J. Automation & Systems Engineering 10-1 (2016): 40-64

**Regular paper**

# Genetic Folding/Programming Toolbox: Analogue Circuit Design Case Study

Ogri J. Ushie[1], Maysam F. Abbod[2], and Brian E. Usibe[3]

[1, 2] Department of Electronic and Computer Engineering, College of Engineering, Design and Physical Sciences, Brunel University, London, UK. Ogri.Ushie@brunel.ac.uk / ushjames@yahoo.com
[1,3] Department of Physics, Faculty of Science, University of Calabar, Nigeria.

***Abstract*** *- This work introduces the concept of genetic folding/programming combination to develop a standalone optimisation tool and the developed algorithm is tested with four different benchmark functions. Most GP used in circuit evolution interface two software packages but this work only Matlab is used which reduces the time used for transferring the simulation between the two platforms. To enhance testing of the algorithm and automatic Netlist creation, the expression is extracted with the aid of genetic folding. The automatically simulated Netlist is fed to modified symbolic circuit analysis in Matlab that translate it to matrices to enhance frequency response. The frequency response is then compared to the set frequency response and the RMS difference gives the error which controls the programme towards the desired solution. One circuit is tested and the algorithm successfully evolved the set frequency response.*

**Keywords -** Combined Genetic Folding and Genetic Programming algorithm, Automatic Netlist Simulation, Modified Symbolic Circuit Analysis in Matlab, and Benchmark testing

## 1 Introduction

In artificial intelligence, genetic programming (GP) is an evolutionary algorithm (EA) -based methodology motivated by biological evolution to search computer programmes that execute a user-defined task. Fundamentally, GP is a set of algorithms and a fitness function to compute how well a computer has implemented a task. GP is a domain-independent, systematic method for getting computers to resolve problems automatically, beginning from what is required to be done as a high-level statement. Using inspirations from biological evolution, GP begins from a randomly generated computer programmes, and gradually refines them through procedures of sexual recombination and mutation, until solutions are obtained. All these processes are carried out without the user having to specify the form or know or structure of the solutions in advance.

A basic introduction to GP that specify how you can create: an individual using terminals and functions, random population using full, grow and ramped-half-and-half is in [1]. The paper also described GP operators and how to evaluate fitness. GP Matlab toolbox that illustrates how it can be represented using Matlab is in [2, 3]. GP algorithms have been applied in different areas: Balasubramaniam and Kumar have used GP as a novel approach to finding a solution to matrix Riccati differential equation for a non-linear singular system. The goal is to reduce calculation effort and results presented show that GP approach is better regarding accuracy as compared to the traditional Runge-Kutta method [4]. Other applications include: GP application in the area of software repairs are in [5, 6], while a fully automated technique to locate and repair bugs in software is illustrated [5]. Also solving iterated functions using GP is in [7]. GP- based feature optimiser integration with patter recognition and fisher criterion methods to non-intrusive load supervising for load identification is illustrated [8].

GP has been applied to automatically synthesise similar human designs in some fields. These include: analogue electrical circuit, antennas, mechanical systems, controllers, quantum computing circuits, optical lens system, bioinformatics, robotics, sorting networks, assembly code generation, scheduling and software repair. Others are: communication protocols, empirical model discovery, reverse engineering and symbolic regression. According to the authors, despite differences in the techniques and representations, results presented shared common features [9, 10]. Hou et al. [11] presented GP based on the tree representation for a passive filter synthesis and the results presented show that their method can generate both economical and compliant passive filter circuits. The paper also specifies how the authors intended to add more design objectives such as component value sensitivity and group delay variation to be considered in their future work. Chang et al. applied the same technique as that of Hou et al. and claimed that their technique is better with regard to its efficiency compared to traditional technique and faster than previous work [12].

Evolvable Hardware (EH) is a research field in EA used in electronic circuit simulation with no manual engineering design. It is a combination of autonomous system, fault tolerance, artificial intelligence and reconfigurable hardware. Some of EH's applications in electronic circuit simulations are discussed by different researchers [13-20]. Doboli et al. [21] used very high speed integrated circuit hardware description language-analogue mixed signal (VHDL-AMS) for creating high-level analogue and mixed signal. In the work, many constraints are introduced to the VHDL-AMS instructions and case studies are illustrated. An evolvable hardware simulation which automatically designs analogue circuits using parallel GA was developed by Lohn et al. [22]. The algorithm evolves component values, circuit topology and circuit size. Vural et al. [23] propose three EAs: harmony search (HS), DE and ABCA to optimise CMOS amplifier area. Results presented to demonstrate that the techniques meet specifications, accommodates required functionalities and the design objective.

Other applications of GP as EH in addition to those discussed in Section 2.4.2 include: The use of current – flow analysis and GP for the invention of CMOS amplifier is presented in [24]; the work illustrates how current-flow evaluation corrects and screens circuits utilising topology-independent design rules. The approach is aimed to show how connections are linked between transistors. Also, a tree representation method in circuit design is illustrated by Senn et al. [25]. The authors combined GP and two-port theory for analogue circuit design. The presentation of the circuit as the two-port network enhanced the encoding and evaluating of the circuit's structure. The approach is also applied to active (transistor) and passive linear circuits. Moreover, GP use for the automatic design of analogue electronic circuits by Koza et al. [26] that has transistor as the active filter is presented as part of examples. It uses single technique by applying GP for modelling both circuit topology and sizing. Also, Peng et al. [27] used GP and bond graph (GPBG) in electronic circuit analysis with active components that is an extension of their previous research on passive component design. The analysis covers three models of a transistor, and one model of an op amp are implemented and analysed as two-port BG components. It also uses GP to create BG defining parameters and component topology in the design of the active filter.

## 2 Methodology

This work demonstrates how concept of genetic folding (GF) algorithms and GP are combined and used to develop a standalone optimisation tool and how the developed algorithm is tested with four different benchmark functions. The flowchart shown in Figure 1 summarises the GF and GP combined algorithm for circuit evolution and benchmark testing procedure used in this work. A randomly generated number is used in the start generation to generate population which is calculated to ascertain how well each individually evolving expression is performing with regard to its individual objective function. If the evolving expression satisfies the objective with zero error, the iteration with zero error is taken as solution else a generation continues. The evolving expression from the generation is extracted with the aid of GF and substituted with specified range of values of *X* and *Y*. The same values are being substituted into an original expressions and a RMS difference is used as an error. The procedure continues until a zero error is obtained or the objective function is satisfied. Detailed processes involved are explained in the flowchart shown in Figure 1.

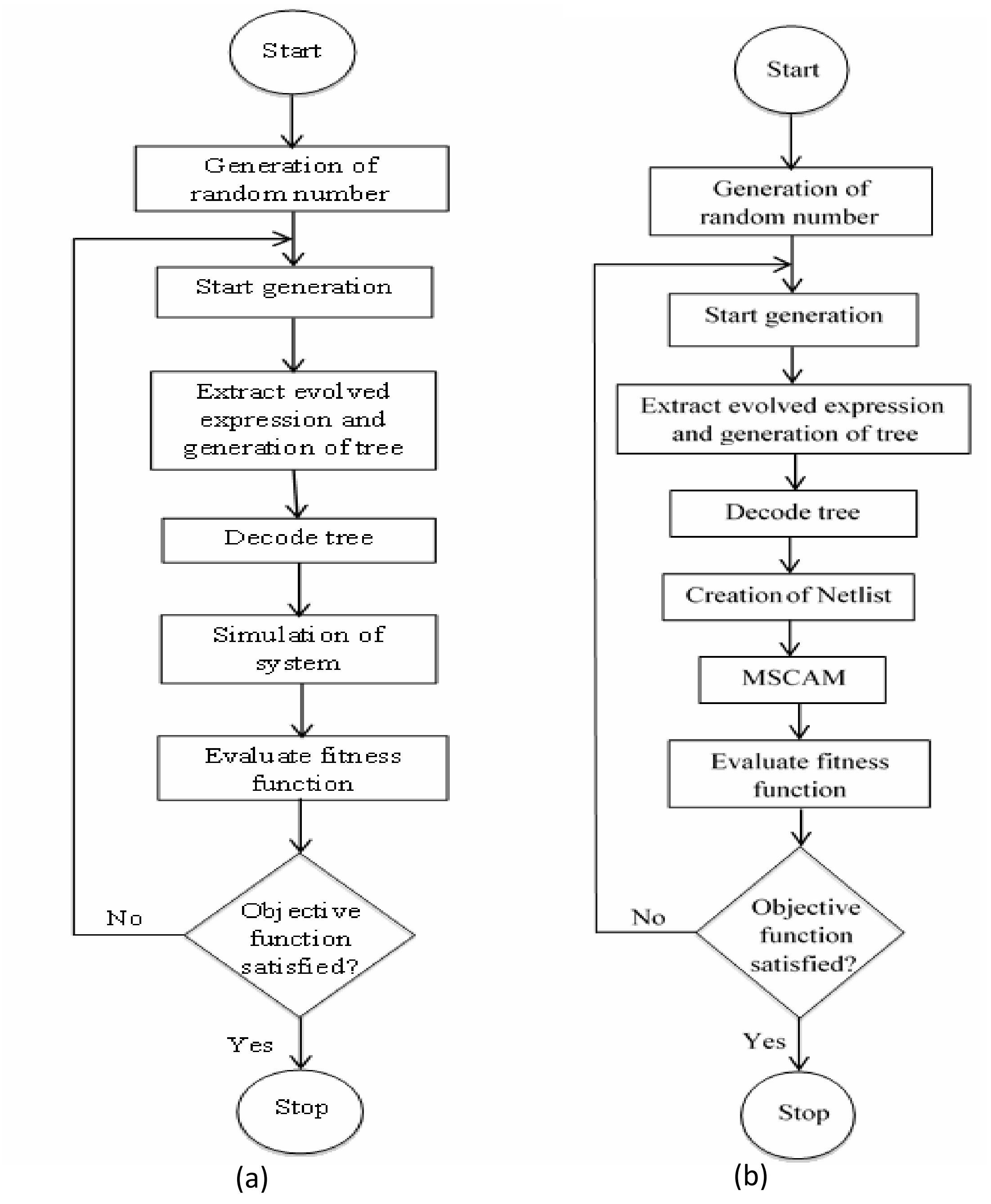
Start
Generation of random number
Start generation
Extract evolved expression and generation of tree
Decode tree
Simulation of system
Evaluate fitness function
Objective function satisfied?
No
Yes
Stop
(a)
Start
Generation of random number
Start generation
Extract evolved expression and generation of tree
Decode tree
Creation of Netlist
MSCAM
Evaluate fitness function
Objective function satisfied?
No
Yes
Stop
(b)

**Figure 1: The GF and GP algorithm for (a) Benchmark testing. (b) Circuit evolution.**

## 2.1 Genetic Programming

GP is the newest concept in the research area of evolutionary computation (EC). It was created by John Koza and originated from the GA. GP differ from the genetic algorithm (GA) in that, GP is represented by variable length structures containing whatever elements are needed to solve the problem, whereas GA is represented by a fixed length of numerical strings. The tree structure (TS) in GP population is used to create neural networks, determine designs for analogue electric circuits and parallelise computer programmes. The TS is great because it can produce solutions of complexity and arbitrary size, as opposed to GA with fixed-length. GP has been used successfully in a different number of applications: arts and entertainment, biology and bio-information, medicine, time series prediction, control, modelling and regression image and signal processing. In GP, a population is randomly created and each individual in the population is evaluated to ascertain its fitness that serves as selection criteria. The best individual is selected and reproduced, mutated or crossover with other individuals to produce new individuals for the next generation [9, 10, 28-30].

