model's Genetic algorithm)	Percentage difference
-0.7566	-0.7566	0.0000%
-0.8355	-0.8355	0.0000%
-3.0000	-3.0000	0.0000%
-4.0000	-4.0000	0.0000%

The software developed using this algorithm has an edge over other schemes in that it is not limited to the type of problems it can handle. For example, a scheme like the Newton-Raphson method would easily solve for the roots (zeros) of a function but can not provide solutions to problems of simultaneous equations. This is due to the way the Newton Raphson method functions. But the Genetic algorithm being a search tool that provides improving solutions at each stage of iteration, can conveniently handle this problem. In fact it can handle all varieties of optimization problems which other schemes cannot. This because they were not developed to be all embracing but problem type specific.

We illustrate the superiority of the algorithm, by the following Example:

***Example* 5:** Simultaneous linear equations.

Consider the set of linear equations:

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots\ldots\ldots + a_{1n}x_n = b_1 \\
&a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots\ldots\ldots + a_{2n}x_n = b_2 \\
&\quad\vdots \qquad\quad \vdots \qquad\qquad\qquad\quad \vdots \qquad \vdots \\
&a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots\ldots\ldots + a_{nn}x_n = b_2
\end{aligned}$$

This can be written in matrix form as:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \; i.e, \; Ax = b$$

where $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix}$; $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ and $b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$

If we multiply both sides of the matrix equation by the inverse of *A,* we have: $A^{-1}.Ax = A^{-1}b$. But $A^{-1}.Ax = I$, therefore $Ix = A^{-1}.B$. The implication is that if we form the inverse of the matrix coefficients and pre-multiply matrix **b** by it we shall determine the matrix of the solutions of **x**

***Numerical data for Example* 5**:

To illustrate the power of this genetic algorithm software, we consider a set of 10 simultaneous linear equations. This is represented in matrix form; Ax = B as follows:

$$\begin{pmatrix} 5 & 8 & 5 & 6 & 6 & 11 & -3 & 5 & 6 & 1 \\ 4 & 9 & 7 & -3 & 11 & 4 & 8 & 6 & 9 & 10 \\ 7 & 9 & 5 & 3 & -4 & 1 & 2 & 12 & 9 & 3 \\ 7 & 8 & 9 & 5 & 3 & 1 & 4 & 6 & 6 & 1 \\ 9 & 7 & 4 & 6 & -6 & 1 & 5 & 2 & 3 & 7 \\ 6 & 8 & 9 & 4 & 1 & 6 & 3 & 1 & 4 & 6 \\ 1 & 6 & 3 & 5 & 4 & 3 & 1 & 4 & 1 & 6 \\ 8 & 5 & 7 & 8 & 4 & 5 & -11 & 2 & 3 & 4 \\ 1 & 2 & 6 & 4 & 1 & 3 & 5 & 8 & 5 & 9 \\ 6 & 5 & 1 & 2 & 3 & 1 & 2 & 1 & 5 & 7 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} 21 \\ 8 \\ 7 \\ 9 \\ 12 \\ 17 \\ 21 \\ 16 \\ 4 \\ 11 \end{pmatrix}$$

This is solved using the genetic algorithm software within 3 seconds, and screen shot of the solution is shown in Figure 7.

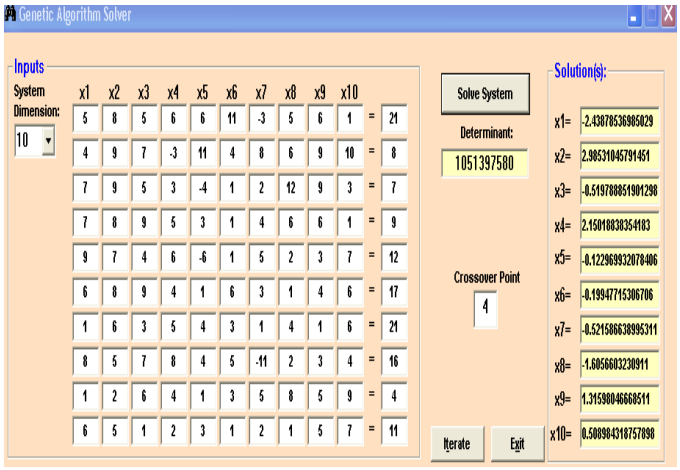

Figure 9: Screen shot of the solution obtained for Example 5 using the Genetic Algorithm software.

