Intelligent Optimisation of Analogue Circuits Using Particle Swarm Optimisation, Genetic Programming and Genetic Folding

by

Ogri James Ushie

A thesis submitted for the degree of

Doctor of Philosophy

Department of Electronic and Computer Engineering

College of Engineering, Design and Physical Sciences

Brunel University London

April 2016

Abstract

This research presents various intelligent optimisation methods which are: genetic algorithm (GA), particle swarm optimisation (PSO), artificial bee colony algorithm (ABCA), firefly algorithm (FA) and bacterial foraging optimisation (BFO). It attempts to minimise analogue electronic filter and amplifier circuits, taking a cascode amplifier design as a case study, and utilising the above-mentioned intelligent optimisation algorithms with the aim of determining the best among them to be used. Small signal analysis (SSA) conversion of the cascode circuit is performed while mesh analysis is applied to transform the circuit to matrices form. Computer programmes are developed in Matlab using the above mentioned intelligent optimisation algorithms to minimise the cascode amplifier circuit. The objective function is based on input resistance, output resistance, power consumption, gain, upper-frequency band and lower frequency band. The cascode circuit result presented, applied the above-mentioned existing intelligent optimisation algorithms to optimise the same circuit and compared the techniques with the one using Nelder-Mead and the original circuit simulated in PSpice. Four circuit element types (resistors, capacitors, transistors and operational amplifier (op-amp)) are targeted using the optimisation techniques and subsequently compared to the initial circuit. The PSO based optimised result has proven to be best followed by that of GA optimised technique regarding power consumption reduction and frequency response.

This work modifies symbolic circuit analysis in Matlab (MSCAM) tool which utilises Netlist from PSpice or from simulation to generate matrices. These matrices are used for optimisation or to compute circuit parameters. The tool is modified to handle both active and passive elements such as inductors, resistors, capacitors, transistors and op-amps. The transistors are transformed into SSA and op-amp use the SSA that is easy to implement in programming. Results are presented to illustrate the potential of the algorithm. Results are compared to PSpice simulation and the approach handled larger matrices dimensions compared to that of existing symbolic circuit analysis in Matlab tool (SCAM). The SCAM formed matrices by adding additional rows and columns due to how the algorithm was developed which takes more computer resources and limit its performance.

Next to this, this work attempts to reduce component count in high-pass, low-pass, and all- pass active filters. Also, it uses a lower order filter to realise same results as higher order filter regarding frequency response curve. The optimisers applied are GA, PSO (the

best two methods among them) and Nelder-Mead (the worst method) are used subsequently for the filters optimisation. The filters are converted into their SSA while nodal analysis is applied to transform the circuit to matrices form. High-pass, low-pass, and all- pass active filters results are presented to demonstrate the effectiveness of the technique. Results presented have shown that with a computer code, a lower order op-amp filter can be applied to realise the same results as that of a higher order one. Furthermore, PSO can realise the best results regarding frequency response for the three results, followed by GA whereas Nelder-Mead has the worst results.

Furthermore, this research introduced genetic folding (GF), MSCAM, and automatically simulated Netlist into existing genetic programming (GP), which is a new contribution in this work, which enhances the development of independent Matlab toolbox for the evolution of passive and active filter circuits. The active filter circuit evolution especially when operational amplifier is involved as a component is of it first kind in circuit evolution. In the work, only one software package is used instead of combining PSpice and Matlab in electronic circuit simulation. This saves the elapsed time for moving the simulation between the two platforms and reduces the cost of subscription. The evolving circuit from GP using Matlab simulation is automatically transformed into a symbolic Netlist also by Matlab simulation. The Netlist is fed into MSCAM; where MSCAM uses it to generate matrices for the simulation. The matrices enhance frequency response analysis of low-pass, high-pass, band-pass, band-stop of active and passive filter circuits. After the circuit evolution using the developed GP, PSO is then applied to optimise some of the circuits. The algorithm is tested with twelve different circuits (five examples of the active filter, four examples of passive filter circuits and three examples of transistor amplifier circuits) and the results presented have shown that the algorithm is efficient regarding design.

Dedication

The thesis is dedicated to God Almighty, for the wisdom and understanding to carry out the research.

Also, I dedicate this doctorate work to my dear wife (Onyodo) and children (Ashikem, Inung and Adaoshi) for their prayers, sacrifices and patience. Besides, I dedicated it to my lovely parents (Father and my late Mother), my brothers and sisters.

Acknowledgements

I would like to appreciate those who supported me in one way or the other to realise the dream of completing this research. A huge thanks to Dr Maysam F. Abbod (Reader), whom I am indebted to, for his continuous guidance and advice during the research. I also appreciate his support, encouragement and patience throughout the research period, and above all, I acknowledged him for the opportunity given to me to do and achieve this doctorate. He is a great mentor, friend and supervisor whom I have learnt so much from during the research. I also appreciate the suggestions, time and dedication of my second supervisor, Dr Tatiana Kalganova.

A big thank you goes to the Department of Electronic and Computer Engineering, Prof Tassos Karayiannis, Dr Ali Mousavi, Dr Philip Collins and the Management of Brunel University London for their support.

Also, I am grateful to Mohammed Al-Shammaa, Mary-Jane Sule, Dr Emeka Dumbili, Jasper Ojoghoro and Pauldy Otermans for their suggestions and editorial assistance.

I am grateful to the former Vice-Chancellor of University of Calabar; Prof James Epoke, who supported me to realise my dream. I would like to appreciate the financial assistance from Tertiary Education Trust Fund (TETFUND) through the University of Calabar, Calabar, Nigeria.

Declaration

I certify that no art of this thesis has been previously submitted for a degree nor has it been submitted as part of the requirements for a degree. I also certify that this thesis has been written by me. Any assistance that I received in my research work and the preparation of the thesis itself has been duly acknowledged and referenced.

Signature of Student

Ogri James Ushie

April 2016, London

Table of Contents

List of Figures

List of Tables

List of Nomenclature

ABCA	Artificial Bee Colony Algorithm
AMS	Analogue Mixed Signal
AI	Artificial Intelligence
AVR	Automatic Voltage Regulator
BFO	Bacterial Foraging Optimisation
BCO	Bee Colony Optimisation
APSO	Accelerated Particle Swarm Optimisation
CSD	Canonical Signed Digit
CFA	Chaotic Firefly Algorithm
CSI	Computational Swarm Intelligence
CRPSO	Craziness-Based Particle Swarm Optimisation
DE	Differential Evolution
DRO	Dynamic Rounding-Off
EWB	Electronic Work Bench
EH	Evolvable Hardware
EP	Evolutionary Programming
ES	Evolutionary Strategies
FA	Firefly Algorithm
GA	Genetic Algorithm
GABFA	Genetic Algorithm Bacterial Foraging Algorithm

GEP	Gene Expression Programming
GF	Genetic Folding
GP	Genetic Programming
HBPSO	Human Behaviours Based Particle Swarm Optimisation
KVL	Kirchhoff's Voltage Law
LBG	Linde-Buzo-Gray
LUT	Look-Up Table
LB	Lower Bound
MSE	Mean Square Error
MSCAM	Modified Symbolic Circuit Analysis in Matlab
NA	Nodal Analysis
NUFB TMUX	Non-Uniform Filter Bank Trans-Multiplexer
Op-amp	Operational Amplifier
OTA	Operational Trans-Conductance Amplifier
PIC	Peripheral Interface Controller
Pc	Crossover Rate
PECS	Power Electronics Circuits
PID	Proportional Integral Derivative
P_m	Mutation Rate
PSO	Particle Swarm Optimisation
QMF	Quadrature Mirror Filter

QPSO	Quantum Particle Swarm Optimisation
RMS	Root Mean Square
SCAM	Symbolic Circuit Analysis in Matlab
SI	Swarm Intelligence
SPICE	Simulated Programme with Integrated Circuit Emphasis
TS	Tree Structure
UB	Upper Bound
VLSI	Very Large Scale Integration
VHDL-AMS	Very-High-Speed Integrated Circuit Hardware Description Language-Analogue Mixed Signal

Chapter 1

Introduction

1.1 Introduction

Electronics is a field in technology and physics concerned with circuit design using microchips and transistors, and with the movement and behaviour of electrons in a vacuum, semiconductor, conductor or gas. The application of electronics to human activities has become a part of life and almost all aspects of human endeavours required it for better functionality or productivity. Electronics is a very rapid growing industry because of its demand in a day-to-day application and the need to overcome some challenges facing the society. The objective of an electronics engineer is to design circuits that are smaller, faster and cheaper. Other objectives are: to reduce power consumption and increase system reliability.

Electronics is growing on a daily basis and one of the latest advances is a terahertz (THz) transmitter [1]. Panasonic, the Japanese National Institute of Information and Communications Technology and Hiroshima University claim to have produced a terahertz (THz) transmitter capable of signal transmission of data at a per-channel rate of over 10 Gbit/s through multiple channels at around 300 GHz. The technology is the latest development in wireless communication which can transmit data at rates ten times higher than existing technology allows [1].

Secondly, in the UK, a collaborative project for developing sensing technologies that monitor machined metal parts is ongoing [2]. The project brings together partners to integrate research and end users, Centre for Process Development (CPI), with the technology and industrial scale manufacturing, BAE Systems, Element Six, Advanced Manufacturing Ltd, The Advanced Manufacturing Research Centre, Printed Electronics Limited, DMG Mori Seiki and The National Physical Laboratory involved. The ‘Intelligent Tooling’ project is developing electronic components and embedded sensors within high-value machining usages in manufacturing sectors such as aerospace, automotive, rail, energy and marine. Part of the research team is aiming at designing and printing the electronic sensors, offering expertise toward integration of printable and conventional electronics [2]. Every day

electronic devices are changing in size; as such, there is a need for more research regarding electronics miniaturisation.

1.2 Research Questions

The world physical interpretation is analogue in nature that makes analogue circuits very vital in circuit design. Although the quantity of digital circuit design outnumbered that of analogue design, most digital circuit modelling requires the analogue module for interfacing to the external world. This thesis intends to combine disparate concepts in new ways to investigate a conventional circuit system optimisation. The research questions are:

- Whether introducing the concept of component count reduction in passive and active filter circuits will reduce the size, power consumption and increase circuit reliability.
- If there is an improvement on existing symbolic circuit analysis in Matlab (SCAM) whether its capacity (matrices dimension) will be enhanced to handle 30 by 30 or more so that it can be used to simulate complex circuits. This is important especially when operational amplifier (op-amp) is involved as circuit component compared existing one that cannot handle matrices dimension more than eight by eight. The SCAM formed matrices by adding additional rows and columns due to how the algorithm was developed which takes more computer resources and limit its performance.
- Whether combining the concepts: genetic programming (GP), genetic folding (GF), modified symbolic circuit analysis in Matlab (MSCAM) and automatically generated Netlist for the evolution of passive and active filter circuits will aid to develop an independent Matlab toolbox. The simulator uses only Matlab compare to existing GP which combine Matlab and PSpice.

Power consumption in electronic circuit designs has been a source of concern for engineers because of its effect on the environment. The more a system is complex, the less reliable it would be or in other words, the fewer the elements in a system, the greater the reliability of the system. Electronic circuit's minimisation increases system reliability, reduces power consumption and reduces component count and size. Also, optimisation can be used to vary component values if the desired component values are not available in a developing country where some component values seem to be a problem.

The process of converting electrical circuit into its equivalent matrices requires tedious mathematical computation. Because it is a human method, it is also prone to error or may take longer time and each circuit has to undergo the same process each time a circuit has to be solved. Instead of combining two platforms (PSpice and Matlab) for electronic circuit simulation, only Matlab is used in this research to develop an algorithm for the evolution of filter circuits. This reduces elapsed time used for transferring the simulation to and from the two packages, and can also serve as a useful tutorial on how to use GP in Matlab to design analogue circuits. Simulation programme with integrated circuit emphasis (Spice) is also known as PSpice and the latter is used throughout the thesis.

1.3 Circuits Optimisation

A circuit optimisation is a process of finding the best or an alternative design for existing electronic circuits. Circuit optimisation helps to reduce component count, cost, size and increases system reliability in circuits. There have been advances in analogue circuit optimisation. Among such advances in recent time is the introduction of a look-up table (LUT) based analogue design automation. The LUT algorithm is used to extract circuit parameters from complex physics-based models of transistor used by PSpice [3].

1.4 Motivations

The motivation of this research is centred on the fact that most optimisation packages are expensive, not independent, not flexible, and not open access. Therefore, this work is motivated to develop a standalone optimisation algorithm that is flexible, open source and less expensive because only one simulator is required. The Matlab toolbox uses only Matlab software compare to existing GP which combine Matlab and PSpice software packages which will reduce payment of subscription to two software.

1.5 Aim and Objectives

The overall aim of this piece of research is to embark on analogue electronic circuit optimisation in terms of component count reduction, improve on existing algorithms and to develop a toolbox or an independent algorithm that can be used as a tutorial in Matlab for circuit evolution.

- This research objective is to survey optimisation methods to identify the best method and use it to optimise analogue circuit.

- This work intends to search for independent optimisation techniques if any and possible improvement if any.
- Lastly, one of its objectives is to develop an automated independent optimisation algorithm for electronic circuit evolution without combining with other packages.

1.6 Thesis Contributions

To the best of my knowledge regarding this research and since most of my contributions to knowledge or the ideas have been published by me, several findings from the research are considered significant. The following summarises the main contributions of this research:

- The work presents genetic algorithm (GA), firefly algorithm (FA), bacterial foraging optimisation (BFO), artificial bee colony (ABC), and particle swarm optimisation (PSO). These algorithms are used because there are intelligent methods to justify the research topic. Analogue electronic circuits are optimised using cascode amplifier by applying all these artificial intelligent algorithms for the purpose of identifying the best algorithm.
- The research attempts to reduce component count in high, low, and all pass active filters. Also, a lower order filter is simulated to achieve the same results as that simulated with higher order ones as regards their frequency response.
- Modified Matlab symbolic circuit analysis and simulation tool that generates matrices that make use of Netlist from PSpice are presented. The matrices can be applied to calculate circuit components or for optimisation.
- The research introduces the use of GF, MSCAM and GP for the evolution of active and passive filter circuits. Instead of combining PSpice and Matlab in electronic circuit simulation, the work only used Matlab. This reduces elapsed time use for transferring simulation between the software packages and reduces cost of subscription.

1.7 Thesis Overview

This thesis is comprised of seven chapters. Chapter 1 (this chapter) which is made up of the following sections: introduction, research questions, circuit optimisation, motivations, aim & objectives, thesis contributions, thesis overview and list of publications. The remaining chapters of the thesis are briefly described below:

In Chapter 2, a detailed background theory, literature survey, the concept, principle of inspiration of various evolutionary algorithms and artificial intelligent methods are presented. The developer and the period in which the algorithms were developed are also stated and some area applied.

Chapter 3 describes MSCAM and its applications in an electronic circuit simulation are illustrated. It improves on existing SCAM so that the developed algorithm can handle matrices dimension more than a matrix size of eight by eight.

Chapter 4 presents circuit optimisation. One circuit is used as an example to implement five different artificial intelligent methods (GA, FA, BFO, ABCA, and PSO) with a sole aim to determine the best method among them.

In Chapter 5, the use of GF, MSCAM, GP and automatically generated symbolic Netlist to develop an independent algorithm is presented. The benchmark testing of the developed algorithm for its efficiency with four mathematical functions is implemented. The algorithm evolved the expression in the form of tree structure illustrated in the chapter.

In Chapter 6, application of EA in analogue circuit evolution as an evolvable hardware (EH) is described. The chapter emphasises the use of GF, MSCAM, GP and automatically generated symbolic Netlist for the evolution of analogue circuit such as low-pass, high-pass, band-pass and band-stop for both passive and active filter circuit. In other words, the developed algorithm in Chapter 5 is modified and applied in analogue circuit evolution.

Chapter 7 is the conclusions and future work. It illustrates deduction based on results & discussion and possible future work.

1.8 List of Publications

The highlights of the publications are:

Conference Papers/Posters

- O. J. Ushie and M. F. Abbod, "Intelligent Optimization Methods for Analogue Electronic Circuits: GA and PSO Case Study," *International Conference on*

Machine Learning, Electrical and Mechanical Engineering (ICMLEME'2014), on Jan. 8-9, 2014 Dubai (UAE), pp. 193-199, 2014.

- O. J. Ushie "Intelligent Minimisation Methods for Analogue Electronic Circuits Using PSO and GA," *Brunel University Research Student Conference,* 11 – 12 March, 2014.

- O. J. Ushie "Intelligent Minimisation Methods for Analogue Electronic Circuits Using PSO and GA," *School of Engineering and Design Research Student Conference, Brunel University London,* 23 – 26 June, 2014.

Journal Papers

- O. J. Ushie, M. F. Abbod, E. C. Ashigwuike and S. Lawan, "Constrained Nonlinear Optimization of Unity Gain Operational Amplifier Filters Using PSO, GA and Nelder-Mead'' The International Journal of Intelligent Control and Systems (IJICS), vol. 20, pp. 26-34, 2015.
- O. J. Ushie, M. Abbod, and E. C. Ashigwuike, "Naturally Based Optimisation Algorithm for Analogue Electronic Circuits: GA, PSO, ABC, BFO, and Firefly a Case Study," Journal of Automation and Systems Engineering (JASE), vol.9 issue 3, pp 173-184, 2015
- O. J. Ushie, M. F. Abbod, and E. C. Ashigwuike, "Matlab Symbolic Circuit Analysis and Simulation Tool using PSpice Netlist for Optimization," International Journal of Engineering and Technology Innovation vol. 5, pp. 75-86, 2015.
- O. J. Ushie, M. Abbod, and Brian E. Usibe, "Genetic Folding/Programming Toolbox: Analogue Circuit Design Case Study," Journal of Automation and Systems Engineering (JASE), vol.10 issue 1, pp 40-64, 2016
- O. J. Ushie, M. F. Abbod, and Julie C. Ogbulezie, "The Use of Genetic Programming to Evolve Passive Filter Circuits" International Journal of Engineering and Technology Innovation, (submitted 13/01/2016 under review).
- O. J. Ushie, M. F. Abbod, and E. C. Ashigwuike, "Evolution of Active Filter Circuits Design Using Genetic Programming" International Journal of Electronics and Communications, Elsevier (submitted 19/05/2016 under review).

Chapter 2[1]

Literature Review

2.1 Introduction

Optimisation is a process of finding an optimal solution for a model [6]. As long as the society continues to exist, the need to improve its standard of living will also continue. Societal problems arise from a diverse field such as engineering, manufacturing, finance, music, medicine, computational art, chemistry and physics. The desire to obtain the best solution is faced in day-to-day life and cannot be overemphasised. Optimisation is applied in our everyday activities to minimise or maximise something. An organisation minimises cost, maximises profits and maximises performance. Tourists maximise their enjoyment to a minimal cost during a holiday.

This chapter surveys various intelligent optimisation techniques applied in this research to optimise analogue electronic circuits. The methods surveyed are: GA, GP, GF, PSO, FA, ABCA and BFO.

2.2 Artificial Intelligence

Artificial intelligence (AI) which is the study of computer techniques that emulate aspects of human intelligence or writing computer programmes that emulate the behaviour of organisms to solve a problem [7]. It can also be defined as the enterprise of constructing a physical symbols system that can reliably pass the Turning test [8]. Turning test was named after a famous man who was working as a director of programming at Manchester University, who contributed to artificial intelligence. Turning developed a concept known as 'Turning test'. The test involves a person communicating through teletype with an unidentified party that might be either a computer or another person. If the computers at the other destination response in a humanlike way, it may fool the person into thinking it is another human [9]. AI involves three things: knowledge representation, search and application of these ideas.

Search is the process of solving a problem where the basic technique or method being applied involves examining many possibilities while finding a solution. Planning for

[1] Majority of Chapter 2 has been published in [4, 5].

Christmas vacation involves search. One may decide to visit wife's family, one's family or travel to the US. If one decides to travel to the US, one may search for a flight, may rent a car or a hotel and so on. The knowledge representation in AI involves the research studies of the problem to find a language to encode the ideas so that the computer can use it. Knowledge representation and search form the core of the AI. The application involves natural language processing and vision [9].

The optimisation methods are developed based on the behaviour of organisms that are translated into algorithms. These algorithms are applied to write computer programmes to optimise the analogue circuits and used for other applications mentioned in the introduction of this chapter. Classification of the algorithm gives a clear understanding on how to analyse it and how it works. Algorithms that apply similar problem-solving technique can be grouped together. Algorithm can be labelled or classified as:

- Deterministic versus randomised: Deterministic algorithms yield on a given set of input the same results and always follow the same computational steps. On the other hand, randomised or stochastic algorithm has some randomness introduced by random function. Solutions in the population are always different each time the programme is run due to the random function. Although the final results may be of no big difference, the path of individual is not repeated on the same input. The randomised algorithms have the effect of disturbing the input, easy to implement and superior to deterministic regarding runtime [10].
- Offline versus online: online algorithms do not know their input initially, but the input is supplied online; an example of an online algorithm is ski, whereas offline algorithms know their input at the beginning [10].
- Exact versus approximate versus heuristic versus operational: Exact algorithm aims at computing an optimal solution given such a specified goal. Often, it is quite expensive regarding memory, run time and not possible for large input. Approximation algorithm aims at computing the solution that is never worse than a factor or guaranteed factor worse than optimal solution example travelling salesman problem. Heuristic algorithms find the optimal solution without providing a guarantee which they always do. The operational algorithm does not optimise the objective function but chain a sequence of computational operations directed by expert knowledge, for example ClustalW [10].

- Other classes of algorithms according to the main concept are listed as: divide and conquer algorithms, greedy algorithms, simple recursive algorithms, dynamic programming algorithms, backtracking algorithms, and branch and bound algorithms [10].

2.3 Evolutionary Computing

An algorithm is defined as any well-defined computational process that takes a set of values or some values as input and yield set of values or some values as output. In other words, an algorithm is a series of computational steps that change the input into output [10]. An algorithm is referred to be correct if, for every set of input instance, it produces the correct output as solution. It can be detailed as a hardware design or even as in English as computer code. The only condition is that the description must provide precise steps of the computational process to be followed. An optimisation algorithm is the process that is executed iteratively and also involves comparing of various solutions till a satisfactory or an optimum solution is found [11].

An evolutionary algorithm (EA) has been in use for the past decades for the provision of solutions for many engineering and computer science problems. The EA is classified into: GA [12], GP [13], GF [14], evolutionary programming (EP) [15], evolutionary strategies (ES) [16], gene expression programming (GEP) [17] and differential evolution (DE) [18, 19]. The following operations are involved while considering evolutionary computing: encoding, initialisation, fitness, selection, operators and termination.

2.3.1 Encoding

Encoding is the process of representing strings of genes called potential solutions (individuals as in GA whereas particles as in PSO). GA encodes potential solution typically in a form of a real string or a form of a fixed-length string of binary numbers [20]**.** PSO represents its potential solution known as particles in a form of a fixed-length of real-valued vector or a form of a real string or a form of a fixed-length string of binary numbers. GP represents its potential solutions in a form of a variable-sized of tree structure (TS) of values and functions [21]. The encoding process is dependent on the types of technique to be used and the problem involved.

2.3.2 Initialisation

Initialisation of a set of a population size is a first step in the EA. Each population has a number of particles or individuals, encoding sets and parameters. It is important to specify how many individuals or particles the population has. A small population size may converge too quickly. It is better that the first population size specification should have a large number of individuals or particles to be able to explore the whole search space that may consume the memory resources and time. Therefore, other genetic parameters and the population size are chosen carefully.

2.3.3 Fitness

In EA, the fitness function of an individual is the measured value of an objective function that provides a measure of how well individuals have performed in a problem space. In calculating fitness, the particle or individual has to be first decoded, evaluated and then determine how well it has performed as regard objective function. Therefore, each individual or particle is evaluated depending on how well the particle or individual is closed to the optimal solution called fitness value. These values guide the search to direct the individuals or particles toward an optimal solution. The thesis makes use of the error that is the RMS value of the difference between the objective function and the evolving function within a frequency range.

2.3.4 Selection

GA uses the best solution (solution with high fitness) to pass to the next generation called a selection of the fittest [11]. Selection method determines the number of trials or times an individual is selected for reproduction. In other words, it determines which and how many parents to be chosen, how many offspring to create, and which individuals will be swapped with the next generation. The most widely used method is roulette wheel method. It gives each individual its fitness value and depending on these values, a chosen individual will survive to the next generation.

2.3.5 Operators

The operators are mutation and crossover.

2.3.5.1 Crossover

It is the process of swapping parts of solution(s) with another in chromosomes with the purpose of mixing the solution [11]. Crossover is the procedure of taking two chromosomes and using them to reproduce new offspring. After the selection, the population is made up of better individuals. Selection does not create new ones. Crossover operator is applied to the search domain with the hope of creating good offspring or the main task is to mix the solutions. Crossover involves three steps:

a. Two pairs of chromosomes for the mating are randomly selected. If two parents (A, B) are being selected:

00100/00101110101 (B)

10101\00101101010 (A)

b. The crossover point is created at random along the chromosome length. For a one-point crossover as in this case, the random point is = 5, the head is the left part of the cross point and the remainder is the tail. The swap parts take the form to create new offspring:

Head (B) | Tail (A) = e;

Head (A) | Tail (B) = f;

c. Finally, the chosen parts are replaced between the two chromosomes (A, B) depending on the cross point and connected again by (|) operator as follows:

0010000101101010 (e)

10101**00101110101** (f)

Crossover operator proceeds if the two parents have a percentage for the mating less than the probability value P_c. However, the P_c value is known as the crossover rate that depends on either an adapted value or a fixed value.

2.3.5.2 Mutation

It is the process of changing parts of solution randomly, thereby increasing the diversity of the population and avoiding the algorithm to converge to local optimum [11]. Crossover exploits the newest solution to finding better ones, whereas mutation explores the entire search domain. The mutation operator is introduced to avoid early convergence to local optima by randomly sampling new points in the search domain. The mutation rate is represented as P_m. The mutation operator process as follows:

101011(B)

10**00**1(b)

Notice only two genes have been changed 3rd, 5th.

2.3.6 Termination

The termination condition is the number of generations the algorithm should run or certain conditions the algorithm should satisfy before it stops.

2.4 Evolutionary Algorithm Classification

EA are globally oriented, straightforward to apply in problems where there is little or no knowledge about the solution to the problem. Because EA is random in nature and it needs no derivative information, it is able of searching in the solution domain with greater possibility of locating the global solution. Among the EA mentioned in Section 2.3, the review will only cover GA, GP and GF which are used in this research.

2.4.1 Genetic Algorithm

GA was developed by John Holland and co-workers in the 1960s and 1970s; it applies the Charles Darwin's theory of evolution based on survival of the fittest. Holland was the first to use selection, mutation, recombination and crossover in the study of an artificial system [11]. GA is a population–based stochastic technique that makes use of the principle of survival of the fittest to produce a better solution [22]. Individuals in the population are encoded accordingly as strings. After the decoding, the fitness is evaluated which serve as criteria for selection of pairs of individuals for the next reproduction. During iteration, individuals are selected for reproduction according to their performance in the problem domain evaluated from fitness. GA operators are: selection, mutation, and crossover discussed above.

GA has many advantages over traditional optimisation algorithms. The most famous two are parallelism and ability to deal with complex problems. GA can handle various types of optimisation, whether objective function is stationary or changes with time (non-stationary), continuous or discontinuous linear or nonlinear, or with random noise. Offsprings in the populations behave like independent elements, makes the population explore the search domain in many directions simultaneously. Different groups of encoded strings and different parameters can be implemented or manipulated at the same time, making GA parallelism. In

summary, GA has some advantages over traditional optimisation algorithms, and one of them is its ability to handle complex problems and parallelism. Its disadvantages include: setting its right parameters (mutation, crossover and selection criteria), formulation of population size, and proper fitness function [11]. Also, it is time-consuming because GA requires many generations to converge to a solution and large population sizes.

The GA is summarised as follows:

a. Formulate an objective function
b. Encode a solution into strings
c. Generate an initial population
d. Evaluate the individual's fitness in the population with regard to the objective function
e. Specify GA parameters (crossover, mutation, generation and population size)
f. Perform crossover with probability P_C
g. Perform mutation with probability P_m
h. Select elite for the next generation
i. Update generation
j. End according to criteria

GA is popular among EAs. Application of genetic learning for a combinational logic design that has a case-based memory of past problem-solving attempt that learnt to improve the quality of the result for similar design problems is explained in the paper [23]. The algorithm is applied to parity checker and the presented result has improvement. Zarifia et al. [24] introduced a GA method for neural spike detection. The new approach solves the problem of vulnerability to noise, human intervention and lengthy training by conventional methods of spike detection. Bechouat et al. [25] compared PSO and GA as different approaches for selection and generation of duty cycle to obtain the maximum power in photovoltaic system. Furthermore, the extension of GA and its use to improve circuit's parameters is presented [26]. An automated combinational and digital circuit design using GA is presented [22, 27-29].

It has been successfully applied to automate the process of analogue circuit design [30-32]. Taherzadeh-Sani et al. [33] presented a method for determining the sizes of devices in analogue IC using GA. The efficiency of the GA using the approach is illustrated by the authors; they demonstrated how GA can be used for selection of the best device sizes in

analogue circuits. Some useful guidelines for automated design of analogue circuit by performing evolutionary operations are discussed [34]. Besides GA application to optimise active filter design using electronic work bench (EWB) is presented by Al-Azawi and Abdul-Whab [35]. They stressed the need for component count reduction, especially in op-amp which consumes power, and further emphases that it also reduces cost of design. Also, the paper shows how GA solves complex problem easier as compared to traditional optimisation method. The authors claim GAs to be the best solution. This research compared GA and PSO in component count optimisation and concluded that PSO is the best technique suitable as regard result presented. GA uses to optimise component values selections in references [36-38]. Results presented show a low design error as a result of freedom of component selection allowed by GA and reduces mathematical computation of transfer function.

A modified GA kernel for efficiency improvement on the analogue IC design cycle is illustrated [39]. Furthermore, competitive co-evolutionary DE (CODE), a new algorithm with practical user-defined specifications is proposed to design analogue ICs [40]. A directly performance-constrained template-based automatic layout is retargeting and optimisation for analogue ICs is presented in [41]. In addition, a new CMOS wideband low noise amplifier with gain control is proposed [42]. Besides, a new approach to an optimal analogue test point's selection is analysed [43]. Furthermore, simulation-based approach in which the simulator and the search algorithm are being optimised for analogue circuit synthesis is illustrated [44].

2.4.2 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 GA in that, GP is represented by variable length tree structures containing whatever elements that are needed to solve the problem, whereas GA is represented by a fixed length of numerical strings. The TS in GP population is popular because it 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 [21, 45-47].

In preparation for implementing GP according to Kennedy and Eberhart [48], 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.

The GP algorithm; according to Koza [46], 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:
    (a) Each programme in the population is executed such that a fitness measure that specifies how well the problem is solved is clearly formulated.
    (b) 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.
3. The single best individual in the population produced while the run is taken as the result.

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 [49]. 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 [50, 51]. 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 [52]. Other applications include: GP application in area of software repairs are in [53, 54], while a fully automated technique to locate and repair bugs in software is illustrated [53]. Also solving iterated functions using GP is in [55]. GP- based feature optimiser integration with patter recognition and fisher criterion methods to non-intrusive load supervising for load identification is illustrated [56].

GP has been applied to automatically synthesise similar human designs in a number of 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 [45, 47]. Hou et al. [57] 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 [58].

2.4.3 Genetic Folding

The GF is a class of EA based on numbers of genes structurally organised in order of linear numbers separated by dots [14]. 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.

2.4.3.1 The RNA Alphabet

All organisms on the earth contain one trillion genes that have a complex mission and function to accomplish. Gene is made up of deoxyribo nucleic acid (DNA [59]. The information carried in DNA consists of thousands of genes. The significant genetic information that DNA contains influences the function of cells. DNA is made up of nucleotide units that consist of three components: a deoxyribose sugar, a phosphate and a base. The main unit of a genetic code is the base pair. A base pair consists of two nucleic acid bases, which are chemically bonded. DNA has four different bases: adenine (A), guanine (G), cytosine (C), and thymine (T). The links of four types of bases are: $A \leftrightarrow T, G \leftrightarrow C$. The amino acid is formed by combined DNA base pairs. Sequences of amino acids are assembled into functional proteins or RNA. RNA is another group of nucleic acids whose function is for protein synthesis, information carrier from DNA to ribosomal sites of protein in a cell [119]. The four different bases are adenine, cytosine, guanine, and uracil abbreviated as A, C, G, and U, respectively. The bases are linked to each other in RNA thus: $A \leftrightarrow U, G \leftrightarrow C$. This idea is applied in genetic folding.

2.4.3.2 The Folding Language

As illustrated above, RNA sequences are folded with complementary bases. In GF, the arrangement of chromosomes is understood in terms of the folding procedure. Figure 2.1 demonstrates a simple idea of GF language for the expression sin ((a* b) + (c-d)) that mimics the same process of the RNA folding to be represented in the secondary structure:

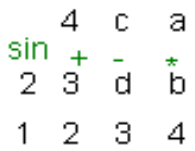

Figure 2.1: The folding language.

The outcomes in each operation are signified by indices to be applied later. Index one is the outcomes of sin (4+3) which is referred to as index '4' and '3'. However, the operator '*sin*' is the addition of the subtraction in index '3' and the multiplication of index '4'. In Figure 2.1 the reading process is from down to up and the overall result is indexed number one. In GF representation, the chromosome comprises of a simple floating string of genes whereas the gene pool comprises of multiple complexes of genes.

2.4.3.3 Genetic Encoding

The TS representation is helpful to explain the encoding and decoding process of the GF algorithm. The elements in the TS can be easily calculated in a recursive manner. For example, equation (2.1) can be encoded in GF as:

$$\sqrt{(p \times q) - (r + s)} \tag{2.1}$$

The TS diagram for expression in (2.1) is represented in Figure 2.2

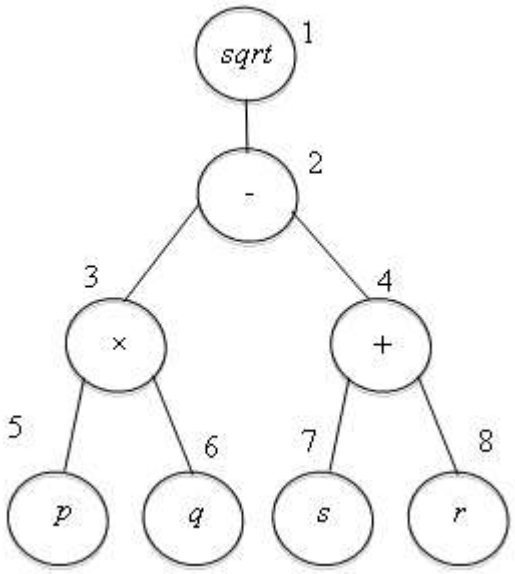

Figure 2.2: TS diagram for equation 2.1.

The TS expression view like (from top to down and from left to right):

$$sqrt - \times + pqrs \tag{2.2}$$

The expression in equation (2.2) is read as the following road map: the "sqrt" is one operand operator with the value (minus) as an input value. Then the minus operator is a two operands operator with two values (× and +); the multiplication operator is a two terminals operator that have two values (*p* and *q*), and the plus operator is a two terminals operator that have two values (*s* and *r*). Two steps involved in representing equation (2.2) using GF is as follows:

Step 1) all elements are given a position number in order

sqrt	−	×	+	*p*	*q*	*r*	*s*
1	2	3	4	5	6	7	8

Step 2) elements are folds over their complementary location genes:

2.0	3.4	5.6	7.8	0.5	0.6	0.7	0.8

The GF in equation (2.2) begins with the operator "sqrt" (position 1) and ends with the element "*s*" (position 8). The "*sqrt*" operator (position 1) calls the "minus" operator (position 2). The "minus" operator has in the left child (LC) (position 3) the "multiplication" operator and in the right child (RC) (position 4) the "plus" operator. The "multiplication" operator (position 3) has the terminals (*p* and *q*) in the LC and RC respectively. The "plus" operator (position 4) has the terminals (*s* and *r*) in the RC and LC respectively. The indices of the element's positions are used to represent terminals.

2.4.3.4 Genetic Decoding

The decoding process in GF is a reverse technique of encoding GF. The decoding technique requires two steps to be followed:

Step 1) use the value of the genes to call the next gene to be read

2.3	4.5	6.7	0.4	0.5	0.6	0.7

Step 2) substitute each gene's position with its appropriate operator

−	×	+	*a*	*b*	*c*	*d*
1	2	3	4	5	6	7

The diagram in Figure 2.3 shows the representation of GF chromosomes in the TS diagram:

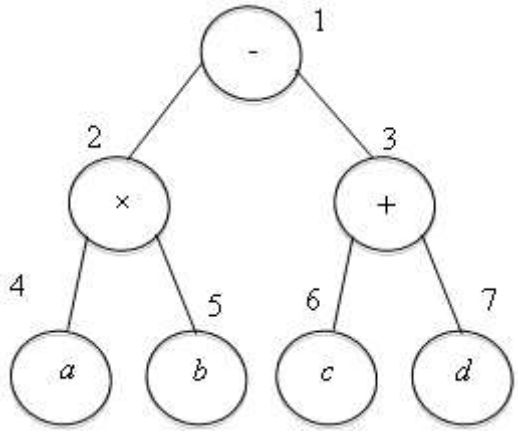

Figure 2.3: TS diagram for genetic decoding illustration.

Drawing the GF chromosome in the TS diagram starts with the gene in the position "1". Position "1" has the minus operator with two values (LC.RC). Starting with the LC refers to position "2". Position "2" is the "multiplication" operator with two values (LC.RC). The LC refers to position "4" which has a value "0.4" value refers to "*a*" which is a terminal value. The RC value refers to position "5". Position "0.5" value refers to "*b*" a terminal value. RC refers to position "3". Position "3" is the "plus" operator with two values (LC.RC). The LC refers to position "6" which has a value "0.6" value refers to "*c*" a terminal value. The RC value refers to position "7". Position "0.7" value refers to "*d*" a terminal value.

2.5 Swarm Intelligence Optimisation Methods

Swarm is defined as a population of interacting individuals or particles that is able to optimise certain global objective through cooperative search of domain. Social behaviour of animals is applied to develop an algorithm to solve problems in the society. Individual amongst the group relates so that the problem is better solved than just each individual contribution. The problem-solving behaviour that develops from the communications amongst swarm of specious is called swarm intelligence (SI) [48]. The inspiration comes from social insects such as termites, ants, wasps, bees, schools of fish and flocks of birds. Some of SI optimisation methods are: PSO, FA, ABCA and BFO.

2.5.1 Particle Swarm Optimisation

PSO is a population-based random optimisation method developed by Kennedy and Eberhart 1995 [60]. It is inspired by a social behaviour of a school of fish or flock of bird. Compared to GA, PSO has no genetic operators such as mutation, reproduction and crossover but dynamically adjusts its velocity. Also, PSO has fewer parameters compared to GA and

does not implement survival of the fittest. In PSO, potential solutions are flown or moved in search of the needed solution in the problem space and each particle is updated in the process. The particles as a whole are referred to as swarm. Suitable parameter selection guide, detailed PSO algorithm and improved PSO algorithm with leadership; during exploration search, a particle is selected as a leader to guarantee that the swarm converges rapidly to the global optimum solution [60-62]. PSO has been used for combinational logic circuits design [63-69]. Also PSO applications and comprehensive survey is presented in [70]. Furthermore, a proposed PSO-based dynamic rounding-off (DRO) technique; which serves an improvement on the existing DRO technique is in [71].

Some studies on the use of PSO in analogue circuit designs are as follow: In power electronics circuits (PECS), PSO technique is used for design and optimisation of PECs with no mathematical analysis required. To enhance population diversity, a simple mutation operator is introduced into PSO. Results presented demonstrate the efficiency of PSO approach in terms of high global search capability, easy implementation and fast convergence [72]. It is used to improve analogue circuit performance and optimal design of analogue circuits using a PSO technique as in [73, 74]. The emphasis is on PSO suitability to solve both single-objective and multi-objective discrete optimisation problem. Furthermore, the usage of PSO in microwave amplifier; the PSO technique used to a single stage amplifier circuit to obtain the best-optimised result in the design in terms of desired low noise and desired gain [75, 76]. Also, PSO application for design of analogue circuits as regard device sizes. Results presented show that PSO method is promising and accurate approach in determining components sizes in an analogue circuit [77]. In addition, the automated discrete component selection of values of capacitor and resistor for analogue active filter synthesis using the craziness-based PSO (CRPSO) is discussed in [78]. A new variant of Human behaviours based PSO (HBPSO) is presented and applied in circuit design for optimisation of switching functioning of inverter circuit [79]. Other applications of PSO in circuit optimisation are illustrated [80-82]. Also, Vural and Yildirim [83] used PSO in analogue active filter to select component values. The work is aimed at optimising overall design error of fourth-order Butterworth low-pass active filter. The same circuit is analysed by the same authors applying PSO, ABCA, GA, in which their performances are evaluated [84].

The first application attempt in the use of swarm intelligence to formulate an optimal power flow problem that consider controllable and uncontrollably distributed generator in

power networks is presented in [85]. The use of adaptive PSO based on clustering that solve the problem of PSO algorithm being trapped into local optima while solving complex multimodal function optimisation problems is given in [86]. Despite all these studies, there is no research that attempts to optimise an operational amplifier filter circuit in terms of component count reduction that this work intends to address.

2.5.1.1 PSO algorithm

The following steps are involved while using PSO.

a. Formulate an objective function
b. Initialise a population of particles with random 'position' and 'velocity' in *n*-dimensions of the problem space *i*=0
c. Calculate the fitness of each particle to obtain *pbest*
d. Compare each particle's fitness with its previous best fitness obtained. If the new value is better than pbest, then set the *pbest* as the new value and *pbest* location as the new location in n-dimensional space
e. Compare *pbest* of particle with each other and update the *gbest* location with the highest fitness
f. Change the position and velocity of the particle according to equations (2.3) and (2.4) respectively
g. Repeat step (c) to (f) until convergence is reached based on designed criteria.

$$x_{t+1} = x_t + v_{t+1} \tag{2.3}$$

$$v_{t+1} = v_t w + c_1\alpha_1(p_t - x_t) + c_2\alpha_2(g_t - x_t) \tag{2.4}$$

where x_t = position, v_t = velocity, w = initial function or weight is taken as constant between 0.5 to 0.9. c_1 and c_2 acceleration constant or the learning parameters and the both take approximate value of 2. α_1 and α_2 are random variable and each takes values between 0 and 1. p_t = *pbest* and g_t = *gbest*. During PSO implementation, the velocity of each particle is adjusted iteratively by the *gbest* (the best position obtained by particles in its neighbourhood) and its *pbest* (personal best position i.e., the best position obtained by particles so far) [87]. The movement of swarm in PSO consist of two main components: a stochastic (random) component and a deterministic component (algorithms follow a rigorous procedure, and values of both design variables and the path and the functions are repeatable).

2.5.1.2 Accelerated PSO

The use of particle best is to increase the variety in the quality solutions; it can also be achieved by simulation applying some randomness so that there is no compelling factor for using personal best unless optimisation problem is multimodal and nonlinear. The global best is only used in accelerated PSO (APSO) that accelerates the convergence of the algorithm. The APSO was developed by Yang in 2008 and further developed by Gandomi et al. [88]. Therefore, in APSO, the formula for the velocity vector is:

$$v_{t+1} = v_t + c_1(\alpha - 1/2) + c_2(gbest - x_t) \tag{2.5}$$

where α is the random variable and the values ranges from 0 to 1. This implies that ½ is out of inconvenience [11], using the standard normal distribution $c_1\alpha_t$, where the second term is replaced by α_t drawn from $N(0,1)$.

$$v_{t+1} = v_t + c_2(gbest - x_t) + c_1\alpha_t \tag{2.6}$$

where α_t can be drawn from any suitable distributions or Gaussian distribution. The position is updated as:

$$x_{t+1} = x_t + v_{t+1} \tag{2.7}$$

Further simplification gives the update of position in single step as:

$$x_{t+1} = (1 - c_2)x_t + c_2 gbest + c_1\alpha_t \tag{2.8}$$

For a typical APSO, c_1 is from 0.1 to 0.4, c_2 is from 0.1 to 0.7, but for most unimodal objective functions c_1 = 0.2, c_2 = 0.5 can be taken as initial values. In order to decrease randomness as iteration progress the APSO use:

$$\alpha_1 = \alpha_0 e^{-\beta t} \tag{2.9}$$

or

$$\alpha_1 = \alpha_0 \beta^t, \quad (0 < \beta < 1) \tag{2.10}$$

where initial value of randomness parameter; α_0 ranges from 0.5 to 1 and the number of iteration is *t*.

2.5.2 Firefly Algorithm

The FA inspired by flashing behaviour and patterns of fireflies was developed by Xin-She Yang lately in 2007 and first published in 2008 [89]. Flashing light characteristics of fireflies that produce short flashes and rhythmic uniquely to a given species is applied to develop an algorithm. The importance of the flashes is to attract mating partners, attract potential prey and serve as a warning to potential predators. The light intensity becomes weaker and weaker as the distance increases because it is being absorbed by air. The reduction in the light intensity limits the visibility to several hundred metres at night, which enable communications among fireflies. Light intensity I of the firefly is inversely proportional to the distance d.

The following features of firefly are used developed FA:

a. Fireflies are being attracted to one another irrespective of their sex (unisex).
b. The attractiveness of fireflies is directly proportional to the brightness of their flash.
c. The brightness of the firefly light is controlled or influenced by the landscape of the objective function. The brightness can simply be in proportionate to the value of the objective function for maximisation of problem while other forms the brightness can be specified similarly to the fitness function in GA.