In preparation for implementing GP according to Kennedy and Eberhart [31], five steps are involved:

1. State the function set
2. State the terminal set.
3. State the fitness measure.
4. Select the system control parameters.
5. State the terminal conditions.

The function set is limited by programming language used to run the GP. The function set includes mathematical functions (cos, sin, tan, exp, etc.), arithmetic operators (+, -, x, #, /, etc.), Boolean operators (AND, OR, NOT, NOR, etc.). The terminal sets composed of variables and constants; for example, in circuit evolution, it comprises of resistors, capacitor, inductors, transistor, diode, op-amps, etc. A fitness measure is often chosen to be inversely proportional to an error produced by programme output or it may be the score of programme achieves in as regard objective function. The two major control parameters are the maximum number of generations and population size. Others parameters used are crossover probability, reproduction probability and mutation. The termination conditions may be the maximum number of generation or if the objective function is achieved.

GP analysis is centred on how the combined GF and GP algorithm is developed and represented for the benchmark testing. The same approach is applied for all benchmark testing; the same parameters are used except variation in Length of parameters and length of the chromosome that is being determined by the length of a TS or a function to be tested. For this case study, benchmark testing expression 1 (i.e. equation 2) in Section 4.1 is used for an illustration. The illustration is based on the following subheadings presented after summarising the GP algorithm:

The GP algorithm, according to Koza [28], is based on the three steps:

1. Generate a random population composed of the original function and termination criteria for the problem.
2. Perform the following sub-steps iteratively until the termination criteria are reached:
3. Each programme in the population is executed such that a fitness measure that specifies how well the problem is solved is clearly formulated.
4. New population is created by selecting individual(s) with probability based on fitness and then these operations are applied:

(i) Reproduction: Copy existing individual to the new population.

(ii) Crossover: Two individuals are created for the new population by randomly recombining chosen parts of two existing individuals.

5. The single best individual in the population produced while the run is taken as the result.

### 2.1.1 Initialisation of Parameters

The following elements are initialised: Length of parameters = 63, population size = 100, maximum number of generation = 500, length of chromosome = Length of parameters multiply by bit group ($63 \times 3 = 189$), mutation = 0.10 and crossover = 0.90. These are the settings that give the best result after several trials. The programme finds the required solution to a given problem whenever it has zero error. The population is randomly generated after parameters initialisation of a size equal to length of the chromosome multiply by the population size.

### 2.1.2 Decoding

The string is coded into +, ×, -, 3, 4, *Y, X* and 7. In this case, a chromosome is divided into a bit group of three, and each is converted to its decimal equivalent. The decimal equivalent is interpreted as:

- '0' represents plus
- '1' represents multiplication
- '2' represents minus
- '3' represents 3
- '4' represents 4
- '5' represents *Y*
- '6' represents *X*
- '7' represents 7

### 2.1.3 Creation

A tree is randomly generated using an operands or terminals (the terminals in this case 3, 4, *Y*, *X* and 7) and operators (+, × and -) defined in Section 2.1.2 above. Beginning with many trees of different sizes and shapes is good. A tree is generated applying a Grow or a Full method:

- Grow – path lengths in TS vary up to a maximum length.
- Full – all branches in TS must reach its maximum depth.
- Ramp half – and - half method – trees of varying depths from a minimum to a maximum depth. Half of the tree is initialised with full and the other with grow. The ramp half - and – half is used.

### 2.1.4 Mutation

Pick a mutation reference point in one parent and swap its subtree with another randomly generated tree. In this research, the mutation rate of 0.1 is used.

### 2.1.5 Crossover

Pick crossover reference points in both parents and then exchange the subtrees. An offspring will be varying even if the parents are the same. The crossover rate of 0.9 is used. A roulette wheel method is used to select two individuals from the present population, and the ten randomly selected subtrees of the parents are swapped to create two offspring.

## 2.2 Genetic Folding

The GF is a class of EA based on numbers of genes structurally organised in order of linear numbers separated by dots [32]. GF is one of the classes of EA based on a generic meta-heuristic optimization technique. The main aspect of the GF algorithm is a population-based methodology motivated by biological evolution. GF imitates the Ribo Nucleic Acid (RNA) secondary structure folding procedure of the complementary bases on itself. In this research, the GF is used to show how elements are structurally linked from beginning to end, so that the expression can be extracted and substituted with respective values of X and Y for benchmark testing or extracted to create an automatic Netlist for circuit evolution. The GF representation of the GP TS of Figure 11 (Figure 11 is used because it is a desired tree structure for equation 2) is shown in Table 1

**Table 1: The GF representation for benchmark testing.**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + | + | + | × | × | × | - | *X* | × | *X* | × | *Y* | *Y* | 7 | × |
| 2.3 | 4.5 | 6.7 | 8.9 | 10.11 | 12.13 | 14.15 | 0.8 | 18.19 | 0.10 | 22.23 | 0.12 | 0.13 | 0.14 | 0.15 |

| 18 | 19 | 22 | 23 | 30 | 31 | 36 | 37 | 38 | 39 | 46 | 47 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| × | × | 3 | × | *X* | 4 | *X* | *Y* | *X* | *Y* | *X* | *Y* |
| 36.37 | 38.39 | 0.22 | 46.47 | 0.30 | 0.31 | 0.36 | 0.37 | 0.38 | 0.39 | 0.46 | 0.47 |

GF is best understood with the following points:

1. The arrangement of the chromosome comprises of float string in the gene and the location of the gene.
2. The gene structure is left child *(LC)* side separated by dot and right child (*RC)* side.

3. The dot stands for and.
4. The operator that has two operands is with *LC* and *RC*.
5. The operator that has one operand is with *LC* and 0 in the *RC*.
6. The terminal has 0 in *LC* and value in the *RC*.

## 2.3 Specifications of the Objective Function and Hardware Requirements

Detail of software environment is as: Matlab version: 8.0.0.783 (R2012b), operating system: Microsoft Windows 7. Others are: RAM: 12 GB, system rating: 64-bit operating system and processor: Intel (R) core (TM) I7-2600 CPU @ 3.40 GHz. X is given a range of value from -10 to 10 with an increment of 1 whereas Y is given a range value from -2 to 2 with an increment of 0.2. This information is used to generate matrices of size 21 by 21 for both an original and an evolving expression. The matrices are reshaped to size 1 by 441 and the RMS difference between the two matrices (the original and the evolving expression matrices) give the error. Mathematically:

- $U$ is the reshaped matrix of size 1 by 441
- $V$ is the reshaped matrix of size 1 by 441

$$W = RMS(U - V) \tag{1}$$

where $W$ is the error, $U$ is the original expression, and $V$ is the GP evolving expression, the error controls the algorithm toward the required solution. The algorithm produces optimal solution when the error is zero.

## 4 Algorithm Benchmark Testing on Mathematical Functions

To test for efficiency, validation and reliability of optimisation algorithm are often performed using a test function or benchmark. Test function is vital to compare, validate and compare the functioning of optimisation algorithms, specifically newly developed ones [33]. For a new GP algorithm developed, it is important to validate its performance by using existing set of benchmarks. The basic requirements on a benchmark according to Feldt et al. [34] are:

- Validity: mistakes that invalidate the expected output should be avoided,
- Comparability: findings should be compared to others researchers findings.

- Reproducibility: experiments and problems should be well documented so that other researchers can reproduce the same solutions to a given problem.

## 4.1 Benchmark Testing Expression 1

$$Z = X^3Y^2 + 3X^2Y + Y^2 - 4X + 7 \qquad (2)$$

Both *X* and *Y* is given a range of values from -50 to 50 with an interval of 1and a three-dimensional plot is represented after each iteration TS representation. The objective function specification is described in Section 4. The GP algorithm evolved an expression with 593.28 error in 1st iteration and the GP TS is shown in Figure 2, its three-dimensional plot is represented in Figure 3.