The solutions, using this genetic algorithm software are (see Figure 9) shown in Table 3:

Table 3: Solution to Example 5; using this Algorithm

Variable	**Solution (from using the the software)**	**Variable**	**Solution (from using the the software)**
X_1	-2.43878536985029	X_6	0.19947715306706
X_2	2.98531045781451	X_7	0.521586638995311
X_3	-0.519788851901298	X_8	-1.6056603230911
X_4	2.1508838354183	X_9	1.31598046668511
X_5	-0.1229699932078406	X_{10}	0.508984318757898

The problem in Example 5 was the solved using the Gauss-Jordan row reduction technique. The solution obtained using the Gauss-Jordan row reduction was exactly the same that obtained using the software. However the Gauss-Jordan row reduction technique cannot be used to find the roots of a polynomial, which this software can easily be used to obtain, see Examples 1-4.

5.0 Discussion

In this work we have considered a number of examples on solving for roots of polynomials. In Figures 3-8, the basics computational steps involved in using the software to solve for the roots of a polynomial. The results obtained in all the cases, showed (0.0%) no percentage deviation from exact analytical solution. This can be clearly seen in Tables 1 and 2. Also we illustrate the superiority of the algorithm by solving a set of 10 dimensional simultaneous equations. The results are shown in Fig. 9 and Table 3. The solution obtained using the Gauss-Jordan row reduction was exactly the same that obtained using the software. The results show that genetic algorithm can be used to optimize and find numerical solutions to mathematical and applied mathematical problems.

Also most of the classic schemes for example; Newton-Raphson Scheme and Gauss-Jordan scheme can only solve for specific problems that they were designed for. Consequently, Newton-Raphson is suited for finding the zeros (roots) of polynomial and Gauss-Jordan for solving set of linear simultaneous equations. But this genetic algorithm programme, can effectively and accurately, solve all classes of optimization problems and solve also for the zeros or roots of polynomial. The problem may be in any field of engineering like Petroleum and Production Engineering among others.

6.0 Conclusion

The aim of this work was to design a software, using genetic algorithm that can evaluate the roots of polynomial equations and solve and optimize equations described the dynamics of a system. From the examples taken, and the results generated, it can it seen that the genetic algorithm could handle these problems in an integrated manner accurately. The software is easy to understand, easy to use and results are generated in a very small amount of time (usually less than 3 seconds). Thus, this genetic algorithm software with crossover substantially out performs all the other classical methods and heuristics

References

[1] Calson, S.E. and Shonkwiler, R. (1995), Annealing a Genetic Algorithm over Constraints, Department of mechanical, Aerospace and Nuclear Engineering Thornton Hall, University of Michigan.

[2] Coit D.W, Smith, A.E. and Tate, D.M (1995), Adaptive Penalty Methods for Genetic Optimization of Constrained Combinational Problems, Accepted for

[3] Darwin, C.R. (1859), On the Origin of Species by Means of Natural Selection and The Descent of Man and Selection in Relation to Sex, third edition, vol. 49 of Great Books of the Western World, Editor in chief: M.J. Adler. Robert P. Gwinn, Chicago, IL, 1991. First Edition John Murray, London, 1859.

[4] Parks, G.T., Li, J., Blazs, M. and Miller, I. (2001), An Empirical Investigation of Elitism in Multi-objective Genetic Algorithms, Page 51-74 of the Journal of Foundations of Computing and Decision Sciences, Vol 26. No.1

[5] Goldberg, D.E. (1989), Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.