Firefly algorithm according to Xin-She Yang [11]

Formulate objective function f(y) y = (y1, . . ., yd)T
Create initial population of m fireflies yi (I = 1, 2, . . ., m)
Light intensity Ii at yi is calculated by f(yi)
Specify absorption coefficient β.
while (p < MaxGeneration),
    for i = 1 : m (all m fireflies)
    for j = 1 : m (all m fireflies) (inner loop)
        if (Ii < Ij)
            Move firefly i toward j
        end if
        Vary attractiveness with distance d via exp[-βd^2]
        Calculate current solution and update light intensity.
    end for j
end for i

Determine the new global best g after ranking the fireflies

end while

Variations of attractiveness and light intensity can be expressed mathematically according to Xin-She Yang [11] as follow:

$$I_{(d)} = \frac{I_S}{d^2} \tag{2.11}$$

where I_s refers to as light intensity at the source, fixed light absorption coefficient β for a given medium, I varies with d.

$$I = I_0 e^{-\beta d} \tag{2.12}$$

where I_0 is original light intensity at $d = 0$. From I_s/d^2 and to avoid singularity at $d = 0$, the combined effect of both absorption and the inverse-square law can be approximated to Gaussian form as follow:

$$I_{(d)} = I_0 e^{-\beta d^2} \tag{2.13}$$

Because light intensity is proportional to firefly's attractiveness, therefore attractiveness δ of a firefly by

$$\delta = \delta_0 e^{-\beta d^2}. \tag{2.14}$$

where δ_0 is the attractiveness at $d = 0$. Because it is faster to calculate $1/(1+d^2)$ than exponential function, the function is approximated as

$$\delta = \frac{\delta_0}{1+\beta d^2}. \tag{2.15}$$

Both equation (2.14) and equation (2.15) define characteristic distance $\lambda = 1/\sqrt{\beta}$ with a range δ_0 to $\delta_0 e^{-1}$ for equation (2.14) or $\delta_0/2$ for equation (2.15). In some applications, the attractiveness function $\delta_{(d)}$ can be decreasing function generalised form as:

$$\delta_{(d)} = \delta_0 e^{-\beta d n}, \quad (n>=1) \tag{2.16}$$

For a fixed β, the characteristic length becomes

$$\lambda = \beta^{-1/n} \quad \to 1, \qquad n \to \infty \tag{2.17}$$

On the other hand, for a given range of distance λ in an optimisation problem, the absorption coefficient β can be used as an initial value. i.e.

$$\beta = \frac{1}{\lambda^n} \tag{2.18}$$

The distance d between every two fireflies i and j at y_i and y_j, respectively, is the Cartesian distance.

$$d_{ij} = \left\| y_i - y_j \right\| = \sqrt{\sum_{k=1}^{m} \left(y_{i,k} - y_{j,k} \right)^2} \tag{2.19}$$

where $y_{i,k}$ is the kth component of the spatial coordinate y_i of ith firefly. In a two-dimensional case, is represented as:

$$d_{ij} = \sqrt{\left(x_i - x_j \right)^2 + \left(y_i - y_j \right)^2} \tag{2.20}$$

Firefly i movement as a result of attraction to another j is determined by

$$y_i^{t+1} = y_i^t + \delta_0 e^{-\beta d_{ij}^2} (y_j^t - y_i^t) + \alpha \varepsilon_i^t \,. \tag{2.21}$$

where the third term is randomisation, with ε_i being a vector of random numbers generated from uniform distribution or Gaussian distribution and α is the random parameters. The second term is due to the attraction.

FA has been popular in recent years and has over 850 publications. Fister et al. [90] presented a comprehensive review of FA. The work reviewed how FA is hybrid or modified and used for different applications. Also, the multi-objective optimisation enhanced FA has been investigated [91-93]. A decentralised algorithm for synchronicity based on firefly features is presented [94]. The FA is applied to the sensor system and the result presented shows that the algorithms can efficiently synchronise sensor network.

The flashing features inspired FA used in optimisation is used to develop an algorithm [95, 96]; the first reference provides a detailed FA while the second reference indicates the development of the algorithm. The result presented shows how PSO betters firefly algorithm.

Furthermore, the use of FA to construct codebook for vector quantisation is in [97]. Higher quality result is presented for reconstructed images better than those obtained from the Linde-Buzo-Gray (LBG), PSO, and quantum-PSO (QPSO). In addition, Gandomi et al. [98] researched on different chaotic maps and compared their performance, and stated that some chaotic maps can improve the performance of FA; some parameters in FA can be replaced with these chaotic maps. A support vector regression with the chaos-based FA presented by Kazemet et al. [99] illustrated its application in stock market price forecasting. On the other hand, Srivastava et al. [100] showed how a modified FA can be applied in software testing. It was demonstrated with generated independent test sequences efficiently which have better performance. In the area of image registration, Zhang and Wu applied FA [101]. Also, Zaman and Matin [102] discovered that FA can perform better than PSO and found global results. A non-convex economic dispatch problem with valve-loading effect was solved by Yang et al. [103] using FA and obtained the best results compared to other techniques. In addition, Imanirad et al. [104] applied FA to create alternative for decision makers with different options. Chaotic FA (CFA) use to optimise time coordination of relays (optimise total operating time and to minimise damage during faults). The algorithm is implemented in Matlab, results are presented and compared to conventional FA [105].

2.5.3 Artificial Bee Colony Algorithm

ABCA of the honey bees inspired the bee algorithm. Many different forms of bee algorithms have been developed which are: the virtual bee algorithm (VBA) [106], the honey bee algorithm (HBA) [107], the honeybee-mating algorithm (HBMA) [108], the artificial bee colony optimisation (ABCO) [109]. Honey bees store honey in constructed colony and live in a colony as forage. Honey bees interact by ‘waggle dance’ and pheromone. Whenever the bees find food source and carry nectar to the hive, the food source is communed by waggle dance but it varies from species to species. ABCA was developed by Karaboga in 2005 [6]. The bees in a colony are grouped into three classes: onlooker bees (observer bees), scouts, and employed bees (forager bees). In a given food source, only a single employed bee is involved. That is to say, the number of food source is equal to the number of employed bee. Employed bee of the abandoned food site becomes a scout to search for new food sources. Employed bees and onlooker bees communicate with each other to enable onlooker bees locate a food source to forage.

It is a population-based, random search technique carries out in the neighbourhood. Detailed survey of ABCA was carried out by Karaboga et al. [110] Honey bee special features such as communication and dance, decision-making, foraging, marriage, nest site selection, allocation of task, mating and reproduction are being transformed into bee algorithm. A modified version of ABC algorithm a best-so-far for solution update; the authors developed improved algorithm that perform better than existing ABC algorithm. Results presented demonstrated the efficiency of the technique in terms of speed of convergence and quality solutions when compared to original ABCA [111]. The ABC algorithm has been used for different areas of applications some of which are: ABCA has been used successfully for scheduling task [112-114] and its usage for reliable and efficient routing algorithm for emergency and disaster management situation has been reported [115]. Hong hybridised several forecasting methods: ABCA, chaotic sequence, recurrent mechanism and seasonal adjustment in power engineering for electric load forecasting. Results presented show that the technique is an alternative method in terms of forecasting accuracy [116].

ABCA application in the area of electronic circuit design includes: its application by Karaboga as an alternative technique in the design of digital filter [117]. In addition, ABCA is used to optimise Nano-CMOS analogue mixed signal (AMS) where the authors claimed that their work was the first to combined bee colony optimisation (BCO) and meta-modelling for AMS design domain exploration. Results presented illustrate the suitability of the technique [118]. Manoj and Elias [119] used ABCA in electronic circuit to design a non-uniform filter bank trans-multiplexer (NUFB TMUX). The filter coefficients are represented in canonical signed digit (CSD) format that is formulated as optimisation problem for ABCA application. The authors presented results that are better than that obtained by rounding coefficients of filters to their nearest CSD number. The results also outperformed that got from using PSO and GA. Also, Agrawal and Shu [120] used ABCA as an alternative design method for two-channel QMF (quadrate mirror filter) bank that has linear phase. Results illustrated show improved performance as regard smallest peak reconstruction error and better results for bigger tap QMF banks when compared to DE, PSO and other conventional optimisation algorithms. Kockanat and Karabogo [121] released the first time proposal of 2D-ABCA adaptive filter algorithm in literature. The use of ABCA for investigation of the design of CMOS inverter and results presented show that the approach is capable of

producing results within a very short time with acceptable error while satisfying all the design conditions [122].

Bee algorithm

State the objective function f(y), y = (y1, y2, . . . yd)T
Encode f(y) into virtual nectar levels
Specify dance route (direction, strength) or protocol
While (criteria)
    For loop over all m dimensions
        Formulate current solution
        Calculate the current solution
    end for
Update the optimal solution set and communicate
end while

2.5.4 Bacterial Foraging Optimisation

Natural selection favours the survival of genes of potential animals that have foraging strategies (techniques of handling, locating and ingesting food) as a result of them obtaining enough food to enhance their reproduction and eliminate animals with weaker foraging strategies. After many generations, weaker animals are reshaped or eliminated, and this led scientists to discover evolutionary principle called foraging theory. A foraging animal optimises the energy obtained per unit time while foraging in the face of environment (risk from predators, destiny of prey and nature of physical search area) and constraints (reasoning capabilities and sensing) [123]. The survival of an animal as regard its mobility and its exploration for food depends upon their fitness criteria referred to as BFO. It is inspired by the chemotaxis characteristics of bacteria which make it move away or toward specific signals as it senses any chemical near it. The organism reshapes the poorer organism with weak search experience. The genes of the fittest species are used for development chain and have promised better animals in next generations. The foraging characteristic in evolutionary species is being translated into algorithms that are applied to non-linear optimisation model.

The control mechanism on how bacteria forage is divided into four units: chemotaxis, swarming, reproduction and elimination and dispersal.

a. Chemotaxis: The technique is accomplished by tumbling and swimming by flagella. The rotation of flagella in every bacterium determines whether it should travel in a specified direction (swimming) or different directions (tumbling) carried out in entire lifetime. Mathematically it is defined as;

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(j) \tag{2.22}$$

where $\phi(j)$ is unit length random direction used to specify the direction to move after a tumble. $\theta^i(j,k,l)$ stands for *ith* bacterium at *jth* chemotaxis, *lth* elimination or dispersal step, and *kth* reproduction. *C(i)* is the step size taken in the random direction defined by tumble.

b. Swarming: During the technique of getting to the best food location, the bacterium in the optimal path should send attractive signal to other bacteria for them to swarm together toward the desired location. Swarming is mathematically represented as;

$$J_{cc}(\theta, p(j,k,l)) = \sum_{i=1}^{s} J_{cc}^{i}(\theta, \theta^i(j,k,l))$$

$$= \sum_{i=1}^{s}[-d_{attract}\exp(-w_{attract}\sum_{m=1}^{p}(\theta_m - \theta_m^i)^2)] + \sum_{i=1}^{s}[h_{repellent}\exp(-w_{repellent}\sum_{m=1}^{p}(\theta_m - \theta_m^i)^2)] \tag{2.23}$$

where $J_{cc}(\theta, P(j,k,l))$ is the objective function value to be added to the actual objective function to be minimised. *p* is the number of parameters to be minimised, *s* is the total number of bacterial.

c. Reproduction: The least healthy bacteria are eliminated and the remaining healthiest bacteria each break into two bacteria and are placed in same location. This maintains the population of the bacteria constant.

d. Elimination and Dispersal: The population of bacteria in the search domain changes gradually which may be as a result of consumption of nutrient or an event can happen such that all the bacteria in an area are kill or a group is sent to another new part of environment. This may possibly destroy the chemotactic progress, or enhance chemotactic progress since dispersal may bring bacteria close to good food sources.

Detailed BFO Algorithm, its usage in non-linear optimisation models such as control systems & distributed optimisation and design optimisation of Yagi-Uda Array are reported respectively in [123, 124]. BFO is used in distributed control application in area of sensor network. The authors suggested a general technique to control design applying discrete event-

triggered elements at the high level and hierarchical model consisted of continuous time-triggered elements at the low level [125]. BFA is applied to training Neural Network (NN) and result presented in the paper shows that BFO-NN technique obtains quality result than that of GA-NN in terms of accuracy and convergence [126]. Also, a hybrid GA-BFA proposed for proportional-integral-derivative (PID) controller for an automatic voltage regulator (AVR) system [127, 128]. Result is compared to GA, PSO, and GA-PSO and it proves its potential for optimisation. In addition, BFO application in the area of power system in terms of: optimisation of voltage stability limit and real power loss, optimisation of active power filter for load compensation and for estimation of three winding transformer parameter [129-136].

In the area of communication, an improved adaptive BFA (ABFA) using the method of adaptive delta modulation is applied to optimised both phase and amplitude of linear array of antenna and Multi-colony BFO (MC-BFO) for complex radio frequency identification (RFID) network planning problem was in [137, 138].

This research intends to address the following:

- Reduce component count in passive and active filters.
- Improve on existing SCAM to enable it matrices dimension more than 32 by 32 as regard operational amplifier simulation compared existing that cannot handle matrices dimension more than 8 by 8 and also extend it to handle active components.
- Use GP, GF and MSCAM and automatically generated Netlist for the evolution of passive and active filter circuits.

2.6 Summary

This chapter presents definition and general background of optimisation, optimisation algorithm and their area of applications to the society at large. Also, the elements of evolutionary optimisation algorithm are discussed and illustrated, how they are formulated and used while developing algorithm. This is followed by the review of literature on the various optimisation methods used in this research. It also describes in detail the history of these developed techniques, principle of inspiration and different application areas. Some of their key features are highlighted below.

GA applies the principle of survival of the fittest to solve optimisation problems. GA owns some advantages over traditional optimisation algorithms: its ability to handle complex problems and parallelism. Its disadvantages include, setting its right parameters (mutation, crossover and selection criteria), formulation of population size, proper fitness function. Also, it is time-consuming because GA requires many generations to converge to a solution and large population sizes. In the use of GA for circuits' optimisation, the objective function is formulated specifying the frequency response and called directly from the GA Matlab optimisation toolbox under the fitness function option. The maximum and minimum component values are fed at the upper and lower options respectively. Also, GA parameters are set. The total number of components to be minimised is specified in the dialog box option called number of variable option. The programme is automatically generated, applying generate code option under the file menu in order to rank results.

GP originated from the genetic algorithm. GP differs from 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 TS is great because it can produce solutions of complexity and arbitrary size, as opposed to GA with fixed-length. In this work, GP is for circuits' evolution, the objective function is formulated specifying the lower and upper cut-off frequencies. A Matlab programme is coded using GP algorithm for circuits' evolution, and GP parameters are also initialised.

GF is based on numbers of genes structurally organised in order of linear digits separated by dots. GF imitates the RNA secondary structure folding procedure of the complementary bases on itself. GF is used to show how the TS in GP are linked from the top to the bottom during circuit evolution. The GA, GP and GF are evolutionary computation algorithm.

PSO is inspired by a social behaviour of a school of fish or flock of bird. Compared to GA, PSO has no genetic operators such as mutation, reproduction and crossover but dynamically adjusts its position and velocity. Also, PSO has fewer parameters compared to GA and does not implement survival of the fittest. In PSO, potential solutions are flown or moved in search of the needed solution in the problem space and each particle is updated in the process. PSO is used in this work for circuits' optimisation; the objective function is formulated, specifying the frequency response and being called into a Matlab programme coded using PSO algorithm for circuits' optimisation, which is expected to sample

components values that will produce the frequency response. The maximum and minimum component values are specified. In addition, PSO parameters are set.

FA inspired by flashing behaviour and patterns of fireflies. It is powerful optimisation tool due to the effect of attractiveness function that is unique to firefly behaviour. FA's disadvantages include: it does not keep record of past better solution to guide the search toward better solution; parameters do not change with time and are trapped into several local optima. In this research, FA is used for circuit's optimisation, the objective function is formulated specifying the frequency response and being called into a programme coded in Matlab using FA algorithm for circuits' optimisation which is expected to sample components values that will produce the frequency response. The maximum and minimum component values are specified. Besides, FA parameters are initialised.

ABCA imitates foraging behaviour of bees. It has fewer control parameters, easy to implement, easily hybridise with other algorithm and easy to be modify. Its disadvantages are: it is compromised with its cost-benefit function, slow to converge and sometimes trapped to local optimal. ABCA is used for circuits' optimisation in this work; the objective function is formulated, specifying the frequency response and being called into a Matlab programme coded using ABCA algorithm for circuits' optimisation which is expected to sample components values that will produce the frequency response. The maximum and minimum component values are specified. Also, ABCA parameters are set.

BFO is inspired by the chemotaxis characteristics of bacteria which make it move away or toward specific signals as it senses any chemical near it. The survival of an animal as regard its mobility and its exploration for food depends upon their fitness criteria referred to as bacterial foraging optimisation. Its disadvantage is that it required a large number of parameters. In this research, BFO is used for circuits' optimisation, the objective function is formulated specifying the frequency response and being called into a programme coded in Matlab using BFO algorithm for circuits' optimisation which is expected to sample components values that will produce the frequency response. The maximum and minimum component values are specified. In addition, BFO parameters are initialised. PSO, FA, ABCA, and BFO are some swarm intelligent optimisation methods. Artificial intelligence methods are used due to the following advantages over humans. Hardware advantages include greater parallel speeds and greater serial speeds. Self-improvement advantages include modification of motivational system, improvement of algorithms and design of new

mental modules. Co-operative advantages include transfer of skills, copyability, improved communication and perfect co-operation. Compared to mathematical optimisation: simplex method, branch and bound, Lagrange multipliers, interior point methods and cutting planes that are unable to solve difficult problems due to memory needed and large amount of computational time.

The next section is Chapter 3, which gives a brief review of symbolic circuit analysis and description on how automatically generates matrices from Netlist created from PSpice or automatically generated Netlist from simulation is transformed to matrices.

Chapter 3[2]

Modified Symbolic Circuit Analysis in Matlab and its Applications in Electronic Circuit Simulation

3.1 Introduction

This chapter gives a brief review of symbolic circuit analysis. It describes MSCAM and its usage in an electronic circuit simulation with illustrations. It improves on existing SCAM, so that, the developed algorithm can handle matrices dimension of size more than eight by eight whenever op-amp is involved as a circuit component. Besides, the chapter gives a detailed description of how automatically generated matrices from Netlist; created from PSpice or automatically generated Netlist from the simulation is transformed to matrices. The remaining part of this chapter is subdivided as: electronic circuit simulation techniques, discussed in Section 3.2 and symbolic method illustrated in Section 3.3. Circuit's simulation and results are presented in Section 3.4, the formulation of the objective function and circuit's simulation examples are demonstrated in Section 3.5 and summary of this chapter is in Section 3.6.

A physical model can be explained quantitatively or qualitatively. In the qualitative simulation, analysis of variables in a system is achieved by assigning a given range of qualitative values. Whereas in a quantitative simulator, analysis of variables in a system is achieved by assigning real values and the response to a given excitation is evaluated using analytical (symbolic) or numerical [140]. The two main importance of symbolic simulator in analogue circuits design are:

- It provides insight to novice or student designers, and it also interactively helps experienced designers [141, 142].
- It also aids in automatically generating analytic analogue circuits [143].

[2] The bulk of Chapter 3 has been published in [139].

3.2 Electronic Circuit Simulators

There are many types of simulation software; some are licensed and some are free with limited capacity. A simulator helps students to demonstrate or verify what has been taught in the classroom. Some of the free analogue simulation software include: SPice which is a general-purpose and open source analogue electronic circuit simulation [144, 145].

Furthermore, electric very large scale integration (VLSI) design system is an automation simulator for integrated circuits and printed circuit board [146-148]. Also, gpsim is a Microchip PIC (peripheral interface controller) microcontroller's software designed for PIC circuit simulations [149]. Other simulators include; DoCircuits is a cross-platform virtual learning system, web-based that models an instrument as well as circuits applied in labs to enhance students' implementation of experiments in virtual environment [150]. PartSim a free and easy to use circuit simulator; is another kind of simulator that can be run in web browser [151]. Besides, SimOne is a European's leading software applied for gas transport, optimisation and distribution simulation [152]. Moreover, CircuitLab simulates analogue and digital components side-by-side and also gives precise results for nonlinear circuit [153]. Other digital and analogues (mixed-signal) simulators are: EasyEDA, which is an enthusiast's web-based software for educators, electronics engineers and students [154]. Falstad circuit simulator applet is a tool that boots with animated schematic of LRC circuit. In it, the green indicator signifies positive voltage, grey colour signifies the ground, red signifies negative voltage and moving yellow dots signifies current [155]. GeckoCIRCUITS is a power electronic designing circuit tool with a fast circuit simulation capability [156]. Furthermore, Ngspice is a mixed-levelled and mixed-signal circuit simulator. It is developed based on three software packages: Spice3f5, Cider1b1 and XSpice [157, 158]. Also, NL5 is an analogue simulator that operates with piecewise linear components [159]. SuperSpice from AnaSoft is a circuit tool for both integrated circuit and board-level applications [160]. Also, SIMetrix is a simulator that enhances engineers to simulate and model switching power electronics systems which combine accuracy and speed in model environment [161]. Moreover, Maple is a computer algebra system created to enable users to key in a mathematical symbol in a traditional way [162]. Conclusively, multisim (national instrument (NI)) includes microcontroller simulation that can export and import features to PCB (printed circuit board) [163]. In most of these simulators, there are no detailed insight analyses of the simulation

procedure while processing, most optimisations cannot be carried out directly with them and most researchers have no access.

3.3 Symbolic Method

Symbolic circuit analysis has been used by many researchers in analogue circuit simulation. Gielen et al. [141] introduced a symbolic analysis for analogue circuit design that is interactive. It provides the designer of the analogue circuit with deep insight on how the design process is carried out than mechanical simulators do. In addition, optimisation of analogue circuit design using simulated annealing (SA) and symbolic simulation is also developed by Gielen et al [164]. It is non-fixed-topology and efficient analogue design tool. The method sizes all circuit components to satisfy design objective. Also, a tutorial overview regarding applications and methods of symbolic analysis for analogue circuit is illustrated [165]. Wambacq, et al. [166] illustrated approximated symbolic expression for small signal analysis (SSA) of analogue IC applying symbolic computation. Yu and Sechen [167] developed an approximate symbolic analysis of large analogue IC. This unified approach used two new approximation (create a tree of two-graph and product terms in decreasing order of magnitude in symbolic network function) during computation processes with classical two-graph tree enumeration method. Also, related two-graph methods for symbolic analysis of circuit are illustrated in [168-170]. However, determinant decision diagram (DDD) combines with a canonical symbolic synthesis of the analogue circuit is presented [171, 172]. Also, a symbolic circuit analysis application for the multi-physical system is presented as a new modelling approach. The technique reduces the complexity of the symbolic equations and solution by mixing numeric and symbolic strategies. The approaches potential is demonstrated on usage for modelling and analysis of gas-pipeline nets and mixed mechanical and electrical systems [173]. The use of symbolic analysis has aided in analogue realisation; the fraction power of a realistic transfer function was introduced in analogue circuit domain as already existed in digital domain. It helps in the design of analogue circuits for such fractional order controller and solves the problem of running continued fraction expansion algorithm every time [174]. Besides, SA and GA based technique for symbolic description of analogue circuits is developed. It is a Matlab tool based on evolutionary algorithms and modified nodal analysis of analogue circuits containing MOSFETs for automatic simplified symbolic SSA [175].

The process involved in transforming electrical circuit into matrices required tedious calculations and the same process has to be performed each time a circuit is to be resolved. Also, transforming circuit's equations by the human method may take a longer time or result in an error to solve a simple circuit. The existing automated method (SCAM) [176, 177] cannot handle some components such as MOSFET and BJT. Also to transform a circuit whose matric is more than eight by eight tend to be very slow because it requires large memory and in some circuit applications, it is inefficient.

The symbol(s) to be used in the code has to be symbolised. The essence is for the computer to treat it as symbolic to form an equation with it and to enable the users to execute a variety of symbolic evaluations in Science and Mathematics. The data structure which stores a string representation of a symbol is called symbolic object. For illustration, symbolising *e, f, g* and *y* is as follows:

syms e f g y, but if only one character is involved the *sym* is singular.

Two set of equations is formed from circuits either by nodal analysis method or mesh analysis method [178].

3.3.1 Mesh Analysis

The mesh can be defined as a loop that does not comprise of any other loop. It uses mesh current as circuit variables. The mesh analysis is most useful when the circuit has mostly voltage sources. It applies Kirchhoff's voltage law (KVL), Ohm's law and simultaneous equations to find unknown currents in a circuit. It does not apply Kirchhoff's current law which makes it different from branch current method. It is useful to solve a circuit with fewer variables and less simultaneous equations and to be useful to solve a circuit without a calculator. Detailed procedures are as follow:

a. Assign a current loop
    - Specify a direction of loops
    - Given circuit's number of loop = number of branches – 1
    - Loop current can overlap
    - All branches are covered in the loop
    - Each loop is referred to as mesh
b. Write the KVL equation each mesh and if loop current overlap

- Currents are added if in the same direction
- Currents are subtracted if in the opposite direction
- Voltage sources are added if in the direction of mesh current
- Voltage sources are subtracted if opposite of the mesh current

c. The simultaneous equations are solved for the mesh current
   - Obtain each branch currents from the mesh current
   - Voltages are evaluated from the current

3.3.2 Nodal Analysis

The modified nodal analysis is called modified nodal analysis (MNA) for easy formation of the algorithm. For the algorithm to handle components such as transistor and operational amplifier, the transistor and operational amplifier are converted to SSA. The op-amp representation differs from the one represented [177]. The new operational SSA representation reduces the size of memory occupied, increases speed and can handle larger matrix dimension size. Detailed illustration of the approach is explained below.

3.3.3 Development of Algorithm for New Modified Nodal Analysis

New modified nodal analysis (NMNA) applied to a circuit with op-amps, capacitor, transistors, inductor, resistors, independent voltage and current sources results in a matrix equation of the form:

$$AX = I \tag{3.1}$$

For a given circuit with p number of nodes, the following illustrate how matrices *A, I* and *X* are formed.

3.3.3.1 The A Matrix:

A matrix is $p \times p$ and comprises only known quantities, (the values of the gain of the operational amplifier and the passive elements (the capacitors, inductors and resistors). Component connected to ground only appears on the diagonal; while non-grounded component appears both on and off the diagonal as summarised below:

a. *A* matrix $p \times p$ in size and holds only known quantities.
b. Have both active and passive elements
c. Components connected to ground only appear on the diagonal

d. Components not connected to ground appear both on and off the diagonal terms.
e. The op-amp is coded such that 1 is added to the operational amplifier output (i.e. *A(p,p)=A(p,p)+1),* and negative input is added with *Av* while positive input is subtracted with *Av*.

3.3.3.2 The X Matrix:

Is a matrix of size $p \times 1$ vector that contains the unknown quantities (node voltages)

3.3.3.3 The I Matrix:

a. The I matrix is a $p \times 1$ vector that comprises only known quantities
b. It holds summation of current sources in a loop due to node voltage. The current source is as a result of the voltage source or as a result of independent source.
c. The circuit is solved using Matrix manipulation below:

$$X = A^{-1}I \longrightarrow \text{provided } A \text{ is non-singular.} \tag{3.2}$$

The computer can easily accomplish the matrices evaluation that may be difficult by the human method.

3.3.3.4 Presentation

The convention of representation obviously does not change the solution. However, the procedure described below simplifies the formation matrices necessary for the solution of the circuit.

a. The ground or reference node is labelled 0.
b. The other remaining nodes are labelled in sequential order from 1 to p.
c. Voltage at node 1 is call *v_1,* at node 2 is referred to as *v_2* and so on.
d. The naming of independent voltage sources is quite flexible; it starts with the letter "*V*" and one node must be unique from another node name.
e. The current as a result voltage source is labelled with "I_1, I_2 I_3 and so no" whereas the current source due to the voltage source is V_1/R_{111} that is voltage source over impedance. The current in particular branch is the sum of these current sources.

3.4 Circuits Simulation and Results

The developed algorithm discussed in Section 3.3.3 is used to write computer code in Matlab. The programme is implemented with four different examples to demonstrate its

efficiency. Before, the continuation of the results analysis, a brief introduction to filter is made.

3.4.1 Filter

The electronic and electrical filter design is a vital device in a modern communication system and electronic. A filter is an electronic device that permits certain range of frequencies to be transmitted and it is classified into four as regard frequencies passage or rejection:

- A low-pass filter suppresses high frequencies but allows only the low frequencies to be transmitted.
- Whereas a high-pass filter suppresses low frequencies but allows only the high frequencies to be transmitted.
- A band-pass filter is a filter that passes a certain range of frequencies and rejects frequencies outside the given range.
- Band-rejection filter is a device which passes most frequencies unchanged but suppresses those in a specific range to very low levels. It is opposite of a band-pass filter.

A digital filtering technique is worthy of mention since it is widely applied and increasingly important. This filter executes the filtering task applying digital and analogue components in addition to digital to analogue (D/A) converters, analogue to digital (A/D) converters, shift registers, multiplexers and multiplier. Filter can also be classified in terms of component combinations as passive and active filter.

3.4.1.1 Passive Filter

Passive filters are designed from the combination of inductance, capacitance and resistance and can be designed to cover a range of frequency from 10 Hz to 500 MHz. The passive filters have low sensitivity to component variation and do not need an external power supply which makes it advantageous over active circuits. They are disadvantageous when size and cost are considered because of the bulky coil.

3.4.1.2 Active Filter

The active filters are constructed from an op-amp (active source), capacitor and resistor. The active filters have low output and high input impedances also, eliminate bulky

coils and expensive nature of passive filters. The major setback posed by the active filter is that its gain reduces at high frequencies above 50 kHz, and also it requires a power source [179, 180]. Operational transducer amplifier and a step-by-step method applied in the design of active filter [181, 182].

3.4.2 Operational Amplifier and its Small Signal Analysis

The op-amp is a versatile electronic circuit that can perform basic mathematical operations, such as division, multiplication subtraction and addition. It can also be applied to do differentiation and integration. The op-amp is used as the integral element in filters, amplifiers flip-flops and oscillators. The op-amp has the following properties: infinite input resistance, zero output resistance, zero offsets voltage, infinite open-loop gain (A), infinite frequency response and infinite common-mode rejection ratio.

The SSA is the procedure of replacing non-linear elements with linear ones. Also, in some cases, the elements are being replaced by their internal operation of the components. The small signal operation of an op-amp is shown in Figure 3.1.

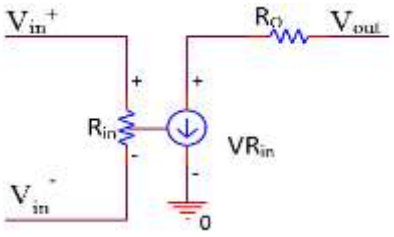

Figure 3.1: Small signal analysis of operational amplifier.

The SSA of op-amp has VR_{in} as a not-defined voltage that makes it difficult in the simulation. The second option of the SSA of the op-amp explained in [183] is being used. The summary of the op-amp nodal analysis detailed process is illustrated with the following equations:

$$V_{out} = A_v(V_{in}^{+} - V_{in}^{-}) \tag{3.3}$$

For instance, if an op-amp's inputs are: $V_{in}^{+} = V_3$, $V_{in}^{-} = V_4$ and its output is $V_{out} = V_5$, then the output node voltage is given as

$$V_5 = A_v(V_3 - V_4) \tag{3.4}$$

From equation (3.3),

$$A_v V_{in}^{-} - A_v V_{in}^{+} + V_{out} = 0 \tag{3.5}$$

From equation (3.5) analysis, if the output of an op-amp is in *p* node, then the matrix of *A(p,p)* is added with 1. i.e.

$$A(p,p) = A(p,p) + 1 \tag{3.6}$$

In this particular case, as V_{in}^{+} is at *p-2* node and V_{in}^{-} is at *p-1* node, the expression will be:

$$A(p,p-1) = A(p,p-1) + A_v \tag{3.7}$$

$$A(p,p-2) = A(p,p-2) - A_v \tag{3.8}$$

In other words, the op-amp's output is added with 1, V_{in}^{+} input is added with ($-A_v$) and V_{in}^{-} input is added with A_v as explained above. The algorithm is coded in such a way that each op-amp is substituted as illustrated above.

3.4.3 Transistor Amplifier

The transistor amplifier will be discussed based on its SSA and frequency response using common-emitter and common-source for illustration. The transistors are being replaced by their SSA. The values of R_{pi}, R_o, g_m, C_{pi} and C_{mu} are usually specified in the simulated result from PSpice or obtained in literature or data sheet. Also, MOSFET is also transformed into its SSA in a similar way.

3.4.3.1 Frequency Response of Common-Emitter Amplifier

The common-emitter amplifier is capable of producing relatively high voltage and current gains. The input resistance is medium and independent of load resistance R_L. Its output resistance is high and independent of source resistance. The coupling capacitor, C_1, couples the biasing network to the source voltage V_s. The coupling capacitor C_2, connects the load resistor R_L, to the collector resistance R_C. The by-pass capacitance C_e is applied to increase the mid-band gain as it short circuit the emitter resistance Re at mid-band frequency. The R_e is for bias stability. The external capacitors C_e, C_2, C_1 are responsible for low frequency response whereas the internal capacitances (C_{pi} and C_μ) are responsible for high cut-off. C_{pi} is the emitter-base capacitance, the collector-based capacitance is C_μ, R_x is the

resistance of the silicon material between the base region and the intrinsic or internal base. The common-emitter amplifier circuit is in Figure 3.2 whereas SSA is in Figure 3.3.

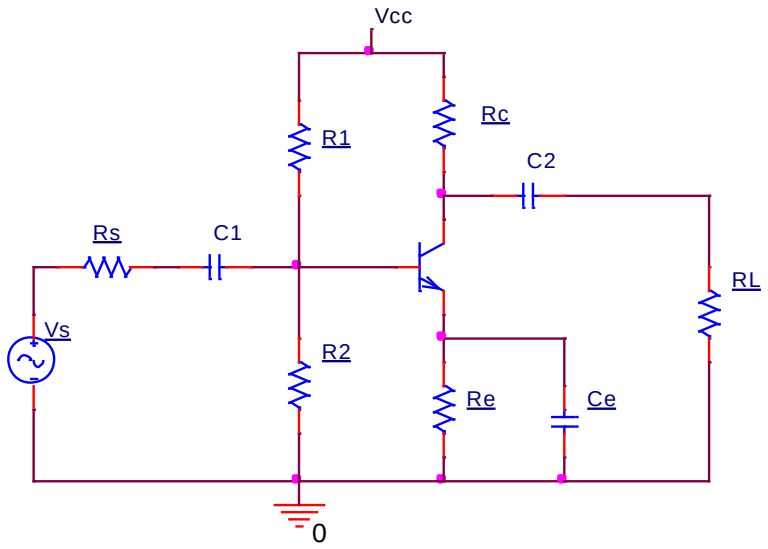

Figure 3.2: Common-emitter amplifier.

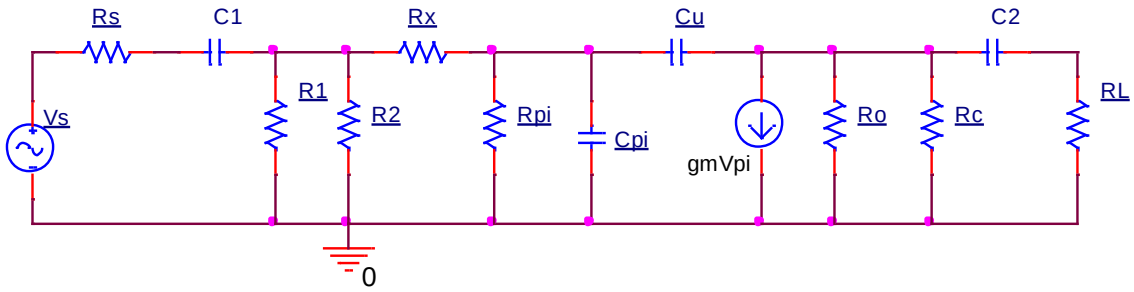

Figure 3.3: SSA of common-emitter amplifier.

3.4.3.2 Frequency Response of Common-Source Amplifier

The common-source amplifier has similar features to that of the common-emitter amplifier explained in Section 3.4.3.1. The only difference is that common-source amplifier input resistance is higher than that of the common-emitter amplifier. The common-source amplifier circuit is in Figure 3.4 whereas SSA is in Figure 3.5.

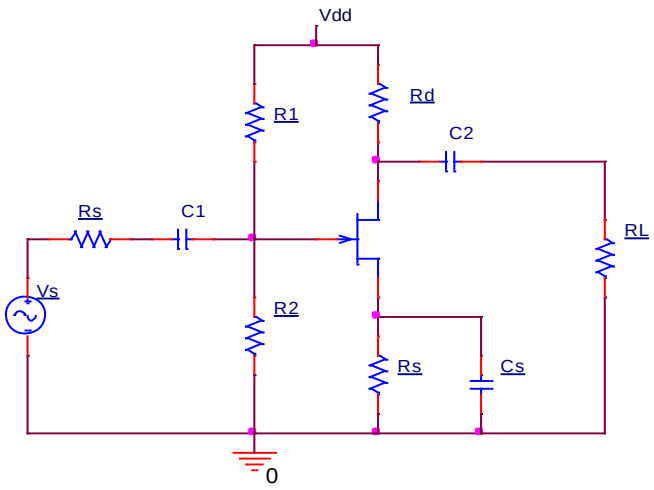

Figure 3.4: Common-source amplifier.

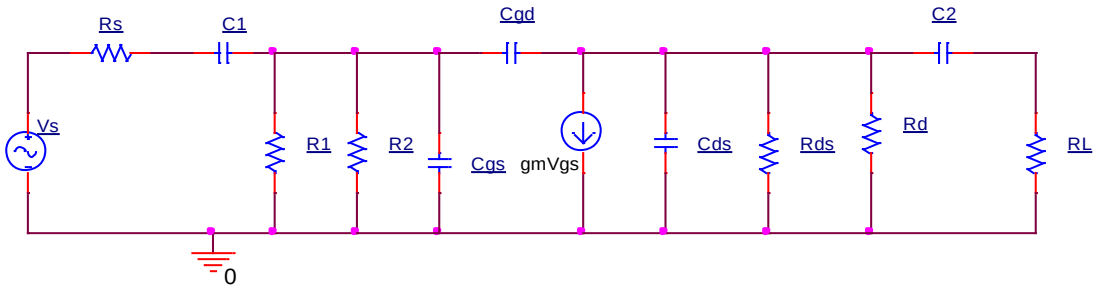

Figure 3.5: SSA of common-source amplifier.

3.5 Formulation of Objective Function and Circuits Simulation Examples

Detail of the 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. The same system specification is used throughout this research. The circuit simulation process is summarized in flowchart shown in Figure 3.6. Four different circuits are considered in order to demonstrate the potential of the developed algorithm. Example 1 (seventh order Chebyshev filter) is used to illustrate the formulation of objective function.

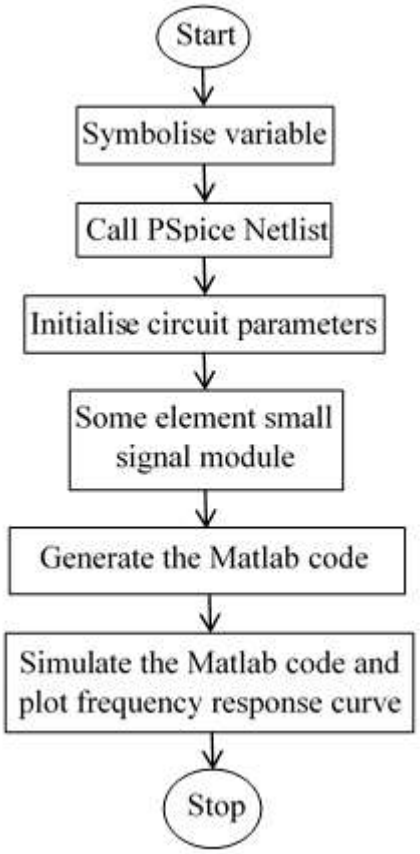

Figure 3.6: The proposed MSCAM algorithm.

3.5.1 Seventh Order Chebyshev Circuit Objective Function Specifications

The symbolic matrices A and B obtained from MSCAM are used to formulate the fitness function as:

$$A = [(V_S \div R_{111});0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0] \quad (3.9)$$

$$B=\begin{bmatrix}
b_{11} & b_{12} & 0 & 0 & 0 & b_{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
b_{21} & b_{22} & b_{23} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & b_{32} & b_{33} & b_{34} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & b_{43} & b_{44} & b_{45} & 0 & b_{47} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & b_{54} & b_{55} & b_{56} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
b_{61} & 0 & 0 & 0 & b_{65} & b_{66} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & b_{74} & 0 & 0 & b_{77} & b_{78} & 0 & 0 & 0 & b_{712} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & b_{87} & b_{88} & b_{89} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{98} & b_{99} & b_{910} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{109} & b_{1010} & b_{1011} & 0 & b_{1013} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{1110} & b_{1111} & b_{1112} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & b_{127} & 0 & 0 & 0 & b_{1211} & b_{1212} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{1310} & 0 & 0 & b_{1313} & b_{1314} & 0 & 0 & 0 & b_{1318} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{1413} & b_{1414} & b_{1415} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{1514} & b_{1515} & b_{1516} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{1615} & b_{1616} & b_{1617} & 0 & b_{1619} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{1716} & b_{1717} & b_{1718} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{1813} & 0 & 0 & 0 & b_{1817} & b_{1818} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{1916} & 0 & 0 & b_{1919} & b_{1920} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_{2019} & b_{2020}
\end{bmatrix} \quad (3.10)$$

where $b_{11}=j\times\omega\times C_1+(1\div R_{111})+(1\div R_4)+(1\div R_6)$

$b_{12}=-((j\times\omega\times C_1)+(1\div R_4))$

$b_{16}=-1\div R_6$

$b_{21}=-(A_v+(j\times\omega\times C_1)+(1\div R_4))$

$b_{22}=j\times\omega\times C_1+(1\div R_2)+(1\div R_4)+1$

$b_{23}=-1\div R_2$

$b_{32}=-1\div R_2$

$b_{33}=j\times\omega\times C_2+(1\div R_2)$

$b_{34}=-j\times\omega\times C_2$

$$b_{43} = -(A_v + j \times \omega \times C_2)$$

$$b_{44} = j \times \omega \times C_2 + (1 \div R_3) + (1 \div R_7) + 1$$

$$b_{45} = -1 \div R_3$$

$$b_{47} = -1 \div R_7$$

$$b_{54} = -1 \div R_3$$

$$b_{55} = (1 \div R_3) + (1 \div R_5)$$

$$b_{56} = -1 \div R_5$$

$$b_{61} = -1 \div R_6$$

$$b_{65} = -(A_v + (1 \div R_5))$$

$$b_{66} = (1 \div R_5) + (1 \div R_6) + 1$$

$$b_{74} = -1 \div R_7$$

$$b_{77} = j \times \omega \times C_3 + (1 \div R_7) + (1 \div R_{10}) + (1 \div R_{12})$$

$$b_{78} = -((j \times \omega \times C_3) + (1 \div R_{10}))$$

$$b_{712} = -1 \div R_{12}$$

$$b_{87} = -(A_v + (j \times \omega \times C_3) + (1 \div R_{10}))$$

$$b_{88} = j \times \omega \times C_3 + (1 \div R_8) + (1 \div R_{10}) + 1$$

$$b_{89} = -1 \div R_8$$

$$b_{98} = -1 \div R_8$$

$$b_{99} = j \times \omega \times C_4 + (1 \div R_8)$$

$$b_{910} = -j \times \omega \times C_4$$

$$b_{109} = -(A_v + j \times \omega \times C_4)$$

$$b_{1010} = j \times \omega \times C_4 + (1 \div R_9) + (1 \div R_{13}) + 1$$

$$b_{1011} = -1 \div R_9$$

$$b_{1013} = -1 \div R_{13}$$

$$b_{1110} = -1 \div R_9$$

$$b_{1111} = (1 \div R_9) + (1 \div R_{11})$$

$$b_{1112} = -1 \div R_{11}$$

$$b_{127} = -1 \div R_{12}$$

$$b_{1211} = -(A_v + 1 \div R_{11})$$

$$b_{1212} = (1 \div R_{11}) + (1 \div R_{12}) + 1$$

$$b_{1310} = -1 \div R_{13}$$

$$b_{1313} = j \times \omega \times C_5 + (1 \div R_{13}) + (1 \div R_{16}) + (1 \div R_{18})$$

$$b_{1314} = -((j \times \omega \times C_5) + (1 \div R_{16}))$$

$$b_{1318} = -1 \div R_{18}$$

$$b_{1413} = -(A_v + (j \times \omega \times C_5) + (1 \div R_{16}))$$

$$b_{1414} = j \times \omega \times C_5 + (1 \div R_{14}) + (1 \div R_{16}) + 1$$

$$b_{1415} = -1 \div R_{14}$$

$$b_{1514} = -1 \div R_{14}$$

$$b_{1515} = j \times \omega \times C_6 + (1 \div R_{14})$$

$$b_{1516} = -j \times \omega \times C_6$$

$$b_{1615} = -(A_v + j \times \omega \times C_6)$$

$$b_{1616} = j \times \omega \times C_6 + (1 \div R_{15}) + (1 \div R_{19}) + 1$$

$$b_{1617} = -1 \div R_{15}$$

$$b_{1619} = -1 \div R_{19}$$

$$b_{1716} = -1 \div R_{15}$$

$$b_{1717} = (1 \div R_{15}) + (1 \div R_{17})$$

$$b_{1718} = -1 \div R_{17}$$

$$b_{1813} = -1 \div R_{18}$$

$$b_{1817} = -(A_v + 1 \div R_{17})$$

$$b_{1818} = (1 \div R_{18}) + (1 \div R_{18}) + 1$$

$$b_{1916} = -1 \div R_{19}$$

$$b_{1919} = j \times \omega \times C_7 + (1 \div R_{19}) + (1 \div R_{20})$$

$$b_{19120} = -((j \times \omega \times C_7) + (1 \div R_{20}))$$

$$b_{2019} = -(A_v + (j \times \omega \times C_7) + (1 \div R_{20}))$$

$$b_{2020} = j \times \omega \times C_7 + (1 \div R_{20}) + 1$$

$$C = B^{-1} \times A \tag{3.11}$$

where C holds the unknown voltage drops across all the nodes to be evaluated. C_n is the voltage drops across the last node (n) being calculated to get its frequency response. The values of each component are substituted to determine the unknown.

3.5.2 Circuits Simulation Results

3.5.2.1 Example 1: Seventh Order Chebyshev Filter Circuit

Seventh order Chebyshev filter circuit is used as example 1; it is an online tutorial titled "Minimising Component-Variation Sensitivity in Single Op-amp Filters" [184]. The circuit is in Figure 3.7 and the PSpice simulation that produce the frequency response curve shown in Figure 3.8. The circuit's netlist from PSpice is fed to the modified symbolic circuit analysis (MSCAM) program developed to simulate the frequency response curve, which is the same as that of PSpice as shown in Figure 3.8 indicated with different colours and line style. Same technique is applied to other examples. The MSCAM simulation is with the cut-off frequency of 7100 Hz while that of PSpice simulation is 7600 Hz with error of 6.57% and a gain of 1.