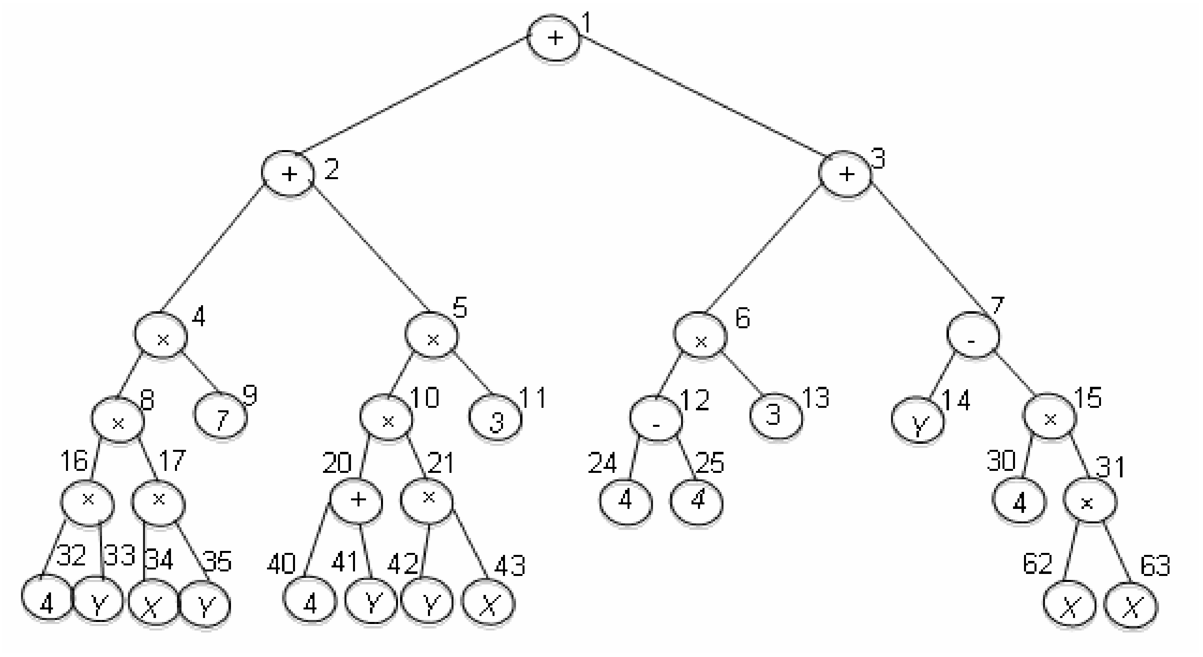

**Figure 2: 1st iteration GP evolved TS for expression in equation 2 with 593.28 errors.**

Simplification of the above TS gives;

$$Z = 7(4 \times Y) \times (X \times Y) + (4 + Y) \times (Y \times X)3 + 3(4 - 4) + Y - 4(X \times X))$$

$$Z = 28XY^2 + 12XY + 3XY^2 + Y - 4X^2$$

$$Z = 31XY^2 + 12XY + Y - 4X^2$$

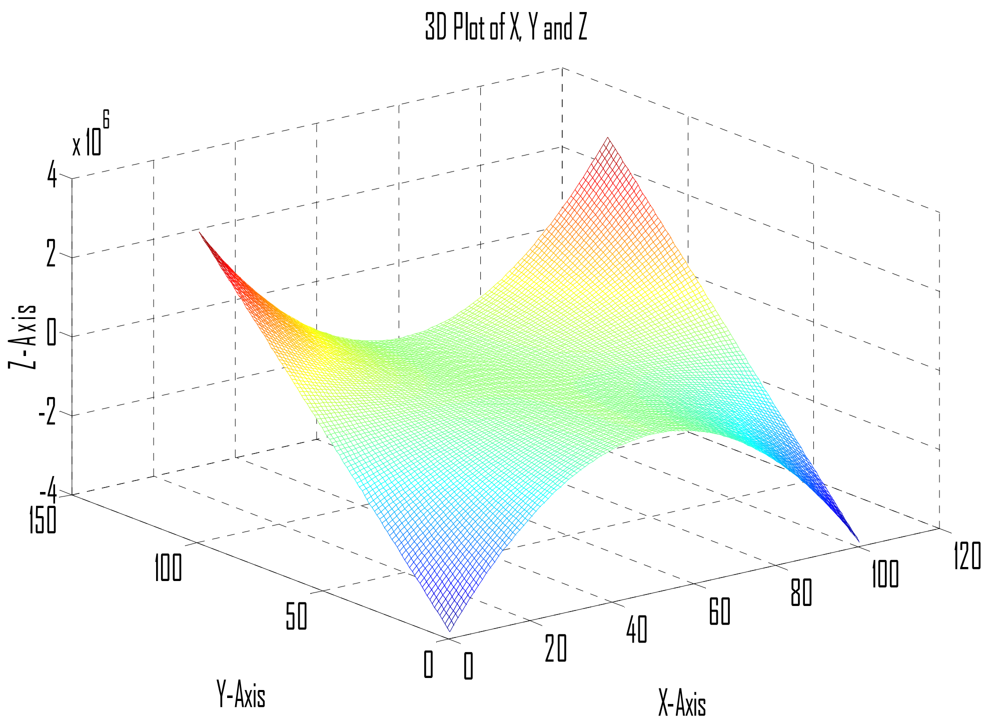

**Figure 3: Three-dimensional plots for expression in equation 2 for 1st iteration with 593.28 error.**

The GP algorithm also evolved another expression with 83.72 errors in the 20th iteration and the GP TS is in Figure 4, its three-dimensional plot is represented in Figure 5.

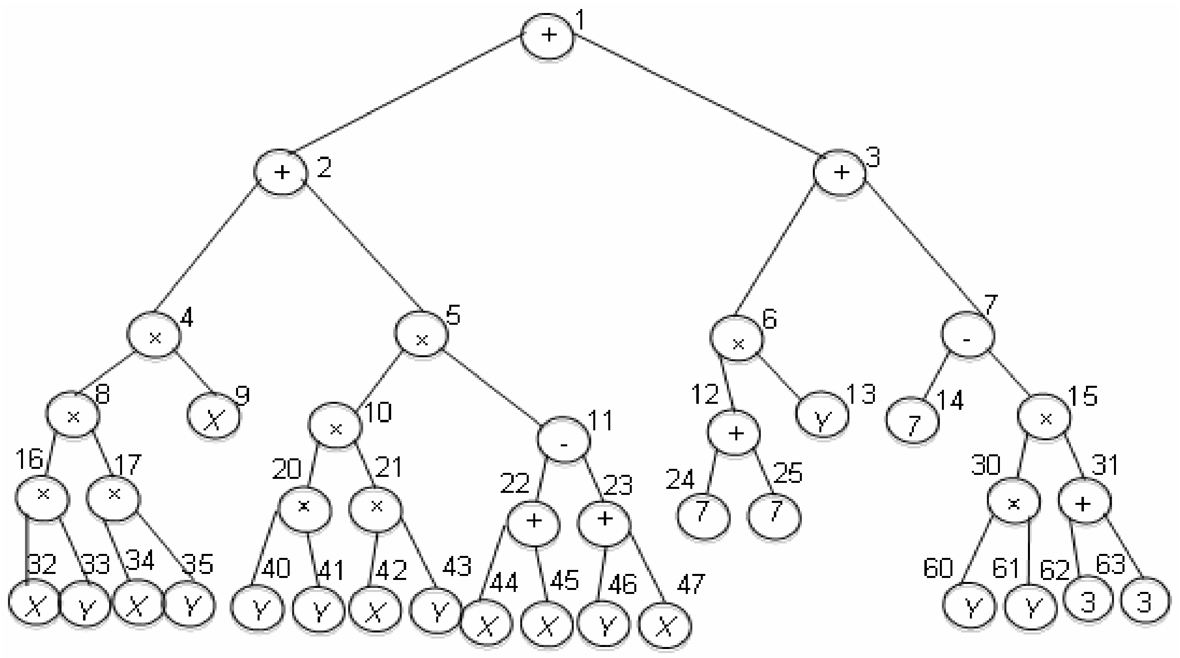

**Figure 4: 20th iteration GP evolved TS for expression in equation 2 with 83.72 errors.**

Evaluation of the above TS gives;

$$Z = X(X \times Y) \times (X \times Y) + Y^3 X(2X - (X + Y)) + 14Y + 7 - 6Y^2$$

$$Z = -Y^4 X + X^3 Y^2 + Y^3 X^2 + 14Y - 6Y^2 + 7$$

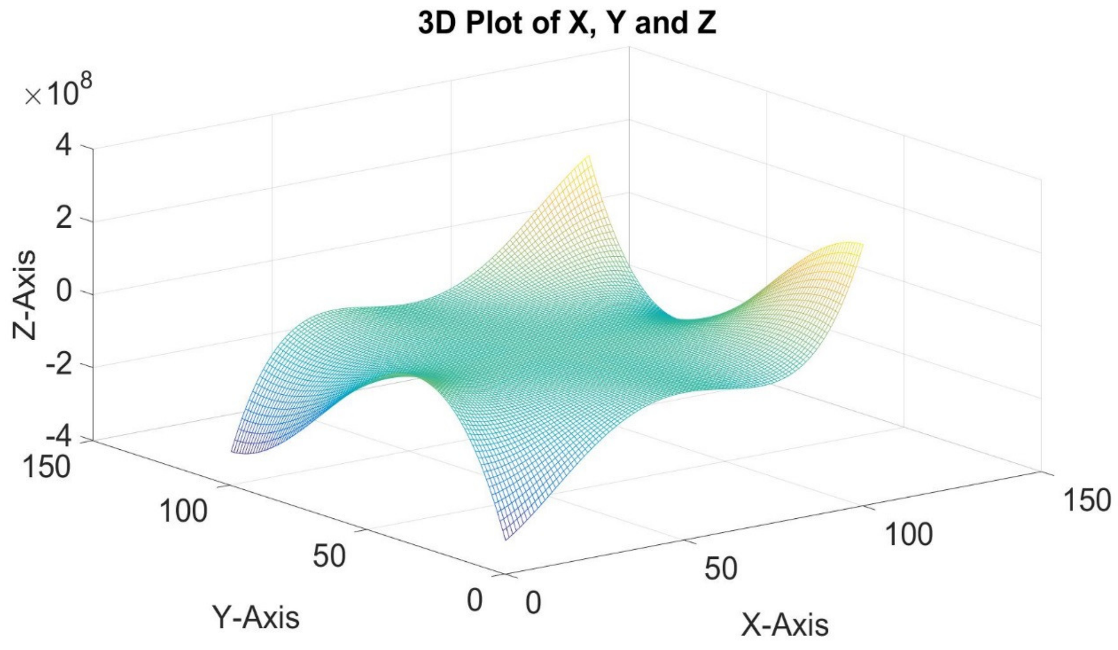

**Figure 5: Three-dimensional plots for expression in equation 2 for the 20th iteration with 83.72 errors.**

The GP algorithm also evolved different expression in the 41st iteration with 3.63 errors and the GP TS is in Figure 6, its three-dimensional plot is represented in Figure 7.