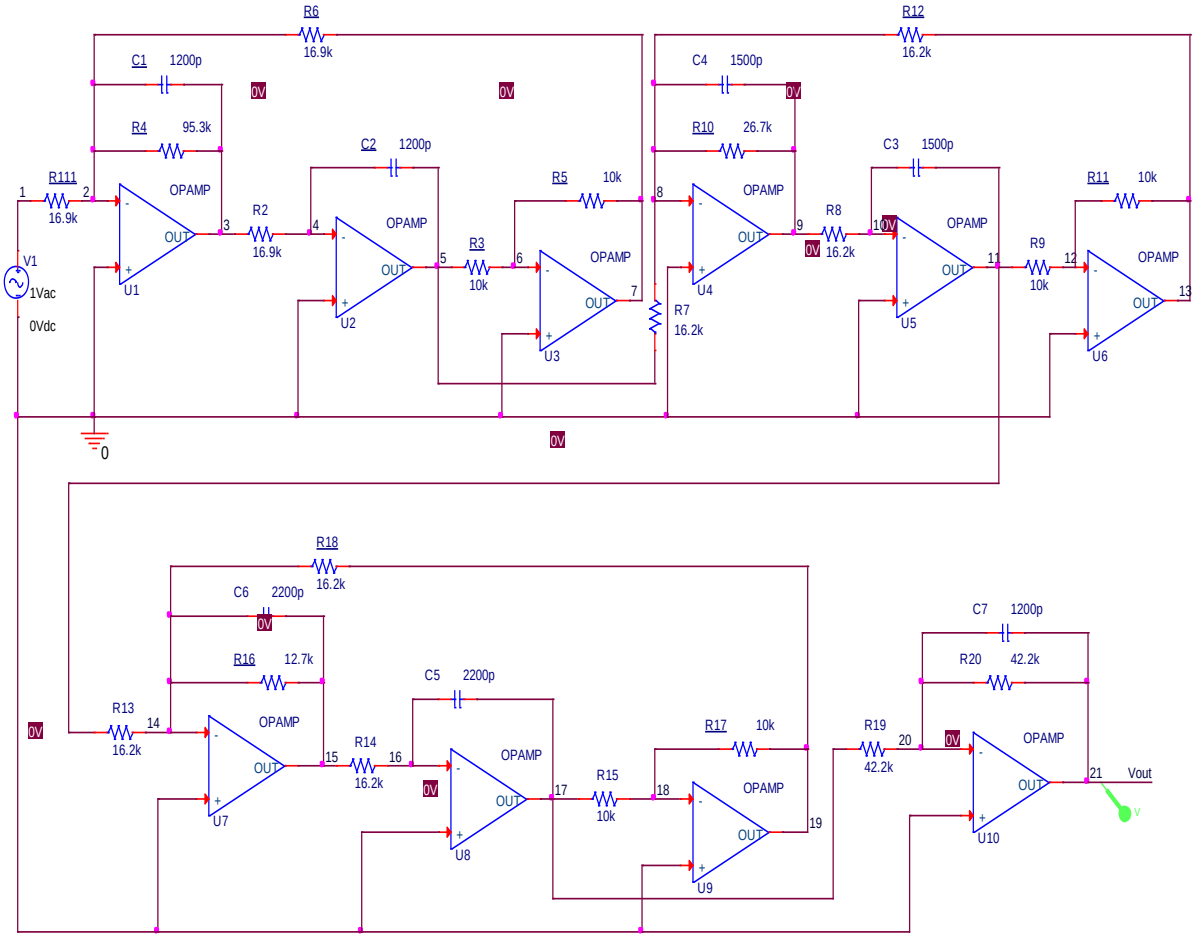

Figure 3.7: Seventh order Chebyshev circuit [184].

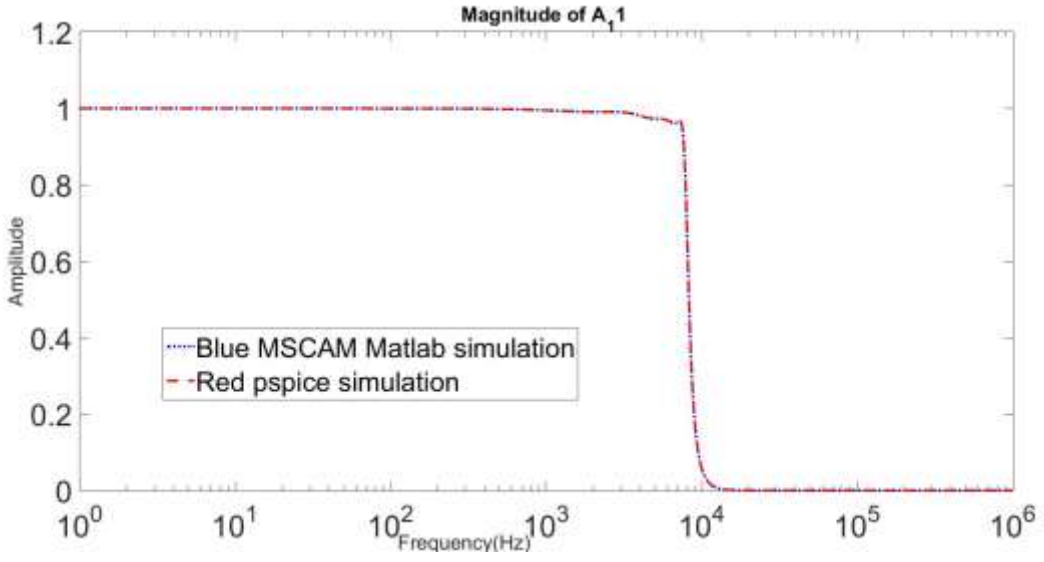

Figure 3.8: Seventh order Chebyshev PSpice (red) and MSCAM (black) frequency response.

3.5.2.2 Example 2: Common-Emitter Circuit

A common-emitter circuit is used as example 2 to demonstrate how a transistor can be transformed to its SSA that replaces the internal operation of transistor. The common emitter circuit is in Figure 3.9. Its small signal equivalent circuit is shown in Figure 3.10 where *gm* = 69.6e-3 and VR_{pi} = 0.713. The SSA frequency response simulated in PSpice and MSCAM simulation is presented in Figure 3.11. The MSCAM and PSpice simulation of the SSA have the same lower and upper cut-off frequencies at 32.5 Hz and 26.1 MHz respectively with approximately zero error and a gain of 75.29.

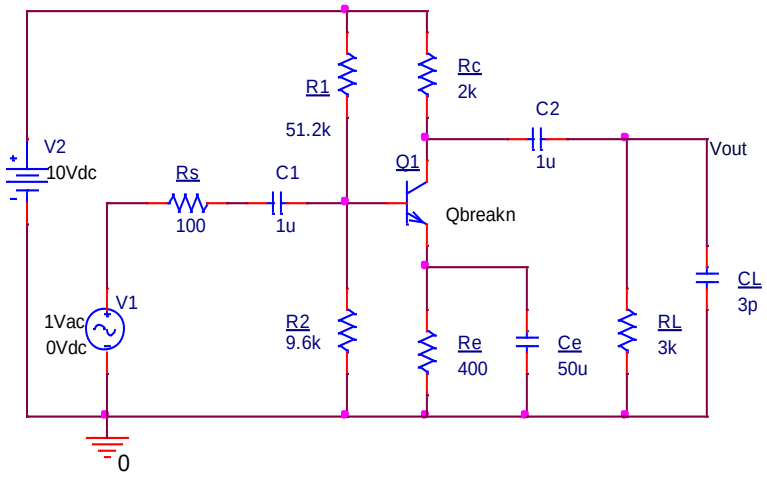

Figure 3.9: Common-emitter circuit.

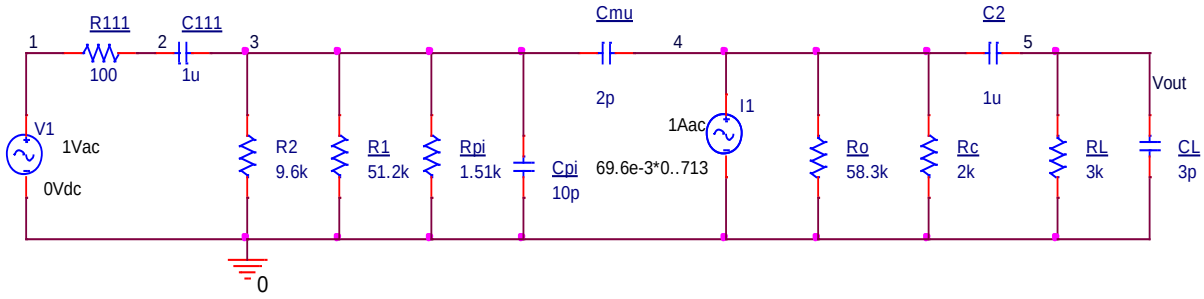

Figure 3.10: Common-emitter SSA.

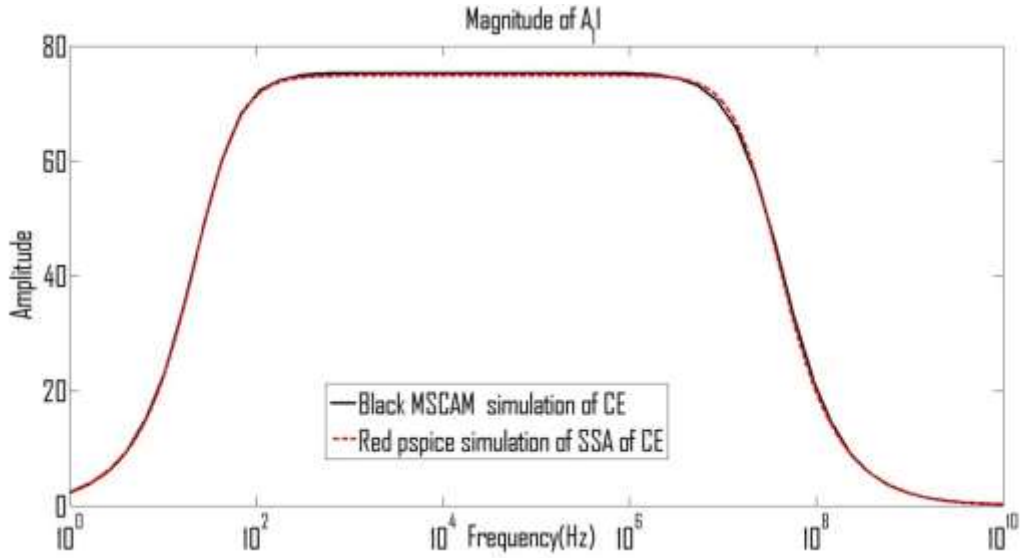

Figure 3.11: Common-emitter SSA PSpice (black) and MSCAM (red) frequency response.

3.5.2.3 Example 3: Combined Operational Amplifier and Transistor Circuit

Example 3 is a circuit that combines op-amp and transistor. It demonstrates how the Matlab code can be applied to implement a circuit that the component comprises of both transistor and op-amp in figure 3.12. Its SSA equivalent circuit is shown in Figure 3.13. The PSpice simulation of the SSA and MSCAM simulation frequency responses are shown in Figure 3.14. The different line colours and styles differentiate the frequency response curve. The MSCAM, SSA and PSpice simulation have the same cut-off frequency of 185 kHz with MSCAM having 0.2 error due to difference in frequency at 100 Hz due to scaling between the software packages and a gain of 3.169.

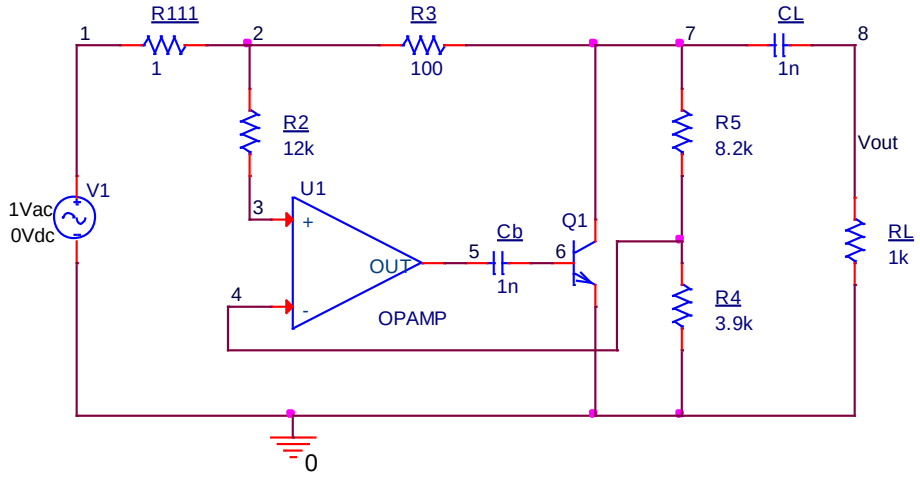

Figure 3.12: Example 3 circuit.

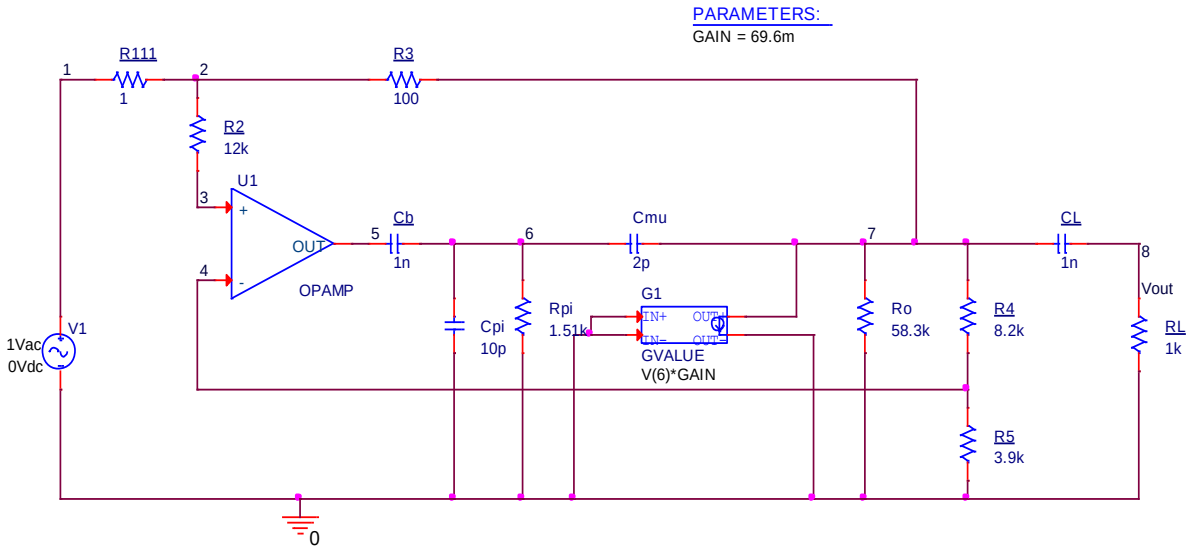

Figure 3.13: Example 3 SSA circuit.

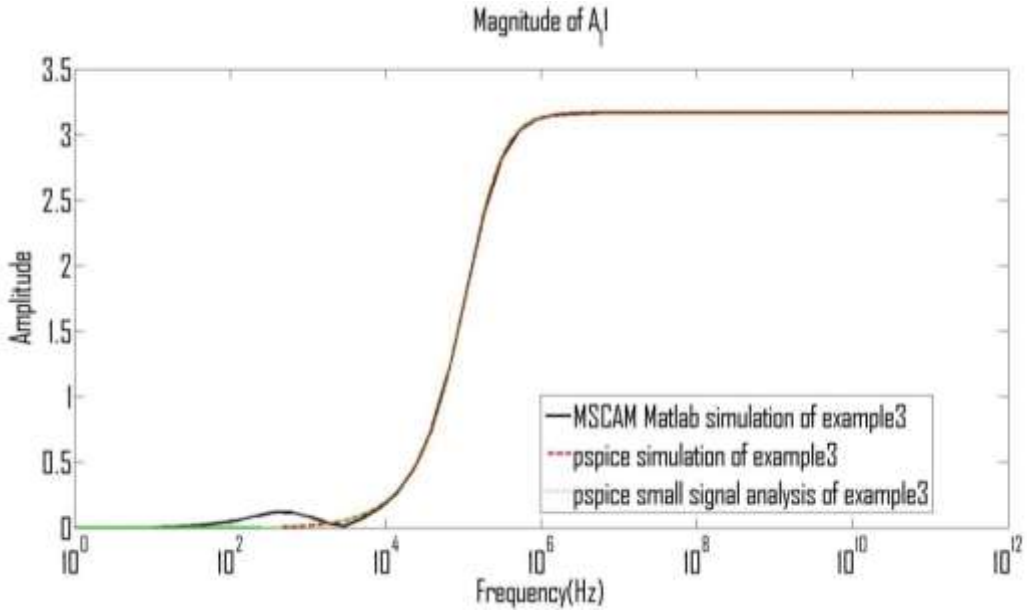

Figure 3.14: Example 3 original circuit PSpice (red), SSA PSpice (green) and MSCAM (black) frequency response curves.

3.5.2.4 Example 4: Common-Source Amplifier Circuit

The common-source amplifier is as example 4 circuit shown in Figure 3.15. The equivalent SSA is illustrated in Figure 3.16. The frequency response curve of both PSpice simulation and MSCAM analysis of the FET amplifier circuit indicated with different colours and line styles in Figure 3.17. The MSCAM and PSpice simulation of the SSA have the same lower and upper cut-off frequencies at 30.9 Hz and 25.7 MHz respectively with approximately zero error and a gain of 2.68.

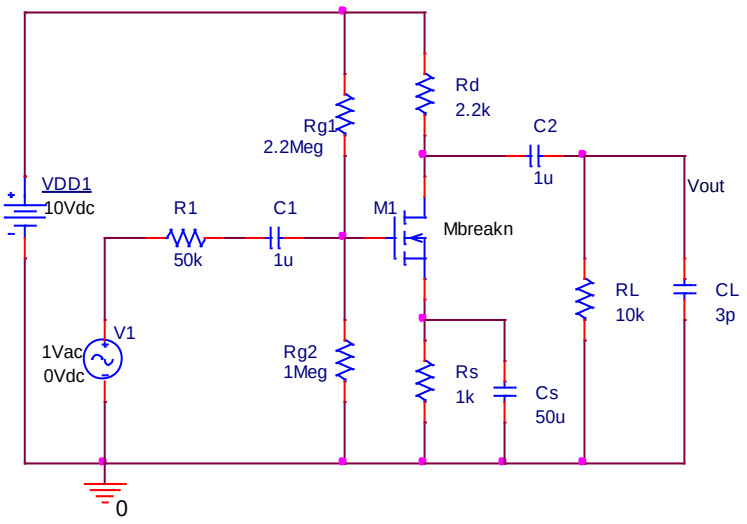

Figure 3.15: Common-source amplifier.

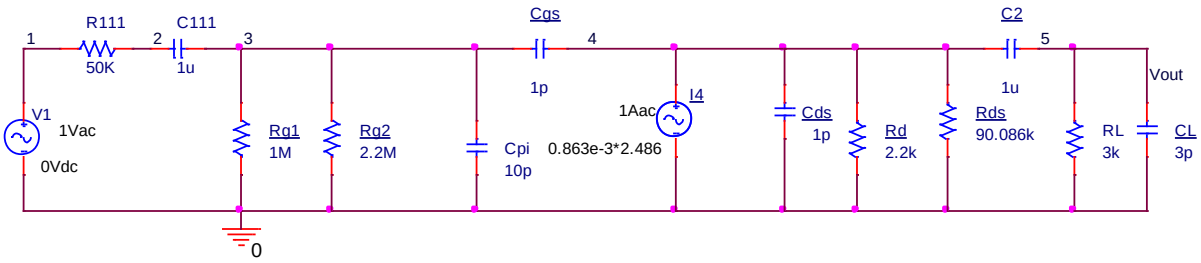

Figure 3.16: Common-source amplifier SSA.

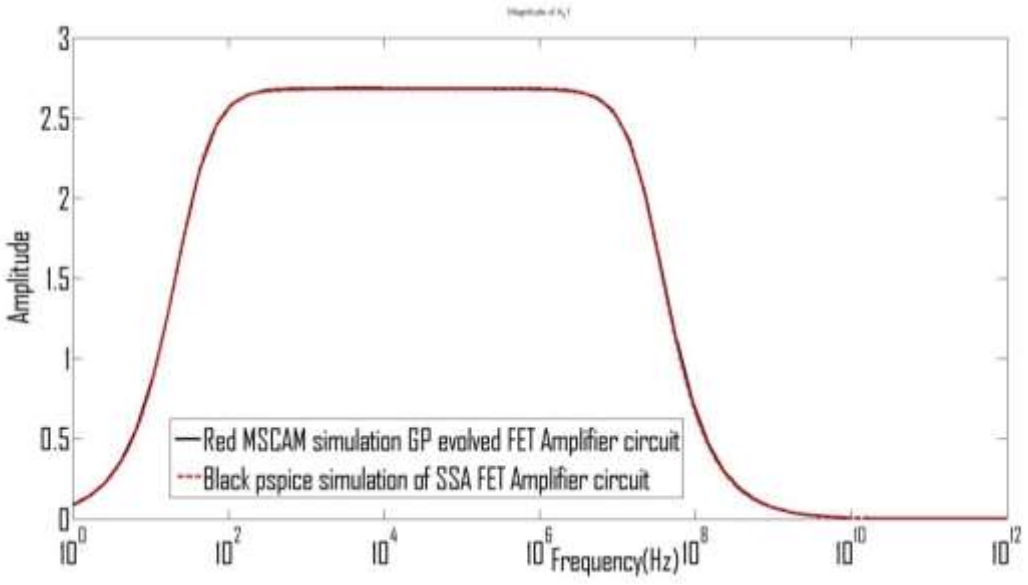

Figure 3.17: Example 4 Pspice (red) and MSCAM (black) SSA frequency response.

Table 3.1 shows the elapsed time for each example and technique. The SCAM simulation capability is limited in terms of the size of circuit and component it can simulate.

Table 3.1: Simulation results time summarised.

Examples	PSpice small signal analysis	MSCAM matrix size /time	SCAM matrix size /time
Example 1 7^{th} order	None but large signal analysis 0.16 sec	20 by 20 0.025 sec	20 by 20 Unable to simulate
Example 2 CE Amp	0.05 sec	5 by 5 0.008 sec	Cannot handle BJT transistor
Example 3 Op-amp/BJT	0.3 sec	8 by 8 0.031 sec	Cannot handle BJT transistor
Example 4 FET Amp.	0.3 sec	5 by 5 0.03 sec	Cannot handle FET transistor

3.6 Summary

A modified symbolic circuit analysis in Matlab (MSCAM) that enhanced circuit's capacity simulation (circuit's matrices dimension) so that it can be used to simulate complex circuits is introduced and a detailed review of symbolic analysis is presented also in this chapter. Brief introduction of electronic circuit simulation techniques is also carried out. This chapter also analysed the SSA of transistor and op-amp. Description of how four different circuits illustrate the developed algorithm using modified nodal analysis in combination with the newly introduced SSA of transistors and Op-amp. Due to the introduction of the transistor SSA, the MSCAM can also simulates circuits with fewer number of BJT and FET. These four examples are presented to demonstrate the efficiency of the developed algorithm. Appendix subdivided into A and B is included in case someone wants to test the four examples.

The next chapter is Chapter 4, which describes five different optimisation tools and how there are applied to optimised analogue circuits

Chapter 4[3]

Analogue Circuit Optimisation

4.1 Introduction

The application of electronics by human activities has become part of life and almost all aspects of human endeavours required it for better functionality or productivity. Electronics is a very rapid growing industry because of its demand in a day-to-day application and the need to overcome some problems or challenges in the society. The objective of an electronics engineer is to design circuits that are smaller, faster, cheaper, and having the ability to communicate wirelessly, to reduce power consumption and increase reliability of the circuits. Human methods of electronic circuit design require tedious mathematical calculations and at times prone to errors. The intelligent optimisation techniques are easy to implement and do not require tedious mathematical computations.

This chapter describes how five intelligent optimisation techniques are used in analogue circuit optimisation. The different optimisation techniques analysed here are: PSO, ABCA, BFO, FA and GA. Besides, Nelder-Mead which is easy and straight forward to be used from the optimisation toolbox is described and used for comparison purpose only and to justify why one should write a complicated programme for the same similar solution. The remaining part of this chapter is subdivided as: the use of Nelder-Mead to minimise analogue circuits, which is discussed in Section 4.2 and application of GA to minimise analogue circuits illustrated in Section 4.3. PSO used to minimise analogue circuits is explained in Section 4.4 while use of BFO to minimise analogue circuits is demonstrated in Section 4.5. Application of FA to minimise analogue circuits is discussed in Section 4.6, ABCA used to minimise analogue circuits is demonstrated in Section 4.7, methodology explained in Section 4.8, results and discussion are illustrated in Section 4.9 and summary of this chapter is in Section 4.10.

[3] A substantial part of Chapter 4 has been published in [5, 185].

4.2 Use of Nelder-Mead to Minimise Analogue Circuits

In Nelder-Mead; fminco-constrained nonlinear minimisation, an objective function is formulated as illustrated in Section 3.5.1 or the one discussed under Section 4.8 and the same objective function is used for all the methods for the same circuit. The objective function is being called right from the optimisation toolbox in the dialog box option called objective function option. The following options are set: algorithm option is set at the interior point and derivative option is set at approximated by solver. The maximum component values are fed at start point option. The maximum and minimum component values are fed at the upper and lower options respectively. The X-tolerance is set at 1E-8 whereas function tolerance is set at 1E-4. These are settings that give the best result.

4.3 Use of Genetic Algorithm to Minimise Analogue Circuits

In the case of GA, the objective function is formulated as illustrated in Section 3.5.1 and it is called directly from Matlab GA optimisation toolbox under the fitness function option. The maximum and minimum component values are fed at the upper and lower options respectively. Also, GA parameters settings include: probability of crossover P_C = 0.8, population size N = 50, probability of mutation P_m = 0.05 and the number of iterations N_{it} = 100. The total number of components to be minimised is specified in the dialog box option called number of variable option. The programme is automatically generated applying generate code option under the file menu in order to rank results. The summaries of parameters applied are in Table 4.1.

Table 4.1: Summary or definition of GA's symbols used.

Symbols	Meaning	Value used	Remarks
P_C	probability of crossover (single point)	0.8	probability of crossover value used to obtain the best approximate frequency response curve
P_m	probability of mutation (Gaussian)	0.05	probability of mutation value used to obtain the best approximate frequency response curve
N_{it}	number of iteration	100	number of iterations used to obtain the best approximate frequency response curve
N	number of individuals (chromosome-binary)	50	number of individuals used to obtain the best approximate frequency response curve

NC	number of components to be minimised	7	number of components to be optimised

4.4 Use of Particle Swarm Optimisation to Minimise Analogue Circuits

As earlier defined, PSO is a population-based stochastic optimisation method developed by Eberhart and Kennedy 1995 [60]. It was inspired by social interaction of flock of bird and school of fish. In comparison with GA, PSO does not execute genetic operators, crossover, reproduction and mutation, but adjusts its velocity dynamically. Also, PSO needs fewer control parameters to obtain the same result as GA control parameters and it does not apply survival of the fittest principle. In PSO, particles explore for the required solution in the problem space the potential solution is updated while the process is ongoing. A swarm is the collection of particles that make up the population. PSO suitable parameter or element selection guide as well as improved PSO algorithm with leadership and detailed PSO algorithm are illustrated in [60, 61].

The following steps are implemented while applying PSO to minimise filter circuits:

a. Formulation of an objective function to calculate fitness as in Section 3.5.1

b. Initialisation of particles using cascode amplifier design component value to illustrate the PSO. Number of iteration N_{it} = 30, number of components to be minimised NC = 7, number of particle N = 20, acceleration constants c_1 = c_2 = 1.49618 (these are values used to obtain the best approximate frequency response curve after several trials), component maximum values = 5 3 1 11 16 16 40 and component minimum values = 0.01 0.3 0.5 0.1 14 14 0.1. The component maximum and minimum values are got in the same magnitude above and below the initial component values but the increment/decrement not more than half of the initial value.

   Each particle generates random component values within a specified range according to expression:

$$X_P = MinValue + (MaxValue - MinValue) \times rand(N \times NC) \quad (4.1)$$

   where *Xp* represents the randomly generated population of size N_C by *N*, *MinValue* is the component Minimum Values, *MaxValue* is the component Maximum Values whereas *N* and N_C as defined in (b).

c. The fitness values of all the particles are calculated.

   The personal best (*pbest*) of each particle is evaluated with respect to the objective function.

d. The fitness of each particle is compared with its former best fitness found. If the current value obtain is better than *pbest*, then set the *pbest* as the current value and *pbest* position as the current position.

e. Compare *pbest* of all the particles with one another and update the *lbest* position with the highest fitness
f. The position and velocity of the particles are altered according to equations (4.2) and (4.3) respectively.

Steps (c) to (f) are repeated until the design criteria are met which is the number of iteration used.

$$x_{t+1} = x_t + v_t \tag{4.2}$$

$$v_{t+1} = v_t w + c_1 \alpha_1 (p_t - x_t) + c_2 \alpha_2 (l_t - x_t) \tag{4.3}$$

where x_t is the position, v_t is the velocity, w is the initial weight, c_1 and c_2 are the acceleration constant, α_1 and α_2 are the random variable, p_t is the *pbest* and l_t is the *lbest*.

4.5 Use of Bacterial Foraging Optimisation to Minimise Analogue Circuits

The use of BFO to minimise circuits involves the following steps:

1. *Initialisation:* In this research, the following were considered: number of bacteria $S = 8$, number of iteration in chemotactic loop $(N_c > N_s)$ $N_c = 5$, number of components $p = 7$, number of swimming length $N_s = 4$, maximum number of dispersal or elimination imposed on bacteria $N_{ed} = 2$, maximum number of reproduction $N_{re} = 4$, probability that dispersal and elimination will continue $P_{ed} = 0.25$ attraction coefficient $w_{attract} = 0.2$, attraction coefficients $d_{attract} = 0.1$, repulsion coefficient $w_{repellent} = 10$, *and* repulsion coefficient $h_{repellent} = 0.1$ component maximum values = 5 3 1 11 16 16 40 and component minimum values = 0.01 0.3 0.5 0.1 14 14 0.1.The objective function is formulated as illustrated in Section 3.5.1
2. $l = l + 1$; dispersal-elimination loop
3. $k = k + 1$; reproduction loop
4. $j = j + 1$; chemotactic loop that is subdivided into:

i. For each bacterium i = 1, 2, 3, …, S, evaluate objective function $J(i, j, k, l)$

a. let $J_{sw}(i, j, k, l) = J(i, j, k, l) + J_{CC}(\theta^{i(}(j, k, l), P(j, k, l))$ (4.4)

where

$$J_{CC} = \sum_{i=1}^{s} [-d_{attract} \exp(-w_{attract} \sum_{m=1}^{p} (\theta_m - \theta_m^i)^2)] + \sum_{i=1}^{s} [h_{repellent} \exp(-w_{repellent} \sum_{m=1}^{p} (\theta_m - \theta_m^i)^2)] \tag{4.5}$$

where $d_{attract}$ is the depth attraction coefficients, $w_{attract}$ is the width attraction coefficient, $h_{repellent}$ is the height repulsion coefficient, $w_{repellent}$ is the width repulsion coefficient θ_m is a given cell location, θ_m^i is the neighbouring cell location, *S* is the number of bacteria and *p* is the number of components.

b. Let $J_{last} = J_{sw}(i, j, k, l)$ to save the value since better cost via run.

c. Loop end

ii. Tumble: create a random vector $\Delta(i), \varepsilon \Re^p$ each element between (-1,1)

iii. Move: let $\phi(i) = \frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$ (4.6)

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(j) \quad (4.7)$$

This is the results in step size $C_{(i)}$ in tumble direction of the *i* th bacterium.

iv. Evaluate $J(i, j, k, l)$ and let

$$J_{sw}(i, j, k, l) = J(i, j, k, l) + J_{CC}(\theta^{i(}(j, k, l), P(j, k, l)) \quad (4.8)$$

v. Swim:

a. let $m = 0$

b. while $m < N_s$

i. let $m = m+1$

ii. $J_{sw}(i, j, k, l) < J_{last}$ *then* $J_{last} = J_{sw}(i, j, k, l)$, $\theta^i(j+1, k, l) = \theta^i(j, k, l) + C(i)\phi(j)$ and use the new $\theta^i(j+1, k, l)$ to compute $J(i, j, k, l)$ as in step 4(iv)

iii. Else $m = N_s$

vi. Increment counter to next bacteria *(i+1)* until all the bacteria undergoes chemotaxis

5. Reproduction

a. For a given *l* and *k*, for each *i* = 1, 2, 3, …, S, let $J_{health}^i = \sum_{j=1}^{j=N_C+1} J_{sw}(i, j, k, l)$ is the *ith* bacterium health sorted in ascending order.

b. The bacteria with minimum health (J_{health}) values split and replaced their parents' location and those highest values die.

6. If $k < N_{re}$, repeat step 2 to 5 until maximum number of reproduction step is complete or next chemotactic loop is started.

7. Dispersal-elimination: for i = 1, 2, 3, …, S, a random number is created and if equal or less than P_{ed}, disperse the bacterium to new location otherwise it should maintain its position.
8. $l < N_{ed}$, repeat step 1; else stop.

4.6 Use of Firefly Algorithms to Minimise Analogue Circuits

1. Initialisation Parameter: The following parameters are initialised: number of components = 7, Max Generation = 100, absorption coefficient $\beta = 0.6$ attractiveness δ_o = 1, randomisation parameter α = 0.01 (α can be varied (decreases gradually as solution approaches optimal) to improve convergence). These are values determined as best values. Component maximum values = 5 3 1 11 16 16 40 and component minimum values = 0.01 0.3 0.5 0.1 14 14 0.1. The objective function is also formulated as illustrated in Section 3.5.1.
2. Light intensity I_i at y_i which is directly proportional to attractiveness is calculated by $\delta = \frac{\delta_0}{1+\beta d^2}$. (4.9)
3. Update firefly positions:
   a. Calculate the square root of the distance between the first firefly and the second firefly as

$$d_{ij} = \left\| y_i - y_j \right\| = \sqrt{\sum_{k=1}^{m} \left(y_{i,k} - y_{j,k} \right)^2} \quad (4.10)$$

   b. Calculate δ with the value of δ_o, and β

$$\delta = \delta_0 e^{-\beta d^2} \quad (4.11)$$

   c. Move firefly *i* toward *j* if (I_i<I_j) according to equation (4.12) bearing in mind that if $+\delta_o$ is attraction and $-\delta_o$ is repulsion.

$$y_i^{t+1} = y_i^t + \delta_0 e^{-\beta d_{ij}^2} (y_j^t - y_i^t) + \alpha \varepsilon_i^t \quad (4.12)$$

   d. Update firefly position
4. Repeat step 2 until maximum iteration

4.7 Use of Artificial Bee Colony Optimisation to Minimise Analogue Circuits

The steps involved are as the following:

1. Initialisation Parameter: The parameters initialise are: colony size (employed bees and onlooker bees) *NP* = 20, the number of food sources is equivalent to half of the colony size (*NP/2*), a food source is discarded by its employed bee if could not be improved through "limit" trials (limit = 100), max Cycle = 2500, runtime = 100, number of parameters of the problem to be optimized (*D* = 7), component maximum values = 5 3 1 11 16 16 40 and component minimum values = 0.01 0.3 0.5 0.1 14 14 0.1.
2. *Search by employed bees*: An employed bee finds a new food source within the vicinity of a current food source by changing one randomly selected position variable value and maintaining other variables unaltered in each iteration. Let the location of the *ith* food source be; $S_i = (S_{i,0}, S_{i,1}, \ldots, S_{i,Ns-1})$ where $S_{i,0}, S_{i,1}, \ldots, S_{i,Ns-1}$ are N_s variables of solution and optimisation problem. To search for neighbouring food source S'_i, a randomly chosen variable S_{ij} in Si altered as S'_{ij}

$$S'_{ij} = S_{ij} + \alpha(S_{ij} - S_{kj}) \qquad (4.13)$$

where α is the uniform random number (-1,1) and S_{kj} stands for the variable at the *jth* location in randomly chosen food source S_k which is one of the employed bees other than S_i. S'_{ij} is set to extreme value in the range if its position falls outside acceptable range. Whenever a new food source is located, the quantity of nectar of both new and current food source are compared. If the quantity of nectar in the new food source is more than the current food source, the employed bee relocates to the new source.
3. *Choice of onlookers:* When all employed bees undergo the process in section 2 above, they share the nectar information about the food sources with onlookers in hive. The onlooker bee chooses a food source based on the quantity of nectar. In other words, good sources acquire more onlookers than bad sources.
4. *Search by onlooker bees:* As each onlooker chooses food source, also it locates a new food source close to the food source. This is achieved by altering a randomly chosen position variable of the source as illustrated in employed bee. The quantities of nectar found in the new food source by onlookers at a food source are compared to determine the best neighbouring food source. If the quantity of nectar found in the best neighbouring food source is better than the food source, the position is moved to the best food source.
5. *Search by scouts:* If all the onlookers end their search, some employed bees become scout controlled by limit. When onlookers and employed bee associated with a

particular food source cannot determine a better neighbouring food source in iteration limit, the source is discarded and the employed in that particular food source becomes a scout. The scout randomly looks for the position of new food source. Whenever it determines a new food source, it again becomes the employed bee in that food source.

6. *Update the best solution:* If all the scouts become employed bees, the location of the best food source discovered so far is updated.
7. *Termination:* The algorithm stops if termination criterion is satisfied, otherwise repeat steps 2-6.

4.8 Methodology

Cascode amplifier circuit is used to illustrate the methodology as a case study. Its original circuit is presented in Figure 4.1 while its minimised circuit is shown in Figure 4.2. The PSpice simulation of SSA module of the original circuit of the cascode amplifier is used to specify the objective function for the minimised circuit and Matlab code is written to get equivalent component values that satisfy the specifications. The minimised circuit is converted into its SSA as in Figure 4.3. The mesh analysis technique is applied to transform the circuit into its matrices form. BFO, Firefly and ABC, GA and PSO algorithms are applied to get equivalent component values for the minimised cascode amplifier circuit. The best individual or particle for a given iteration that satisfied the objective function specification is taken as solution. The same component values of the circuits and the objective function specification are analysed with Nelder-Mead constrained nonlinear minimisation to analyse the effectiveness of the technique. The optimise unit of the circuit is considered as a black box, and output/input impedance is optimised such that they are not affected.

The input impedance *Rs* and output impedance R_L of the circuit are taken as constants and are given fixed value while others remaining components are given range of values as demonstrated in Table 4.2. The upper and lower limits of the component values enable the programme to choose their values at random to evolve a circuit which satisfies the objective function. The multi-objective optimisation function is based on gain, power, upper-frequency band, and lower-frequency band. Figure 4.4 summarises the technique in the form of a flowchart.

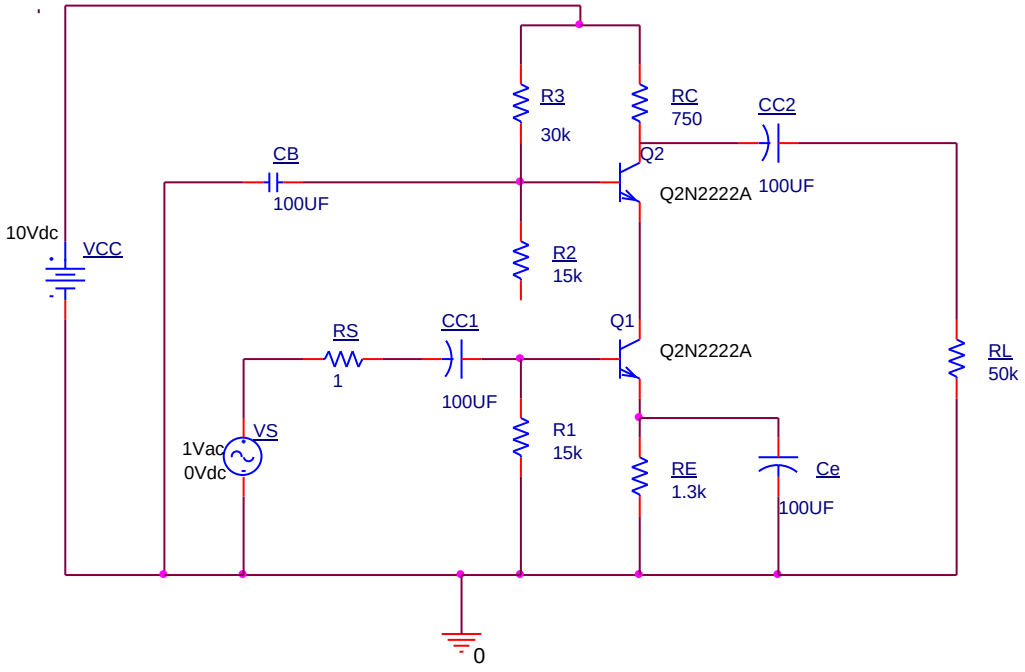

Figure 4.1: Cascode amplifier initial circuit [186].

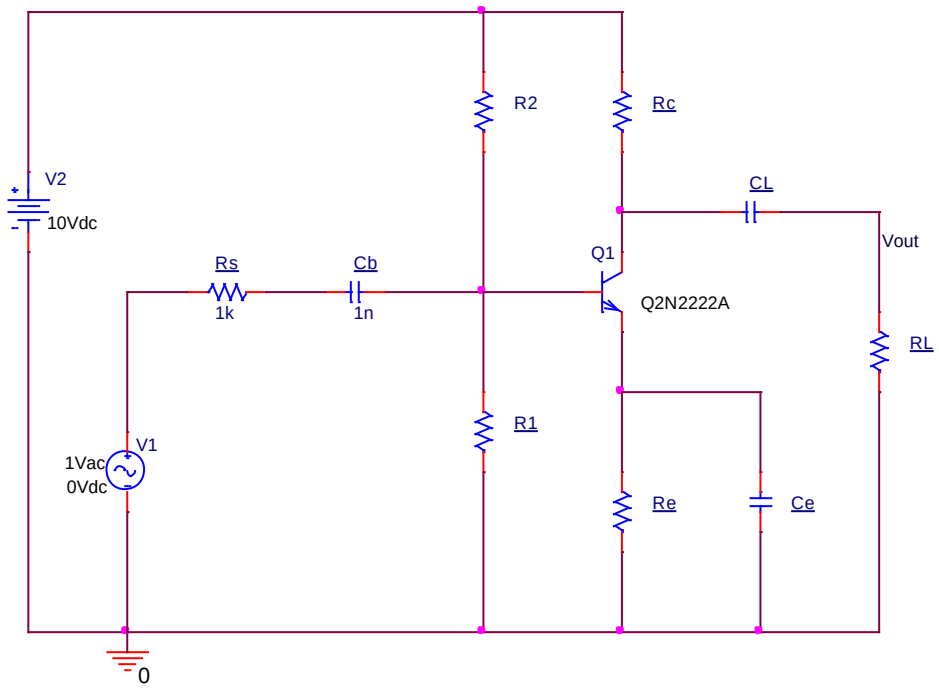

Figure 4.2: Cascode amplifier minimised circuit.

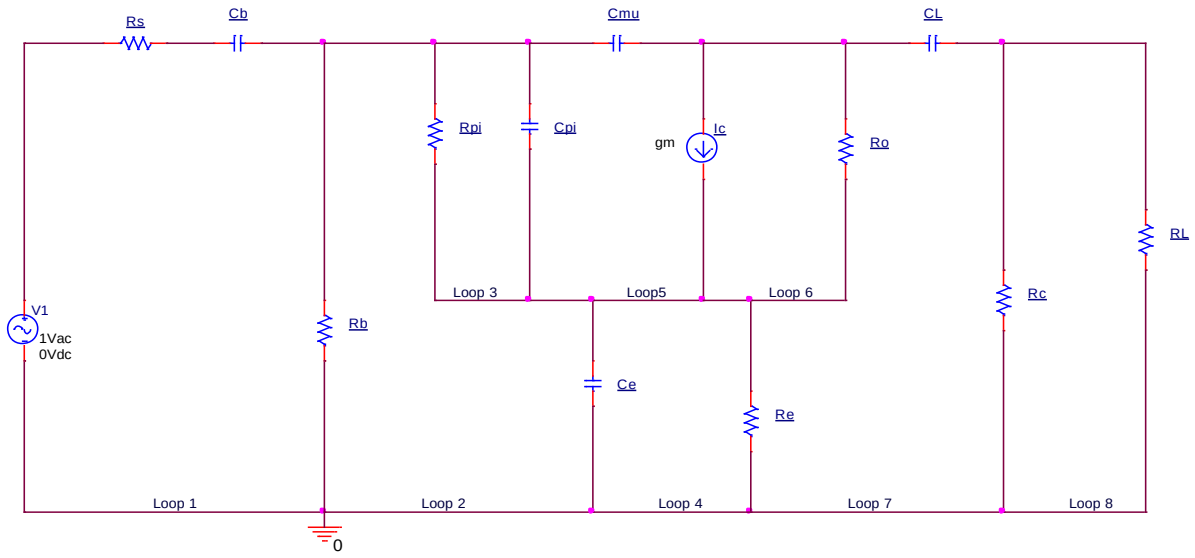

Figure 4.3: SSA for the minimised cascode amplifier circuit.

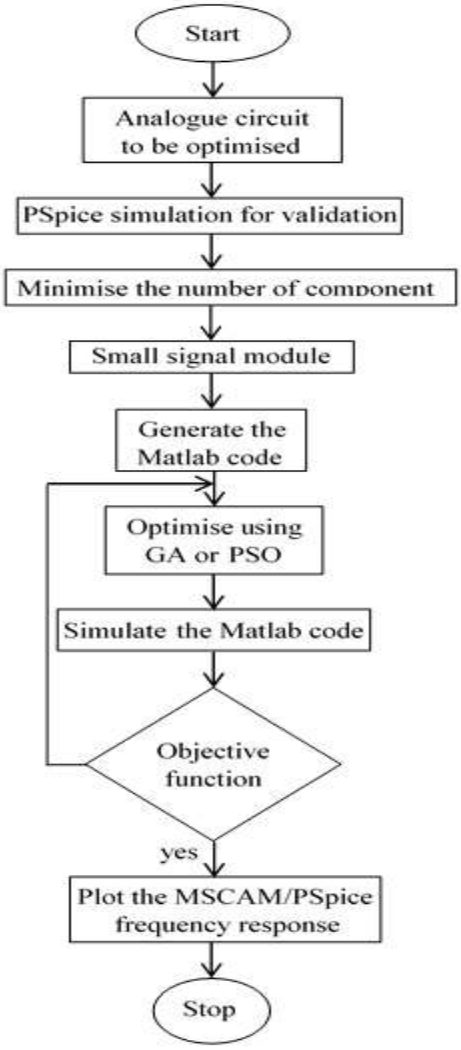

Figure 4.4: The proposed algorithm flow chart.

The weighed objective function is formulated based on:

$$O_f = (cf_{11}/1E^4) + (cf_{12}/1E^8 + (\max\ power \times 10) + (A_v/10) \quad (4.14)$$

where O_f represents objective function, cf_{11} is the difference in value between the achieved and targeted lower-frequency response, cf_{12} is the difference in value between the achieved and targeted upper-frequency response, max-power is the difference in value between the achieved and targeted max-power needed by the circuit, A_v is the range specified for the amplifier gain to be minimised, while R_{out} and R_{in} are the ranges specified for the output and input resistances of the amplifier to be minimised respectively. The components that are given value ranges shown in Table 4.2. Their specified values used in the optimisation are as follow:

cf_{11} = targeted (cf_{11}) – 32.544 Hz

cf_{12} = (targeted (cf_{12}) – 20.444E6)/E6 Hz

Max-power = 0.7236 mW, A_v = 20 to 46, R_{out} = 650 Ω to 800 Ω, R_{in} = 7 kΩ to 7.5 kΩ.

The constants for this work are:

C_{pi} = 10 pF, C_{mu} = 2 pF, Rs = 100 Ω, R_L = 15 kΩ, R_o = 100 kΩ, V_{cc} = 10 V, V_1 = 1 V, V_{be} = 0.648 V, I_C = 1 mA, h_{FE} = 250, q = 1.6E-19 C, k_b = 1.38e-23 J/K, Temperature = 300 K.

Table 4.2: Components ranges.

Component name	Minimum value	Maximum value
C_L (F)	0.01E-6	5E-6
Re (Ω)	0.3E3	3E3
Rc (Ω)	0.5E3	1E3
R_1 (Ω)	14E3	16E3
R_2 (Ω)	14E3	16E3
Cb (F)	0.1E-6	11E-6
Ce (F)	0.1E-6	40E-6

Formulas used in the simulation are:

$$V_T = k_b \times (Temp/q) \tag{4.15}$$

$$R_b = (R_1 \times R_2)/(R_1 + R_2) \tag{4.16}$$

$$\omega = 2 \times \pi \times f \tag{4.17}$$

$$G_m = I_c / V_T \tag{4.18}$$

$$I_B = I_c / h_{FE} \tag{4.19}$$

$$R_{pi} = h_{FE} / G_m \tag{4.20}$$

$$R_{in} = (R_b \times R_{pi})/(R_b + R_{pi}) \tag{4.21}$$

$$R_{out} = (R_c \times R_o)/(R_c + R_o) \tag{4.22}$$

$$A_i = I_C / I_B \tag{4.23}$$

$$A_v = G_m \times R_c \times R_o /(R_c \times R_o)) \tag{4.24}$$

$$A^T = [1;0;0;0;0;0;0;0] \tag{4.25}$$

$$B = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & 0 & a_{35} & 0 & 0 & 0 \\ 0 & a_{42} & 0 & a_{44} & 0 & 0 & a_{47} & 0 \\ 0 & 0 & a_{53} & 0 & a_{55} & a_{56} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} & a_{67} & 0 \\ 0 & 0 & 0 & a_{74} & 0 & a_{76} & a_{77} & a_{78} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{87} & a_{88} \end{pmatrix} \tag{4.26}$$

where $a_{11} = (R_S + R_b + (1/(j \times \omega \times C_b)))$

$a_{12} = -R_b$

$a_{21} = -R_b$

$a_{22} = (R_{pi} + R_b + (1/(j \times \omega \times C_e)))$

$a_{23} = -R_{pi}$

$a_{24} = -(1/(j \times \omega \times C_e))$

$a_{32} = -R_{pi}$

$a_{33} = (R_{pi} + (1/(j \times \omega \times C_{pi})))$

$a_{35} = -(1/(j \times \omega \times C_{pi}))$

$a_{42} = -(1/(j \times \omega \times C_e))$

$a_{44} = (R_e + (1/(j \times \omega \times C_{pi})))$

$$a_{47} = -R_e$$

$$a_{53} = -(1/(j \times \omega \times C_{pi}))$$

$$a_{55} = -I_C$$

$$a_{56} = -(1/G_m)$$

$$a_{65} = -(1/G_m)$$

$$a_{66} = R_o$$

$$a_{67} = -R_o$$

$$a_{74} = -R_e$$

$$a_{76} = -R_o$$

$$a_{77} = (R_o + R_c + R_e$$

$$a_{78} = -R_c$$

$$a_{87} = -R_c$$

$$a_{88} = (R_L + R_c + (1/(j \times \omega \times C_L)))$$

$$C = B^{-1} \times A \qquad (4.27)$$

where C contains unidentified voltages in all the nodes to be found. C_n is the voltage across the load resistor at node n being analysed to determine its frequency response within a given range of frequency (1 Hz to 1 GHz). The frequency response curves in Figure 4.5 are plot of gain in magnitude against its frequency for original circuit, and that of circuits optimised using Nelder-Mead, BFO, Firefly and ABC, GA and PSO. Firefly parameters are specified as: runtime = 100, number of iteration = 150, generation (N) = 40, α = 0.5, γ = 1, β = 0.2. Also, ABCA parameters are set as follow: number of colony size (NP) = 20, food source

which must be improved (limit) = 100, food number = *NP/2*, runtime = 100 and number of cycle for foraging (maxcycle) = 2500.

GA parameters are specified as: crossover (P_c) = 0.8, pop-size = 50, mutation (P_m) = 0.05 and generation = 100. Whereas PSO parameters are set as: initial weight (w = 0.7298), acceleration constants ($c_1 = c_2$ = 1.49618), number of particles = 20 and number of iteration = 30. In addition, BFO parameters selections are: number of bacteria in pop (S) = 8, number of bacteria reproductions (splits) per generation (S_r) = S/2, algorithm runtime = 100, Limits the length of a swim when it is on a gradient (N_s) = 4, number of chemotactic step per bacteria lifetime (N_c) = 5, number of reproduction steps, N_{re} = 4, probability that each bacteria is eliminated (P_{ed}) = 0.25 and number of elimination-dispersal events (N_{ed}) = 2.

4.9 Results and Discussion

The optimised and original circuits are simulated in PSpice to get their frequency response. The 0.707 (-3dB) of the maximum gain is used to set the cut-off frequencies, which serves as reference. Example 1 shows cascode amplifier circuit, Example 2 describes a high-pass filter; Example 3 illustrates a low-pass one and Example 4 for an all-pass filter.

4.9.1 Example 1: Cascode Amplifier Circuit

The results got from example 1 are summarised in Table 4.3. The original circuit simulated in PSpice enabling to obtained bandwidth, power and frequency response. Figure 4.5 shows the frequency response curves with unique line styles for the Nelder-Mead, original circuit, BFO, ABCA, FA, PSO and GA optimised circuits. Results presented have demonstrated that GA and PSO are useful optimisation tools for electronics. However, PSO has proved to be the best among the five-swarm algorithm techniques regarding power reduction and frequency response. Also, results presented revealed that FA, Nelder-Mead, ABCA and BFO are not good enough because they have narrow bandwidth in terms of frequency response and power consumption instead of reduction, it rather increases. It further revealed that PSO is better than GA regarding higher power reduction and better frequency response and they have least mean and standard deviation errors as in Table 4.3. The components mean and standard deviation are presented in Table 4.4.

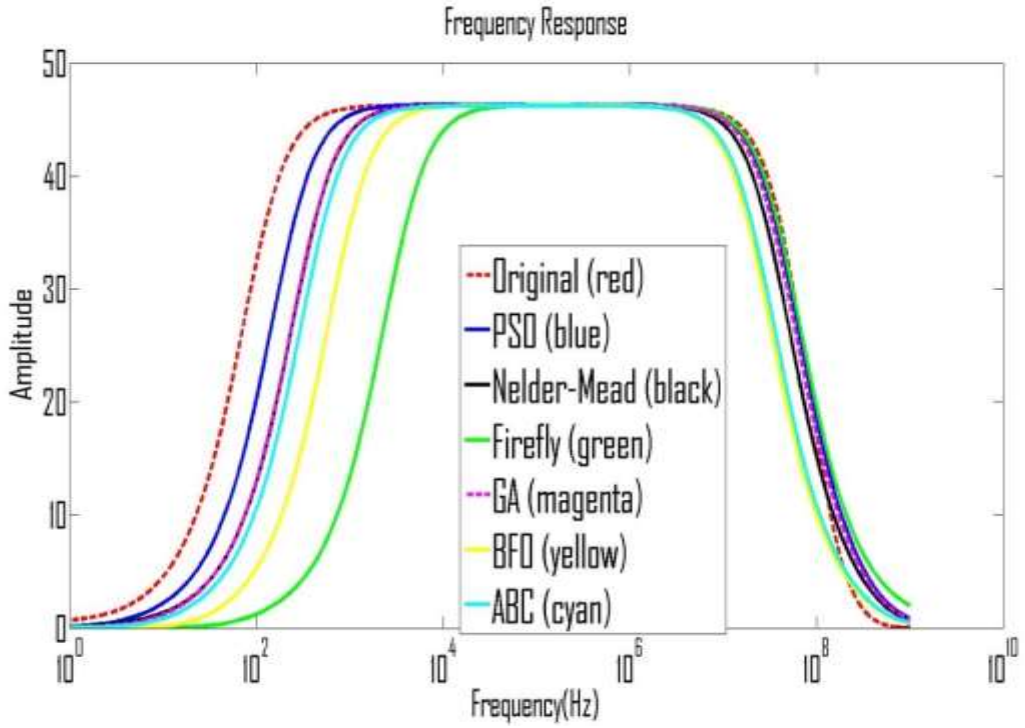

Figure 4.5: Frequency response curve for all the optimised circuits and initial cascade circuit.

Table 4.3: Results obtained from Example 1 simulation.

Circuit element	Initial Circuit	Nelder-Mead	Firefly	ABC	BFO	GA	PSO
C_L (µF)	100	18.74	13.07	11.98	24.79	6.86	14.66
Re (kΩ)	1.3	1.91	1.92	0.99	2.70	1.43	2.76
Rc (Ω)	750	504	990	621	533	665	549
Cb (µF)	100	9.76	0.11	5.49	1.57	6.13	11
R1 (kΩ)	15	14.75	0.50	4.99	12.82	4.86	9.57
R2 (kΩ)	15	14.75	4.20	7.10	3.81	14.13	12.12
Ce (µF)	100	48.73	26.42	50	24.57	30.58	50
R_3 (kΩ)	30	-				-	-
C_4 (µF)	100	-				-	-
Q2N2222	2	1	1	1	1	1	1
Rs (Ω)	1	1	1	1	1	1	1
R_L (kΩ)	50	50	50	50	50	50	50
V_1ac (volt)	1	1	1	1	1	1	1
V_2dc (volt)	12	12	12	12	12	12	12

Earth	1	1	1	1	1	1	1
cf_1 (Hz)	102	338.80	3.27E3	410.87	831.80	338.80	204.20
cf_2 (MHz)	48.95	36.31	47.70	26.02	23.24	45.69	47.75
Power (mW)	22.10	25.5	23.6	42.4	31.9	18.4	18.0
Percentage of power change		+15.4%	+6.79%	+91.86%	+44.3%	-16.7%	-18.6%
Number of component	16	13	13	13	13	13	13
Objective function error		6.74E3	53.44	7.47E3	6.98E3	6.83E3	6.70E3
Number of trials			100	100	100	100	100
Mean error			56.316	6454	5123	26.658	26.587
Std error			0.2026	5304	39.106	0.0043	0.0255

Table 4.4: Mean and standard deviation results obtained from Example 1 simulation.

Device	Firefly		ABC		BFO		GA		PSO	
	mean	std	mean	std	mean	std	mean	std	mean	std
CL (F)	10.50	3.214	9.368	5.842	10.63	4.270	1.257	1.014	20.00	0
Re (Ω)	0.655	0.501	1.005	0.571	9.156	5.887	0.343	0.171	0.013	0.030
Rc (Ω)	4.234	0.021	4.935	0.595	4.175	7.230	5.041	0.585	4.013	0.126
Cb (F)	5.056	1.526	5.031	2.985	8.659	5.606	6.057	2.736	9.861	0.816
R1 (Ω)	0.508	0.017	0.948	0.310	21.56	7.116	1.022	0.285	0.506	0.036
R2 (Ω)	8.985	0.186	8.971	0.301	6.464	8.398	9.106	0.302	8.500	0
Ce (F)	3.292	6.296	15.73	8.424	14.81	4.272	0.153	0.053	0.102	0.000

4.9.2 Example 2: High-Pass Filter Circuit

The original circuit (high-pass filter) [187] is in Figure 4.6 while the minimised Nelder-Mead circuit is in Figure 4.7. The optimised and original circuit's frequency response curve with unique line style is shown in Figure 4.8. The component values for minimised PSO, GA circuit and summary of results obtained simulated for the high-pass filter is in Table 4.5.

For high and low unity gain filters,

$$A_V = 1 = -2Q^2 \tag{4.28}$$

which means that Q = 0.707. The analysis simply implies that, the quality factor is positioned at 0.707.

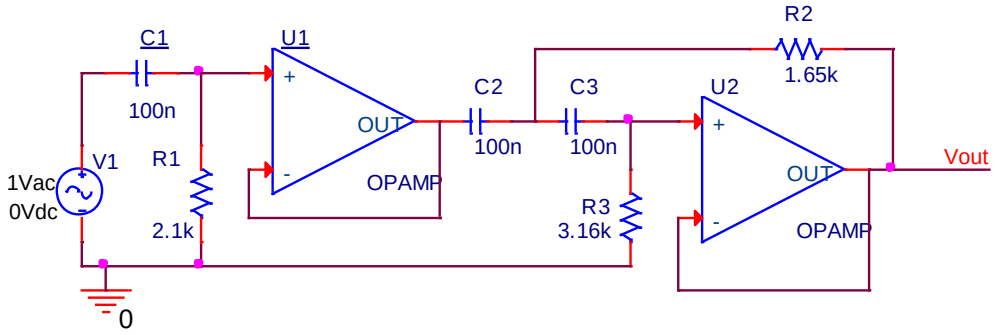

Figure 4.6: Original high-pass filter circuit [187].

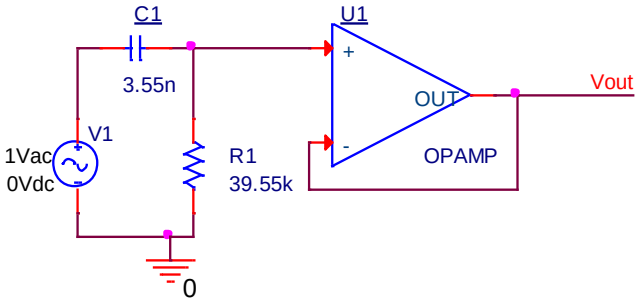

Figure 4.7: Nelder-Mead optimised high-pass filter circuit.

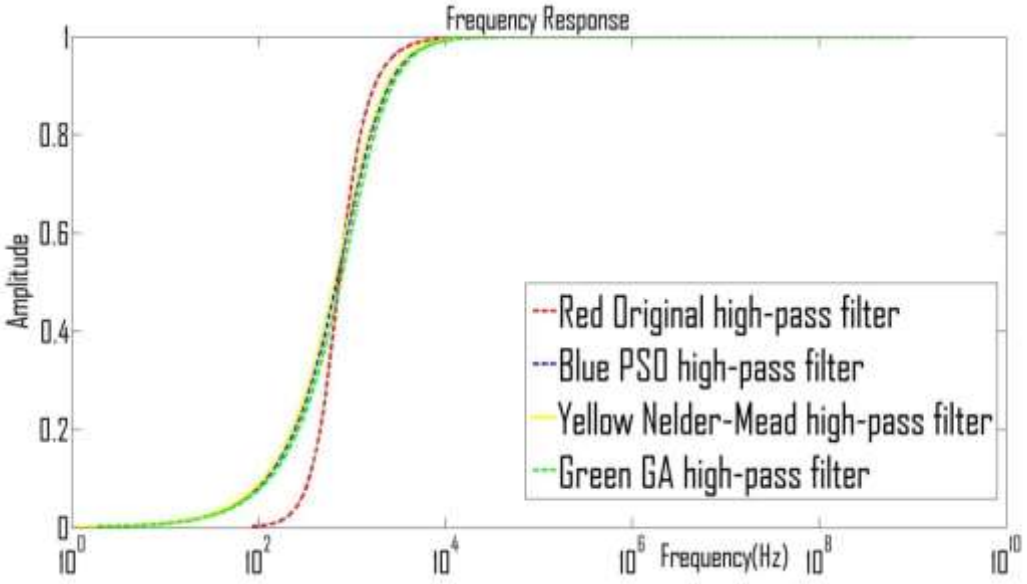

Figure 4.8: Frequency response curve for the high-pass filter.

Table 4.5: Results obtained from Example 2 simulation.

Circuit element	Initial Circuit	Nelder-Mead	GA	PSO
C_1 (nF)	100	3.55	3.16	3.60
C_2 (nF)	100	-	-	-
C_3 (nF)	100	-	-	-
R_1 (kΩ)	2.1	39.55	38.16	37.00
R_2 (kΩ)	1.65	-	-	-
R_3 (kΩ)	3.16	-	-	-
Op amp U_1	1	1	1	1
Op amp U_2	1	-	-	-
Ground	1	1	1	1
V_1 (ac volt)	1	1	1	1
No. of Components	10	5	5	5
Component reduction percentage	-	50%	50%	50%
Elapsed time	-	-	3.36 seconds	0.66 seconds
Objective function error	-	4.82E3	3.2E3	3.2E3

4.9.3 Example 3: Low-Pass Filter Circuit

The original circuit (low pass filter) [187] is shown in Figure 4.9, while the minimised Nelder-Mead circuit is in Figure 4.10. The optimised and original circuit's frequency response curve with unique line style is shown in Figure 4.11. The summary of simulated results and component values for minimised PSO, GA circuit for the high pass filter are in Table 4.6.

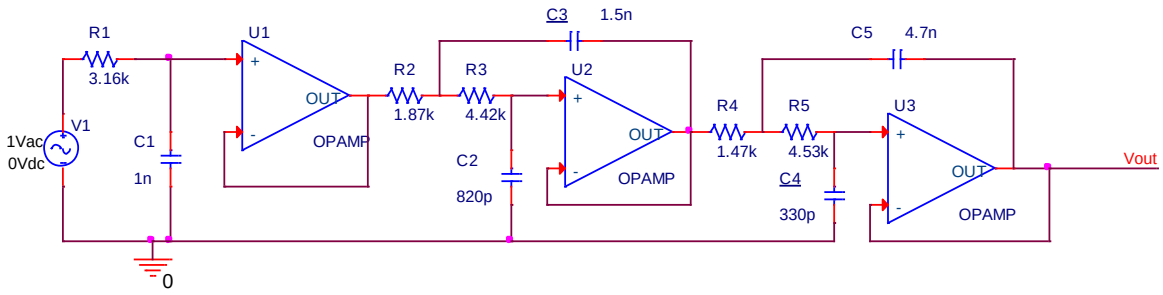

Figure 4.9: Original low-pass filter circuit [187].

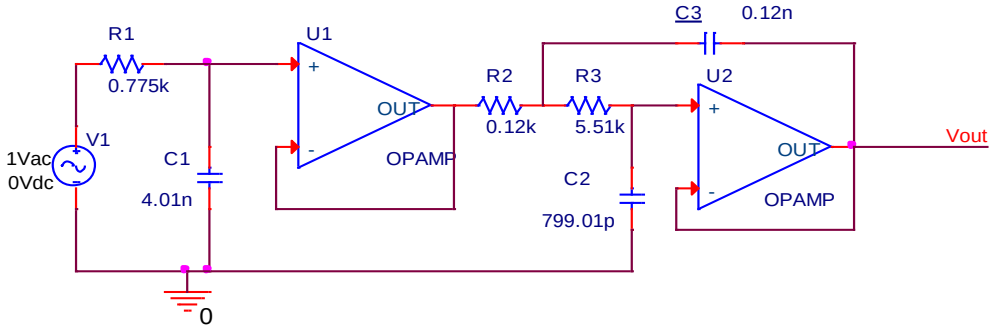

Figure 4.10: Nelder-Mead optimised low-pass filter circuit.

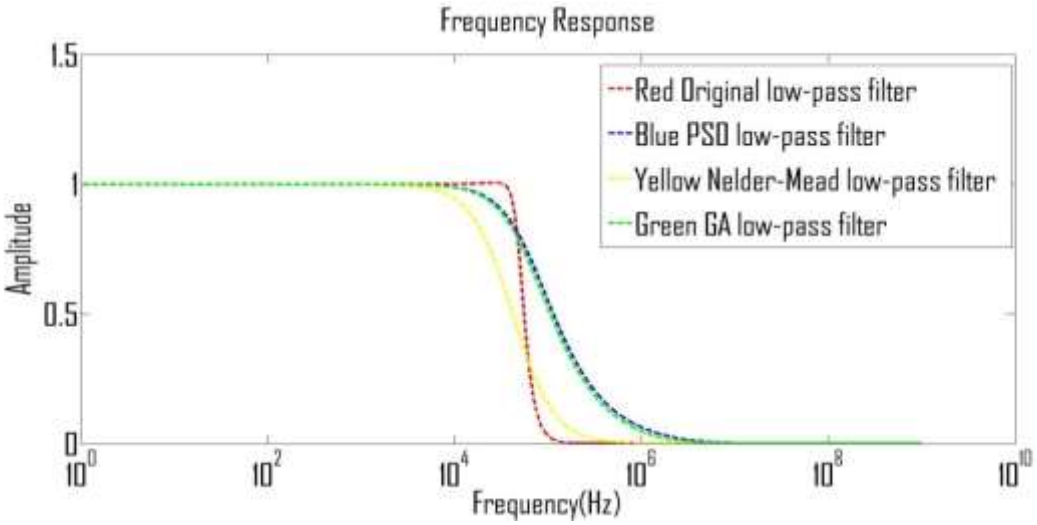

Figure 4.11: Frequency response curve for the low-pass filter.

Table 4.6: Results obtained from Example 3 simulation.

Circuit elements	Initial Circuit	Nelder-Mead	GA	PSO
C_1 (nF)	1	4.01	0.24	0.1
C_2 (pF)	820	799.01	786.66	793.55
C_3 (nF)	1.5	0.12	0.02	0.2
C_4 (pF)	330	-	-	-
C_5 (nF)	4.7	-	-	-
R_1 (kΩ)	3.16	0.775	0.54	0.54
R_2 (kΩ)	1.87	0.12	0.09	0.02
R_3 (kΩ)	4.42	5.51	3.19	3
R_4 (kΩ)	1.47	-	-	-
R_5 (kΩ)	4.53	-	-	-
Ground	1	1	1	1
Op amp U_1	1	1	1	1
Op amp U_2	1	1	1	1
Op amp U_3	1	-	-	-
V_1 (ac volt)	1	1	1	1
No. of Components	15	10	10	10
Component reduction percentage	-	33.33%	33.33%	33.33%
Elapsed time	-	-	4.98 seconds	0.99 seconds
Objective function error	-	843.03	792.03	792.03

4.9.4 Example 4: All-Pass Filter

The original circuit (all-pass filter) [187] is in Figure 4.12, while the minimised Nelder-Mead circuit is in Figure 4.13. The optimised and original circuit's frequency response curve with unique line style is shown in Figure 4.14. The summary of simulated results and component values for minimised PSO, GA circuit for the all-pass filter are in Table 4.7.

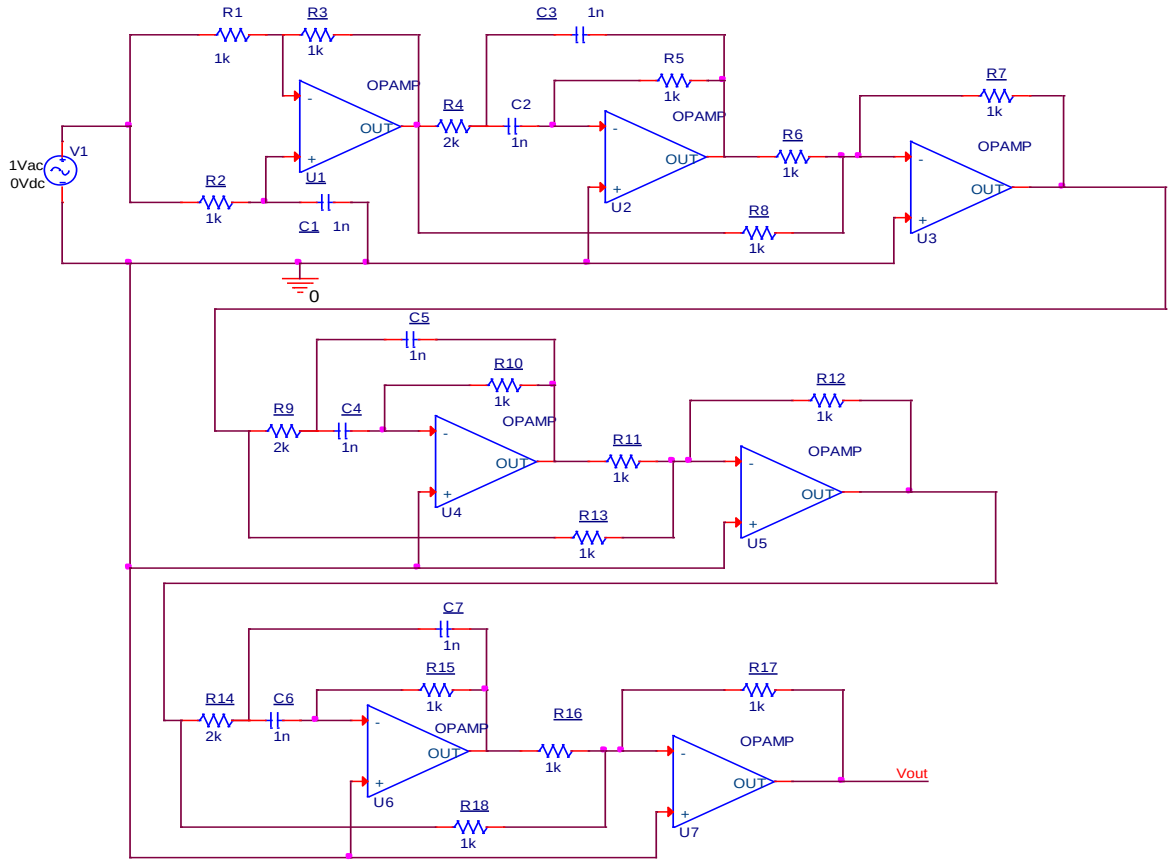

Figure 4.12: Original 7th order all-pass filter circuit [187].

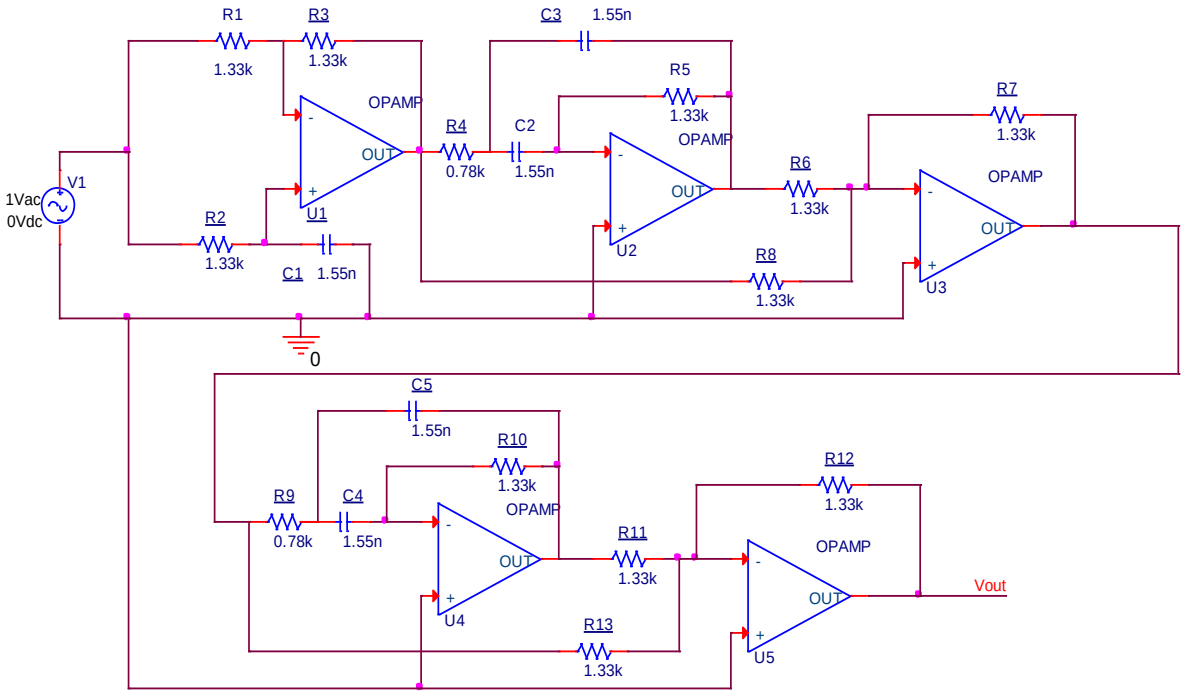

Figure 4.13: Nelder-Mead optimised all-pass filter of the 7th order circuit.

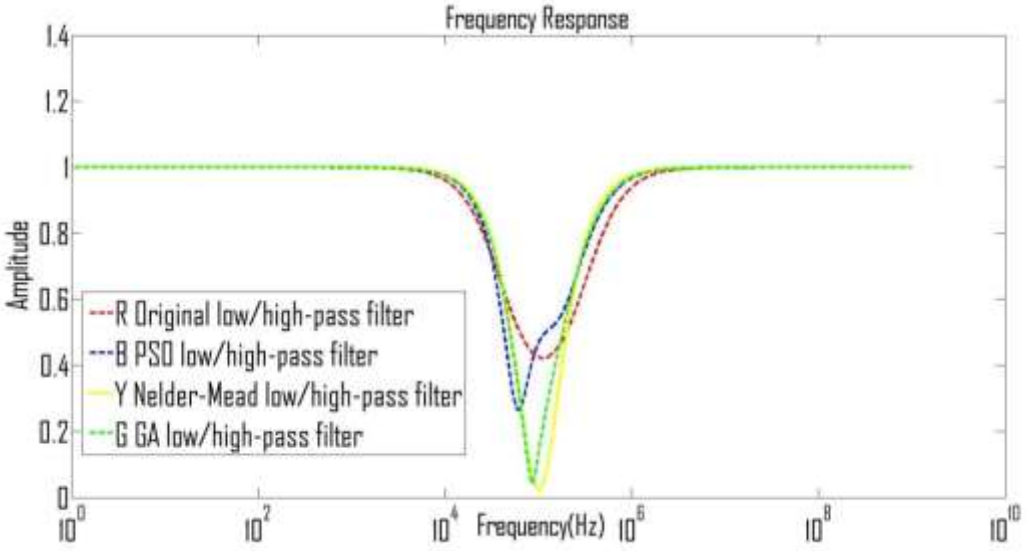

Figure 4.14: Frequency response curve for the all-pass filter.

Table 4.7: Results obtained from Example 4 simulation.

Circuit element	Initial Circuit	Nelder-Mead	GA	PSO
C_1 (nF)	1	1.55	1.76	1.82
C_2 (nF)	1	1.55	1.52	1
C_3 (nF)	1	1.55	1.52	1
C_4 (nF)	1	1.55	1.58	1.09
C_5 (nF)	1	1.55	1.58	1.09
C_6 (nF)	1	-	-	-
C_7 (nF)	1	-	-	-
R_1(kΩ)	1	1.33	0.65	1.01
R_2 (kΩ)	1	1.33	0.64	0.54
R_3 (kΩ)	1	1.33	0.65	1.01
R_4 (kΩ)	2	0.78	0.84	1
R_5 (kΩ)	1	1.33	1.76	1.59
R_6 (kΩ)	1	1.33	0.58	0.5
R_7 (kΩ)	1	1.33	0.59	0.5
R_8 (kΩ)	1	1.33	0.59	0.5
R_9 (kΩ)	2	0.78	0.99	1
R_{10} (kΩ)	1	1.33	0.74	1.32
R_{11} (kΩ)	1	1.33	1.76	2
R_{12} (kΩ)	1	1.33	1.97	1.43

R_{13} (kΩ)	1	1.33	1.97	1.43
R_{14} (kΩ)	2	-	-	-
R_{15} (kΩ)	1	-	-	-
R_{16} (kΩ)	1	-	-	-
R_{17} (kΩ)	1	-	-	-
R_{18} (kΩ)	1	-	-	-
Ground	1	1	1	1
Op amp U_1	1	1	1	1
Op amp U_2	1	1	1	1
Op amp U_3	1	1	1	1
Op amp U_4	1	1	1	1
Op amp U_5	1	1	1	1
Op amp U_6	1	-	-	-
Op amp U_7	1	-	-	-
V_1 (ac volt)	1	1	1	1
No. of Components	34	25	25	25
Component reduction percentage	-	26.47%	26.47%	26.47%
Elapsed time	-	-	18.55 seconds	4.05 seconds
Objective function error	-	3.377E5	2.188E5	2.188E5

Table 4.8 illustrates the cut-off frequencies located at the quality factor points. Results presented have revealed that, this technique can be used to reduce components in high, low and all pass filters, particularly in appliance where a phase angle change has no effect.

Table 4.8: Showing the cut-off frequencies for the original and optimised filter circuits.

Circuit type	High-pass filter	Low-pass filter	All-pass filter circuit	
			Low-pass	High-pass
Original	1 kHz	33 kHz	10.30 kHz	100.5 kHz
Nelder-Mead	1.1 kHz	29 kHz	10.32 kHz	100.2 kHz
GA	1.3 kHz	34 kHz	10.31 kHz	100.3 kHz
PSO	1.2 kHz	34 kHz	10.30 kHz	100.4 kHz

4.10 Summary

This work presents four different swarm optimisation and GA algorithms for analogue circuit optimisation. The results of the first example in this chapter illustrated showed how equivalent analogue circuit can be found in terms of cut-off frequencies but do not give exact performance as the initial circuit. The examples further showed that component count reduction is achieved in analogue circuit same as it has been accomplished in digital circuits. The original circuit has sixteen (16) components count whereas the optimised five methods (Nelder-Mead, FA, BFO, ABC, PSO, and GA) all have thirteen components count but at different level of power consumption as illustrated in Table 4.3. In this approach, BFO, Nelder-Mead, ABC, and FA increased power consumption indicated at different values. However, PSO and GA reduced power consumption indicated at different levels.

Op-amp filter circuits are minimised in the other three examples. In the high-pass filter circuit, a third order filter is minimised to a single stage op-amp and it obtains an equivalent result as that of third order with a component count reduction of five. In the low-pass filter circuit, the fifth order filter circuit minimised to a three-stage op-amp filter and it obtains an equivalent result in terms of cut-off frequencies as that of a fifth order one with a component count reduction of five. In the all-pass filter, the seventh order filter is minimised to a five stage op-amp with a component count reduction of nine. It means that with computer programme, a lower order op-amp filter can be coded in such a way to realise a higher order op-amp filter by finding the quality factor of 0.707 with its corresponding frequency as detailed by the original circuit. In addition, PSO offers the best results regarding frequency response for the four examples, followed by GA whereas Nelder-Mead gives the worst result.

Next is Chapter 5, which describes GP algorithm developments and how it is tested with four different benchmark functions.

Chapter 5[4]

Genetic Programming

5.1 Introduction

In artificial intelligence, GP is an 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.

This chapter demonstrates how GP and GF algorithms are used to develop a standalone optimisation tool and how the developed algorithm is tested with four different benchmark functions. The remaining part of this chapter is subdivided as: GP which is discussed in Section 5.2 and GF which is illustrated in Section 5.3. Specifications of the objective function for benchmark testing that is presented in Section 5.4 while algorithm benchmark testing on mathematical functions which is demonstrated in Section 5.5 and the summary of this chapter is in Section 5.6.

The GP and the GF algorithms discussed in Sections 2.4.2 and 2.4.3 respectively are used to develop a computer code. The GF, MSCAM, and automatically simulated Netlist is introduced into existing GP which is a new contribution in this work, it enhances the development of independent Matlab toolbox. The simulator uses only Matlab compare to existing GP which combine Matlab and PSpice. The newly developed code is then tested for its efficiency using four benchmark expressions. A flowchart shown in Figure 5.1 summarises the GP algorithm and the benchmark testing procedure used in this work. A randomly generated population is calculated to ascertain how well each individually evolving expression is performing with regard to its individual objective function. If the evolving

[4] The bulk of Chapter 5 has been published in [188]

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 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 5.1.

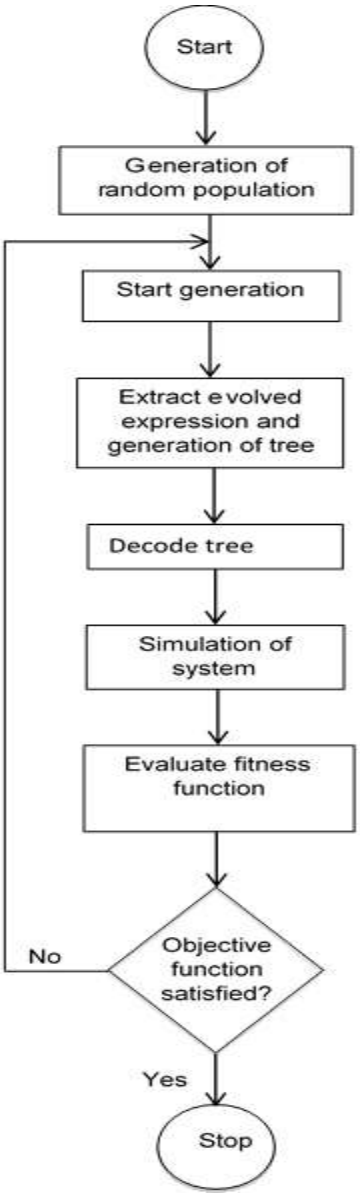

Figure 5.1: The GP algorithm for benchmark testing.

5.2 Genetic Programming

Detailed information about GP is discussed in Section 2.4.2. Here the GP analysis is centred on how the 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 chromosome that is being determined by the length of a TS or a function to be tested. For this case study, benchmark testing Expression 5.2 in Section 5.5.1 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 [46], 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:

    a) Each programme in the population is executed such that a fitness measure that specifies how well the problem is solved is clearly formulated.
    b) 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.

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

5.2.1 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 equation 5.2 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.

5.2.2 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

5.2.3 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 5.2.2 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.

5.2.4 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.

5.2.5 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.

5.3 Genetic Folding

As discussed earlier in Section 2.4.3, GF is the structural arrangement of genes in order of linear numbers separated by dots. In this research, the GF is used to show how elements are structurally linked from beginning to end, so that the expression can be substituted with respective values of *X* and *Y*. The GF representation of GP TS of Figure 5.11 is shown in Table 5.1 (Figure 5.11 is used in this case because it is the desired TS).

Table 5.1: 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**

The GF begins with the series operator in location 1 and ends with element *'Y'* in position 47. The 1st plus operator has the 2nd plus operator in location 2 and the 3rd plus operator in location 3. The plus operator in location 2 has multiplication operators in location 3 and 5. Also, the plus operator in location 3 has multiplication operator in location 6 and minus operator in location 7. The multiplication operator in location 4 has multiplication operator in location 8 and 7 as terminal in location 9 (terminal always end a branch) and so on. Terminals are defined using their indices location. 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 stand 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*.

5.4 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. The same system specification is used in Chapter 6. *X* is given a range of value from -10 to 10 with 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:

- *Z'* is the reshaped matrix of size 1 by 441
- *Z* is the reshaped matrix of size 1 by 441

$$W = RMS(Z' - Z) \tag{5.1}$$

where *W* is the error, *Z'* is the original expression, and *Z* is the GP evolving expression, the error controls the algorithm toward the required solution. The algorithm produces optimal solution when the error is zero.

5.5 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 [189]. 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. [190] 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.

5.5.1 Benchmark Testing Expression 1

$$Z' = X^3Y^2 + 3X^2Y + Y^2 - 4X + 7 \quad (5.2)$$

Both *X* and *Y* is given a range of values from -50 to 50 with an interval of 1 and a three-dimensional plot is represented after each iteration TS representation. The objective function specification is described in Section 5.4. The GP algorithm evolved the expression with 593.28 errors in 1st iteration and the GP TS is shown in Figure 5.2, its three-dimensional plot is represented in Figure 5.3 and the plot of errors against generations is shown in Figure 5.4.

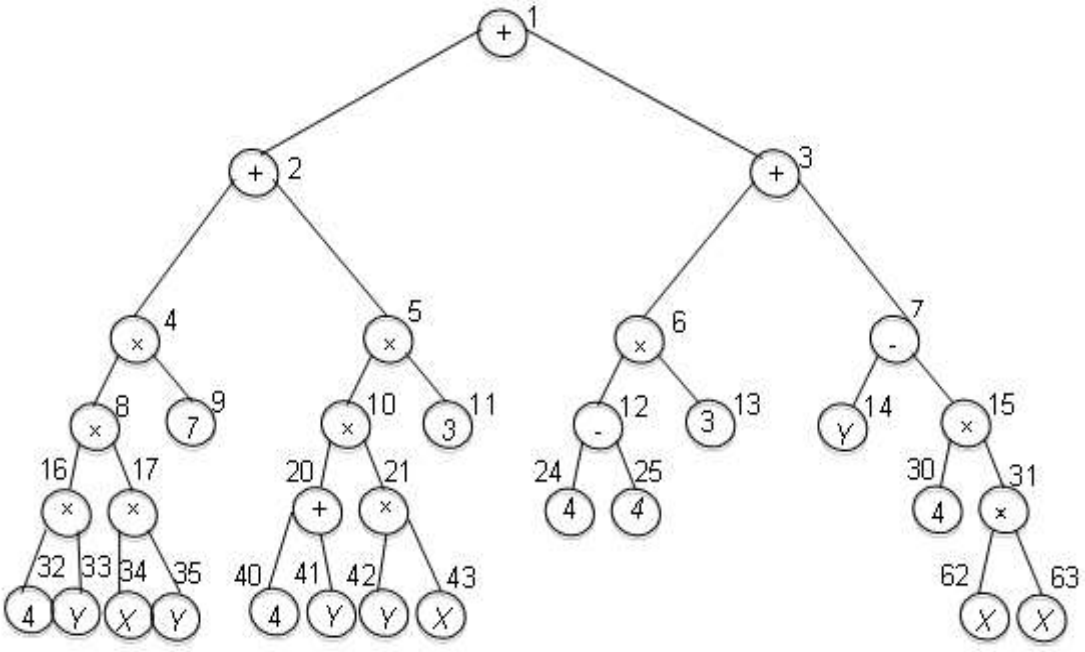

Figure 5.2: 1st iteration GP evolved TS for expression in equation 5.2 with 593.28 errors.

Simplification of the above TS gives;

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

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

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

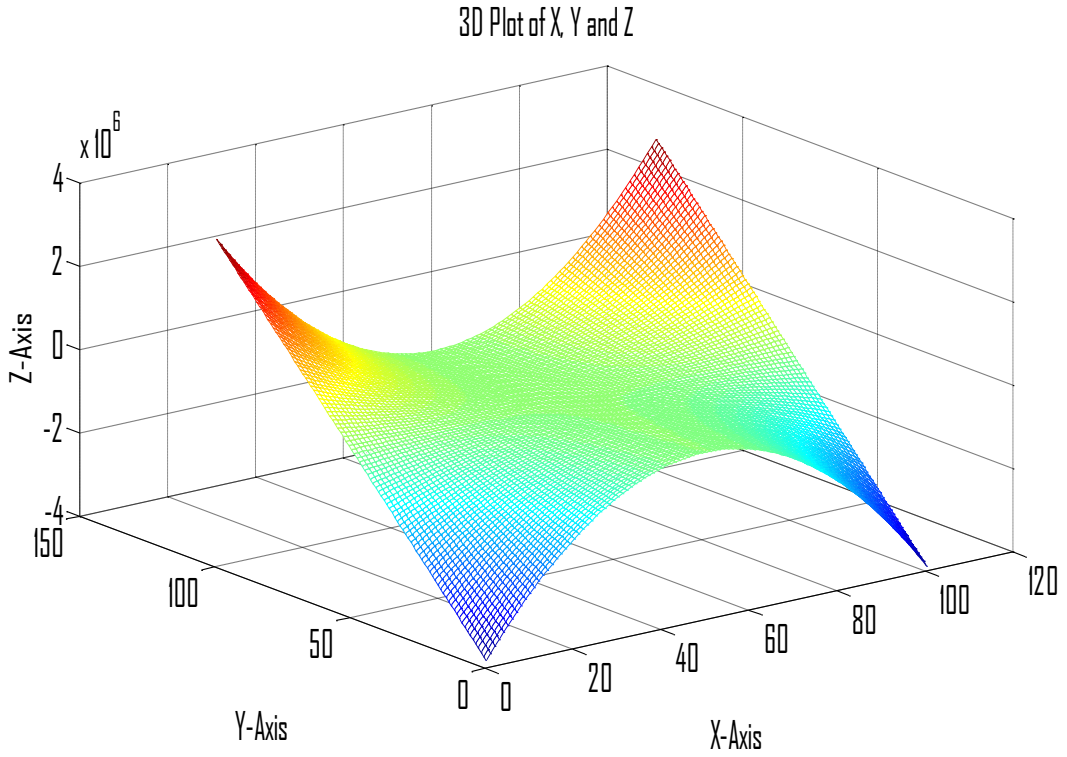

Figure 5.3: Three-dimensional plots for expression in equation 5.2 for 1st iteration with 593.28 errors.