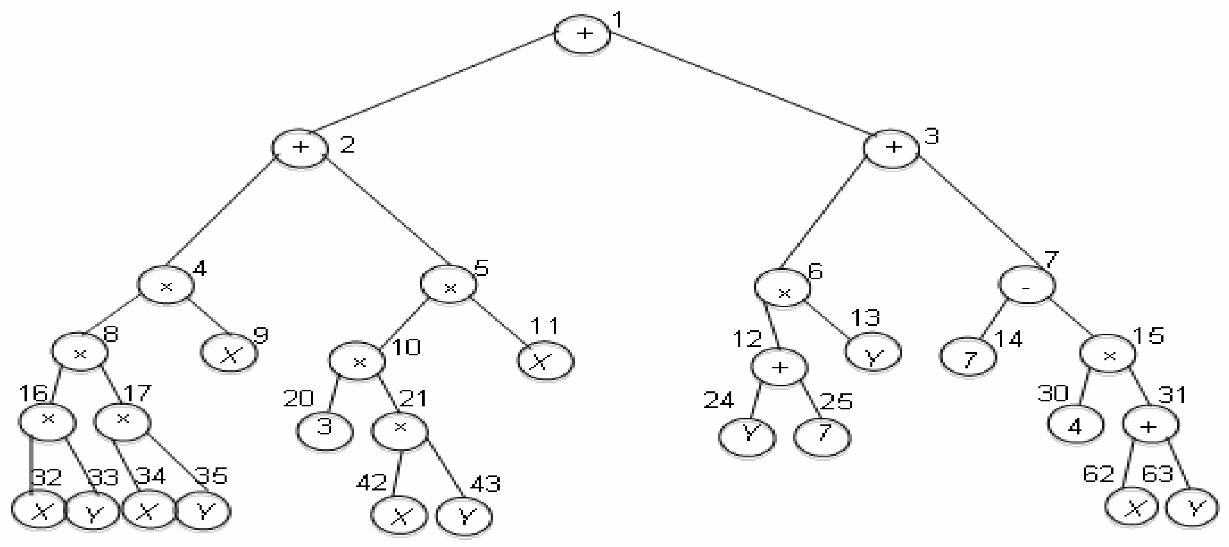

**Figure 6: 41st iteration GP evolved TS for expression in equation 2 with 3.63 errors.**

Its mathematical expressions are of form.

$$Z = X(X \times Y) \times (X \times Y) + YX(3X) + Y(Y+7) + 7 - 4(X+Y)$$

$$Z = X^3Y^2 + 3X^2Y + Y^2 + 3Y - 4X + 7$$

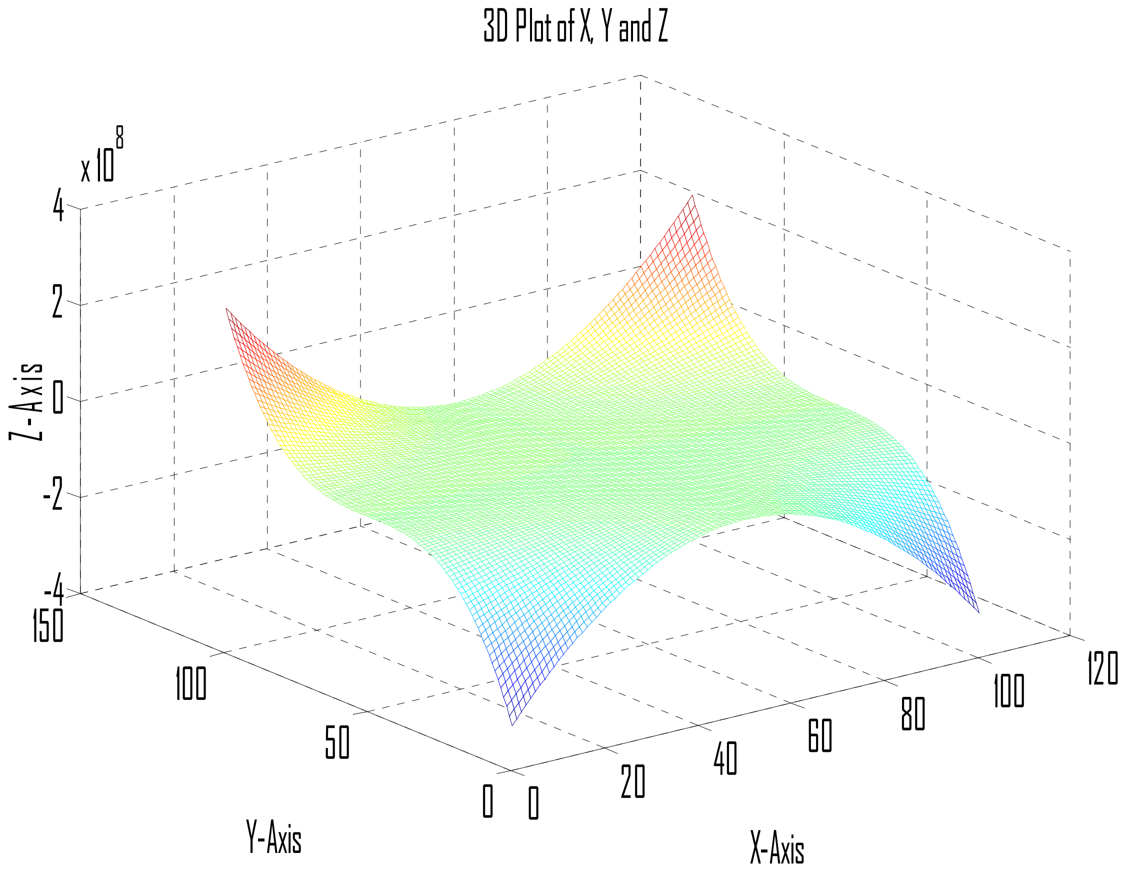

**Figure 7: Three-dimensional plots for expression in equation 2 for the 41st iteration with 3.63 errors.**

The GP algorithm finally evolved the desired expression with optimal solution in the 52nd iteration with zero errors and the GP TS is in Figure 8, its three-dimensional plot that is the same as that of original expression is represented in Figure 9 and the plot of errors against generations is shown in Figure 10.

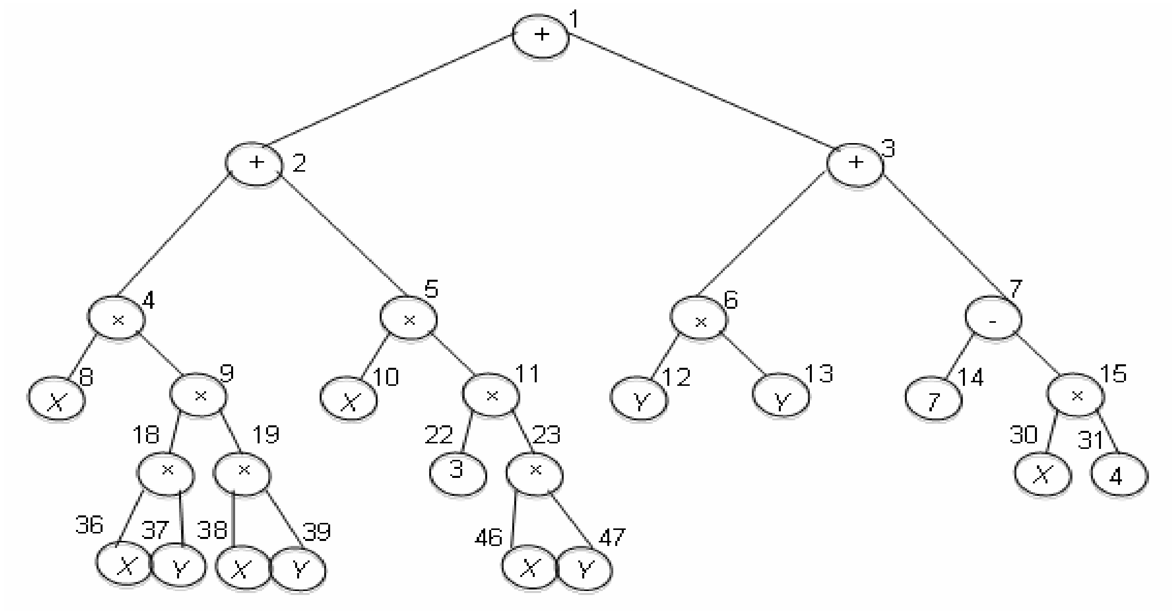

**Figure 8: 52nd iteration GP evolved TS for expression in equation 2 with zero error.**

Critical analysis of the evolved TS of Figure 8 gives the expression simplified bellow:

$$Z = X(X \times Y) \times (X \times Y) + X(3) \times (X \times Y) + (Y \times Y) + 7 - (X \times 4))$$

$$Z = X(X^2Y^2) + 3X(XY) + Y^2 + 7 - 4X = Z = X^3Y^2 + 3X^2Y + Y^2 - 4X + 7$$

From the investigation of the TS or transforming the TS into equation, we can deduce that the algorithm is efficient because it has successfully evolved the original equation.