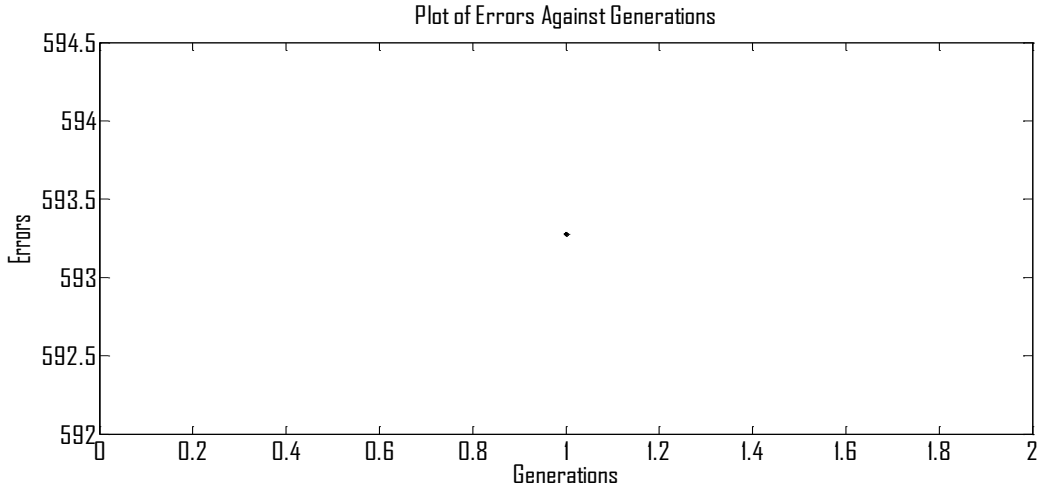

Figure 5.4: 1st iteration plot of errors against generations for expression in equation 5.2.

The GP algorithm also evolved the expression with 83.72 errors in the 20th iteration and the GP TS is in Figure 5.5, its three-dimensional plot is represented in Figure 5.6 and the plot of errors against generations is shown in Figure 5.7.

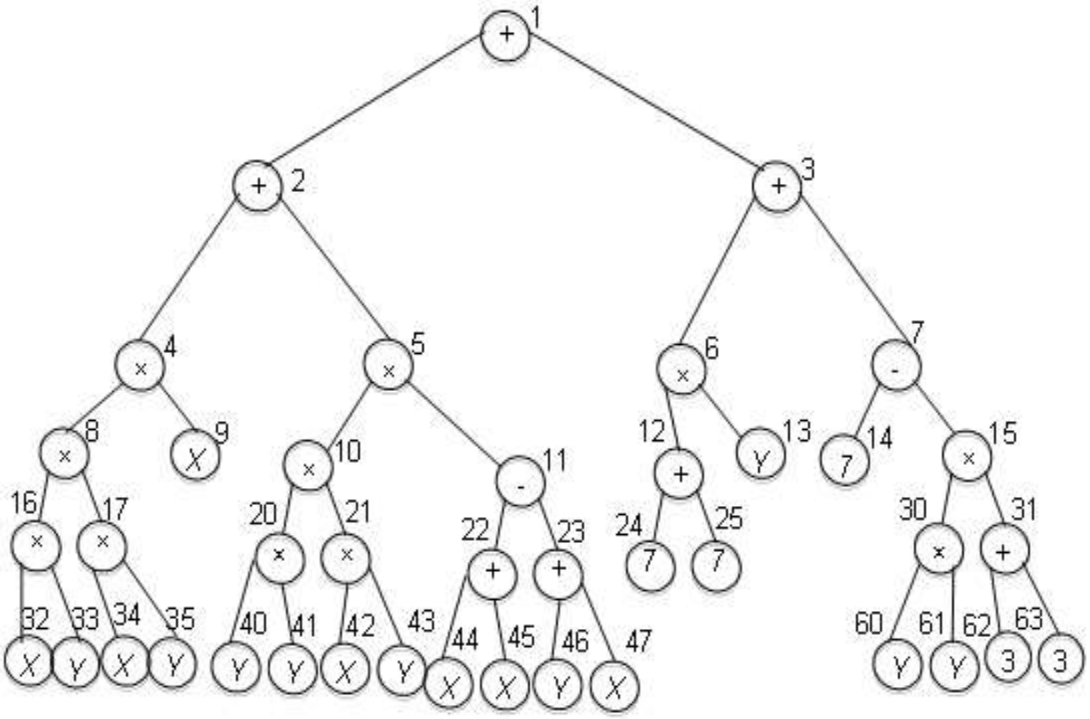

Figure 5.5: 20th iteration GP evolved TS for expression in equation 5.2 with 83.72 errors.

Evaluation of the above TS gives;

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

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

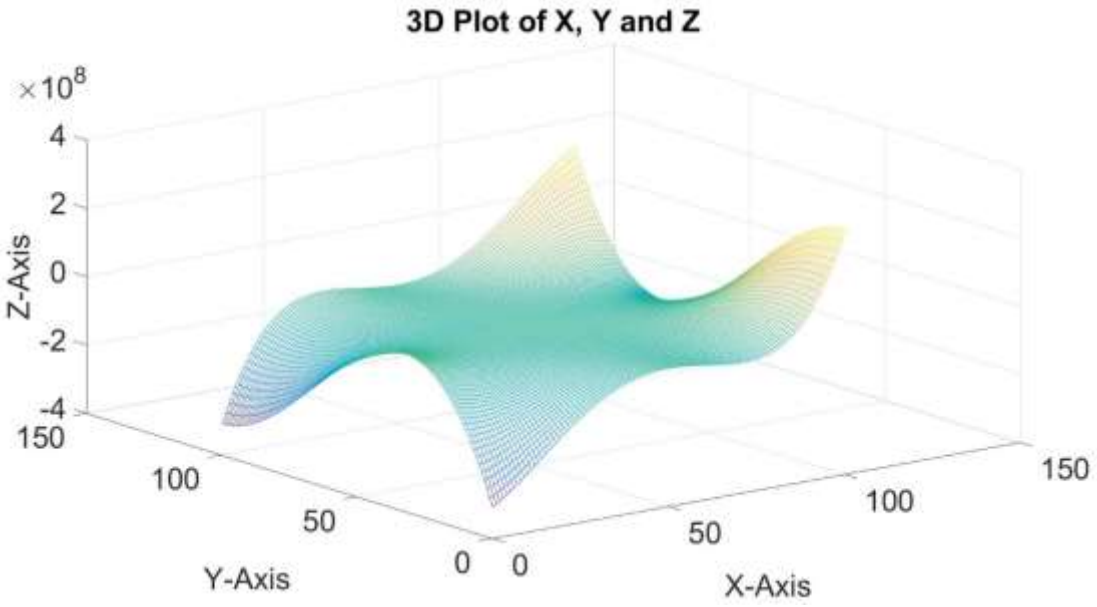

Figure 5.6: Three-dimensional plots for expression in equation 5.2 for the 20th iteration with 83.72 errors.

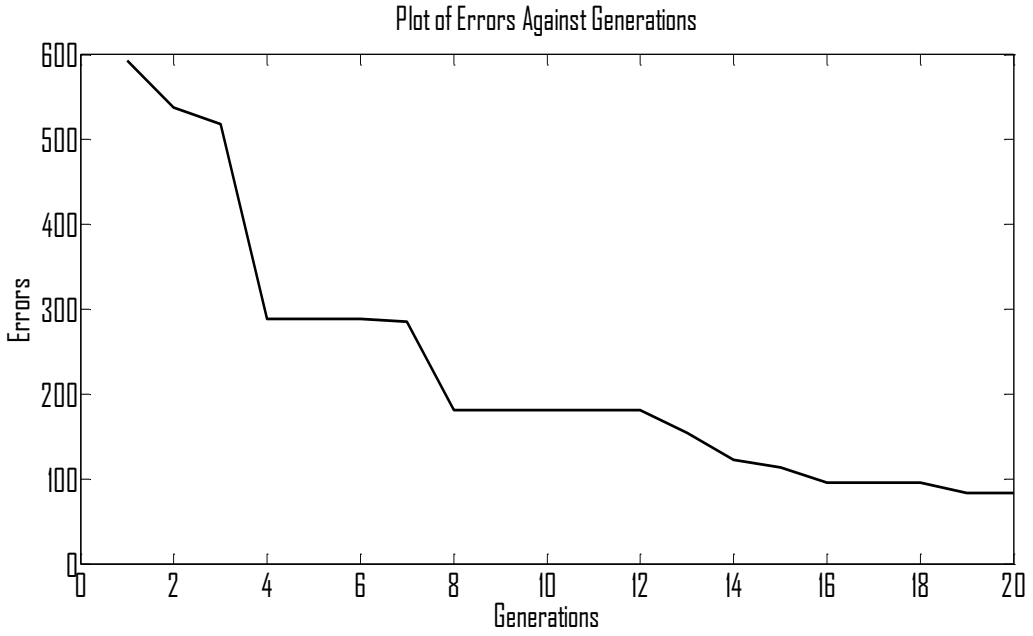

Figure 5.7: 20th iteration plot of errors against generations for expression in equation 5.2.

The GP algorithm evolved the expression in the 41st iteration with 3.63 errors and the GP TS is in Figure 5.8, its three-dimensional plot is represented in Figure 5.9 and the plot of errors against generations is shown in Figure 5.10.

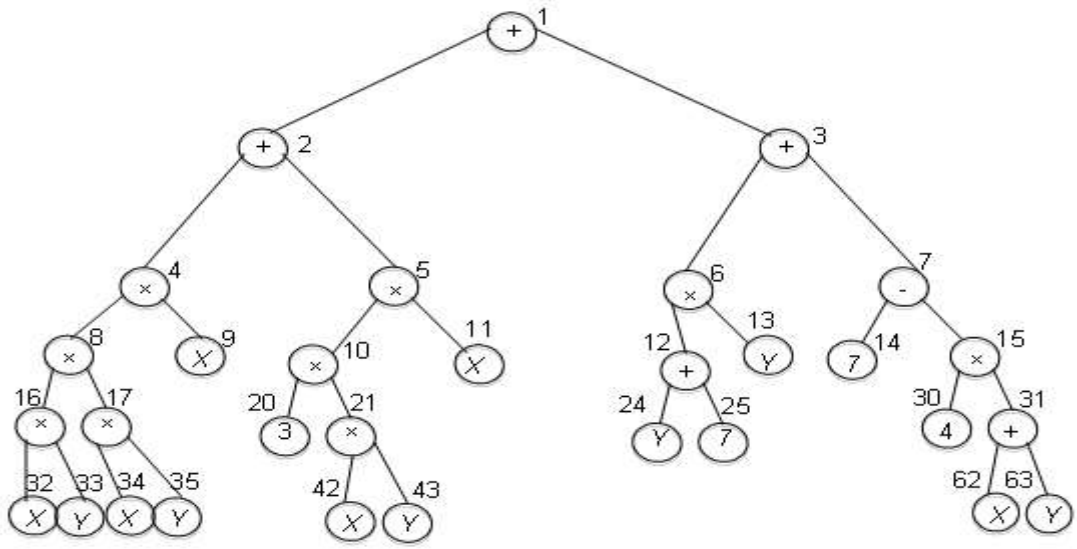

Figure 5.8: 41st iteration GP evolved TS for expression in equation 5.2 with 3.63 errors.

Its mathematical expressions are of form.

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

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

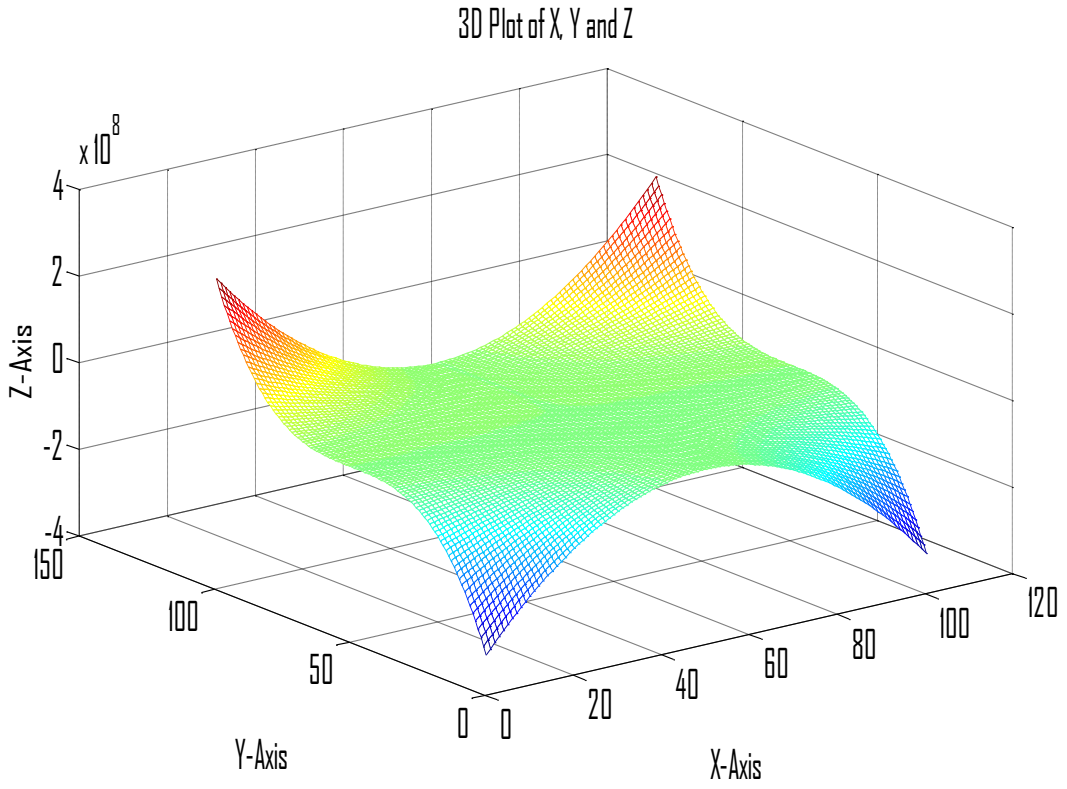

Figure 5.9: Three-dimensional plots for expression in equation 5.2 for the 41st iteration with 3.63 errors.

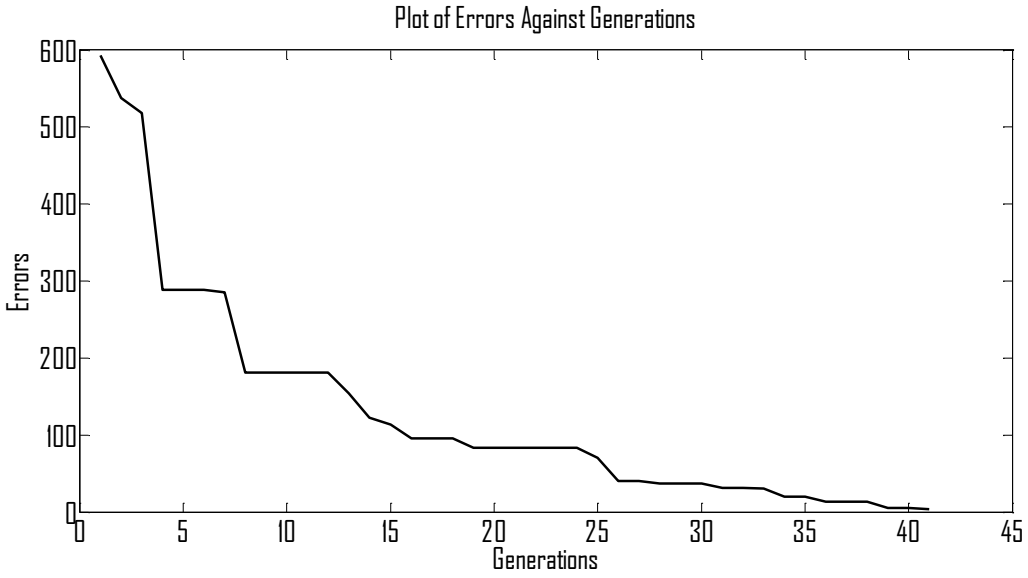

Figure 5.10: 41st iteration plot of errors against generations for expression in equation 5.2.

The GP algorithm finally evolved the expression with optimal solution in the 52nd iteration with zero errors and the GP TS is in Figure 5.11, its three-dimensional plot that is the same as that of original expression is represented in Figure 5.12 and the plot of errors against generations is shown in Figure 5.13.

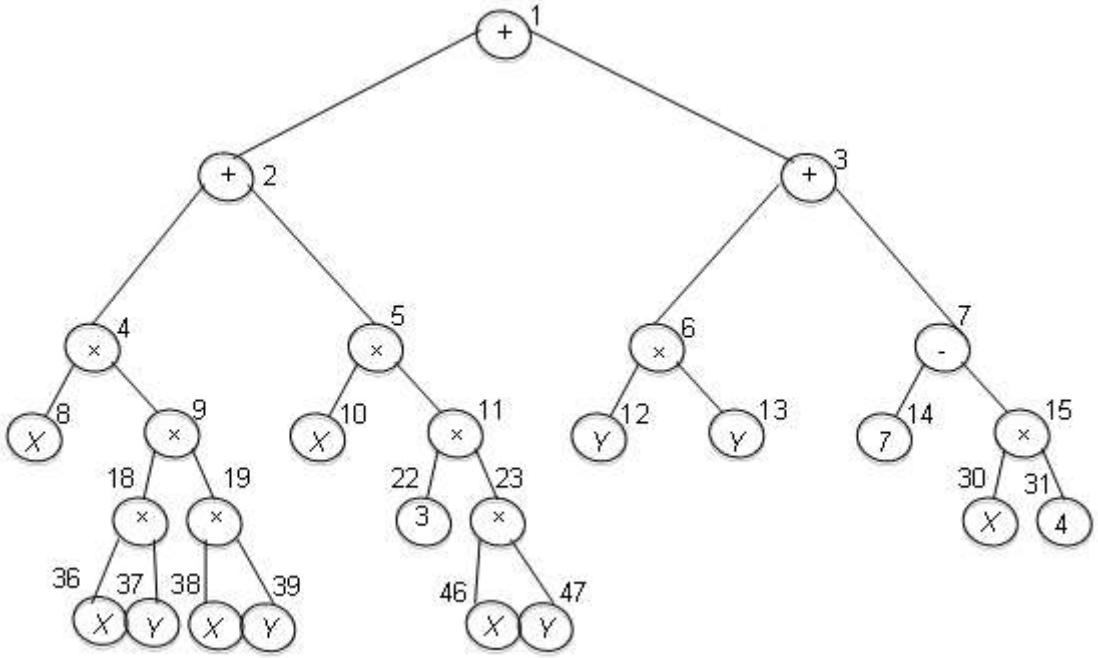

Figure 5.11: 52nd iteration GP evolved TS for expression in equation 5.2 with zero error.

Critical analysis of the evolved TS of Figure 5.11 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.

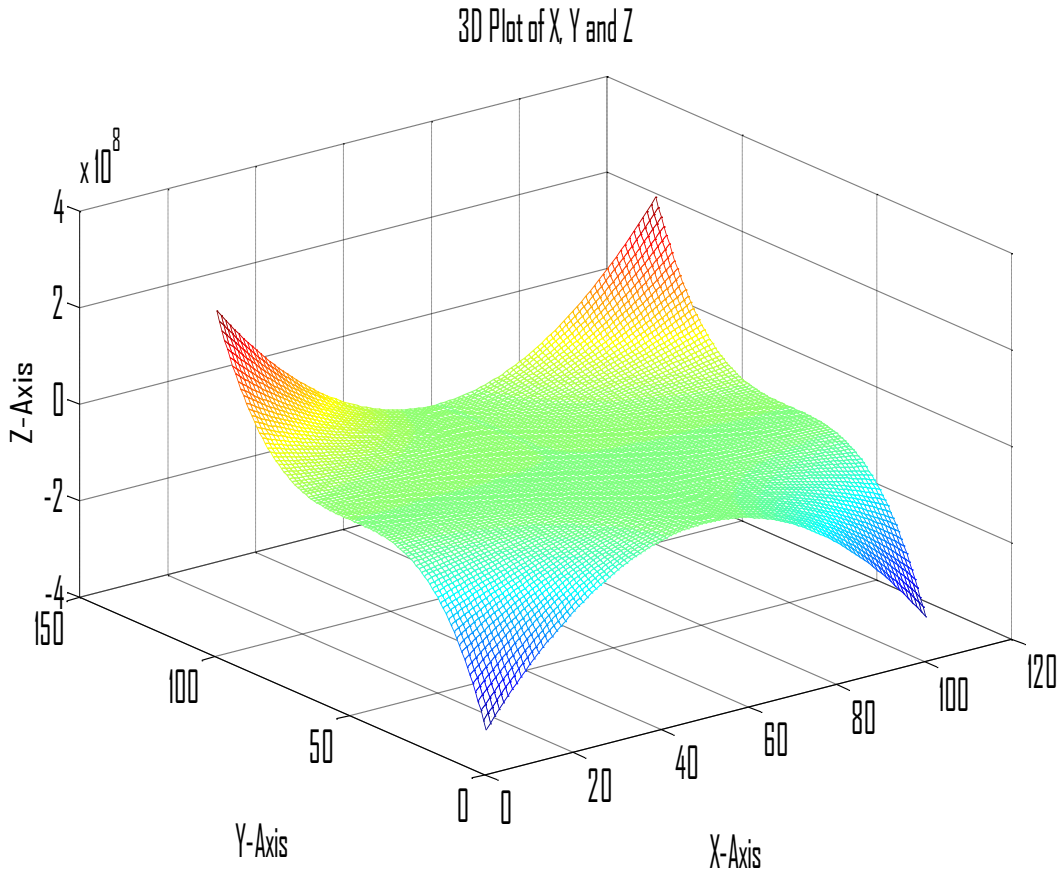

Figure 5.12: Three-dimensional plots for expression in equation 5.2 for the 52nd iteration with zero error and the same as original expression.

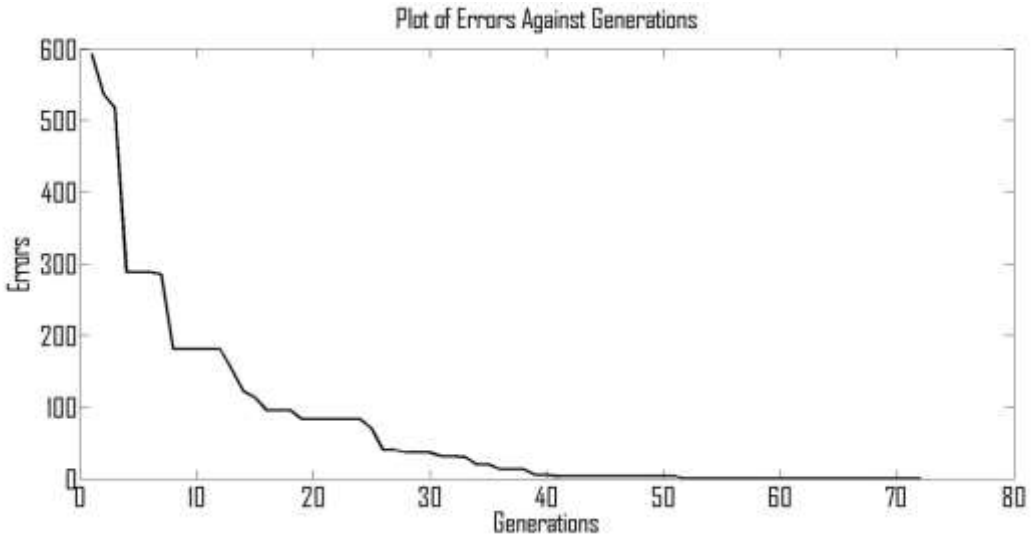

Figure 5.13: Plot of errors against generations for expression in equation 5.2.

5.5.2 Benchmark Testing Expression 2

$$Y' = X^4 + 3X^3 + 4X^2 + 5X + 6 \qquad (5.3)$$

The objective function specification is similar to that formed in Section 5.4. The GP algorithm evolved the expression with optimal solution in the 65th iteration with zero errors and the GP TS of Figure 5.14. *X* is given a range of values from -10 to 10 with interval of 1 and the plot is represented in Figure 5.15, and the plot of errors against generations is shown in Figure 5.16.

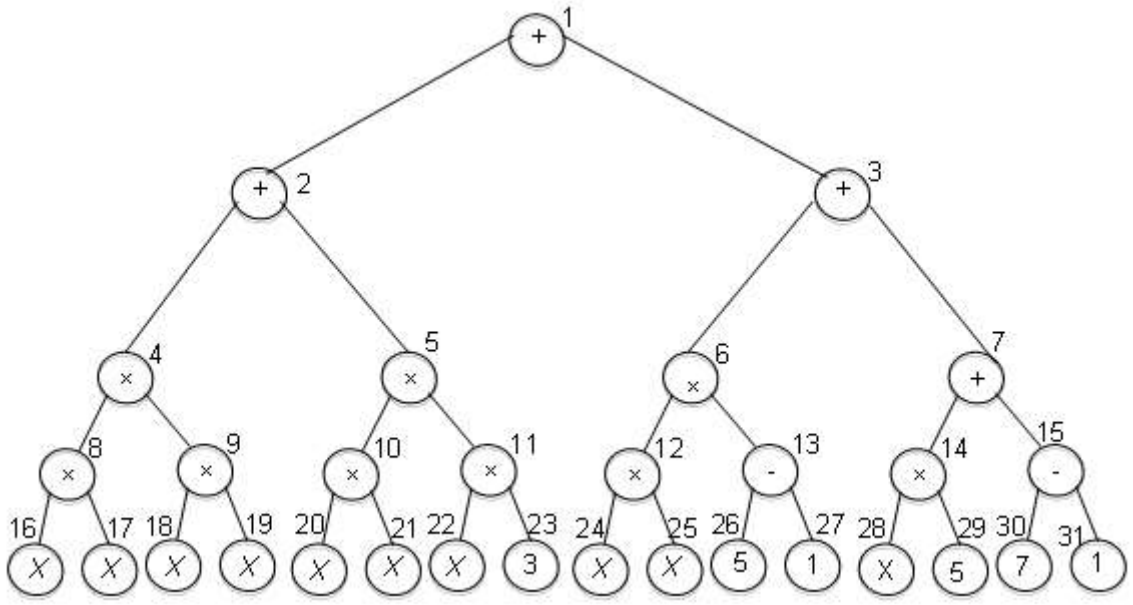

Figure 5.14: 65th iteration GP evolved TS for expression in equation 5.3 with zero error.

Careful analysis of the evolved TS of Figure 5.14 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 equation, we can conclude that the algorithm is efficient because it has successfully evolved the original equation.

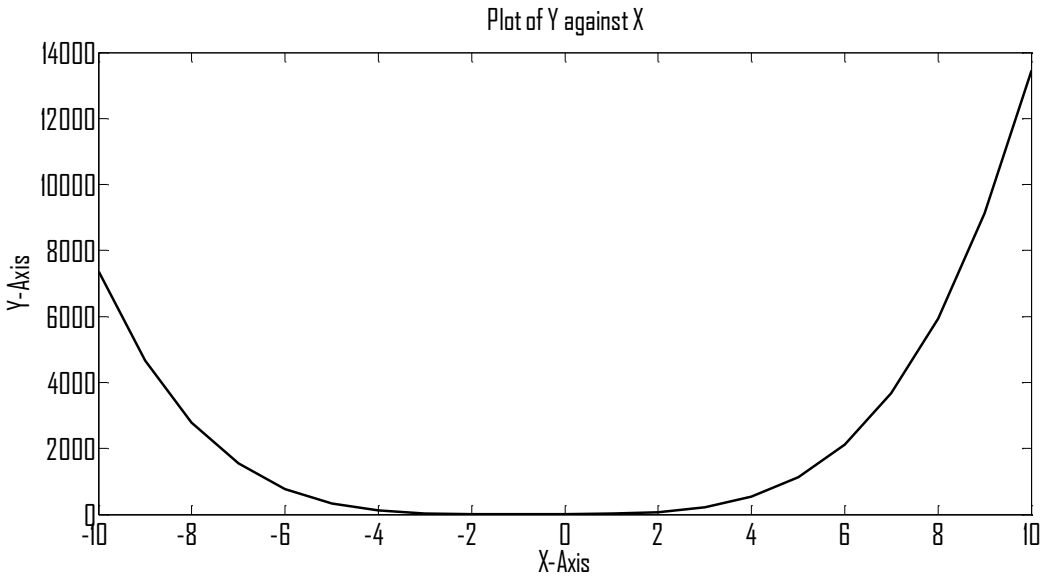

Figure 5.15: Plot of *Y* against *X* for expression in equation 5.3 with zero error.

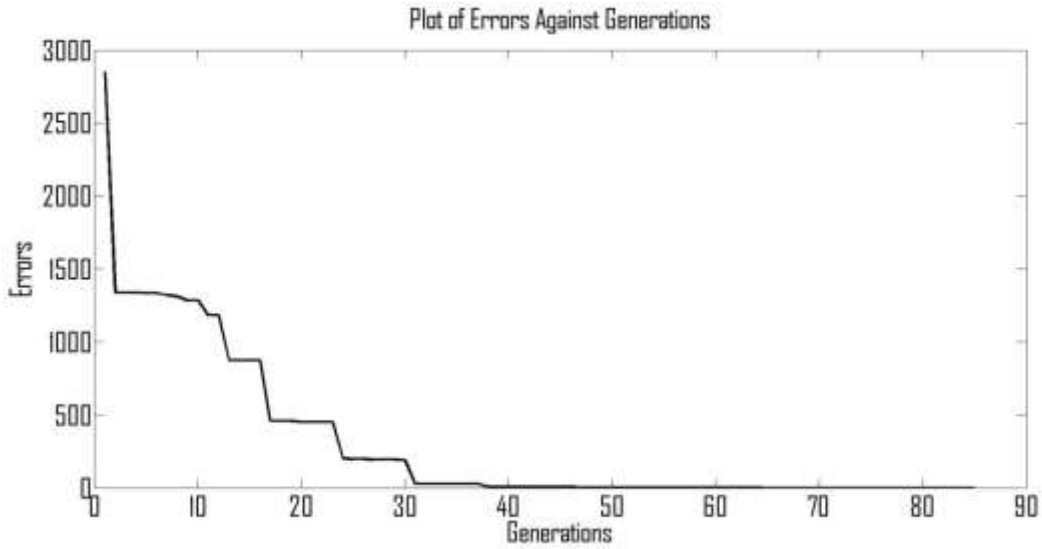

Figure 5.16: Plot of errors against generations for expression in equation 5.3.

5.5.3 Benchmark Testing Expression 3

$$Y' = X^4 - 2X^2 + 1 \qquad (5.4)$$

The objective function specification is similar to that formed in Section 5.4. The GP algorithm evolved the expression with optimal solution in the 30th iteration with zero errors and the GP TS of Figure 5.17. *X* is given a range of values from -10 to 10 with an interval of 1. The plot is represented in Figure 5.18 and the plot of errors against generations is shown in Figure 5.19.

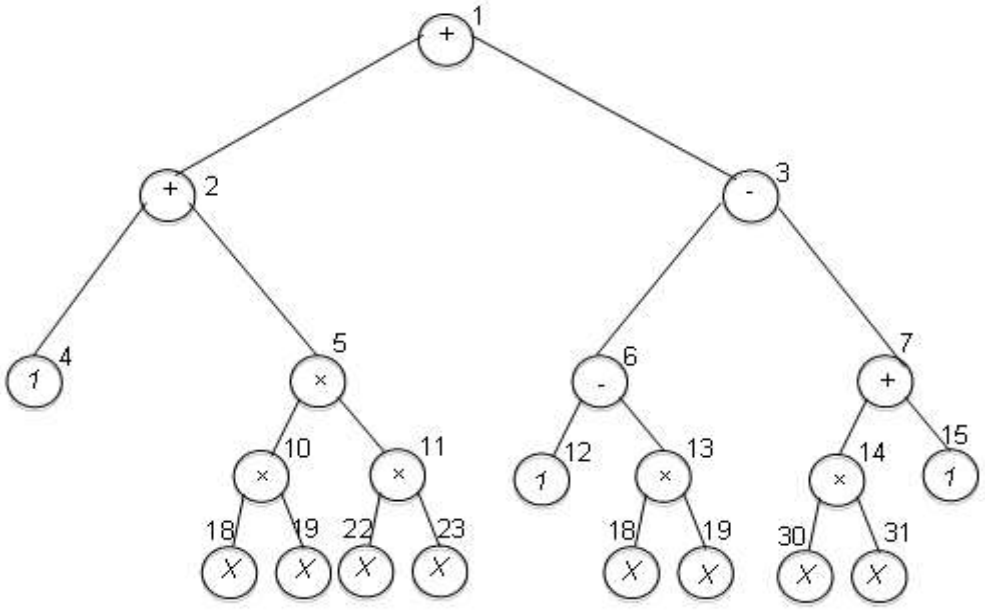

Figure 5.17: 30th iteration GP evolved TS for expression in equation 5.4 with zero error.

Critical analysis of the evolved TS of Figure 5.17 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.

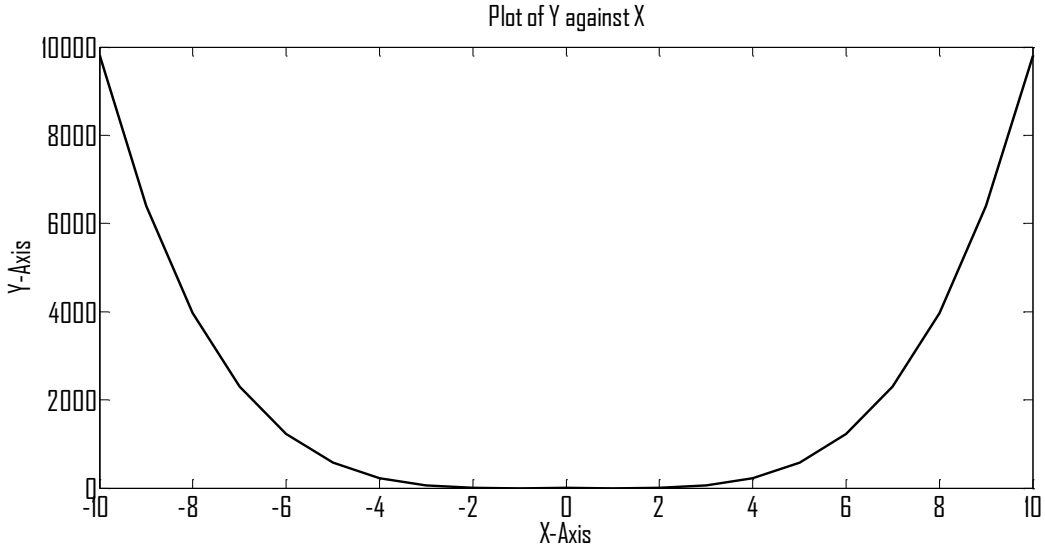

Figure 5.18: Plot of *Y* against *X* for expression in equation 5.4 with zero error.

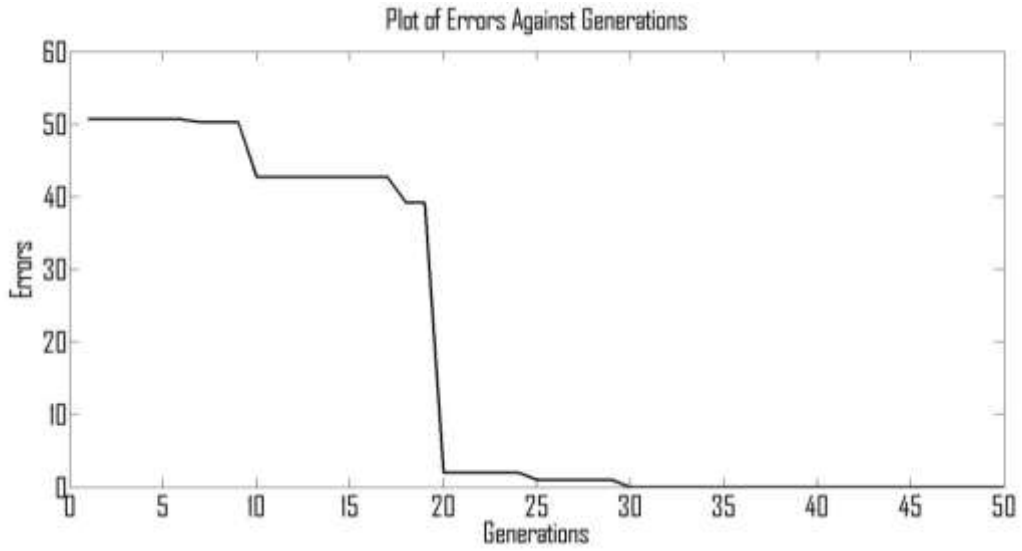

Figure 5.19: Plot of errors against generations for expression in equation 5.4.

5.5.4 Benchmark Testing Expression 4

$$Y' = X^4 - 7X^3 + 3X^2 + 29X + 42 \qquad (5.5)$$

The objective function specification is similar to that formed in Section 5.4. The GP algorithm evolved the expression with optimal solution in the 86th iteration with zero errors and the GP TS of Figure 5.20. *X* is given a range of values from -10 to 10 with an interval of 1. The plot is represented in Figure 5.21 and the plot of errors against generations is shown in Figure 5.22.

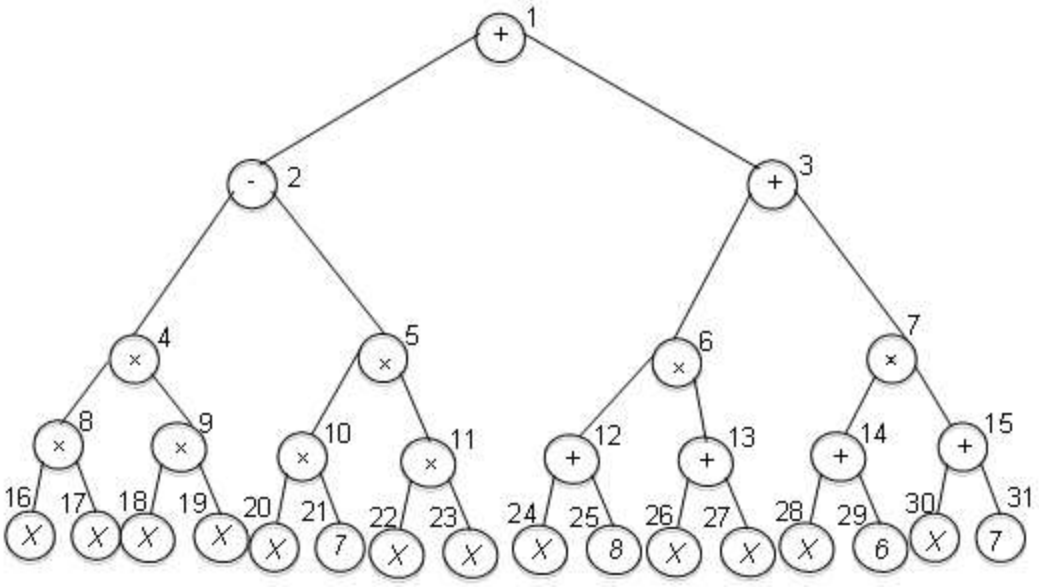

Figure 5.20: 86th iteration GP evolved TS for expression in equation 5.5 with zero error.

Careful analysis of the evolved TS of Figure 5.20 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 equation, we can deduce that the algorithm is efficient because it has successfully evolved the original equation.

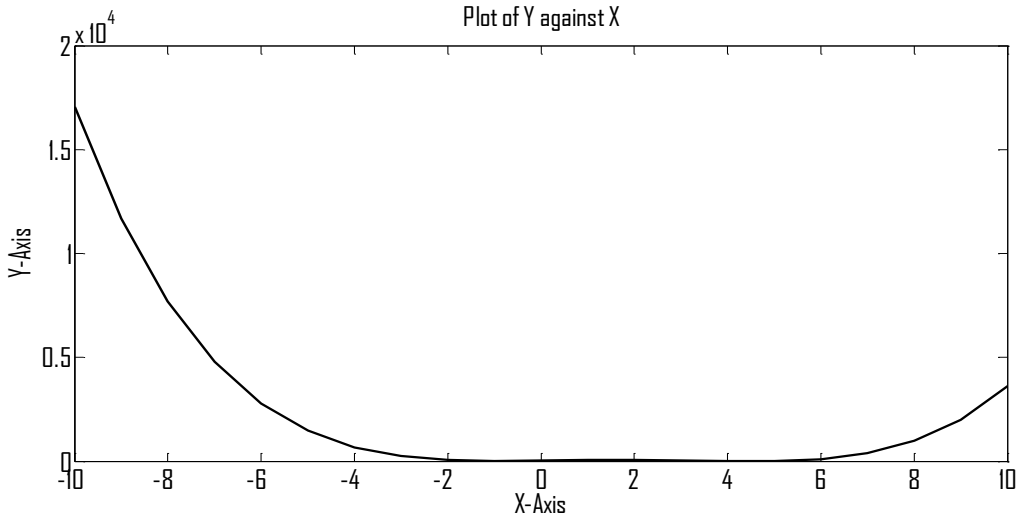

Figure 5.21: Plot of *Y* against *X* for expression in equation 5.5 with zero error.

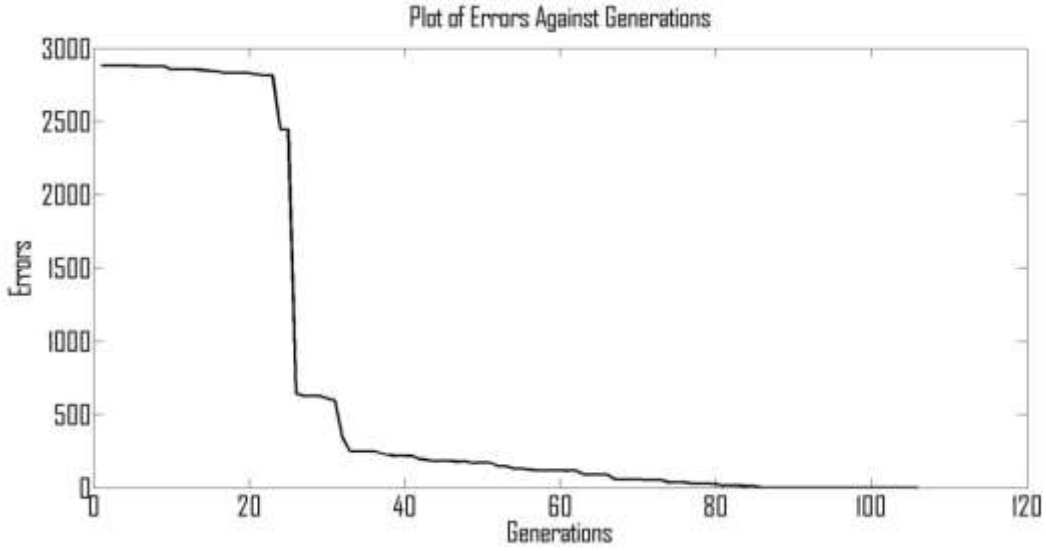

Figure 5.22: Plot of errors against generations for expression in equation 5.5.

5.6 Summary

This chapter described how the GP algorithm is formulated by explaining the different units involve: initialisation of parameters, decoding, creation, mutation, crossover and formulation of objective function. It also illustrated how GF is incorporated into the GP by showing how the TS are structurally linked from the first element to the last. The developed GP algorithm is tested with four different benchmark functions. The algorithm is efficient because it successfully evolved the original benchmark equations.

The next chapter to be discussed is Chapter 6; it describes how the developed GP algorithm is modified and applied to evolve analogue filter circuits.

Chapter 6

Application of Evolutionary Computing in Analogue Circuit Evolution (Evolvable Hardware)

6.1 Introduction

This chapter is an extension of Chapter 5. The developed and tested algorithm in Chapter 5 is modified and applied to analogue circuit evolution. The remaining part of this chapter is presented as follow: evolvable hardware is discussed in Section 6.2, methodology is illustrated in Section 6.3, results and discussions are presented in Section 6.4 and summary of this chapter is in Section 6.5.

As mention earlier, an analogue circuit design is important because of the fact that the world physical reading is analogue in nature. Although the volume of analogue circuit designs is far less compared to digital circuit designs, the majority of digital circuit designs require an analogue circuit for interfacing the outside world. Instead of using two platforms (Matlab and PSpice) for analogue circuit simulation, only Matlab is proposed in this research. This helps to reduce elapsed time needed to transfer simulation between the software packages. The traditional techniques of filter design require tedious mathematical computation.

6.2 Evolvable Hardware

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 [191-198]. Doboli et al. [199] 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. [31]. The algorithm evolves component values, circuit topology and circuit size.

Vural et al. [200] proposed three EAs: harmony search (HS), DE and ABCA to optimise CMOS amplifier area. Results are 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 [201]; 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. [202]. The authors combined GP and two-port theory for analogue circuit design. The presentation of 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. [203] 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. [204] 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 active filter.

Many studies on the use of an EA to evolve passive filters exist, but little has been done or more work is needed in the field of active filters, specifically on op-amp as a part of active filter components.

6.3 Methodology

The detailed steps involved in modification or development of GP algorithm for analogue circuit evolution are illustrated in the flowchart of Figure 6.1. Randomly generated population is computed to know individually evolved circuit's performance. If the evolving circuit satisfies the objective function with less than zero without further reduction in error or zero error, the circuit is referring to as the desired circuit otherwise generation continues. The procedures regenerate until less than zero without further reduction in error or zero error is got. Further processes involved are demonstrated in the flowchart in Figure 6.1 below.

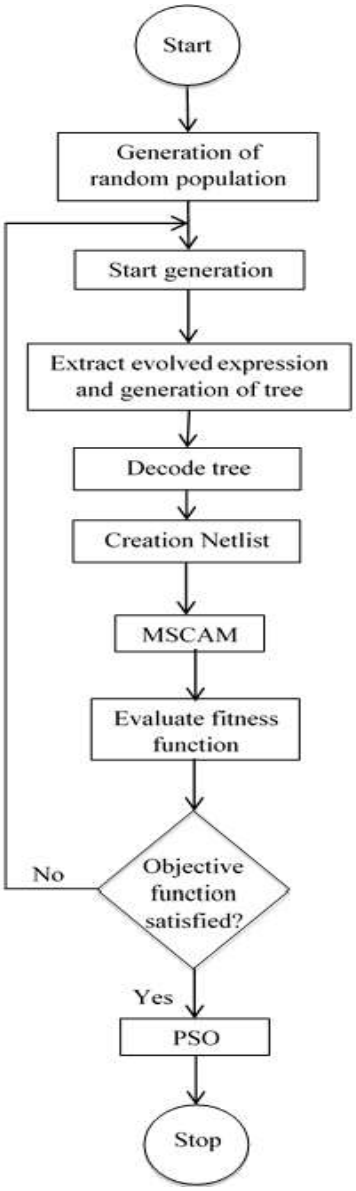

Figure 6.1: The GP algorithm.

6.3.1 Genetic Programming

The aim of GP is to code a computer programme, initialise all the parameters required, and it has the ability to solve a stated problem. GP can find a solution to problems that can be compared and measured regarding fitness. As stated earlier, GP originated from the genetic algorithm. GP differ from 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 TS is great because it can produce solutions of complexity and arbitrary size, as opposed to GA with a fixed-length. GP generates population randomly and each individual is evaluated to determine fitness. An individual with the highest fitness is chosen to perform crossover, reproduction or mutation,

with other individuals to produce other individuals for next generation.

GP algorithm in combination to GF, MSCAM and automatically generated symbolic Netlist are applied to evolve passive and active filter circuits. Also, the algorithm is tested with SSA circuit of a common-collector, FET and transistor common-emitter. The first circuit is used for illustration of detailed technique involved in the approach as show below:

6.3.1.1 Initialisation

The GP algorithm is initialised with the following parameters: crossover = 0.90, mutation = 0.10, length of parameters = 8191, length of chromosome = 24573 and population size (*PS*) = 100. The randomly created population of matrix dimension = length of chromosome by *PS*. These initial values give the best solutions.

6.3.1.2 Coding of Circuit's Components

The voltage source is fixed but added before the Netlist formation to reduce the length of the chromosome. *T*he chromosome is divided into bit group of three, and each is converted to its equivalent decimal translated as:

- '0' for series part (+)
- '1' for parallel part (|)
- '2' for capacitor (*X*)
- '3' represents inductor (*Y*)
- '>3' for operational amplifier (*Z*)

6.3.1.3 Tree Creation

Operators (| and +) and terminals (*L*, *C* and op-amp) defined in Section 6.3.1.2. are used to create tree randomly which may be grow or full technique:

6.3.1.4 Mutation

Mutation point is selected in one individual's subtree and exchanges its subtree with a randomly created subtree. In this research, the mutation rate of 0.1 is used.

6.3.1.5 Crossover

Ten Crossover locations in both parents are selected and the subtrees are interchanged. It creates offspring which are always different from their parents. The crossover rate used is 0.9.

6.3.2 Genetic Folding

In GF genes are structure in order of linear numbers separated by dots. GF is applied in this work to show how the chromosomes are structurally linked from beginning to end to enhance evolving circuit extraction to generate the Netlist. Figure 6.2 shows an active fourth order low-pass filter GP representation, whereas its GF representation is demonstrated in Table 6.1.

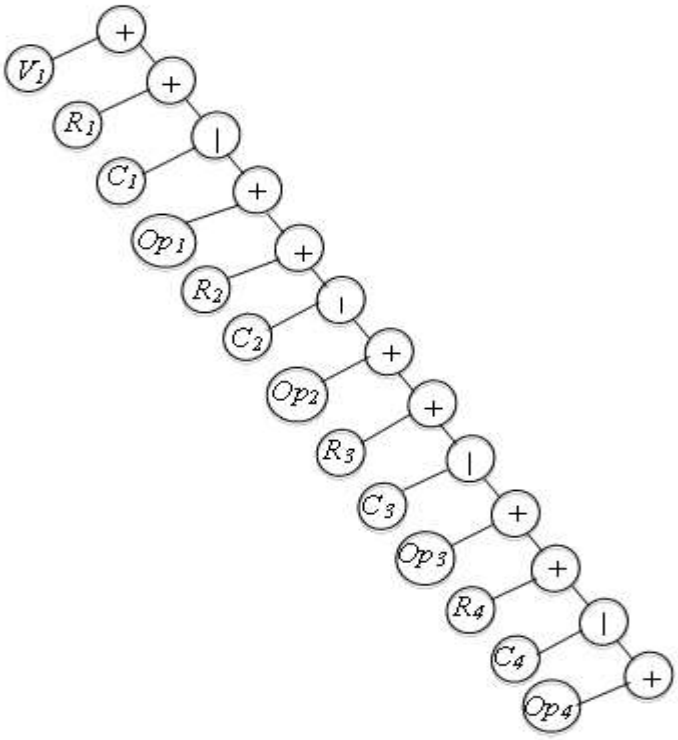

Figure 6.2: Tree representations of active fourth order low-pass filter.

The TS expression is read from top to bottom and from left to right, where + stands for series and | stands for parallel:

$$+V_1+R_1|C_1+Op_1+R_2|C_2+Op_2+R_3|C_3+Op_3+R_4|C_4+Op_4 \quad (6.1)$$

The expression in equation (6.1) is interpreted as follow: the 1st series operator is two operands operator that have values (V_1, series). The V_1 is a terminal. The 2nd series operator is

two operands operators that have values (R_1, parallel). The R_1 is a terminal. The 1st parallel operator is two operand operators that have values (C_1, series). The C_1 is a terminal. The 3rd series operator is a two operands operator that has values (OP_1, series) and so on. The expression also is represented using GF in Table 6.1. Each element is given numbers in order as in the 1st row, the circuit's elements are in the 2nd row and the elements are folded over their complementary location genes in the 3rd row.

Table 6.1: The GF representation for circuit evolution.

1	2	3	6	7	14	15	30	31	62	63	126	127	254
+	R_1	|	C_1	+	OP_1	+	R_2	|	C_2	+	OP_2	+	R_3
2.3	0.2	6.7	0.6	14.15	0.14	30.31	0.30	62.63	0.62	126.127	0.126	254.255	0.254

255	510	511	1022	1023	2046	2047	4094	4095	8190
|	C_3	+	OP_3	+	R_4	|	C_4	+	OP_4
510.511	0.510	1022.1023	0.1022	2046.2047	0.2046	4094.4095	0.4094	8190.0	0.8190

V_1 and the 1st series (+) are fixed to shorten the length of chromosome and are not included Table 6.1. The GF begins with the series operator in location 1 and ends with element *'OP_4'* in position 8190. The 1st series operator has the R_1 terminal in location 2 (and terminal always end a branch) and parallel operator in location 3. The parallel operator in location 3 has terminals C_1 in location 6 and series operator in location 7. The series operator in location 7 has terminals OP_1 in location 14 and series operator in location 15 and so on. The terminals are defined using their indices location. GF is best understood with the following points:

7. The arrangement of the chromosome comprises of float string in the gene and the location of the gene.
8. The gene structure is left child *(LC)* side separated by dot and right child (*RC)* side.
9. The dot stand for and.
10. The operator that has two operands is with *LC* and *RC*.
11. The operator that has one operand is with *LC* and 0 in the *RC*.
12. The terminal has 0 in *LC* and value in the *RC*.

6.3.3 Creation of Netlist

To evaluation how well evolving circuit has performed in the population with regard to the desired circuit, evolving circuits are extracted and converted into a symbolic Netlist. Variable are represented in Matlab programme as single variable as described in Section 6.3.1.2 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, resistor and capacitor) 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 where a component is 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 above evolving circuit is as follow:

$$+V+R|C+Z+R|C+Z+R|C+Z+R|C+Z \tag{6.2}$$

In equation (6.2), *Z* stands for op-amp and replacing the values of parallel and series part into equation (6.2) form equation (6.3) as:

$$0V1R0C2Z3R0C4Z5R0C6Z7R0C8Z9 \tag{6.3}$$

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
- $OAmp_2$ 5 4 5
- $OAmp_3$ 7 6 7
- $OAmp_4$ 9 8 9
- V 0 1 component value
- R_1 1 2 component value
- R_2 3 4 component value
- R_3 5 6 component value
- R_4 7 8 component value
- C_1 0 2 component value
- C_2 0 4 component value
- C_3 0 6 component value
- C_4 0 8 component value

6.3.4 Symbolic Circuit Analysis in Matlab

Gielen and Sansen [140] demonstrated how symbolic simulation is very useful when creating a large part of analytical prototype automatically. In this section, the MSCAM discussed in detail in Chapter 3 uses Netlist automatically generated from simulation described in Section 6.3.3 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.

6.3.5 Objective Function Specifications for the Active Fourth-Order Low-Pass Filter

Voltage gain in other words, frequency response is utilised to analyse fourth-order active filter circuit.

a. Frequency range; 1 Hz to 1 MHz is set for the circuit and a cut-off frequency of 70 kHz is specified. Impedance of each operand node is evaluated using 3 items: frequency, component value and component type.
b. Each operand node brings impedance upward. The operator node computes the corresponding parallel or series (arithmetic) to acquire its impedance after getting impedance from its branches and the process proceeds till the final circuit impedance is computed.
c. The current flowing in the tree is got by division of the source voltage over the circuit impedance. Starting from source, the current flowing in the series node is the same as current flowing in from the source. Whereas in the parallel node, the current flowing in is divided inversely proportional to the branches' impedance and proceeds until the node terminates.
d. The node voltage is the product of impedance and the current, and the voltage gain is obtained by division of the voltage across the set output node by source voltage. The procedure is demonstrated mathematically below.

The symbolic matrices *A* and *B* (equation (6.4) and equation (6.5) respectively), automatically generated from MSCAM simulation is used to formulate fitness thus:

$$A = [(V_S \div R_{111}); \quad 0; \quad 0; \quad 0; \quad 0; \quad 0; \quad 0; \quad 0] \tag{6.4}$$

$$B = \begin{bmatrix} b_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_{21} & b_{22} & b_{23} & 0 & 0 & 0 & 0 & 0 \\ 0 & b_{32} & b_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & b_{43} & b_{44} & b_{45} & 0 & 0 & 0 \\ 0 & 0 & 0 & b_{54} & b_{55} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & b_{65} & b_{66} & b_{67} & 0 \\ 0 & 0 & 0 & 0 & 0 & b_{76} & b_{77} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_{87} & b_{88} \end{bmatrix} \tag{6.5}$$

where $b_{11} = j \times \omega \times C_1 + R_{111}$

$b_{21} = -A_v$

$b_{22} = A_v + 1 + (1 \div R_2)$

$$b_{23} = -1 \div R_2$$

$$b_{32} = -1 \div R_2$$

$$b_{33} = j \times \omega \times C_2 + R_2$$

$$b_{43} = -A_v$$

$$b_{44} = A_v + 1 + (1 \div R_3)$$

$$b_{45} = -1 \div R_3$$

$$b_{54} = -1 \div R_3$$

$$b_{55} = j \times \omega \times C_3 + R_3$$

$$b_{65} = -A_v$$

$$b_{66} = A_v + 1 + (1 \div R_4)$$

$$b_{67} = -1 \div R_4$$

$$b_{76} = -1 \div R_4$$

$$b_{77} = j \times \omega \times C_4 + R_4$$

$$b_{87} = -A_v$$

$$b_{88} = A_v + 1$$

$$C = B^{-1} \times A \qquad (6.6)$$

where C holds an unknown voltage drop across all the nodes to be determined, and C_n is the voltage drops across the last node (n) being calculated to get its frequency response. The *eval* command in Matlab is used to substitute or replace the values of variables in the

automatically generated symbolic matrices (matrices *A* and *B*). The command (logspace) in Matlab is used to span the frequency range over 50 intervals for both evolving and targeted circuits' frequency response. The difference between the *RMS* of GP evolving circuit frequency response and targeted frequency response is referred to as error. The relationship is presented in equation (6.7):

$$W = rms(f_{11} - f_{22}) \tag{6.7}$$

where *W* stands for error, f_{11} is the targeted frequency response and f_{22} is the GP evolving circuit frequency response.

The use of PSO to minimise filter circuit discussed in Section 4.4 is then applied to individual circuit. Part of the circuit is critically observed and certain number of components are removed which is the mostly repeated pattern towards the end of circuit. The remaining components are given range of values so that PSO can select the value within the limit.

6.4 Results and Discussion

The results are explained under three main headings: active filters, transistor amplifier and passive filter circuits

6.4.1 Active Filters Circuits

Five different active filter circuits are used to illustrate the algorithm's efficiency. The algorithm successfully evolves all the 5 circuits with zero or less than zero error. Results presented are the same as the objective function frequency response. The component values are specified over a range of values while PSO selects value within a specified range.

6.4.1.1 Example 1: Fourth-Order Active Low-Pass Filter Circuit

In the 1st iteration, the GP evolved circuit TS is shown in Figure 6.3 whereas its equivalent circuit is shown in Figure 6.4. The MSCAM frequency response of the evolved circuit is shown in black colour; whereas the GP evolved PSpice circuit simulation is indicated in red colour as shown in Figure 6.5. The MSCAM simulation is with cut-off frequency at 160 kHz while that of the original circuit specifications cut-off frequency (PSpice simulation) is 67.9 kHz with error of 0.1736 and a gain of 1. It can be deduced from the frequency response curve that the desired circuit is not achieved because there is variation

between the two curves. The algorithm is constraint to initially with at least the first element type to enable formation of Netlist.

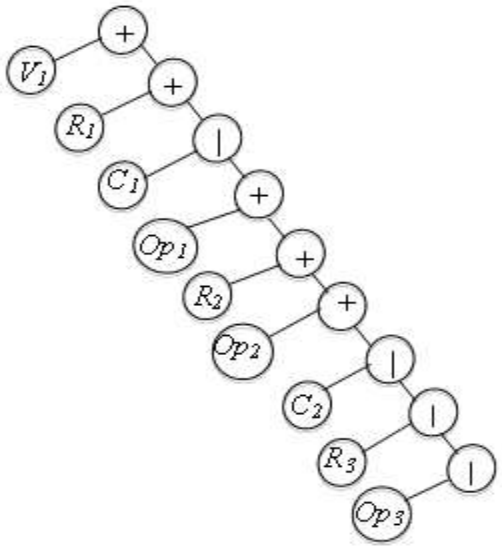

Figure 6.3: 1st iteration tree representations of the active fourth order-low pass filter.

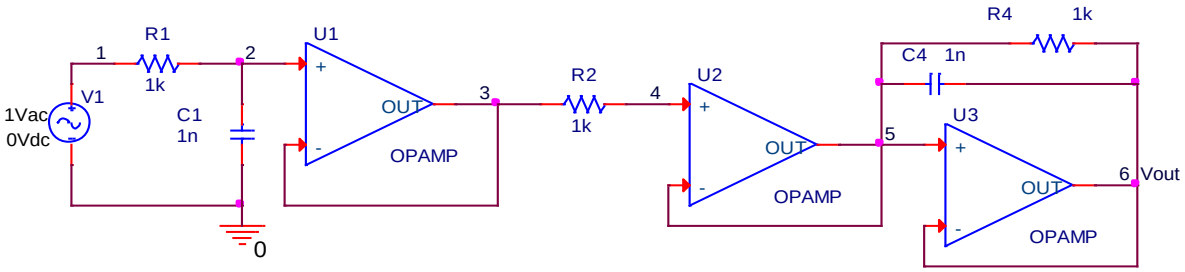

Figure 6.4: 1st iteration GP evolved circuit for the active fourth order low-pass filter.

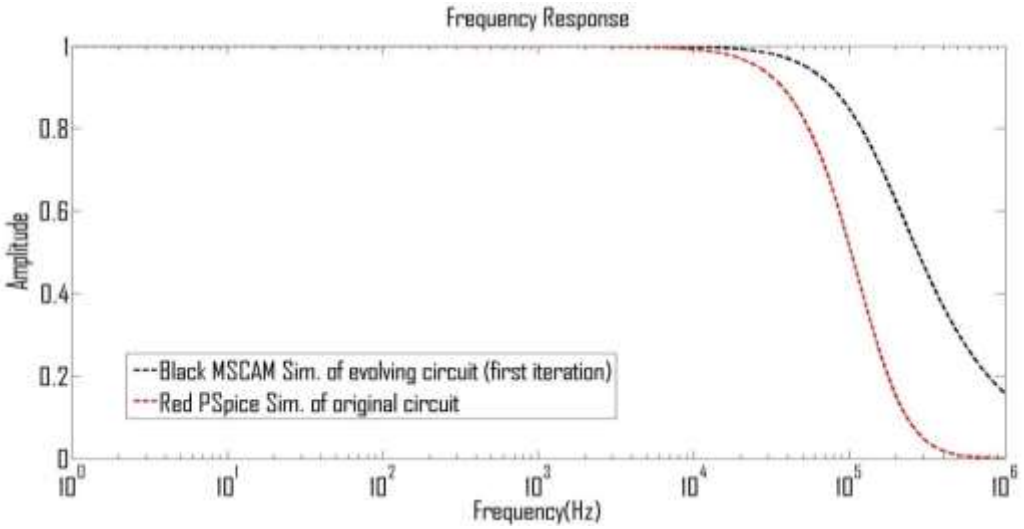

Figure 6.5: 1st iteration frequency response for GP evolved circuit (black), and the PSpice simulation of original circuit (red) for the active fourth order low-pass filter.

For the 7th iteration, the GP evolved circuit TS, which is shown in Figure 6.6 whereas its equivalent circuit is shown in Figure 6.7. The MSCAM frequency response of the evolved circuit is shown in black colour; whereas the optimised GP evolved PSpice circuit simulation is shown in red colour as indicated in Figure 6.8. The MSCAM simulation is with the cut-off frequency at 82 kHz while that of the original circuit specifications cut-off frequency (PSpice simulation) is 67.9 kHz with error of 0.0342 and a gain of 1. Also, it can be inferred from the frequency response curve that the desired circuit is not realised because there is variation between the two curves.

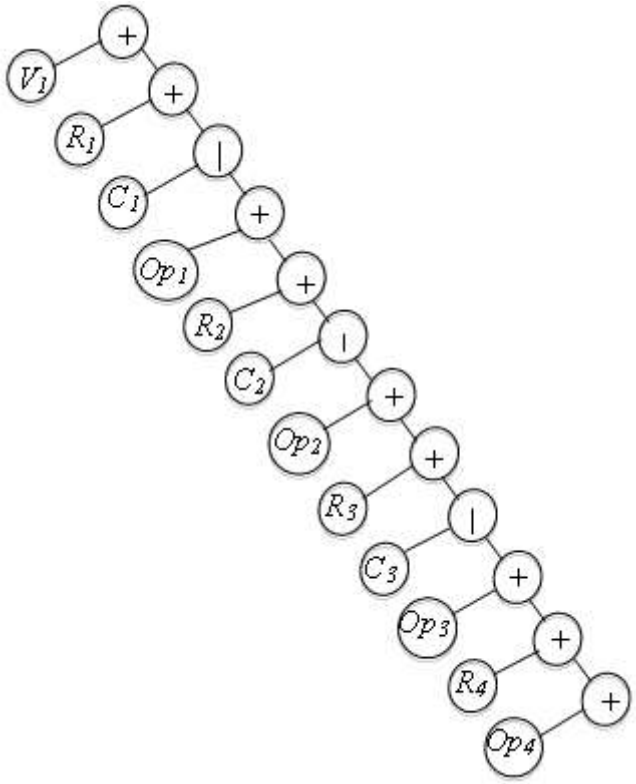

Figure 6.6: 7th iteration tree representations of the active fourth order low-pass filter.

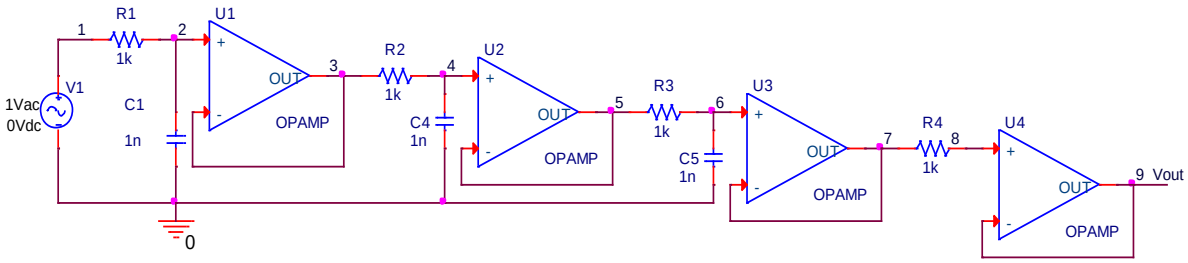

Figure 6.7: 7th iteration GP evolved circuit for the active fourth order low-pass filter.

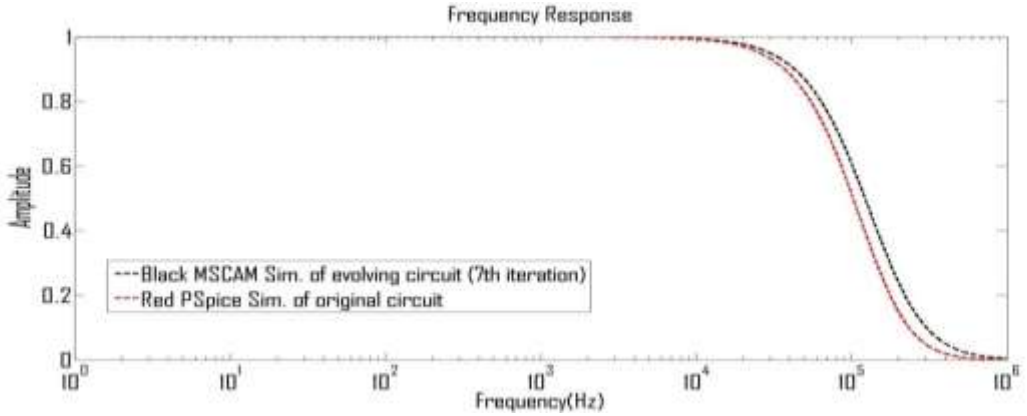

Figure 6.8: 7th iteration frequency response for GP evolved circuit (black), and the PSpice simulation of original circuit (red) for the active fourth order low-pass filter.

For the 12th iteration, the GP evolved circuit TS is shown in Figure 6.9 whereas its equivalent circuit is shown in Figure 6.10. The MSCAM frequency response of the evolved circuit is shown in black colour whereas the optimised GP evolved PSpice circuit simulation is shown in red colour indicated in Figure 6.11. The MSCAM simulation is with the cut-off frequency at 81.13 kHz while that of the original circuit specifications cut-off frequency (PSpice simulation) is 67.9 kHz with error of 9.7298E-7 and a gain of 1. Likewise, it can be concluded from the frequency response curve that the desired circuit is not yet realised even though the response is getting closer but there is variation between the two curves.

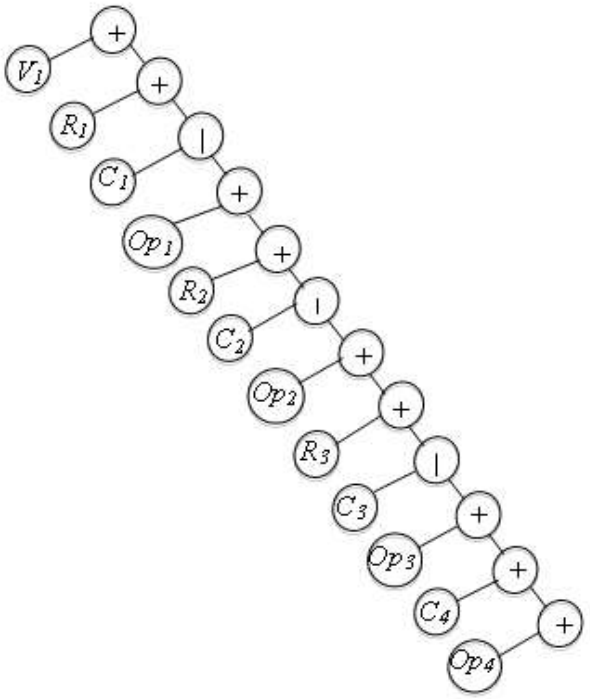

Figure 6.9: 12th iteration tree representations of the active fourth order-low pass filter.

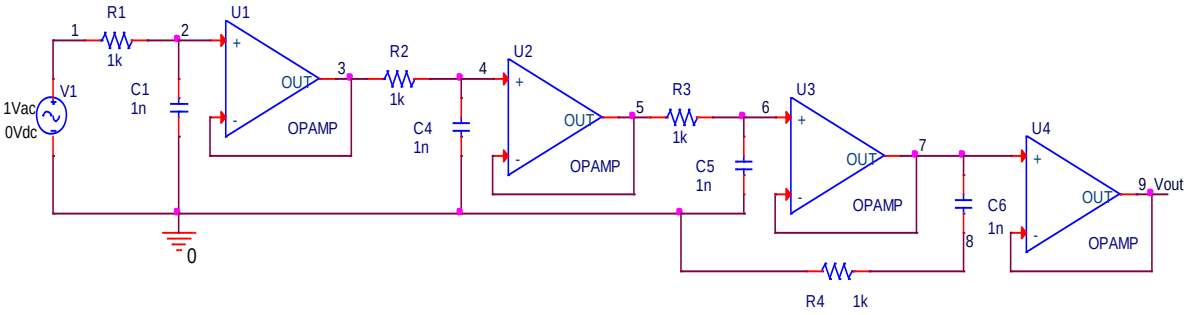

Figure 6.10: 12th iteration GP evolved circuit for the active fourth order low-pass filter.

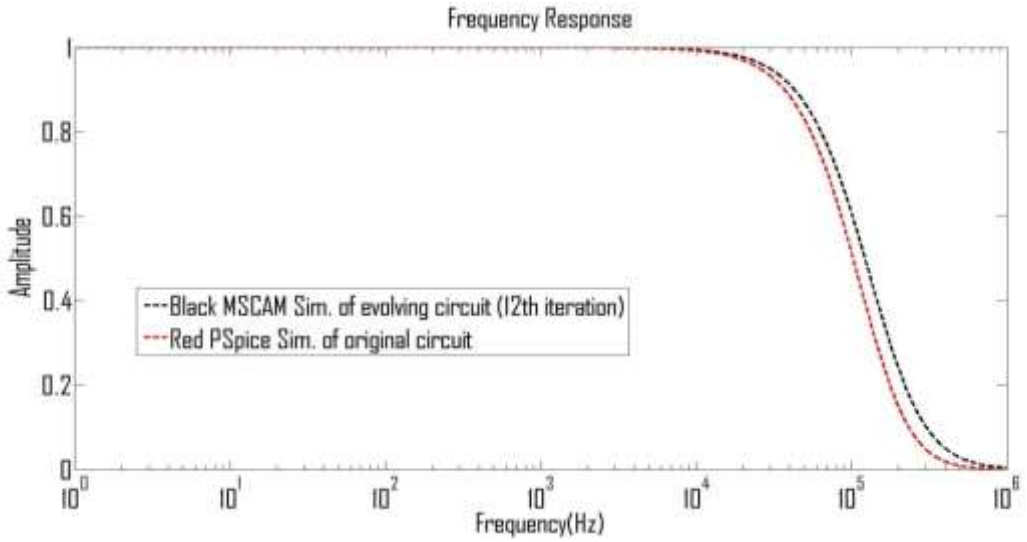

Figure 6.11: 12th iteration frequency response for GP evolved circuit (black), and the PSpice simulation of original circuit (red) for the active fourth order low pass filter.

The GP evolved the desired circuit which is the same as that of original circuit in terms of frequency response curve and components arrangements. Its TS is shown in Figure 6.2 that is used for the illustration of the method, whereas its equivalent circuit is shown in Figure 6.12. The minimised circuit is shown in Figure 6.13. It takes thirty-six minutes to evolve the circuit after eighteen iterations. The MSCAM frequency response of the evolved circuit is shown in black colour whereas the optimised GP evolved PSpice circuit simulation is shown in red colour as indicated in Figure 6.14. The MSCAM, original circuit specifications and optimised circuit simulation with PSO application are with cut-off frequencies at 67.9 kHz while that of optimised circuit simulation without PSO application has cut-off frequency at 81.15 kHz with error of 8.6921E-10 and a gain of 1. The total number of component count reduction is 3. Here it can be inferred from the frequency response curve that the desired circuit is realised because there is no variation between the two curves.

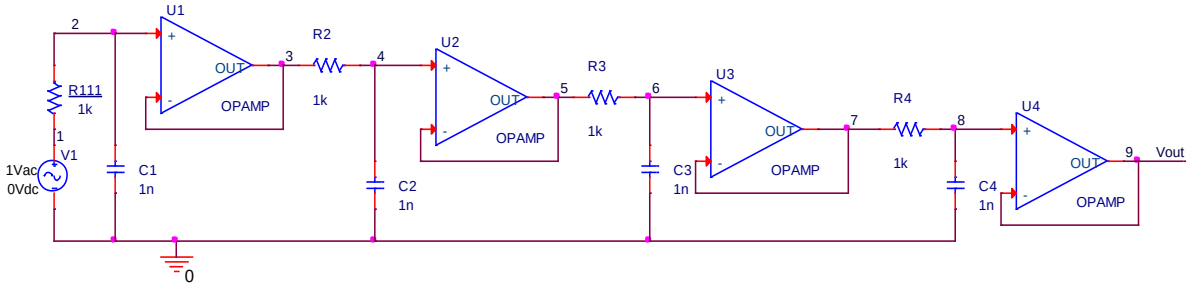

Figure 6.12: 18th iteration GP evolved circuit for the active fourth order low-pass filter [187].

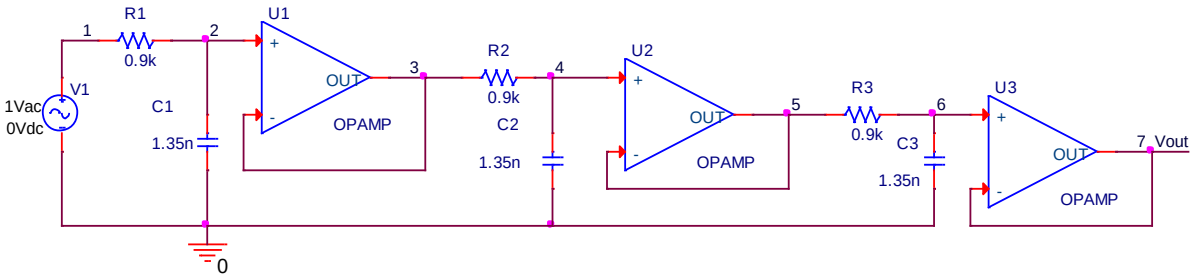

Figure 6.13: GP evolved/reduced/PSO component adjusted circuit for the active 4th-order low-pass filter.

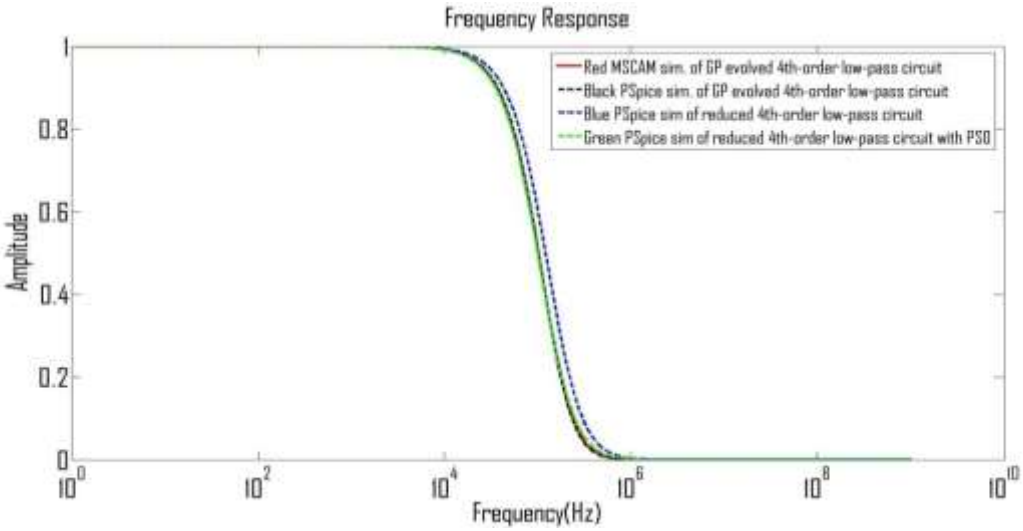

Figure 6.14: 18th iteration Frequency response for GP evolved circuit (Red), PSpice simulation of GP evolved circuit (black), PSpice simulation of reduced GP evolved circuit without PSO (blue) and with PSO (Green) for the active 4th-order filter circuit.

6.4.1.2 Example 2: Fifth-Order Active Low-Pass Filter Circuit with Feedback

The GP evolved desired circuit TS is shown in Figure 6.15, whereas its equivalent circuit is shown in Figure 6.16. The optimised GP evolved circuit is shown Figure 6.17. It takes twenty minutes to evolve the circuit and ten iterations. The MSCAM frequency response of the evolved circuit is shown in blue colour, the optimised GP evolved PSpice circuit simulation is indicated in black colour and the optimised circuit simulation with PSO

is shown in red colour as indicated in Figure 6.18. 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 and optimised circuit simulation with PSO application are with cut-off frequencies at 47.1 kHz while that of optimised circuit simulation without PSO application has cut-off frequency at 32.04 kHz with error of 5.1093E-7 and a gain of 1. The total number of component count reduction is 5.

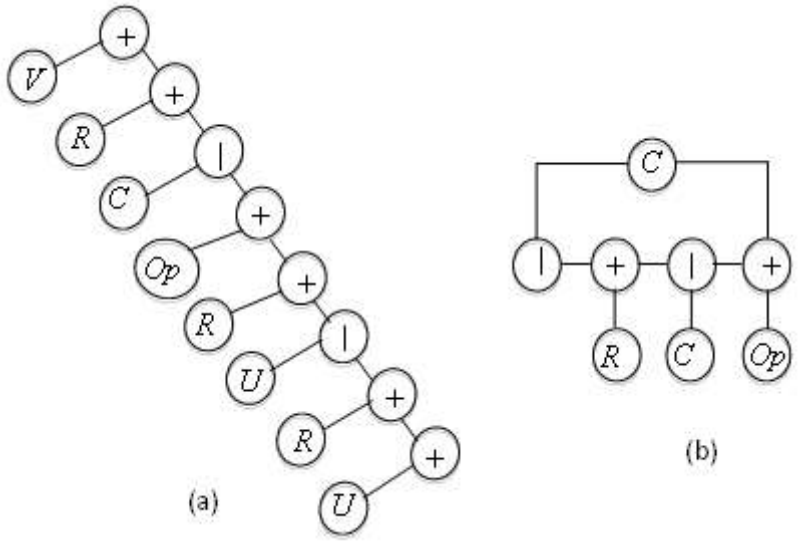

Figure 6.15: (a) GP evolved TS for the active low-pass filter with feedback and (b) *U* representation.

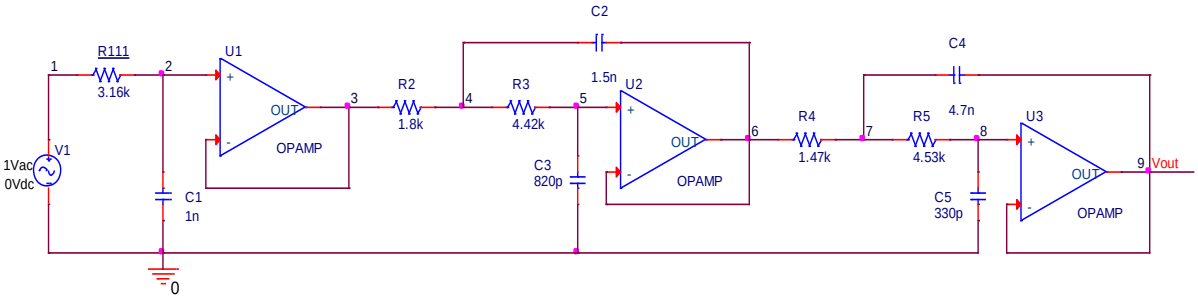

Figure 6.16: GP evolved circuit for the active low-pass filter with feedback [187].

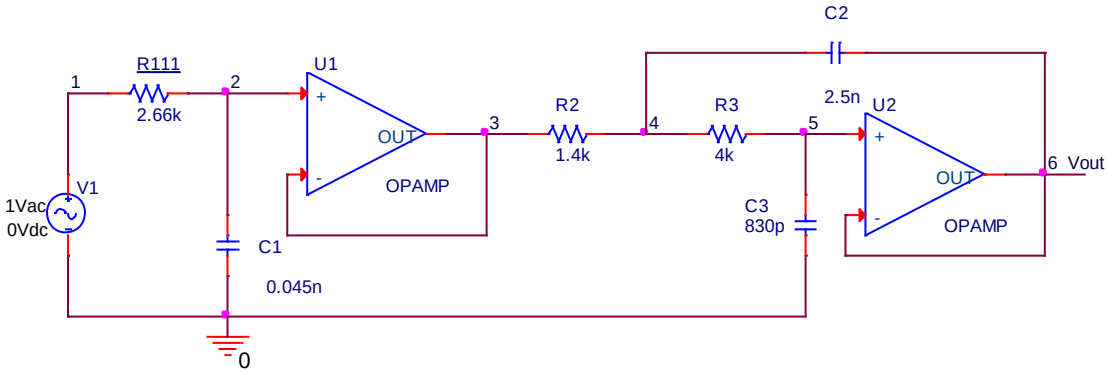

Figure 6.17: GP evolved/reduced/PSO component adjusted circuit for the active low-pass filter with feedback.

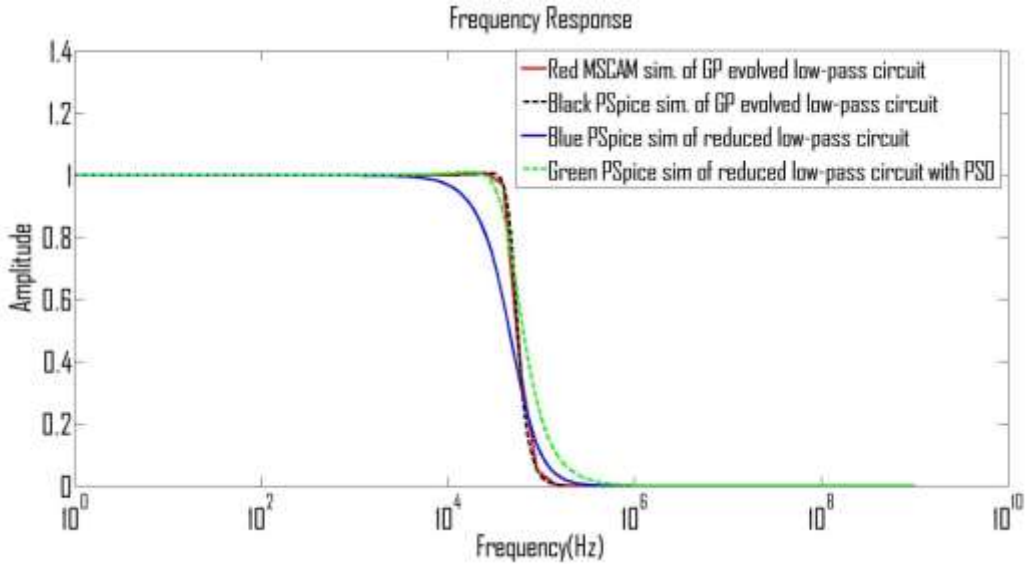

Figure 6.18: Frequency response for GP evolved circuit (Red), PSpice simulation of GP evolved circuit (black), PSpice simulation of reduced GP evolved circuit (blue) and reduced GP evolved circuit with PSO (Green) for the active low-pass filter with feedback.

6.4.1.3 Example 3: Fifth-Order Active High-Pass Filter Circuit with Feedback

The GP evolved desired circuit is shown in Figure 6.19, whereas its equivalent circuit is shown in Figure 6.20. The optimised GP evolved circuit is shown in Figure 6.21. It takes fourteen minutes to evolve the circuit and seven iterations. The MSCAM frequency response of the evolved circuit is shown in blue colour, the optimised GP evolved PSpice circuit simulation is indicated in black colour and the optimised circuit simulation with PSO is shown in red colour as indicated in Figure 6.22. The MSCAM, original circuit specifications and optimised circuit simulation with PSO application are with cut-off frequencies at 1134 Hz while that of optimised circuit simulation without PSO application has cut-off frequency at 1004 Hz with error of 9.6085E-8 and a gain of 1. The total number of component count reduction is 5.

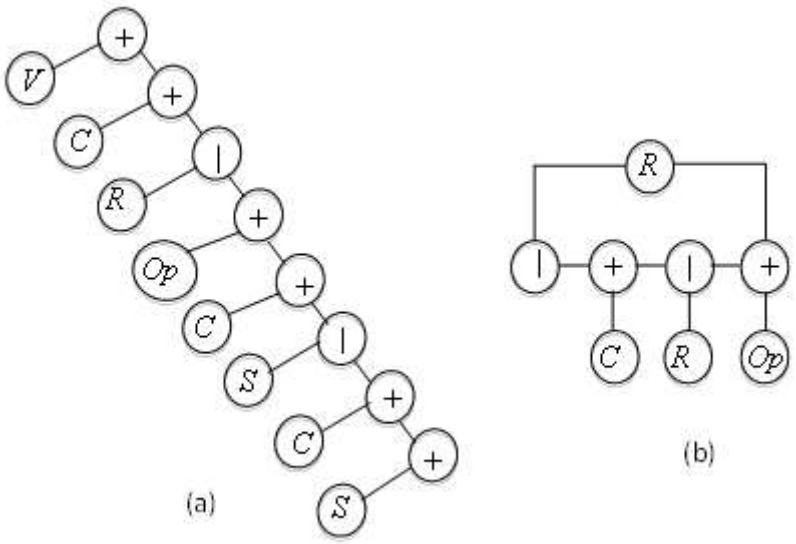

Figure 6.19: (a) GP evolved TS for the active high-pass filter with feedback and (b) *S* representation.

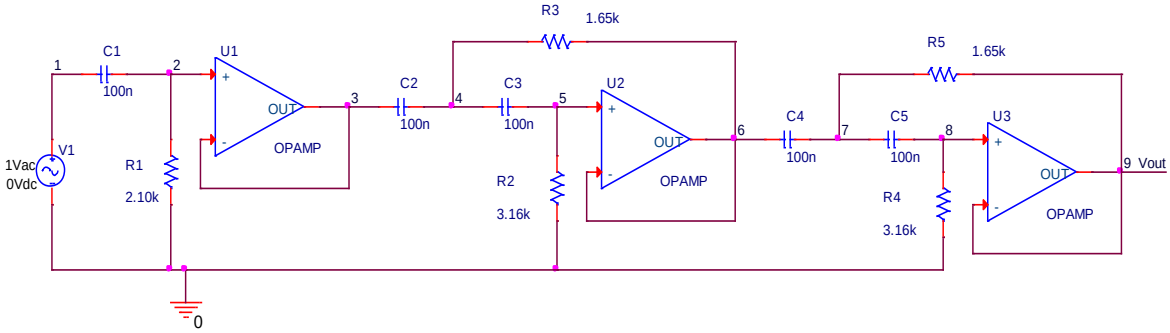

Figure 6.20: GP evolved circuit for the active high-pass filter with feedback.

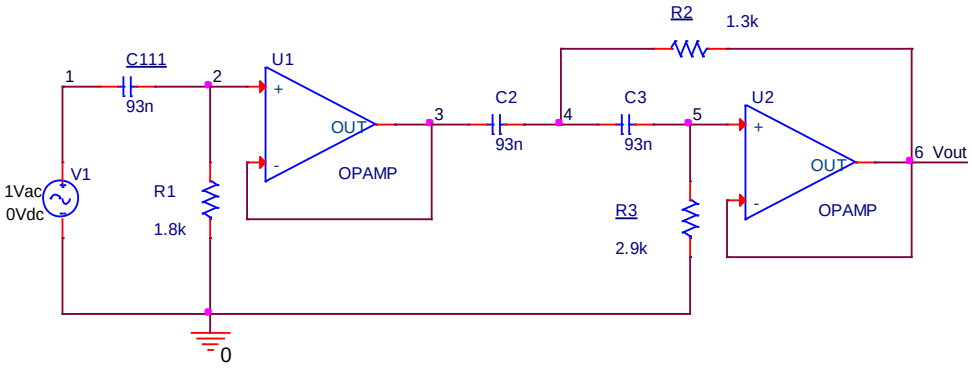

Figure 6.21: GP evolved/reduced/PSO component adjusted circuit for the active high-pass filter with feedback.

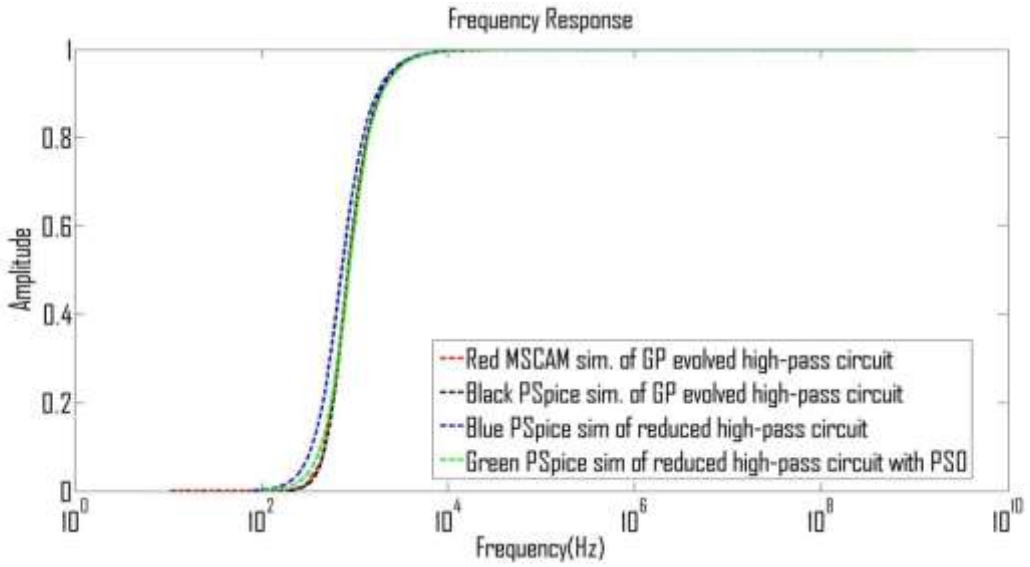

Figure 6.22: Frequency response for GP evolved circuit (Red), PSpice simulation of GP evolved circuit (black), PSpice simulation of reduced GP evolved circuit (blue) and reduced GP evolved circuit with PSO (Green) for the active high-pass filter with feedback.