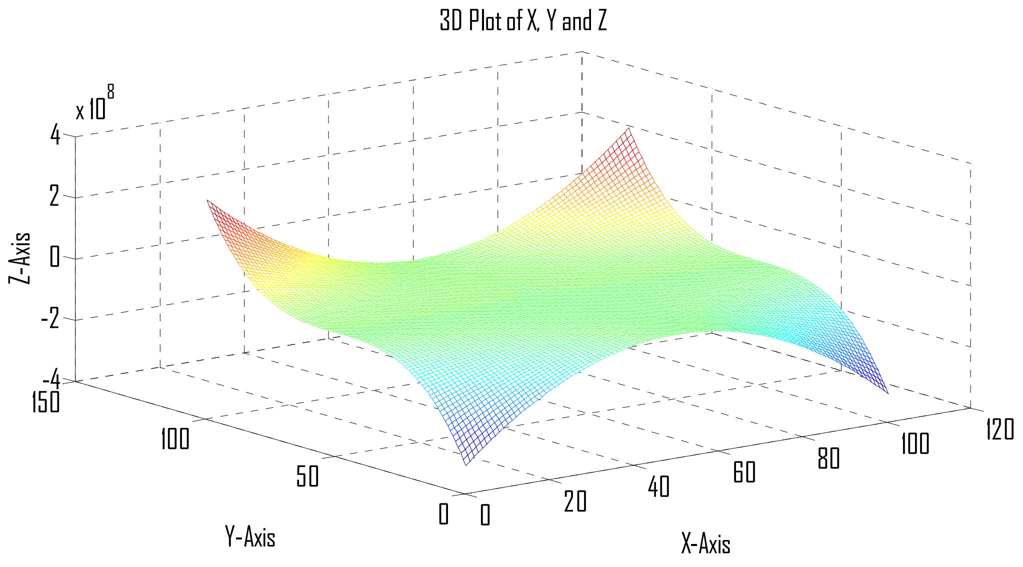

**Figure 9: Three-dimensional plots for expression in equation 2 for the 52nd iteration with zero error and the same as original circuit.**

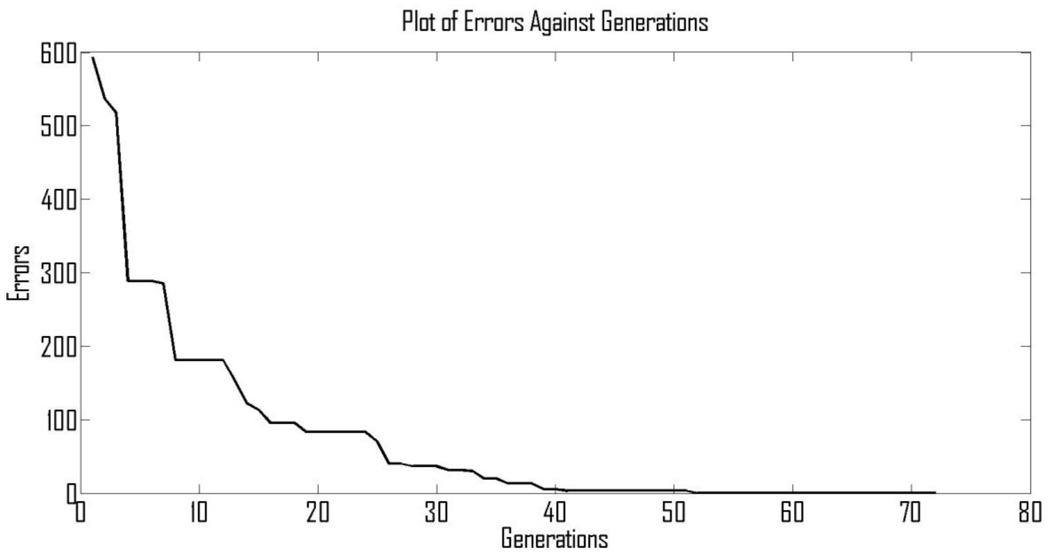

**Figure 10: Plot of errors against generations for expression in equation 2.**

## 4.2 Benchmark Testing Expression 2

$$Y = X^4 + 3X^3 + 4X^2 + 5X + 6$$
(3)

The objective function specification is similar to that formed in Section 4.0 The GP algorithm evolved the desired expression with optimal solution in the 65th iteration with zero errors and the GP TS of Figure 11. *X* is given a range of values from -10 to 10 with an interval of 1 and the plot is represented in Figure 12, and the plot of errors against generations is shown in Figure 13.

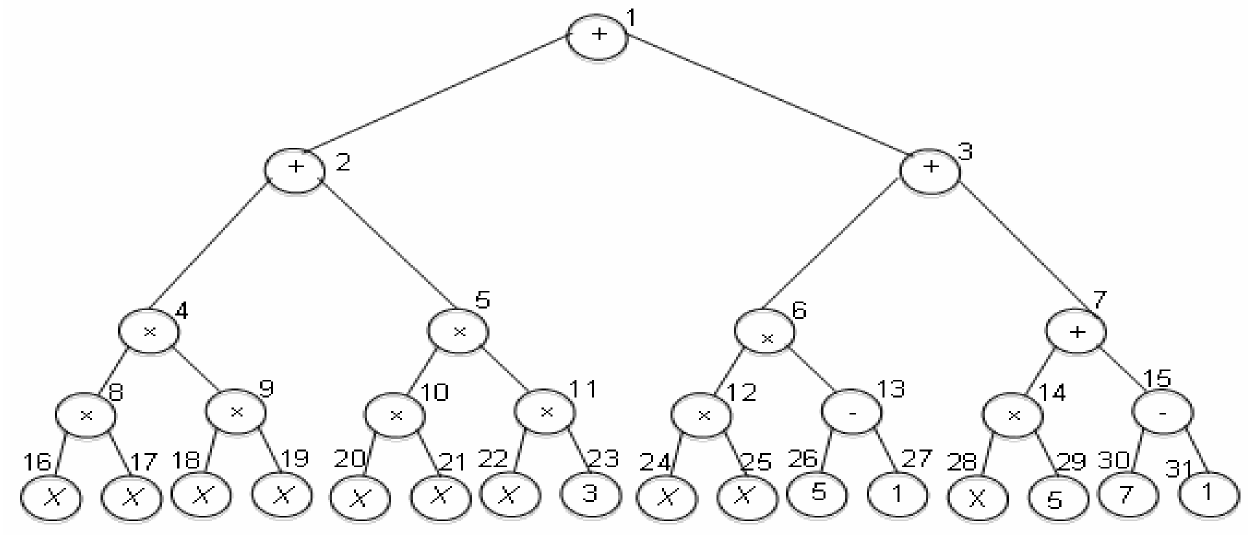

**Figure 11: 65th iteration GP evolved TS for expression in equation 3 with zero error.**

Careful analysis of the evolved TS of Figure 11 produces the expression simplified bellow:

$$Y = ((X \times X) \times (X \times X)) + ((X \times X) \times (X \times 3)) + ((X \times X) \times (5 - 1)) + ((X \times 5) + (7 - 1))$$

$$Y = (X^2 \times X^2) + (X^2 \times (3X)) + (X^2 \times 4) + 5X + 6$$

$$Y = X^4 + 3X^3 + 4X^2 + 5X + 6$$

From the above examination of the TS or transforming the TS into the equation, we can conclude that the algorithm is efficient because it has successfully evolved the original equation.

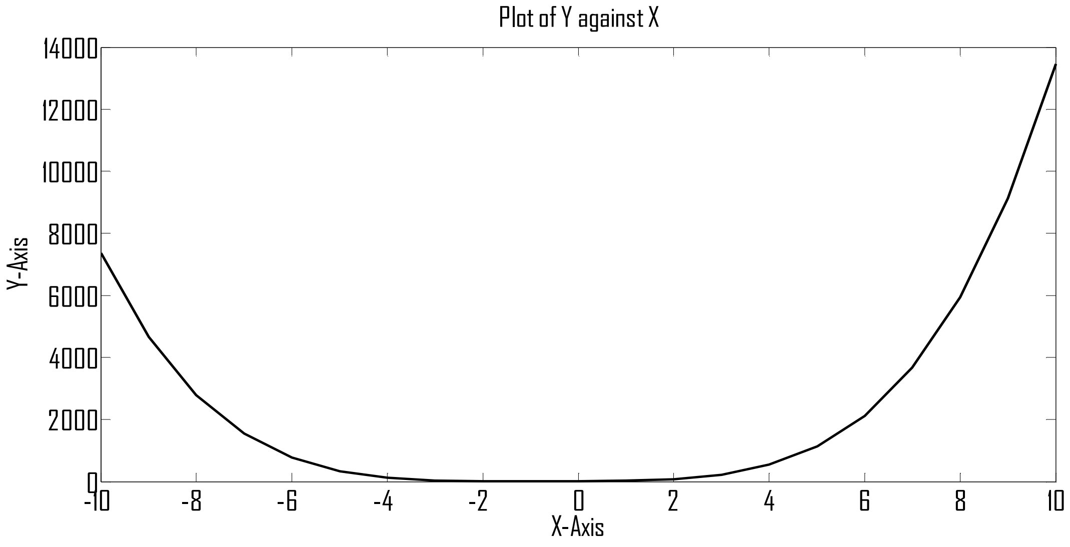

**Figure 12: Plot of Y against X for expression in equation 3 with zero error.**

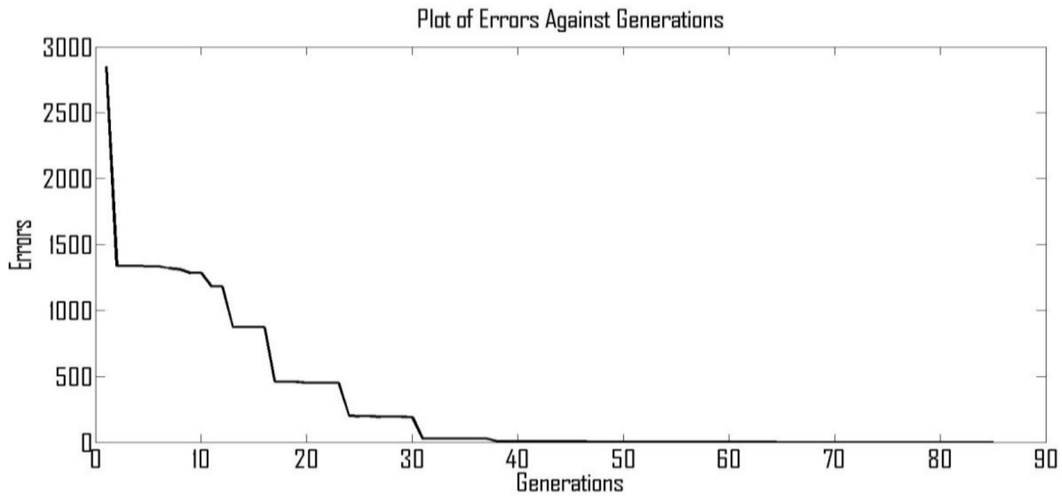

**Figure 13: Plot of errors against generations for expression in equation 3.**

## 4.3 Benchmark Testing Expression 3

$$Y = X^4 - 2X^2 + 1 \tag{4}$$

The objective function specification is similar to that formed in Section 4.0 The GP algorithm evolved the desired expression with optimal solution in the 30th iteration with zero errors and the GP TS of Figure 14. *X* is given a range of values from -10 to 10 with an interval of 1. The plot is represented in Figure 15 and the plot of errors against generations is shown in Figure 16.

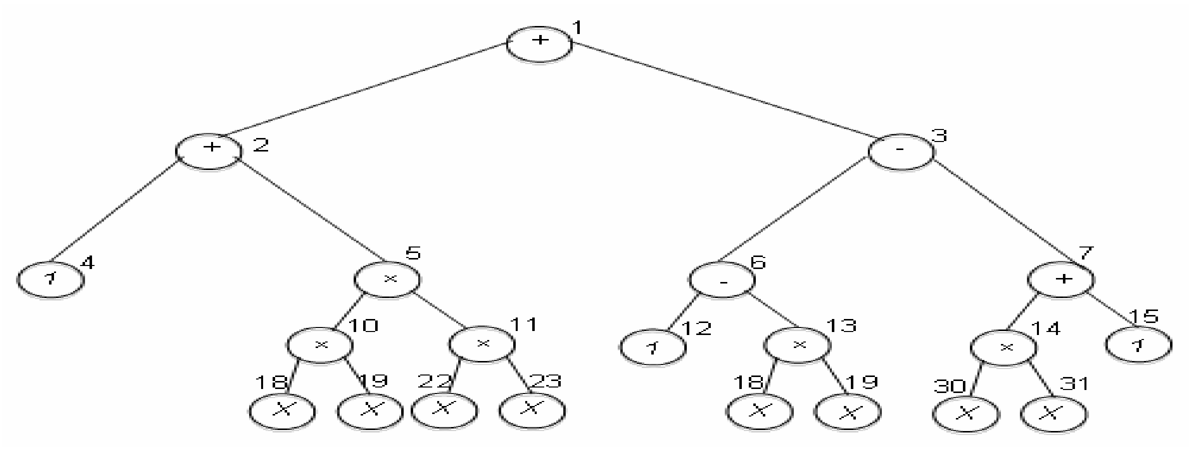

**Figure 14: 30th iteration GP evolved TS for expression in equation 4 with zero error.**

Critical analysis of the evolved TS of Figure 14 gives the expression simplified bellow:

$$Y = 1 + (X^2 \times X^2) + 1 - X^2 - (X^2 + 1) = Y = X^4 - 2X^2 + 1$$

From the investigation of the TS or transforming the TS into equation, we can deduce that the algorithm is efficient because it has successfully evolved the original equation.

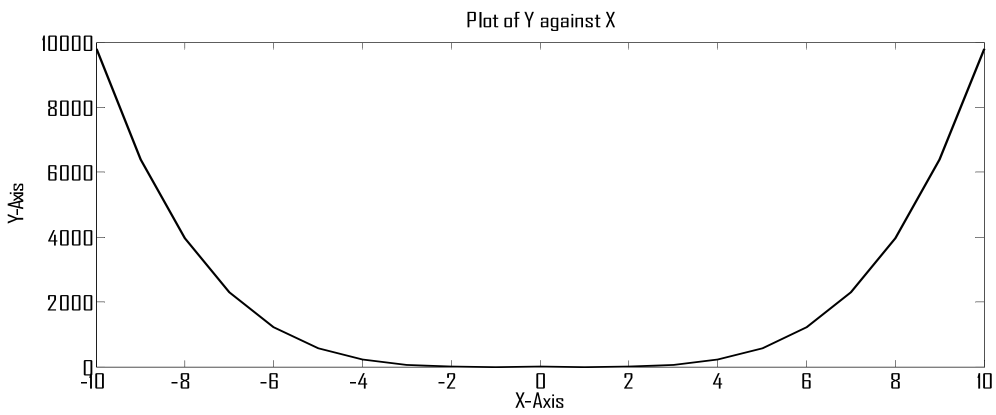

**Figure 15: Plot of Y against X for expression in equation 5.4 with zero error.**

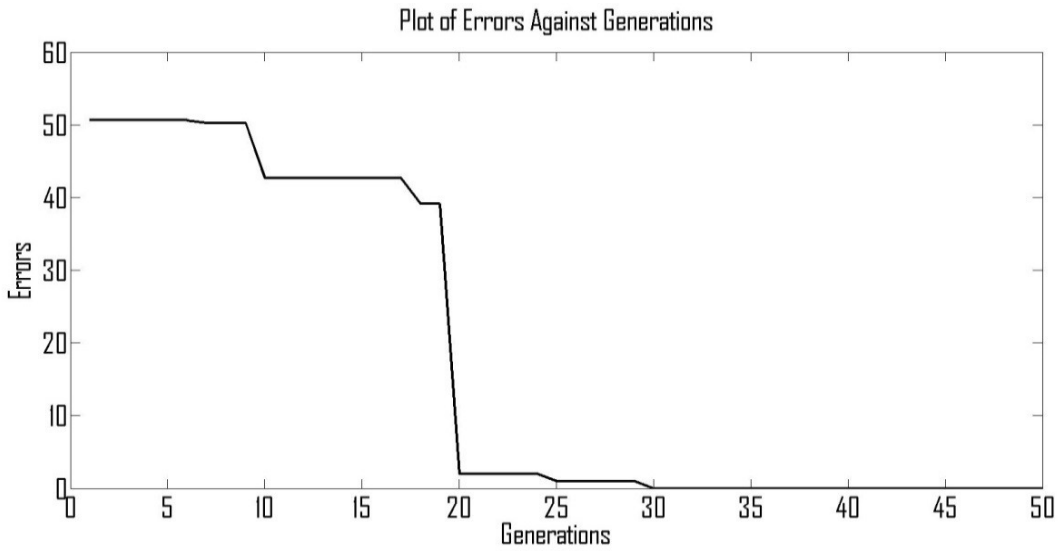

**Figure 16: Plot of errors against generations for expression in equation 5.4.**

## 4.4 Benchmark Testing Expression 4

$$Y = X^4 - 7X^3 + 3X^2 + 29X + 42 \qquad (5)$$

The objective function specification is similar to that formed in Section 4.0 The GP algorithm evolved the desired expression with optimal solution in the 86th iteration with zero errors and the GP TS of Figure 17. *X* is given a range of values from -10 to 10 with the interval of 1. The plot is represented in Figure 18 and the plot of errors against generations is shown in Figure 19.

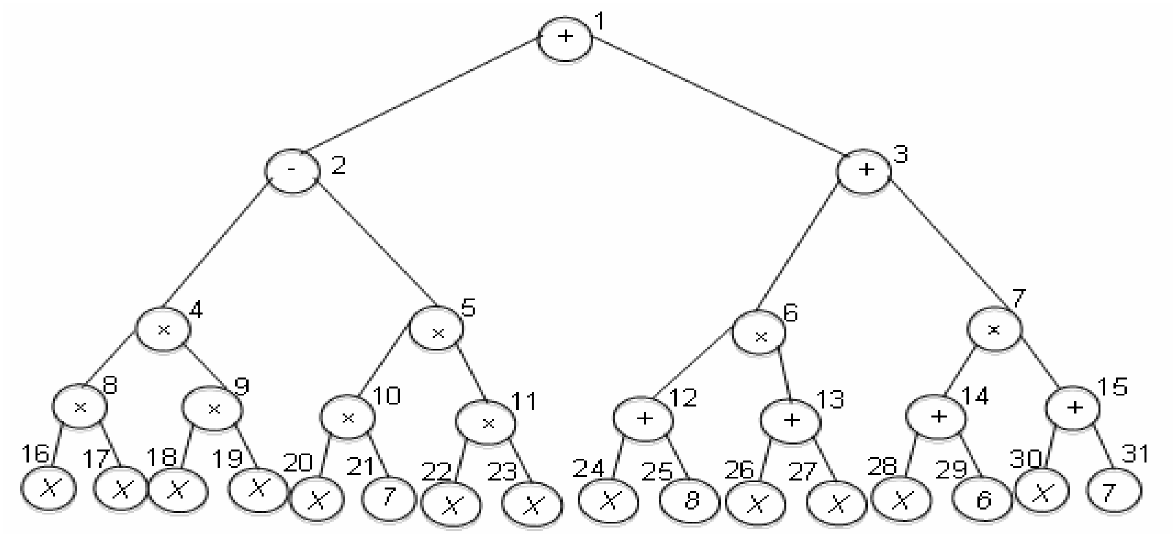

**Figure 17: 86th iteration GP evolved TS for expression in equation 5 with zero error.**

Careful analysis of the evolved TS of Figure 17 gives the expression simplified bellow:

$$Y = (X^2 \times X^2) - (7X \times X^2) + ((X+8) \times (X+X)) + ((X+6) \times (X+7))$$

$$Y = X^4 - 7X^3 + (8X + 8X + X^2 + X^2) + (X^2 + 7X + 6X + 42)$$

$$Y = X^4 - 7X^3 + 3X^2 + 29X + 42$$

From the examination of the TS or transforming the TS into the equation, we can deduce that the algorithm is efficient because it has successfully evolved the original equation.