6.4.1.4 Example 4: Active Band-Pass Filter Circuit

The GP evolved desired circuit TS is shown in Figure 6.23 whereas its equivalent circuit is shown in Figure 6.24. The optimised GP evolved circuit is shown in Figure 6.25. It takes twenty-six minutes to evolve the circuit after thirteen iterations. The MSCAM frequency response of the evolved circuit is shown in blue colour, the optimised GP evolved PSpice circuit simulation is indicated in black colour and the optimised circuit simulation with PSO is shown in red colour as indicated in Figure 6.26. The MSCAM, original circuit specifications and optimised circuit simulation with PSO application are with the lower and upper cut-off frequencies at 31.42 Hz and 47.86 kHz respectively while that of optimised circuit simulation without PSO application has lower and upper cut-off frequencies at 24.73 Hz and 64.57 kHz respectively with error of 3.5614E-7 and a gain of 1. The total number of component count reduction is 6.

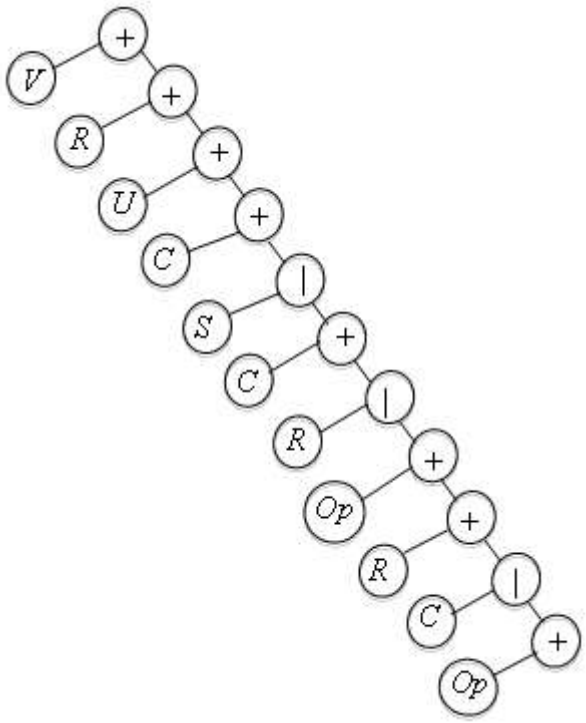

Figure 6.23: GP evolved TS for the active band pass filter (U and S as Figure 6.15 and Figure 6.19).

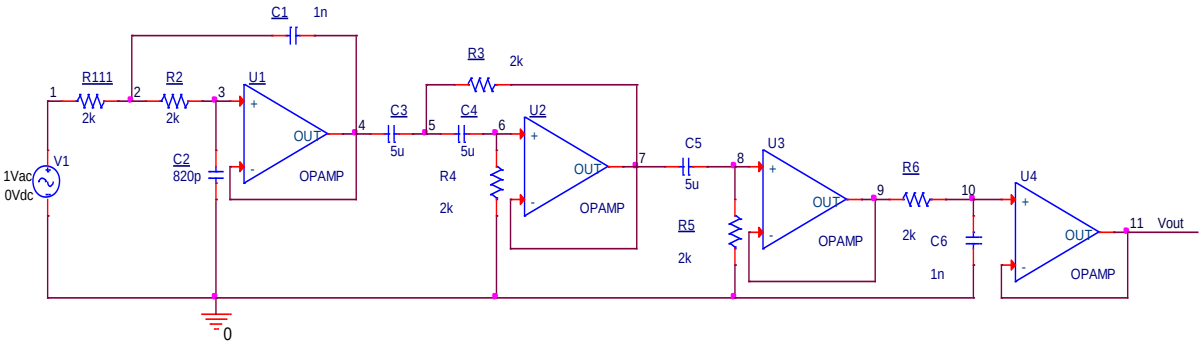

Figure 6.24: GP evolved circuit for the active band-pass filter.

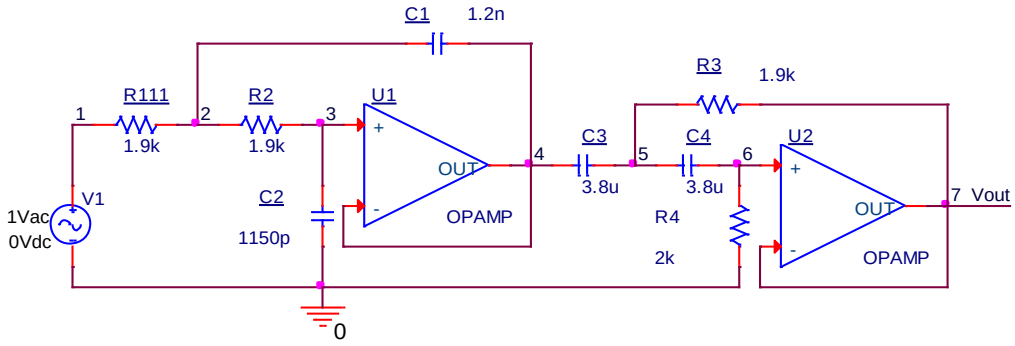

Figure 6.25: GP evolved/reduced/PSO component adjusted circuit for the active band-pass filter.

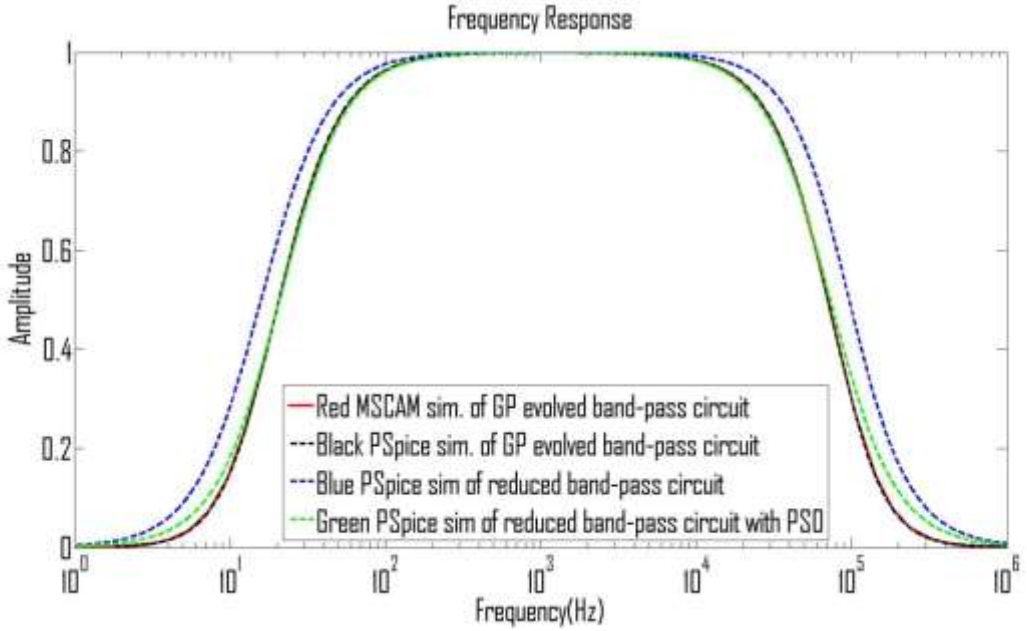

Figure 6.26: Frequency response for GP evolved circuit (Red), PSpice simulation of GP evolved circuit (black), PSpice simulation of reduced GP evolved circuit (blue) and reduced GP evolved circuit with PSO (Green) for the active band-pass filter.

6.4.1.5 Example 5: Active Band-Stop Filter Circuit

The GP evolved desired circuit TS is shown in Figure 6.27, whereas its equivalent circuit is shown in Figure 6.28. The optimised GP evolved circuit is shown in Figure 6.29. It takes twenty-two minutes to evolve the circuit after eleven iterations. The MSCAM frequency response of the evolved circuit is shown in blue colour, the optimised GP evolved PSpice circuit simulation is indicated in black colour and the optimised circuit simulation with PSO is shown in red colour as indicated in Figure 6.30. This complex circuit is evolved within short time because the algorithm did not see it to be complex, what is needed is the right specifications regarding objective function and parameters settings. The MSCAM, original circuit specifications and optimised circuit simulation with PSO application are with lower and upper cut-off frequencies at 45.7 kHz and 453.4 kHz respectively while that of optimised circuit simulation without PSO application has lower and upper cut-off frequencies at 55 kHz and 562 kHz respectively with error of 8.1899E-8 and a gain of 1. The total component count reduction of 9.

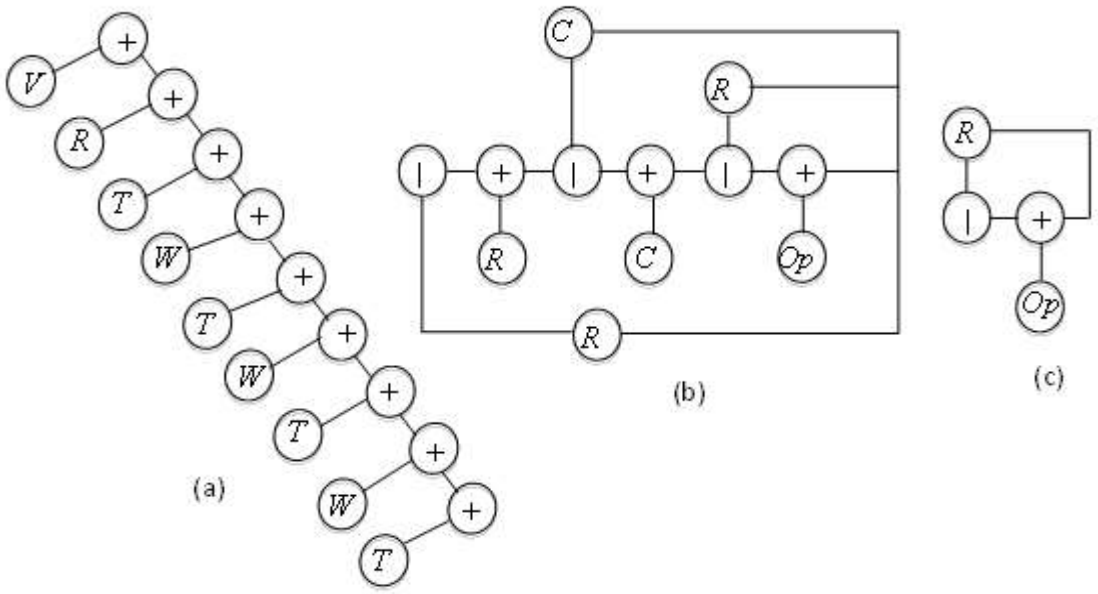

Figure 6.27: (a) GP evolved TS for the band-stop filter (b) *W* representation and (c) *T* representation.

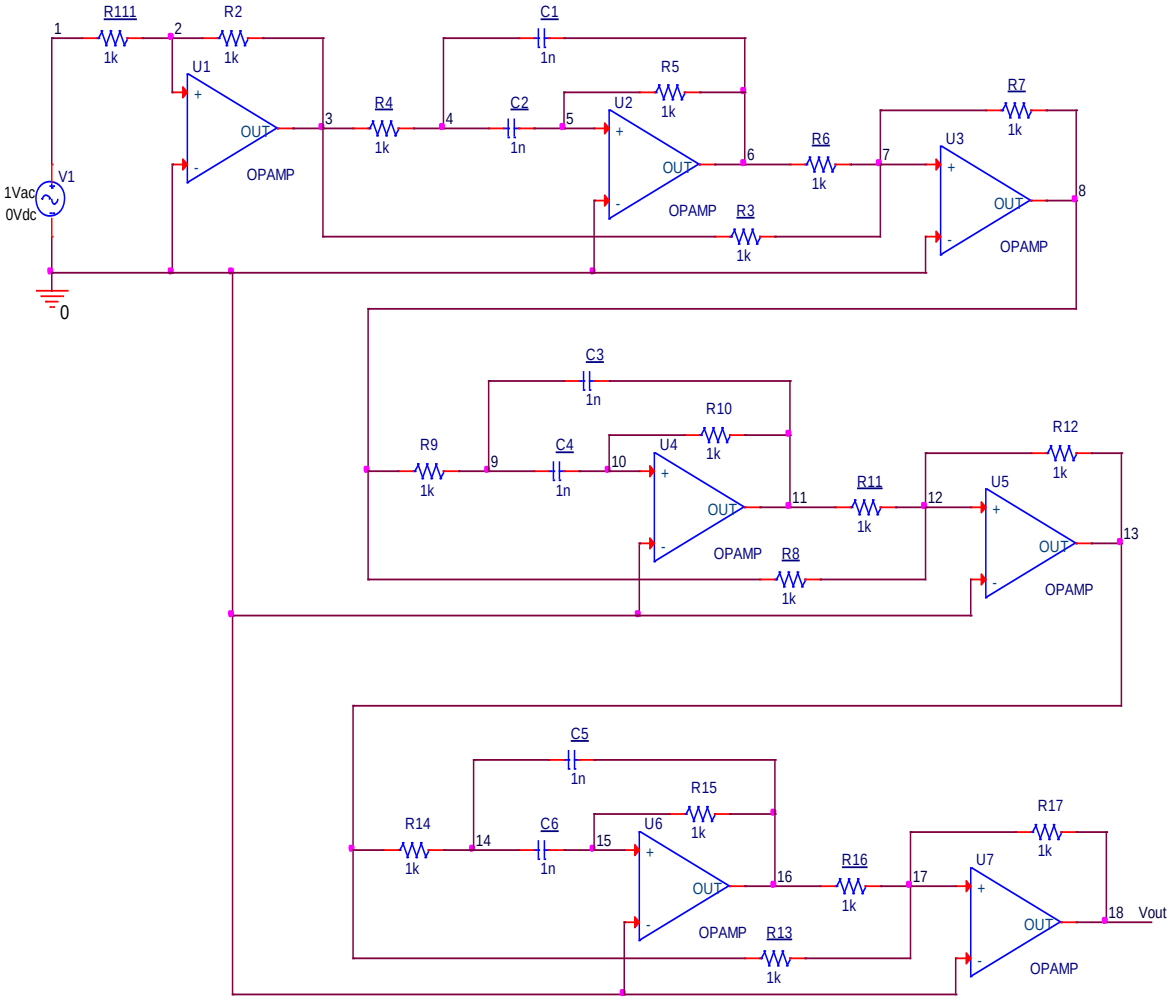

Figure 6.28: GP evolved circuit for the active band-stop filter.

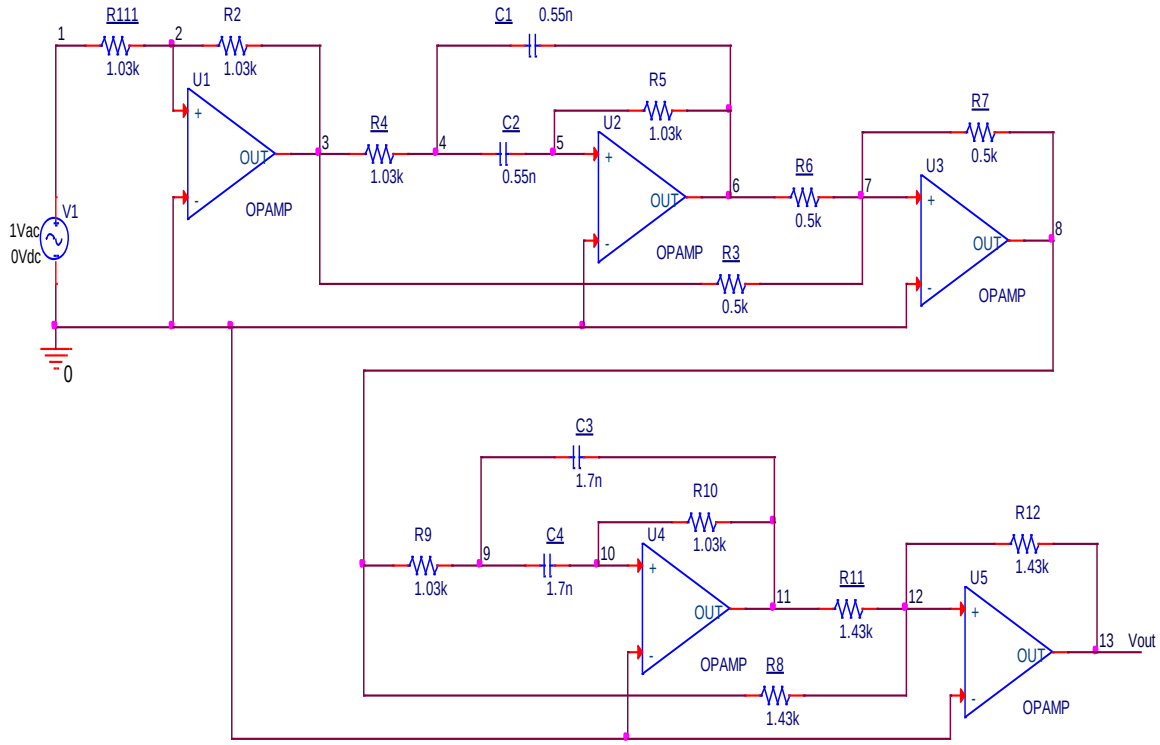

Figure 6.29: GP evolved/reduced/PSO component adjusted circuit for the active band-stop filter.

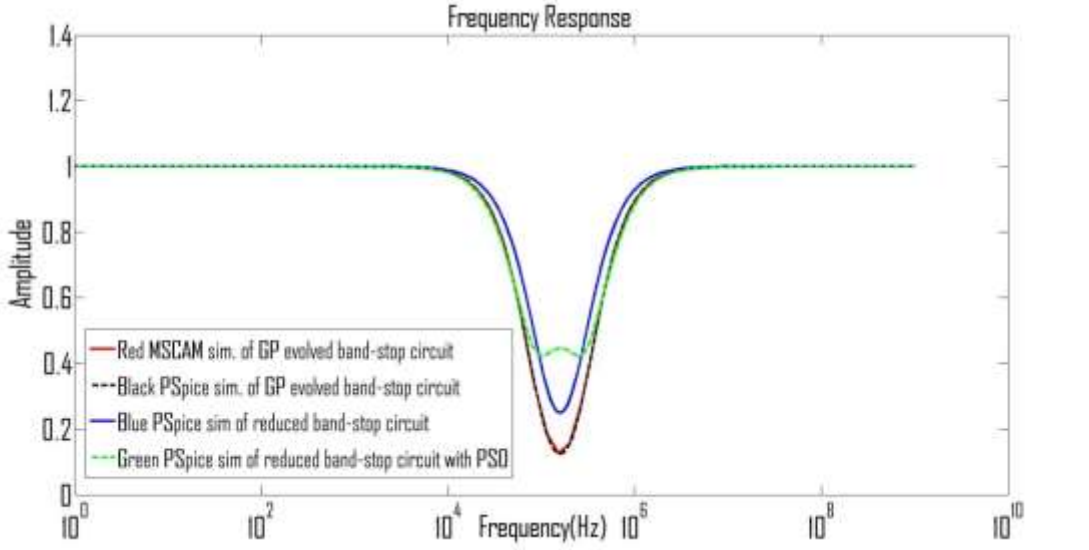

Figure 6.30: Frequency response for GP evolved circuit (Red), PSpice simulation of GP evolved circuit (black), PSpice simulation of reduced GP evolved circuit (blue) and reduced GP evolved circuit with PSO (Green) for the active band-stop filter.

6.4.2 Passive Filter Circuits

Four different examples; two low-pass, one high-pass and one band-pass passive filter circuits are used to demonstrate the efficiency of the algorithm. The algorithm has proved to be efficient as it successfully evolves the four circuits with zero error. The results are the same as the set frequency response specified in the objective function.

6.4.2.1 Example 1: 10th Order Low-Pass Passive Filter Circuit

In example 1, the expected or desired circuit GP representation is shown in Figure 6.31 whereas the equivalent circuit representation is shown in Figure 6.32. It takes fifty-six minutes to evolve the circuit after twenty eight iterations. The MSCAM frequency response of the evolved circuit simulation is shown with the blue colour (solid) whereas the equivalent circuit PSpice simulation is shown with black colour (dash) shown in Figure 6.33. The desired circuit specifications are achieved as regard design and frequency response curve. The MSCAM and original circuit specifications are with cut-off frequencies at 1.07 MHz with zero error and a gain of 1.

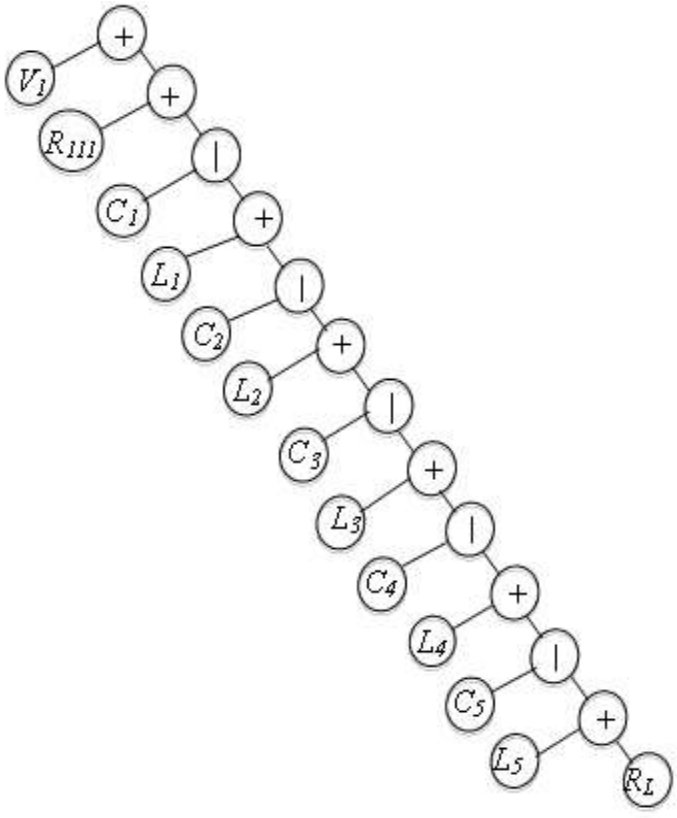

Figure 6.31: Example 1 evolved circuit tree representations.

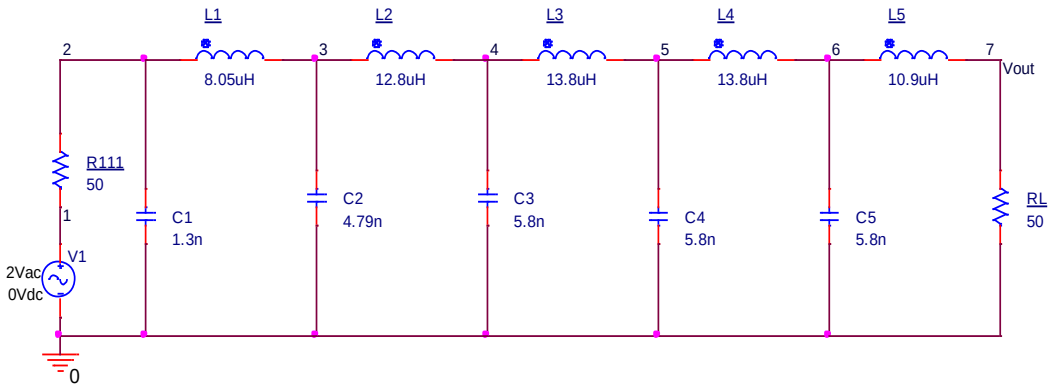

Figure 6.32: Example 1 evolved circuit.

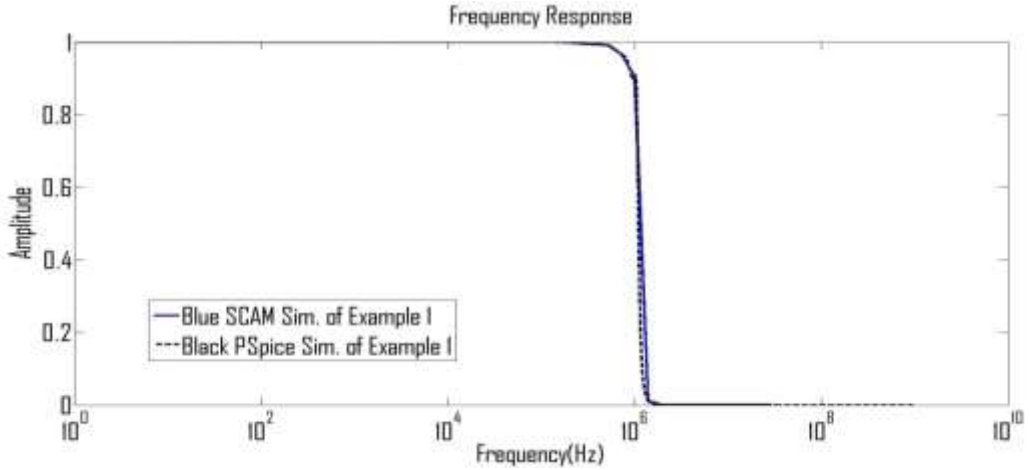

Figure 6.33: Example 1 frequency response curve for MSCAM (blue or solid) and PSpice (black or dashed).

6.4.2.2 Example 2: Low-Pass Passive Filter Circuit

In example 2, the expected or desired circuit GP representation is shown in Figure 6.34 whereas the equivalent circuit representation is shown in Figure 6.35. It takes three hours and twenty-four minutes to evolve the circuit after one hundred and two iterations. The MSCAM frequency response of the evolved circuit simulation is shown in blue colour (solid), whereas the equivalent circuit PSpice simulation is shown in black colour (dash) as in Figure 6.36. The MSCAM and original circuit specifications are with cut-off frequencies at 1.489 kHz with zero error and a gain of 1.

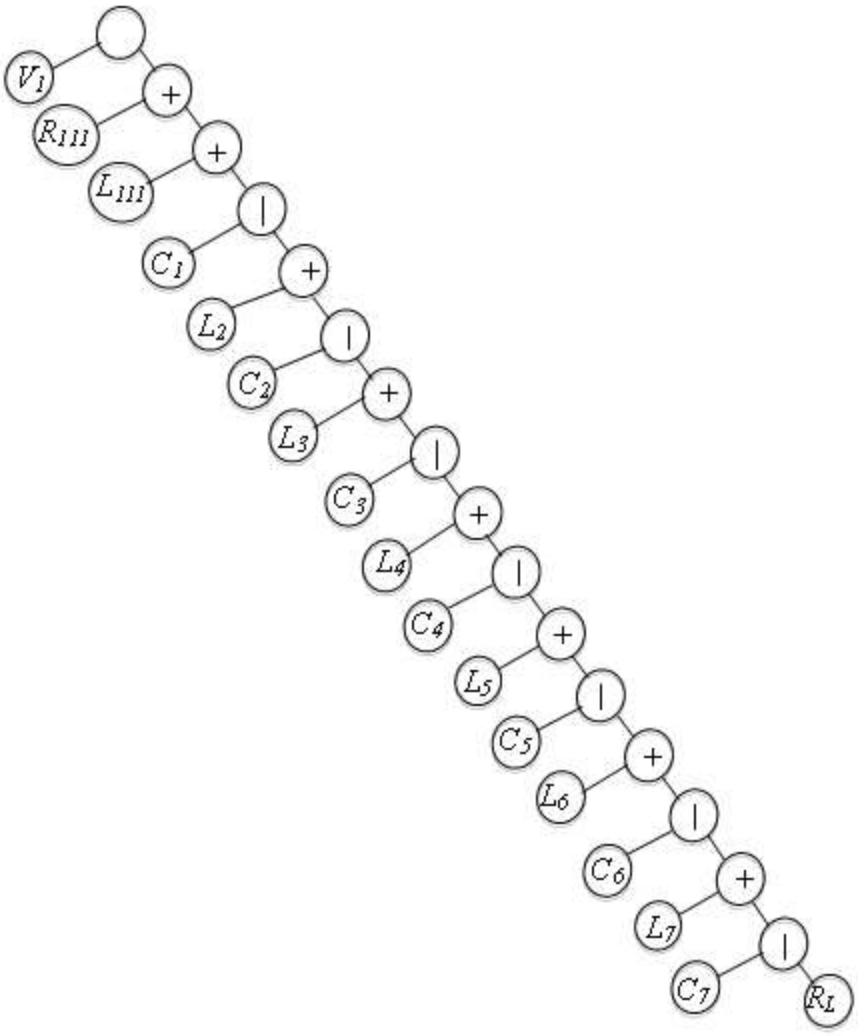

Figure 6.34: Example 2 evolved circuit tree representations.

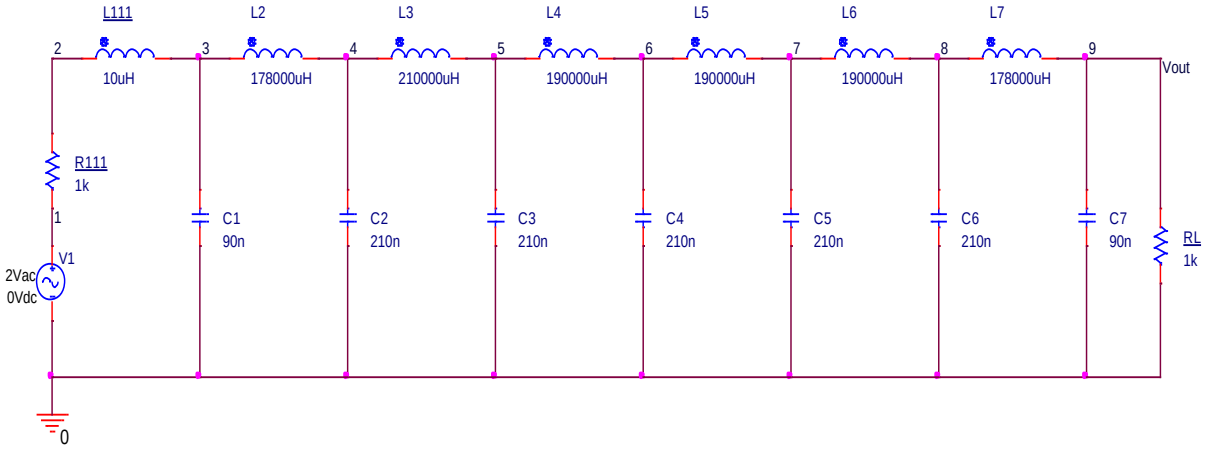

Figure 6.35: Example 2 evolved circuit

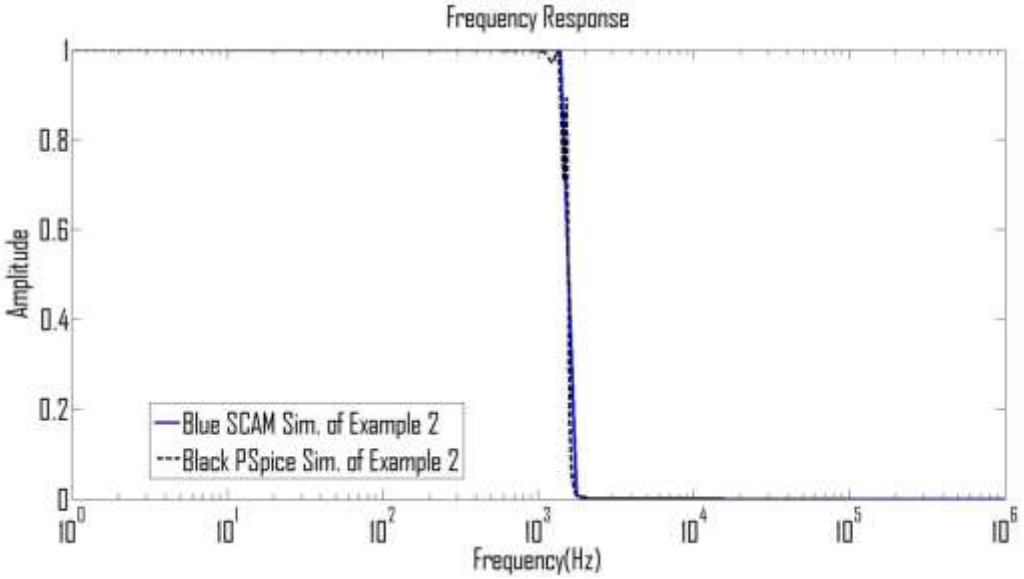

Figure 6.36: Example 2 frequency response for MSCAM (blue or solid) and PSpice (black or dashed).

6.4.2.3 Example 3: High-Pass Passive Filter Circuit

In example 3, the expected or desired circuit GP representation is shown in Figure 6.37, whereas the equivalent circuit representation is shown in Figure 6.38. The optimised circuit is shown in Figure 6.39. It takes thirty-eight minutes to evolve the circuit after one hundred and two iterations. The MSCAM frequency response of the evolved circuit simulation is shown in red colour (dash), whereas the equivalent circuit PSpice simulation is shown in blue colour (dash) as in Figure 6.40. The desired circuit specifications are achieved regarding design wise and frequency response curve. The MSCAM, original circuit specifications, optimised circuit simulation with PSO application, optimised circuit simulation without PSO application have cut-off frequencies at 8.35 MHz with zero error and a gain of 0.0625. The total number of component count reduction is 2.

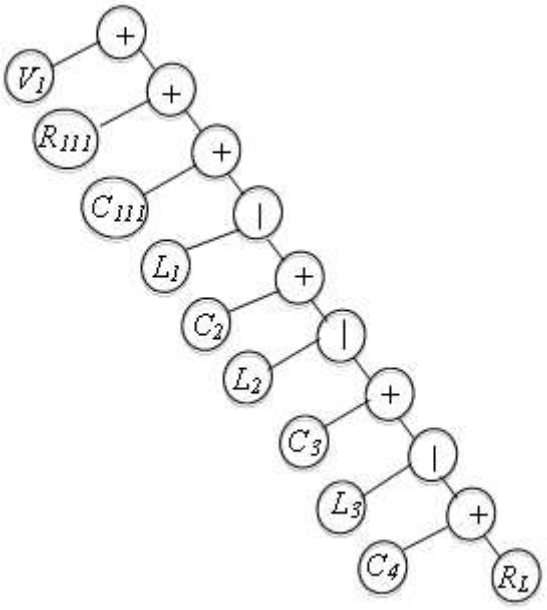

Figure 6.37: High-pass evolved circuit tree representations.

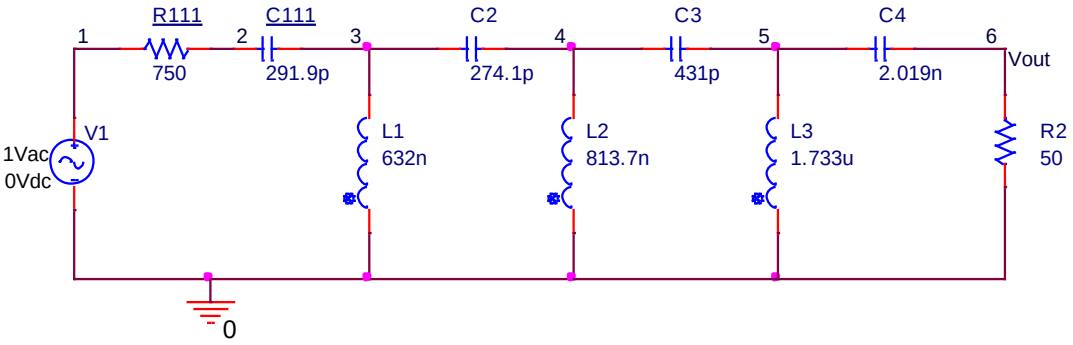

Figure 6.38: GP evolved circuit for the passive high pass filter.

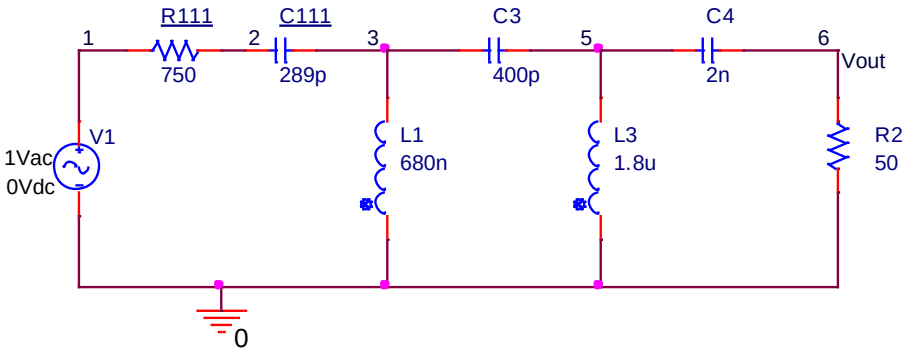

Figure 6.39: GP evolved/reduced circuit for the passive high pass filter.

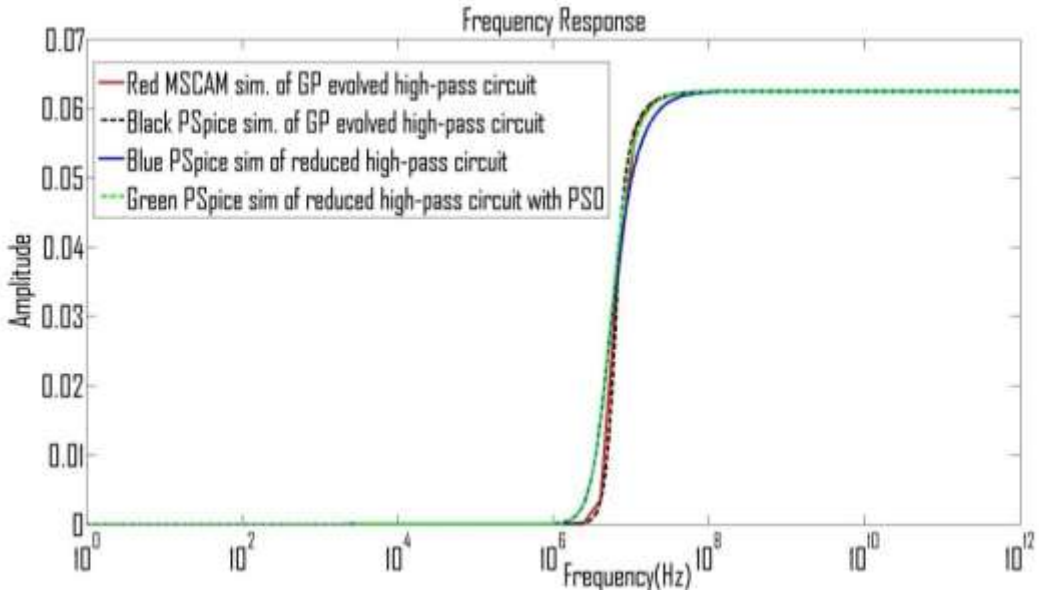

Figure 6.40: High-pass frequency response for MSCAM (blue or solid) and PSpice (black or dashed).

6.4.2.4 Example 4: Band-Pass Passive Filter Circuit

In example 4, the expected or desired circuit GP representation is shown in Figure 6.41 whereas the equivalent circuit representation is shown in Figure 6.42. It takes two hours to evolve the circuit after one hundred and two iterations. The MSCAM frequency response of the evolved circuit simulation is shown with red colour (dash), whereas the equivalent circuit PSpice simulation is shown in blue colour (dash) in Figure 6.43. The same algorithm has been used for all these circuits evolution with little modifications in the cut-off frequency, length of chromosome, bit groups, which are values to play with, and it eliminates mathematical computations involving human methods. The GP evolved MSCAM simulation and original circuit specifications are with the lower and upper cut-off frequencies at 8.242 kHz and 48.59 kHz respectively with zero error and a gain of 0.0625.

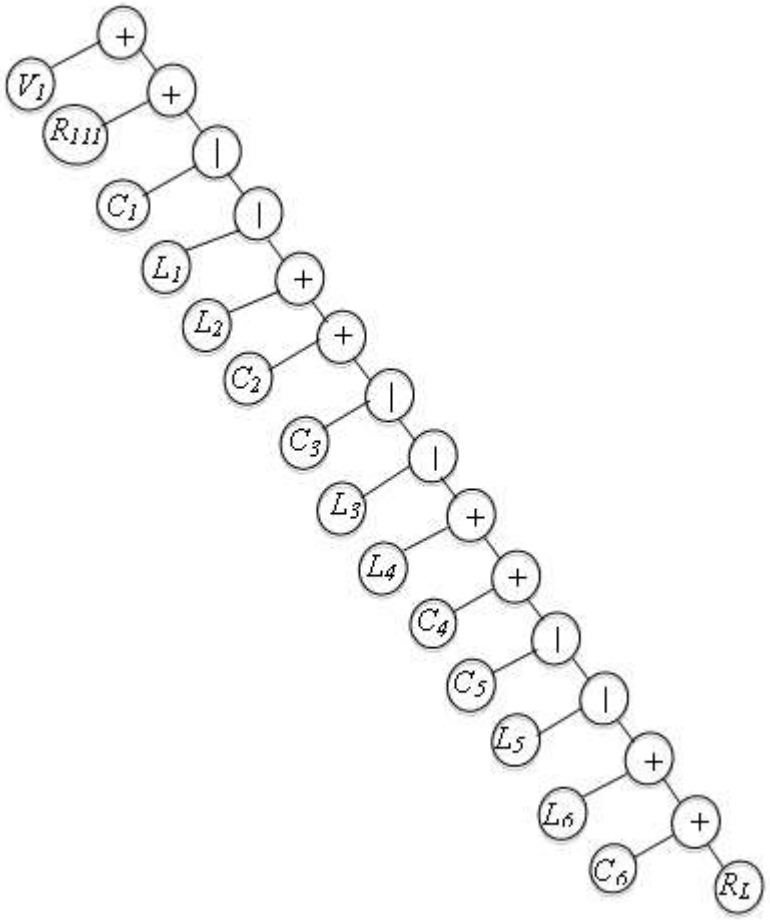

Figure 6.41: Band-pass evolved circuit tree representations.

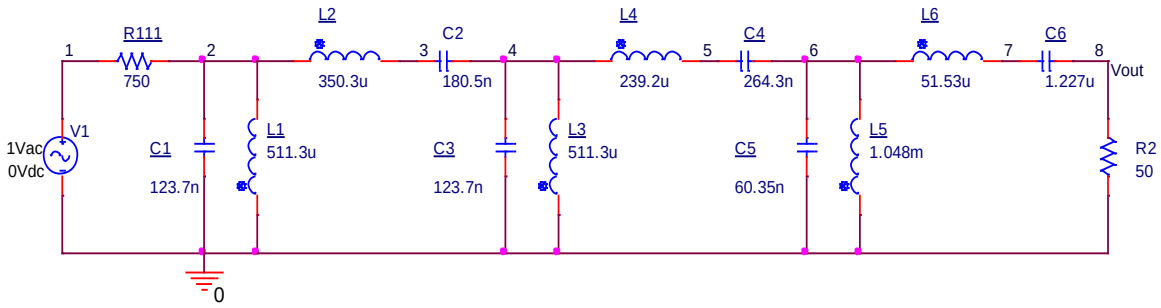

Figure 6.42: Band-pass evolved circuit tree representations.

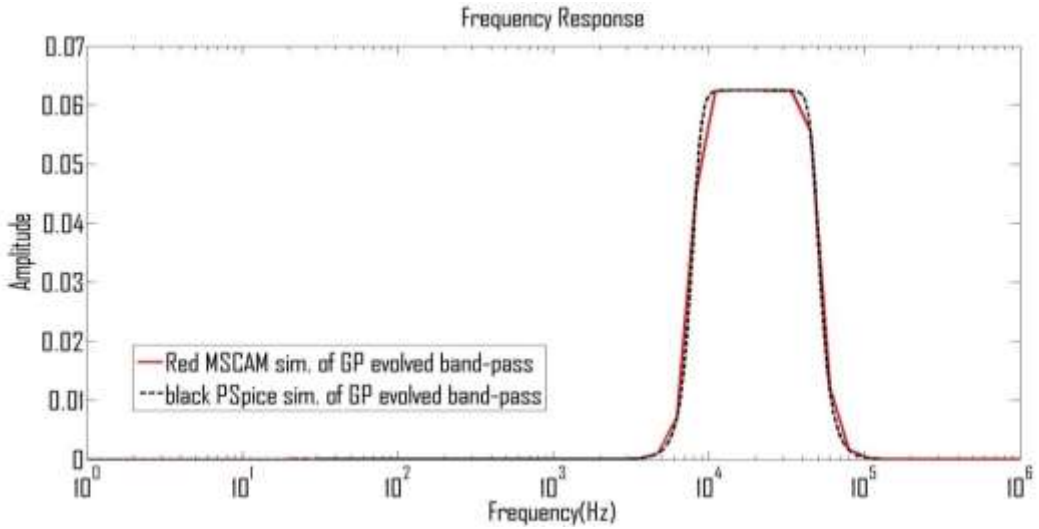

Figure 6.43: Band-pass frequency response for MSCAM (red or solid) and PSpice (black or dashed).

6.4.3 Transistor Amplifier Circuit

The algorithm is also used to evolve SSA of transistor circuits and it is very efficient to evolve them because the frequency responses of all the three evolving GP circuits of the sampled transistors were the same as that of set frequency as illustrated below.

6.4.3.1 Example 1: Common-Collector Transistor Amplifier Circuit

The common collector transistor amplifier circuit is shown in Figure 6.44, GP evolved desired SSA circuit TS is shown in Figure 6.45 whereas its equivalent circuit is shown in Figure 6.46. It takes twenty-four minutes to evolve the circuit and twelve iterations. The MSCAM frequency response of the evolved circuit is shown in black colour and the PSpice simulation of the SSA circuit is indicated in black colour as shown in Figure 6.47. The GP evolved SSA of common collector circuit MSCAM simulation and original circuit specifications have the same lower and upper cut-off frequencies at 46.5 Hz and 95.35 MHz respectively with 2% error and a gain of 0.95.