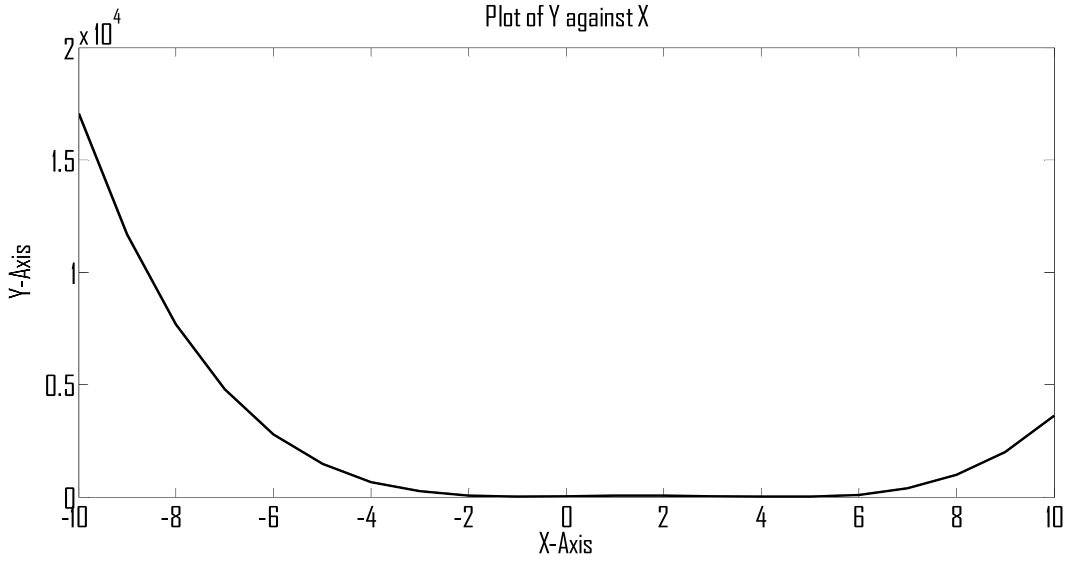

**Figure 18: Plot of Y against X for expression in equation 5 with zero error.**

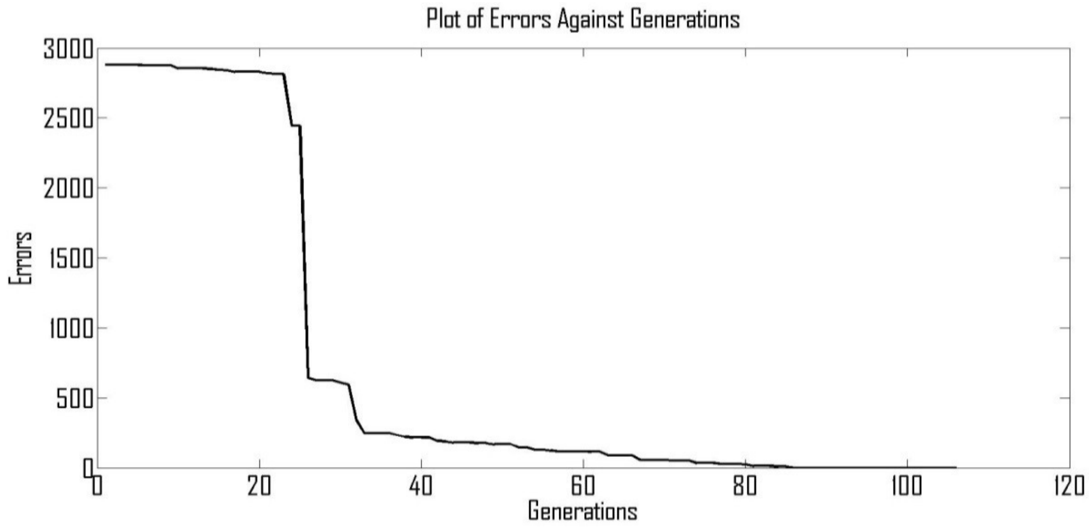

**Figure 19: Plot of errors against generations for expression in equation 5.**

## 5 Case Study: Active filter evolution

To evaluation how well evolving circuit has performed in the population with regard to the desired circuit, evolving circuits are extracted with the aid of GF and converted into a symbolic Netlist. Variable are represented in Matlab programme as single variable as described in Section 2.1.2 and later encoded again to individual element types. The same component type is encoded with unique subscripts to distinguish them if there are more than one element type in the same circuit. The encoding is made symbolically. Using resistor as one component type for illustration, all resistors are labelled *Y*. Supposing there are four resistors (4*Y*), there are being substituted by [‘*a-d’*] so that if an element is chosen and it is ‘*a*’, it is labelled $R_1$, if another element is chosen and it is ‘*b*’, it is labelled $R_2$, and so on. The evolving circuits in the form of TS are described thus: an operand terminates a branch (op-amp, inductor, capacitor and resistor) whereas an operator (parallel or series part) continues the TS. The TS is interpreted from top to bottom and from left to right. The branches that proceed after the operand are swapped with ‘0’. Likewise, the branches that proceed after the ‘0’ are swapped with ‘0’, so that all the branches that proceed after the operands are swapped with ‘0’ up to the maximum length of TS. All the ‘0’ elements are then removed to leave the remainder evolving circuit.

The stack separation evaluation technique is used to rearrange the GP evolved elements as it is connected. The series sets are numbered from 0 to the highest number, whereas the parallel sets are all numbered 0 since all are grounded apart from the special cases connected between nodes. The components labels are distinguished by subscript from 1 to the last element, for example, 4 resistors in a circuit are labelled as $R_1$ $R_2$ $R_3$ $R_4$. The Netlist formation is thus: if an element is picked; it is between a 1st node number and a 2nd node number. It is vital to note that, the series components are always connected to the next node number (in this case) that is not zero. For instance, the extract from the evolving circuit of Figure 24 is as follow:

$$+V+R|C+Z+R|C+R|C+Z+R|C+R|C+Z \qquad (6)$$

In equation (6), *Z* stands for op-amp and replacing the values of parallel and series part into equation (6) form equation (7) as:

$$0V1R0C2Z3R|C4R0C5Z6R|C7R0C8Z \qquad (7)$$

The second and the fourth capacitors are assigned with different letters and programmed in a special way because they are repeated in a regular pattern which makes it easy. The symbolic Netlist is formed thus: It starts with the element name, followed by node1, node2 and followed by its component value. If a circuit has op-amp as component(s), Netlist starts with it and number it from first to the last before other components follow. The formation continues thus: op-amp name, followed by its output node number, inverting node number and non-inverting node number

- $OAmp_1$ 3 2 3
- $OAmp_2$ 6 5 6
- $OAmp_3$ 9 8 9
- $V$ 0 1 component value
- $R_1$ 1 2 component value
- $R_2$ 3 4 component value
- $R_3$ 4 5 component value
- $R_4$ 6 7 component value
- $R_5$ 7 8 component value
- $C_1$ 0 2 component value
- $C_2$ 4 6 component value
- $C_3$ 0 5 component value
- $C_4$ 7 9 component value
- $C_5$ 0 8 component value

Gielen and Sansen [35] demonstrated how symbolic simulation is very useful when creating a large part of analytical prototype automatically. In this section, the modified symbolic circuit analysis in Matlab (MSCAM) discussed in detail in [36]. It uses Netlist automatically generated from simulation described in Section 5 to transform it to symbolic matrices. The symbolic matrices are then substituted with their real values (using the eval command in Matlab) to acquire frequency response. It is then compared with the specified frequency response set in the objective function. The process continues till the set frequency is acquired.

## 6 Circuit Simulation

The GP evolved desired circuit TS is shown in Figure 20, whereas its equivalent circuit is shown in Figure 21. It takes twenty minutes to evolve the circuit and ten iterations. The MSCAM frequency response of the evolved circuit is shown in red and PSpice simulation of the original circuit is indicated with black colour as indicated in Figure 22. The GP algorithm successfully evolved the circuit with feedback loop (that is repeated in regular pattern) just with little modifications in the code. It is easy to modify algorithm of existing circuit for another compared to human method that the whole process has to start over. The MSCAM original circuit specifications with error of 5.1093E-7 and a gain of 1.

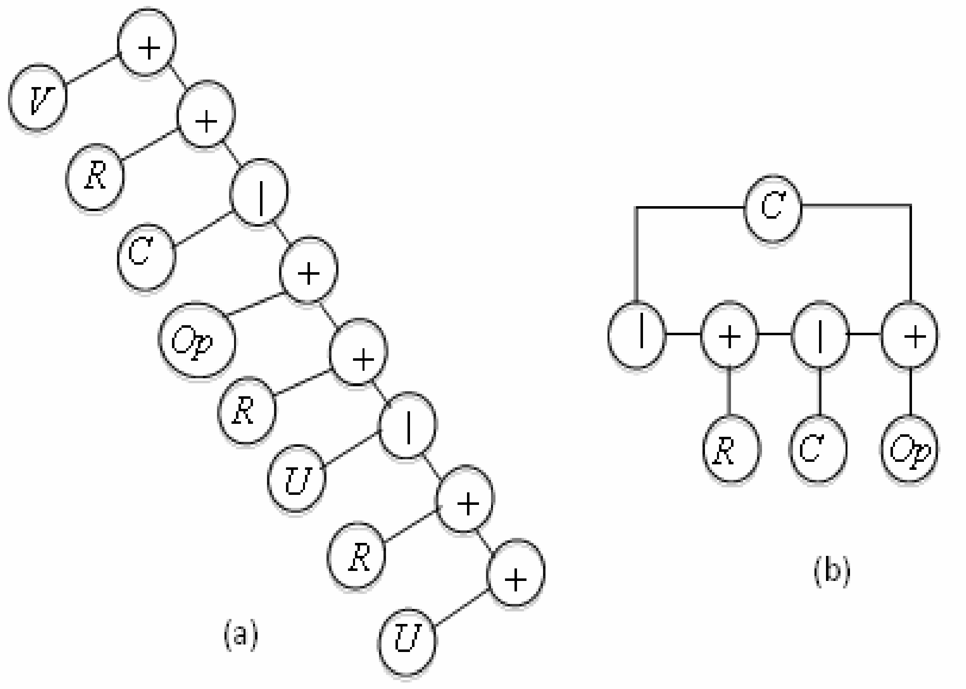

**Figure 20: (a) GP evolved TS for the active low-pass filter with feedback and (b) U representation.**

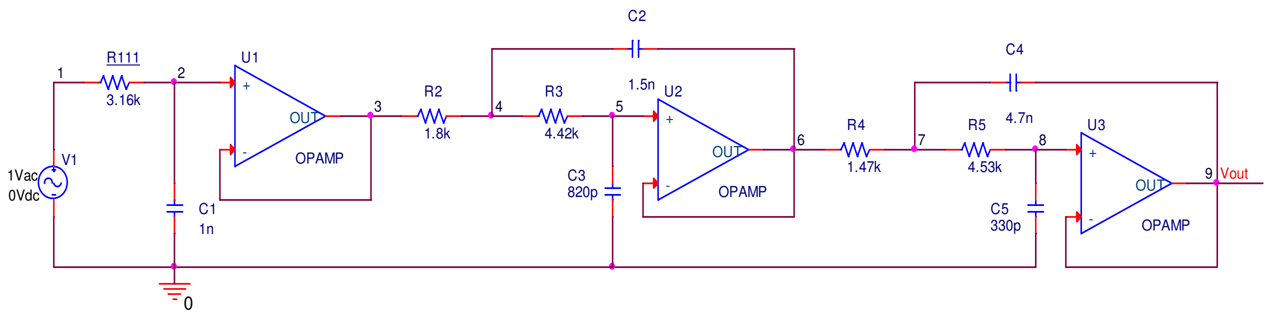

**Figure 21: GP evolved circuit for the active low-pass filter with feedback *[37]*.**

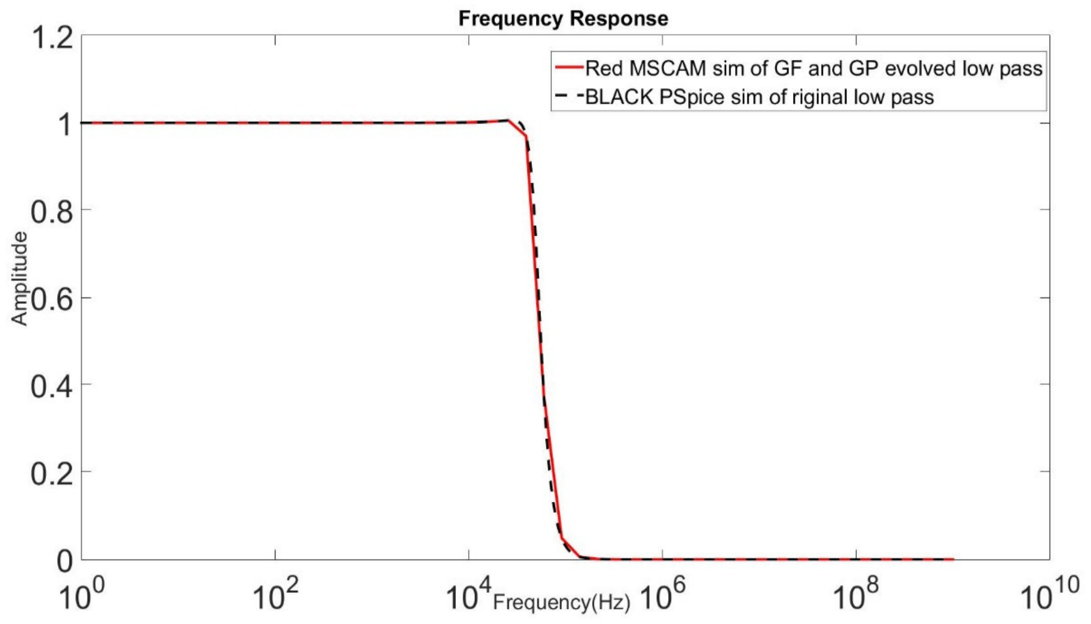

**Figure 22: Frequency response for GP evolved circuit (Red), PSpice simulation of GP evolved circuit (black), PSpice simulation of reduced GP evolved circuit (blue) and reduced GP evolved circuit with PSO (Green) for the active low-pass filter with feedback.**

## 8 Conclusions

This paper introduces GF concept to GP algorithm and used it developed a Matlab toolbox that is tested with four different benchmark functions. The algorithm is efficient because it successfully evolved the original benchmark equations. Also the algorithm was tested on one circuit and it successfully evolved the circuit.

## Acknowledgments

The lead author acknowledges the financial support from Tertiary Education Trust Fund (TetFund) through University of Calabar, Calabar, Nigeria. The authors appreciate the comments provided by the anonymous reviewers and the editor, which has help the authors to improve the quality of the paper.

### References

[1] M. Walker, "Introduction to genetic programming," *Tech.Np: University of Montana,* 2001.

[2] K. Rodríguez and R. Mendoza, "A Matlab Genetic Programming Approach to Topographic Mesh Surface Generation," 2011.

[3] S. Silva and J. Almeida, "GPLAB-a genetic programming toolbox for MATLAB," in *Proceedings of the Nordic MATLAB Conference,* 2003, pp. 273-278.

[4] P. Balasubramaniam and A. V. A. Kumar, "Solution of matrix Riccati differential equation for nonlinear singular system using genetic programming," *Genetic Programming and Evolvable Machines,* vol. 10, pp. 71-89, 2009.

[5] W. Weimer, S. Forrest, C. Le Goues and T. Nguyen, "Automatic program repair with evolutionary computation," *Commun ACM,* vol. 53, pp. 109-116, 2010.

[6] S. Forrest, T. Nguyen, W. Weimer and C. Le Goues, "A genetic programming approach to automated software repair," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation,* 2009, pp. 947-954.

[7] M. D. Schmidt and H. Lipson, "Solving iterated functions using genetic programming," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers,* 2009, pp. 2149-2154.

[8] Y. Lin and M. Tsai, "The Integration of a Genetic Programming-Based Feature Optimizer With Fisher Criterion and Pattern Recognition Techniques to Non-Intrusive Load Monitoring for Load Identification," *International Journal of Green Energy,* vol. 12, pp. 279-290, 2015.

[9] J. R. Koza, S. H. Al-Sakran and L. W. Jones, "Cross-domain features of runs of genetic programming used to evolve designs for analog circuits, optical lens systems, controllers, antennas, mechanical systems, and quantum computing circuits," in *Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference On,* 2005, pp. 205-212.

[10] J. R. Koza, "Human-competitive results produced by genetic programming," *Genetic Programming and Evolvable Machines,* vol. 11, pp. 251-284, 2010.

[11] H. Hou, S. Chang and Y. Su, "Economical passive filter synthesis using genetic programming based on tree representation." in *IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS,* 2005, pp. 3003.

[12] S. Chang and Y. Su, "Automated passive filter synthesis using a novel tree representation and genetic programming," *Evolutionary Computation, IEEE Transactions On,* vol. 10, pp. 93-100, 2006.

[13] K. K. Anumandla, R. Peesapati, S. L. Sabat, S. K. Udgata and A. Abraham, "Field programmable gate arrays-based differential evolution coprocessor: a case study of spectrum allocation in cognitive radio network," *IET Computers & Digital Techniques,* vol. 7, pp. 221-234, 2013.

[14] E. A. Coyle, L. P. Maguire and T. M. McGinnity, "Design philosophy for self-repair of electronic systems using the UML," *IEE Proceedings-Software,* vol. 149, pp. 179-186, 2002.

[15] W. Luo, Z. Zhang and X. Wang, "Designing polymorphic circuits with polymorphic gates: a general design approach," *IET Circuits, Devices & Systems,* vol. 1, pp. 470-476, 2007.

[16] S. Maheshwari, "Analogue signal processing applications using a new circuit topology," *IET Circuits, Devices & Systems,* vol. 3, pp. 106-115, 2009.

[17] J. F. Miller and P. Thomson, "Discovering novel digital circuits using evolutionary techniques," 1998.

[18] A. Tyrrell, R. Krohling and Y. Zhou, "Evolutionary algorithm for the promotion of evolvable hardware," in *Computers and Digital Techniques, IEE Proceedings-,* 2004, pp. 267-275.

[19] S. Vakili, S. M. Fakhraie and S. Mohammadi, "Evolvable multi-processor: a novel MPSoC architecture with evolvable task decomposition and scheduling," *IET Computers & Digital Techniques,* vol. 4, pp. 143-156, 2010.

[20] J. Wang, Q. S. Chen and C. H. Lee, "Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware," *Computers & Digital Techniques, IET,* vol. 2, pp. 386-400, 2008.

[21] A. Doboli and R. Vemuri, "Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions On,* vol. 22, pp. 1504-1520, 2003.

[22] J. D. Lohn, S. P. Colombano, G. L. Haith and D. Stassinopoulos, "A parallel genetic algorithm for automated electronic circuit design," in *Proc. of the Computational Aerosciences Workshop, NASA Ames Research Center,* 2000, .

[23] R. A. Vural, B. Erkmen, U. Bozkurt and T. Yildirim, "CMOS differential amplifier area optimization with evolutionary algorithms," in *Proceedings of the World Congress on Engineering and Computer Science,* 2013, .

[24] T. Sripramong and C. Toumazou, "The invention of CMOS amplifiers using genetic programming and current-flow analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions On,* vol. 21, pp. 1237-1252, 2002.

[25] A. Senn, A. Peter and J. G. Korvink, "Analog circuit synthesis using two-port theory and genetic programming," in *AFRICON, 2011,* 2011, pp. 1-8.

[26] J. R. Koza, F. H. Bennett III, D. Andre, M. A. Keane and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," *Evolutionary Computation, IEEE Transactions On,* vol. 1, pp. 109-128, 1997.

[27] X. Peng, E. D. Goodman and R. C. Rosenberg, "Robust engineering design of electronic circuits with active components using genetic programming and bond graphs," in *Genetic Programming Theory and Practice V*Anonymous Springer, 2008, pp. 185-200.

[28] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing,* vol. 4, pp. 87-112, 1994.

[29] J. R. Koza, F. H. Bennett III and O. Stiffelman, "Genetic programming as a Darwinian invention machine," in *Genetic Programming*Anonymous Springer, 1999, pp. 93-108.

[30] J. R. Koza, F. H. Bennett III and O. Stiffelman, "Genetic programming as a Darwinian invention machine," in *Genetic Programming*Anonymous Springer, 1999, pp. 93-108.

[31] J. F. Kennedy, J. Kennedy and R. C. Eberhart, *Swarm Intelligence.* Morgan Kaufmann, 2001.

[32] M. Mezher and M. F. Abbod, "A new genetic folding algorithm for regression problems," in *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference On,* 2012, pp. 46-51.

[33] M. Jamil and X. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation,* vol. 4, pp. 150-194, 2013.

[34] R. Feldt, M. O'Neill, C. Rayn, P. Nordin and W. B. Langdon, "GP-beagle: A benchmarking problem repository for the genetic programming community," *Late Breaking Papers at GECCO,* 2000.

[35] G. Gielen and W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits.* Springer Science & Business Media, 2012.

[36] O. J. Ushie, M. Abbod and E. Ashigwuike, "Matlab symbolic circuit analysis and simulation tool using PSpice netlist for circuits optimization," 2015.

[37] B. Carter and R. Mancini, "Op Amps for Everyone. [Sl]: Newnes," 2009.