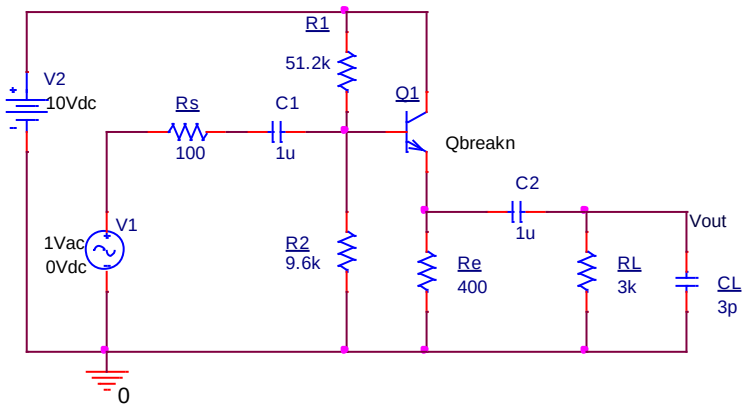

Figure 6.44: Common-collector transistor amplifier circuit.

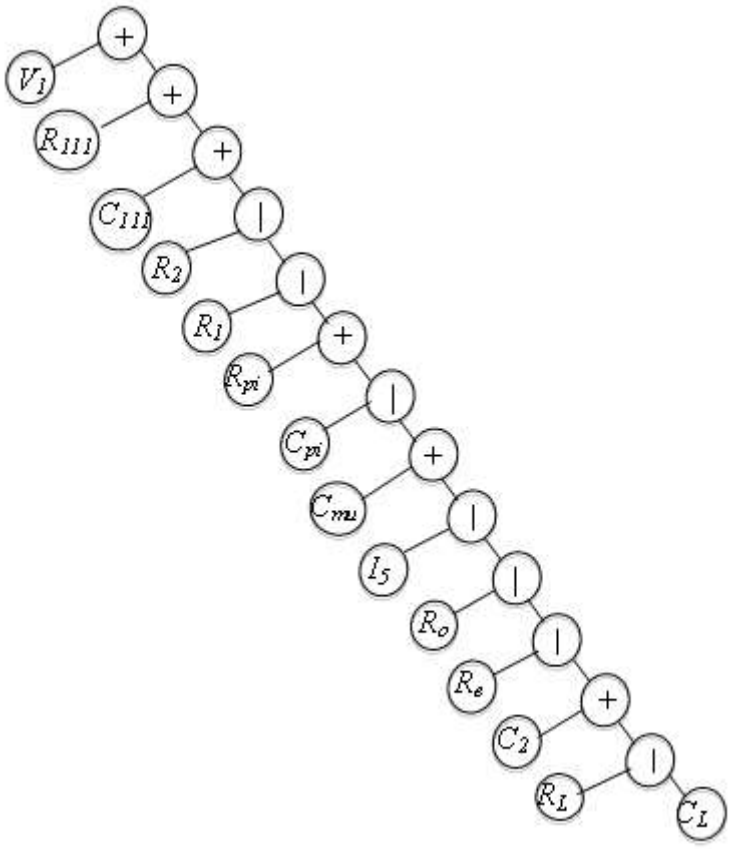

Figure 6.45: GP evolved TS for common-collector transistor amplifier circuit.

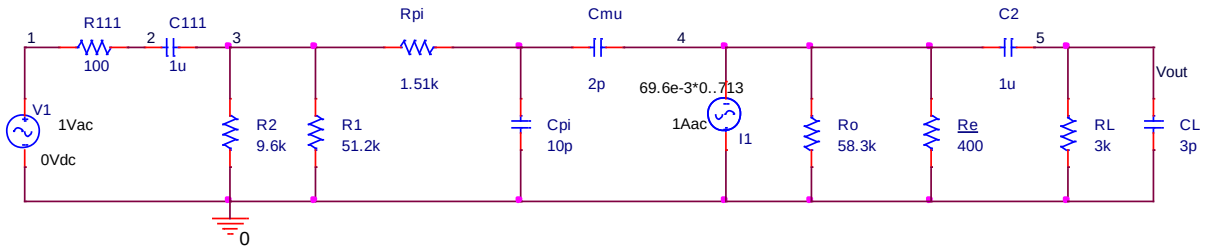

Figure 6.46: GP evolved circuit for the common-collector transistor amplifier circuit.

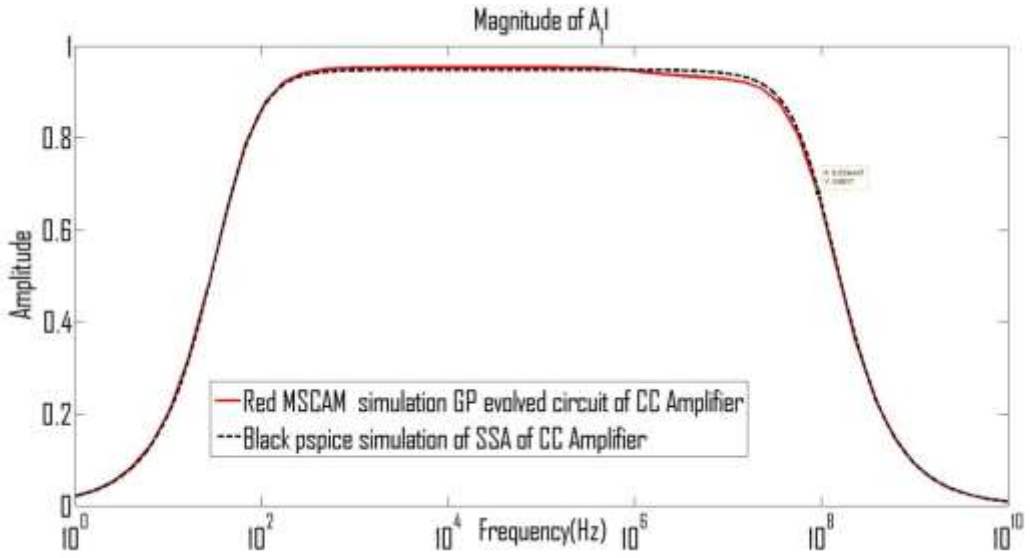

Figure 6.47: Frequency response curve of the SSA simulation (black) and GP evolved circuit (red) for the common-collector transistor amplifier.

6.4.3.2 Example 2: Common-Emitter Transistor Amplifier Circuit

The common-emitter transistor amplifier circuit is shown in Figure 6.48, GP evolved the desired SSA circuit TS which is shown in Figure 6.49 and its equivalent circuit is shown in Figure 6.50. It takes twenty-two minutes to evolve the circuit and eleven iterations. The MSCAM frequency response of the evolved circuit is shown in black colour and the PSpice simulation of the SSA circuit is indicated in the black colour as shown in Figure 6.51. The GP evolved SSA of common-emitter circuit MSCAM simulation and original circuit specifications have the same lower and upper cut-off frequencies at 32.5 Hz and 26.1 MHz respectively with zero error and a gain of 75.29.

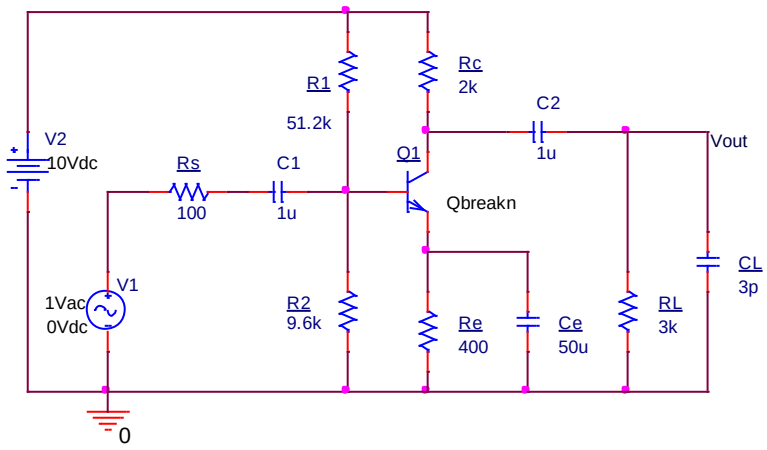

Figure 6.48: Common-emitter transistor amplifier circuit.

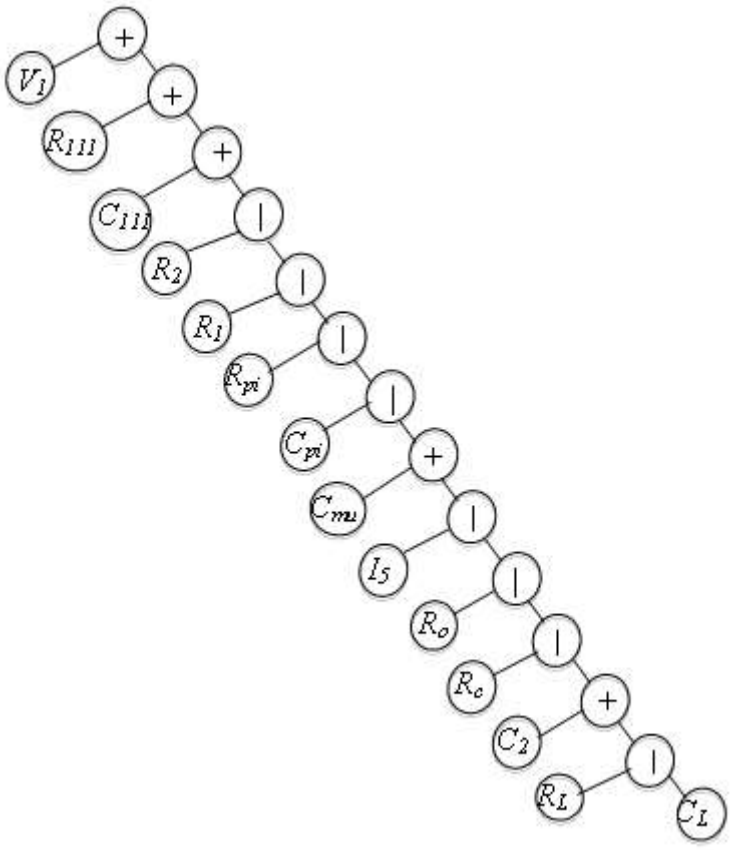

Figure 6.49: GP evolved TS for common-emitter transistor amplifier circuit.

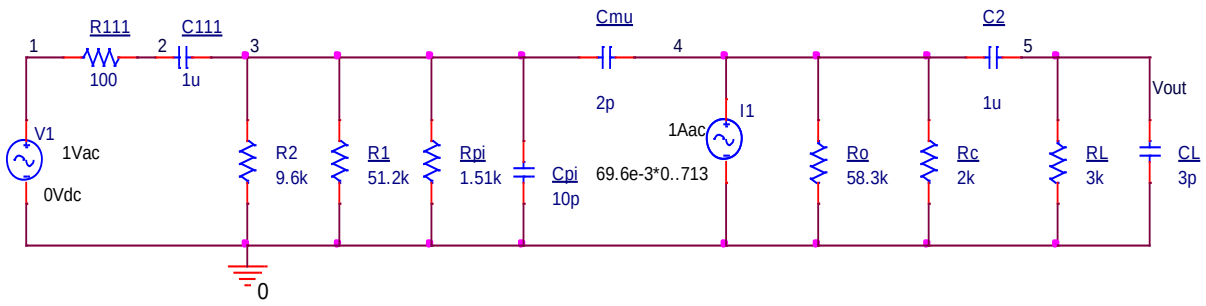

Figure 6.50: GP evolved circuit for the common-emitter transistor amplifier circuit.

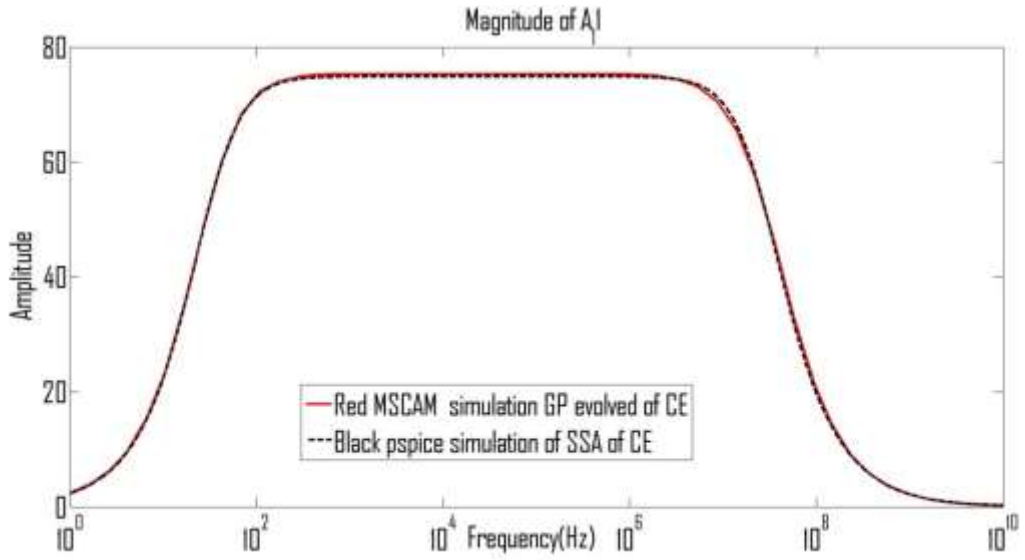

Figure 6.51: Frequency response curve of the SSA simulation (black) and GP evolved circuit (red) for the common-emitter transistor amplifier.

6.4.3.3 Example 3: FET Transistor Amplifier Circuit

The FET transistor amplifier circuit is shown in Figure 6.52, GP evolved the desired SSA circuit TS which is shown in Figure 6.53 and its equivalent circuit is shown in Figure 6.54. It takes twenty-six minutes to evolve the circuit and thirteen iterations. The MSCAM frequency response of the evolved circuit is shown in black colour and the PSpice simulation of the SSA circuit is indicated in the black colour as shown in Figure 6.55. The GP evolved SSA of FET circuit MSCAM simulation, and original circuit specifications have the same lower and upper cut-off frequencies at 30.9 Hz and 25.7 MHz respectively with zero error and a gain of 2.68.

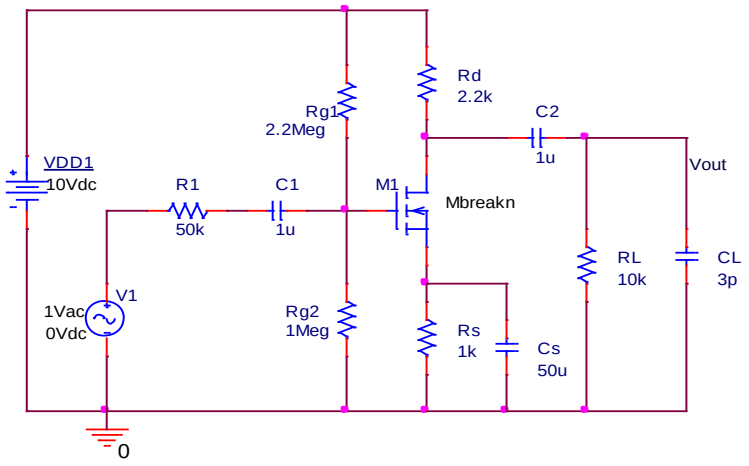

Figure 6.52: Common-source FET amplifier circuit.

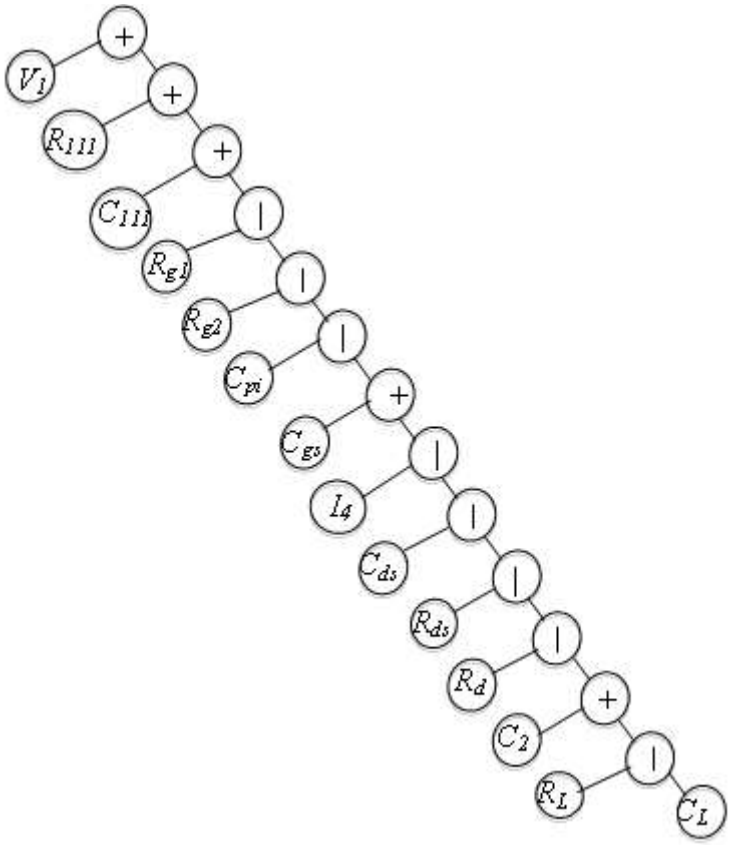

Figure 6.53: GP evolved TS for the common-source FET amplifier circuit.

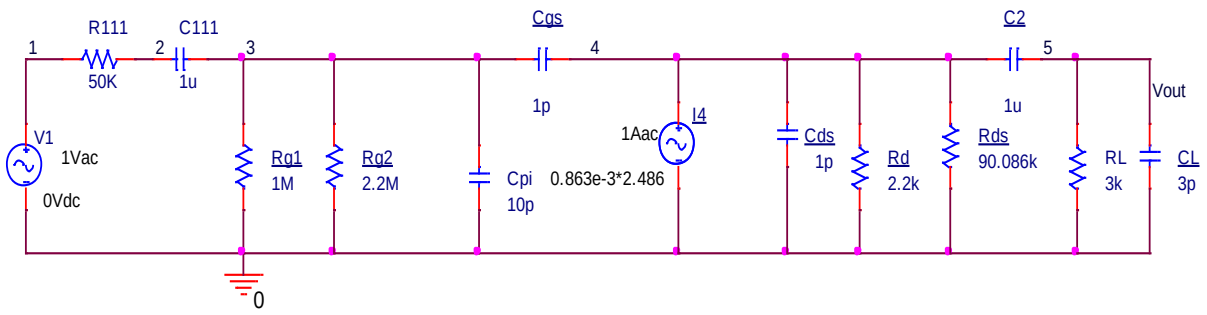

Figure 6.54: GP evolved circuit for the common-source FET amplifier circuit.

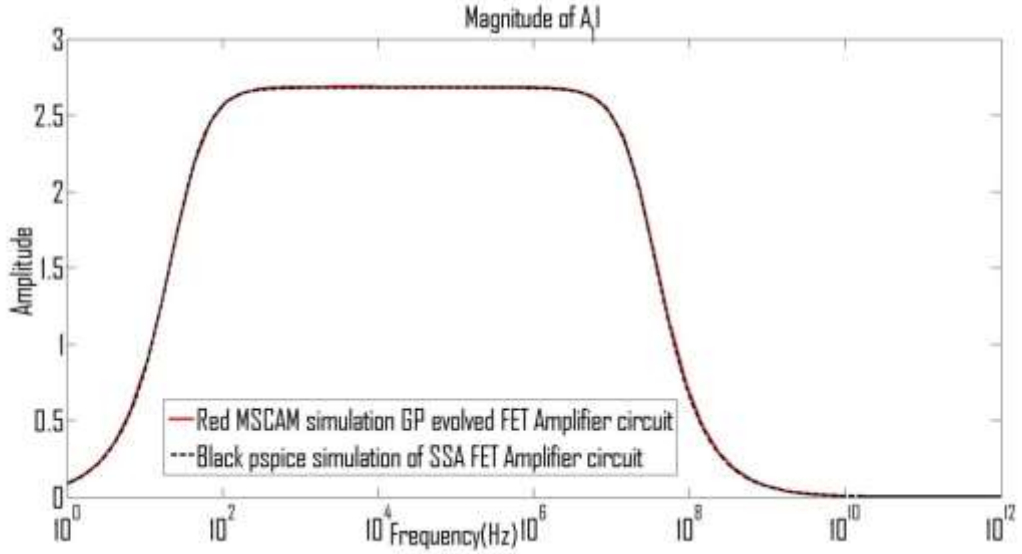

Figure 6.55: Frequency response curve of the SSA simulation (black), and the GP evolved circuit (red) for the common-source FET amplifier.

6.5 Summary

This work introduced the use of GF, MSCAM, automatically simulated Netlist and GP for its first time use for active filter and passive filter circuits' evolution. The developed and tested algorithm in Chapter 5 is modified with the introduction of MSCAM and how automatically simulated Netlist and being used for analogue circuits evolution. The same algorithm has been used for all these circuits evolution with little modifications in the cut-off frequency, length of chromosome, bit groups which are values to play with and it eliminates mathematical computations involving human methods. The research has provided an alternative approach of applying GP for the evolution of passive and active filter circuits or a Matlab toolbox for analogue circuit evolution. The elapsed time used while transferring simulation between software packages is reduced. Twelve different (eight examples of active

filter and four examples of passive filter) circuits are used to demonstrate the efficiency of the approach and algorithm successfully evolved all the circuits.

Chapter 7 is next which is centred on the conclusions and future work.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In Chapter 1, a brief rational for analogue circuit optimisation, the motivation, aim and objective, thesis contribution to knowledge and the list of publication made during this work are presented. Chapter 2 presented definition and general background of optimisation and optimisation algorithm. The review of various optimisation methods used in this research is presented. It also described in detail the history of these developed techniques, principle of inspiration and different application areas. Chapter 2 is concluded by stating the advantages and disadvantages of each method and summary on how each technique can be used to optimise analogue circuit is stated

In Chapter 3, MSCAM is introduced. The description of how the developed algorithm using modified nodal analysis in combination with the newly introduced SSA is illustrated on four different circuits. Results are presented to demonstrate the efficiency of the developed algorithm.

In Chapter 4, four different swarm optimisation and GA algorithms for analogue circuit optimisation are presented. The first result demonstrates how an equivalent analogue circuit can be found by applying the optimisation techniques. The examples further show that component count reduction is achieved in analogue circuit same as it has been accomplished in digital circuits. Other three results show how the approach is used to minimise op-amp filter circuits. In addition, PSO offers the best results regarding frequency response for the four examples.

In Chapter 5, different units involve in the formulation GP algorithm is described. It also illustrates how GF is incorporated into the GP by showing how the TS are structurally linked from first element to the last. The developed GP algorithm is tested with four different benchmark functions. The algorithm is efficient because it successfully evolved the benchmark equations.

In Chapter 6, this unit introduces the use of GF, MSCAM automatically simulated Netlist and GP for its first time use for active filter and passive filter circuits' evolution. The developed and tested algorithm in Chapter 5 is modified with the introduction of MSCAM and how automatically simulated Netlist are being used for analogue circuits evolution. Twelve different results (five examples of active filter, four examples of passive filter circuits and three examples of transistor amplifier circuits) are used to demonstrate the efficiency of the approach and algorithm successfully evolved all the circuits. In this final Chapter 7, conclusions and future research are presented.

The three main contributions to knowledge in this research are:

- This research introduced the concept of component count reduction in passive and active filter circuits which reduce the size, power consumption and increase circuit reliability. It also surveyed five artificial intelligence methods, and have identified PSO algorithm to be the best method among them in terms of power consumption reduction, speed of convergence and use it to optimise analogue circuit demonstrated in Chapter 2 and 4.
- This work presented MSCAM that uses Netlist from PSpice or simulation to generate matrices. These matrices are used to calculate circuit parameters or used for optimisation illustrated in Chapter 3. The MSCAM enhances the matrices dimension of more than 30 by 30 so that it can be used to simulate complex circuits. This is important especially when operational amplifier (op-amp) is involved as circuit component compared existing SCAM that cannot handle matrices dimension more than eight by eight. The SCAM formed matrices by adding additional rows and columns due to how the algorithm was developed which takes more computer resources and limit its performance.
- Also, this work has developed an automated algorithm that combines GF, automatically generated Netlist from simulation, MSCAM and GP for the evolution of active and passive filter circuits demonstrated in Chapter 5 and 6. The GF, MSCAM, and automatically simulated Netlist is introduced into existing GP which is a new contribution in this work, it enhances the development of independent Matlab toolbox. The simulator uses only Matlab compare to existing GP which combine Matlab and PSpice. The newly developed code is then tested for its efficiency using four benchmark expressions.

7.2 Future Work

Although research demonstrated several contributions to Matlab toolbox and circuit optimisation problems, there are still some areas this work can be improved further.

- The developed GP algorithm can be improved further to be able to handle branching or intermediate nodes. In other words, how to improve on the computer code to evolve a circuit system that the node numbering is non- linear. This will enable the algorithm to find variety of circuit to a given circuit specification.
- Also to enhance the computer program to evolve a circuit with multi-feedback loop that do not repeat regular pattern.
- To incorporate the PSO algorithm into the GP algorithm for automatic circuit optimisation rather than applying PSO algorithm after circuit evolution. This will reduce the elapse time used to evolve the truncated part during optimisation before using the PSO algorithm.

References

[1] http://www.newelectronics.co.uk/electronics-news/fibre-optic-speeds-outside-of-fibre/115165/#sthash.Xs0zThsf.dpuf. Feb. 27, 2016.

[2]) http://www.newelectronics.co.uk/electronics-news/printed-sensing-technology-for-metal-tooling-applications/115151/#sthash.PuaqZ8AF.dpuf. Feb.27, 2016

[3] Y. Statter and T. Chen, "A novel high-throughput method for table look-up based analog design automation," *Integration, the VLSI Journal*, vol. 52, pp. 168-181, 2016.

[4] O. J. Ushie and M. Abbod, " " Intelligent Optimization Methods for Analogue Electronic Circuits: GA and PSO Case Study" ," *International Conference on Machine Learning, Electrical and Mechanical Engineering (ICMLEME'2014)*, pp. 193-194-199, on Jan. 8-9, 2014 Dubai (UAE), 2014.

[5] O. J. Ushie, M. Abbod and E. C. Ashigwuike, " Regular paper Naturally Based Optimisation Algorithm for Analogue Electronic Circuits: GA, PSO, ABC, BFO, and Firefly a Case Study " *J. Automation & System Engineering*, vol. 9 -3, pp. 26-60, 2015.

[6] X. Yang, *Engineering Optimization: An Introduction with Metaheuristic Applications.* John Wiley & Sons, 2010.

[7] S. M. Kaplan, *Wiley Electrical and Electronics Engineering Dictionary.* IEEE Press, 2004.

[8] M. Ginsberg, *Essentials of Artificial Intelligence.* Newnes, 2012.

[9] H. Henderson, *Artificial Intelligence: Mirrors for the Mind.* Infobase Publishing, 2007.

[10] T. H. Cormen, *Introduction to Algorithms.* MIT press, 2009.

[11] X. Yang, *Nature-Inspired Optimization Algorithms.* Elsevier, 2014.

[12] John Henry Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT press, 1992.

[13] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT press, 1992.

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

[15] D. Fogel, "Artificial intelligence through simulated evolution," 2009.

[16] I. Rechenberg, "Evolution Strategy: Optimization of Technical systems by means of biological evolution," *Fromman-Holzboog, Stuttgart*, vol. 104, 1973.

[17] C. Ferreira and U. Gepsoft, *What is Gene Expression Programming*, 2008.

[18] A. F. Sheta, "Analogue filter design using differential evolution," *International Journal of Bio-Inspired Computation*, vol. 2, pp. 233-241, 2010.

[19] K. S. Reddy and S. K. Sahoo, "An approach for FIR filter coefficient optimization using differential evolution algorithm," *AEU-International Journal of Electronics and Communications*, vol. 69, pp. 101-108, 2015.

[20] S. Sivanandam and S. Deepa, "Introduction to genetic algorithms. 2008," .

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

[22] M. O'Dare and T. Arslan, "Generating test patterns for VLSI circuits using a genetic algorithm," *Electron. Lett.*, vol. 30, pp. 778-779, 1994.

[23] S. J. Louis, "Genetic learning for combinational logic design," *Soft Computing*, vol. 9, pp. 38-43, 2005.

[24] M. H. Zarifia, N. K. Ghalehjogh and M. Baradaran-nia, "A new evolutionary approach for neural spike detection based on genetic algorithm," *Expert Syst. Appl.*, vol. 42, pp. 462-467, 2015.

[25] M. Bechouat, Y. Soufi, M. Sedraoui and S. Kahla, "Energy storage based on maximum power point tracking in photovoltaic systems: A comparison between GAs and PSO approaches," *Int J Hydrogen Energy*, vol. 40, pp. 13737-13748, 2015.

[26] D. H. Horrocks and Y. M. Khalifa, "Genetic algorithm design of electronic analogue circuits including parasitic effects," in *Proceedings of the First Online Workshop on Soft Computing (WSC1)*, pp. 274-278, 1996.

[27] C. A. Coello, A. D. Christiansen and A. H. Aguirre, "Automated design of combinational logic circuits using genetic algorithms," in *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pp. 335-338, 1997.

[28] J. F. Miller, D. Job and V. K. Vassilev, "Principles in the evolutionary design of digital circuits—Part I," *Genetic Programming and Evolvable Machines*, vol. 1, pp. 7-35, 2000.

[29] A. Aggarwal, T. K. Rawat, M. Kumar and D. Upadhyay, "Optimal design of FIR high pass filter based on L 1 error approximation using real coded genetic algorithm," *Engineering Science and Technology, an International Journal*, 2015.

[30] A. Das and R. Vemuri, "An automated passive analog circuit synthesis framework using genetic algorithms," *IEEE Computer Society Annual Symposium* in *VLSI, ISVLSI'07*, pp. 145-152, 2007.

[31] 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, .

[32] M. Wojcikowski, J. Glinianowicz and M. Bialko, "System for optimisation of electronic circuits using genetic algorithm," *Proceedings of the Third IEEE International Conference* in *Electronics, Circuits, and Systems, 1996. ICECS'96*, pp. 247-250, 1996.

[33] M. Taherzadeh-Sani, R. Lotfi, H. Zare-Hoseini and O. Shoaei, "Design optimization of analog integrated circuits using simulation-based genetic algorithm," *SCS 2003. International Symposium* in *Signals, Circuits and Systems*, pp. 73-76, 2003.

[34] A. Yang, Y. Shan and L. T. Bui, *Success in Evolutionary Computation.* Springer, 2008.

[35] R. J. Al-Azawi and M. Abdul-whab, "Design Active Filter Based on Genetic Algorithm," *IBN Al-Haitham J. for Pure & Appl. SCI, vol.21 (1),* 2008.

[36] D. H. Horrocks and M. C. Spittle, "Component value selection for active filters using genetic algorithms," in *Proc. IEE/IEEE Workshop on Natural Algorithms in Signal Processing, Chelmsford, UK,* pp. 3. 1993,

[37] T. Kaya and M. C. Ince, "The Design of Analog Active Filter with Different Component Value using Genetic Algorithm," *International Journal of Computer Applications,* vol. 45, pp. 43-47, 2012.

[38] R. S. Zebulum, M. A. Pacheco and M. Vellasco, "Artificial evolution of active filters: A case study," *Proceedings of the First NASA/DoD Workshop* in *Evolvable Hardware,* pp. 66-75, *1999.*

[39] M. Barros, J. Guilherme and N. Horta, "Analog circuits optimization based on evolutionary computation techniques," *Integration, the VLSI Journal,* vol. 43, pp. 136-155, 1, 2010.

[40] B. Liu, Y. Wang, Z. Yu, L. Liu, M. Li, Z. Wang, J. Lu and F. V. Fernández, "Analog circuit optimization system based on hybrid evolutionary algorithms," *INTEGRATION, the VLSI Journal,* vol. 42, pp. 137-148, 2009.

[41] L. Zhang and Z. Liu, "Directly performance-constrained template-based layout retargeting and optimization for analog integrated circuits," *Integration, the VLSI Journal,* vol. 44, pp. 1-11, 1, 2011.

[42] S. Wang, Y. Hwang, S. Yan and J. Chen, "A new CMOS wideband low noise amplifier with gain control," *Integration, the VLSI Journal,* vol. 44, pp. 136-143, 2011.

[43] T. Golonek and J. Rutkowski, "Genetic-algorithm-based method for optimal analog test point selection," *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol. 54, pp. 117-121, 2007.

[44] Ö S. Sönmez and G. Dündar, "Simulation-based analog and RF circuit synthesis using a modified evolutionary strategies algorithm," *Integration, the VLSI Journal,* vol. 44, pp. 144-154, 3, 2011.

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

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

[47] 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," *Proceedings. 2005 NASA/DoD Conference* in *Evolvable Hardware,* pp. 205-212, *2005.*

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

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

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

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

[52] 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.

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

[54] 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*, pp. 947-954, 2009.

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

[56] 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.

[57] 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*, pp. 3003, 2005.

[58] 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.

[59] D. W. Mount and D. W. Mount, *Bioinformatics: Sequence and Genome Analysis.* Cold spring harbor laboratory press New York, 2001.

[60] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, pp. 317-325, 2003.

[61] L. Zhou, Y. Shi, Y. Li and W. Zhang, "Parameter selection, analysis and evaluation of an improved particle swarm optimizer with leadership," *Artif. Intell. Rev.*, vol. 34, pp. 343-367, 2010.

[62] S. Das, A. Abraham and A. Konar, "Particle swarm optimization and differential evolution algorithms: Technical analysis, applications and hybridization perspectives," in *Advances of Computational Intelligence in Industrial Systems*, Springer, pp. 1-38, 2008.

[63] J. O. Ushie, J. A. Obu and I. P. Etim, "Optimizing digital combinational circuit using particle swarm optimization technique," *Lat. Am. J. Phys. Educ.*, vol. Vol. 6, pp. 72, 2012.

[64] P. W. Moore and G. K. Venayagamoorthy, "Evolving digital circuits using hybrid particle swarm optimization and differential evolution," *Int. J. Neural Syst.*, vol. 16, pp. 163-177, 2006.

[65] C. A. Coello Coello, E. H. Luna and A. H. Aguirre, "A comparative study of encodings to design combinational logic circuits using particle swarm optimization," *Conference* in *Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD,* pp. 71-78, 2004.

[66] E. H. Luna, C. Coello Coello and A. H. Aguirre, "On the use of a population-based particle swarm optimizer to design combinational logic circuits," in *Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD Conference On,* pp. 183-190, 2004.

[67] C. A. C. Coello, E. H. Luna and A. H. Aguirre, "Use of particle swarm optimization to design combinational logic circuits," in *Evolvable Systems: From Biology to Hardware,* Springer, pp. 398-409, 2003.

[68] V. G. Gudise and G. K. Venayagamoorthy, "Evolving digital circuits using particle swarm," *Proceedings of the International Joint Conference* in *Neural Networks,* pp. 468-472, *2003.*

[69] R. Yadav and M. Gupta, "New improved fractional order integrators using PSO optimisation," *International Journal of Electronics,* vol. 102, pp. 490-499, 2015.

[70] Y. Zhang, S. Wang and G. Ji, "A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications," *Mathematical Problems in Engineering,* vol. 501, pp. 931256, 2015.

[71] M. Kotti, M. Fakhfakh and M. H. Fino, "On the dynamic rounding-off in analogue and RF optimal circuit sizing," *International Journal of Electronics,* vol. 101, pp. 452-468, 2014.

[72] J. Zhang, Y. Shi and Z. Zhan, "Power electronic circuits design: A particle swarm optimization approach," in *Simulated Evolution and Learning,* Springer, pp. 605-614, 2008.

[73] M. Fakhfakh, Y. Cooren, A. Sallem, M. Loulou and P. Siarry, "Analog circuit design optimization through the particle swarm optimization technique," *Analog Integr. Cir. Signal Proc.,* vol. 63, pp. 71-82, 2010.

[74] M. Fakhfakh, Y. Cooren, M. Loulou and P. Siarry, "A Particle Swarm Optimization technique used for the improvement of analogue circuit performances," *Particle Swarm Optimization,* pp. 169-182, 2009.

[75] S. Ulker, "Design of Low Noise Microwave Amplifiers Using Particle Swarm Optimization," *International Journal of Artificial Intelligence & Applications,* vol. 3, 2012.

[76] S. Ulker, "Broadband Microwave Amplifier Design Using Particle swarm optimization," *Journal of Computers,* vol. 11, pp. 2272-2276, 2011.

[77] P. P. Kumar and K. Duraiswamy, "An Optimized Device Sizing of Analog Circuits using Particle Swarm Optimization," *Journal of Computer Science,* vol. 8, pp. 930, 2012.

[78] B. P. De, R. Kar, D. Mandal and S. Ghoshal, "Optimal analog active filter design using craziness-based particle swarm optimization algorithm," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields,* 2014.

[79] N. M. Laskar, P. Paul, S. Nath and K. Baishnab, "Investigating the switching performance of an inverter design using the Human Behavior based PSO," *History,* vol. 43, pp. 53-59, 2015.

[80] G. Karimi, H. Akbarpour and A. Sadeghzadeh, "Multi Objective Particle Swarm Optimization based Mixed Size Module Placement in VLSI Circuit Design," *Appl.Math,* vol. 9, pp. 1485-1492, 2015.

[81] S. Manjula and D. Selvathi, "Optimal Design of Low Power CMOS Power Amplifier Using Particle Swarm Optimization Technique," *Wireless Personal Communications,* vol. 82, pp. 2275-2289, 2015.

[82] S. P. Mohanty, E. Kougianos and V. P. Yanambaka, "Ultra-fast variability-aware optimization of mixed-signal designs using bootstrapped kriging," *16th International Symposium* in *Quality Electronic Design (ISQED),* pp. 239-242, 2015.

[83] R. A. Vural and T. Yildirim, "Component value selection for analog active filter using particle swarm optimization," *International Conference* in *Computer and Automation Engineering (ICCAE),* pp. 25-28, *the 2nd* 2010,

[84] R. A. Vural, T. Yildirim, T. Kadioglu and A. Basargan, "Performance evaluation of evolutionary algorithms for optimal filter design," *Evolutionary Computation, IEEE Transactions On,* vol. 16, pp. 135-147, 2012.

[85] Q. Kang, M. Zhou, J. An and Q. Wu, "Swarm intelligence approaches to optimal power flow problem with distributed generator failures in power networks," *Automation Science and Engineering, IEEE Transactions On,* vol. 10, pp. 343-353, 2013.

[86] X. Liang, W. Li, Y. Zhang and M. Zhou, "An adaptive particle swarm optimization method based on clustering," *Soft Computing,* pp. 1-18, 2014.

[87] C. Blum and X. Li, *Swarm Intelligence in Optimization.* Springer, 2008.

[88] A. H. Gandomi, G. J. Yun, X. Yang and S. Talatahari, "Chaos-enhanced accelerated particle swarm optimization," *Communications in Nonlinear Science and Numerical Simulation,* vol. 18, pp. 327-340, 2013.

[89] X. Yang, *Nature-Inspired Metaheuristic Algorithms.* Luniver press, 2010.

[90] I. Fister, X. Yang and J. Brest, "A comprehensive review of firefly algorithms," *Swarm and Evolutionary Computation,* vol. 13, pp. 34-46, 2013.

[91] B. Amiri, L. Hossain, J. W. Crawford and R. T. Wigand, "Community detection in complex networks: Multi–objective enhanced firefly algorithm," *Knowledge-Based Syst.,* vol. 46, pp. 1-11, 2013.

[92] X. Yang, "Multiobjective firefly algorithm for continuous optimization," *Engineering with Computers,* vol. 29, pp. 175-184, 2013.

[93] M. K. Marichelvam, T. Prabaharan and X. S. Yang, "A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems," *Evolutionary Computation, IEEE Transactions On,* vol. 18, pp. 301-305, 2014.

[94] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems,* pp. 142-153, 2005.

[95] X. Yang, "Firefly algorithms for multimodal optimization," in *Stochastic Algorithms: Foundations and Applications*, Springer, pp. 169-178, 2009.

[96] S. Łukasik and S. Żak, "Firefly algorithm for continuous constrained optimization tasks," in *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, Springer, pp. 97-106, 2009.

[97] M. Horng, "Vector quantization using the firefly algorithm for image compression," *Expert Syst. Appl.*, vol. 39, pp. 1078-1091, 2012.

[98] A. Gandomi, X. Yang, S. Talatahari and A. Alavi, "Firefly algorithm with chaos," *Communications in Nonlinear Science and Numerical Simulation*, vol. 18, pp. 89-98, 2013.

[99] A. Kazem, E. Sharifi, F. K. Hussain, M. Saberi and O. K. Hussain, "Support vector regression with chaos-based firefly algorithm for stock market price forecasting," *Applied Soft Computing*, vol. 13, pp. 947-958, 2013.

[100] P. R. Srivatsava, B. Mallikarjun and X. Yang, "Optimal test sequence generation using firefly algorithm," *Swarm and Evolutionary Computation*, vol. 8, pp. 44-53, 2013.

[101] Y. Zhang and L. Wu, "A novel method for rigid image registration based on firefly algorithm," *International Journal of Research and Reviews in Soft and Intelligent Computing (IJRRSIC)*, vol. 2, 2012.

[102] M. A. Zaman and A. Matin, "Nonuniformly spaced linear antenna array design using firefly algorithm," *International Journal of Microwave Science and Technology*, vol. 2012, 2012.

[103] X. Yang, S. S. S. Hosseini and A. H. Gandomi, "Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect," *Applied Soft Computing*, vol. 12, pp. 1180-1186, 2012.

[104] R. Imanirad, X. Yang and J. S. Yeomans, "Modelling-to-generate-alternatives via the firefly algorithm," *Journal of Applied Operational Research*, vol. 5, pp. 14-21, 2013.

[105] S. Gokhale and V. Kale, "An application of a tent map initiated Chaotic Firefly algorithm for optimal overcurrent relay coordination," *International Journal of Electrical Power & Energy Systems*, vol. 78, pp. 336-342, 2016.

[106] X. Yang, "Engineering optimizations via nature-inspired virtual bee algorithms," in *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, Springer, pp. 317-323, 2005.

[107] D. Karaboga, *An Idea Based on Honey Bee Swarm for Numerical Optimization*, 2005.

[108] A. Afshar, O. B. Haddad, M. A. Mariño and B. Adams, "Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation," *Journal of the Franklin Institute*, vol. 344, pp. 452-462, 2007.

[109] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Applied Soft Computing*, vol. 8, pp. 687-697, 2008.

[110] D. Karaboga, B. Gorkemli, C. Ozturk and N. Karaboga, "A comprehensive survey: artificial bee colony (ABC) algorithm and applications," *Artif. Intell. Rev.*, pp. 1-37, 2012.

[111] A. Banharnsakun, T. Achalakul and B. Sirinaovakul, "The best-so-far selection in artificial bee colony algorithm," *Applied Soft Computing,* vol. 11, pp. 2888-2901, 2011.

[112] K. Ziarati, R. Akbari and V. Zeighami, "On the performance of bee algorithms for resource-constrained project scheduling problem," *Applied Soft Computing,* vol. 11, pp. 3720-3733, 2011.

[113] Q. Pan, M. Fatih Tasgetiren, P. N. Suganthan and T. J. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Inf. Sci.,* vol. 181, pp. 2455-2468, 2011.

[114] C. S. Chong, M. Y. H. Low, A. I. Sivakumar and K. L. Gay, "A bee colony optimization algorithm to job shop scheduling," in *Simulation Conference, 2006. WSC 06. Proceedings of the Winter,* pp. 1954-1961, 2006.

[115] S. Chavan and T. S. Khot, "Efficient and Reliable Routing Algorithm to Enhance Connectivity in Disaster Scenario: ABC Algorithm," *International Journal of Science and Research*, vol. 4 issue 5, 2015.

[116] W. Hong, "Electric load forecasting by seasonal recurrent SVR (support vector regression) with chaotic artificial bee colony algorithm," *Energy,* vol. 36, pp. 5568-5578, 2011.

[117] N. Karaboga, "A new design method based on artificial bee colony algorithm for digital IIR filters," *Journal of the Franklin Institute,* vol. 346, pp. 328-348, 2009.

[118] O. Garitselov, S. P. Mohanty, E. Kougianos and P. Patra, "Bee colony inspired metamodeling based fast optimization of a nano-CMOS PLL," *International Symposium* in *Electronic System Design (ISED),* pp. 6-11, *2011*.

[119] V. Manoj and E. Elias, "Artificial bee colony algorithm for the design of multiplier-less nonuniform filter bank transmultiplexer," *Inf. Sci.,* vol. 192, pp. 193-203, 2012.

[120] S. Agrawal and O. Sahu, "Artificial bee colony algorithm to design two-channel quadrature mirror filter banks," *Swarm and Evolutionary Computation,* vol. 21, pp. 24-31, 2015.

[121] S. Kockanat and N. Karaboga, "A novel 2D-ABC adaptive filter algorithm: A comparative study," *Digital Signal Processing,* vol. 40, pp. 140-153, 2015.

[122] Y. Delican, R. Vural and T. Yildirim, "Artificial bee colony optimization based cmos inverter design considering propagation delays," in *Symbolic and Numerical Methods, Modeling and Applications to Circuit Design (SM2ACD), International Workshop On,* 2010, pp. 1-5, *2010*.

[123] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *Control Systems, IEEE,* vol. 22, pp. 52-67, 2002.

[124] B. Mangaraj, I. Misra and S. Sanyal, "Application of bacteria foraging algorithm for the design optimization of multi-objective Yagi-Uda array," *International Journal of RF and Microwave Computer-Aided Engineering,* vol. 21, pp. 25-35, 2011.

[125] B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert and S. S. Sastry, "Distributed control applications within sensor networks," *Proc IEEE,* vol. 91, pp. 1235-1246, 2003.

[126] Y. Zhang, L. Wu and S. Wang, "Bacterial foraging optimization based neural network for short-term load forecasting," *Journal of Computational Information Systems,* vol. 6, pp. 2099-2105, 2010.

[127] D. H. Kim and J. H. Cho, "Intelligent control of AVR system using GA-BF," in *Knowledge-Based Intelligent Information and Engineering Systems,* pp. 854-859, 2005.

[128] D. H. Kim, A. Abraham and J. H. Cho, "A hybrid genetic algorithm and bacterial foraging approach for global optimization," *Inf. Sci.,* vol. 177, pp. 3918-3937, 2007.

[129] M. Tripathy and S. Mishra, "Bacteria foraging-based solution to optimize both real power loss and voltage stability limit," *Power Systems, IEEE Transactions On,* vol. 22, pp. 240-248, 2007.

[130] M. Tripathy, S. Mishra, L. Lai and Q. Zhang, "Transmission loss reduction based on FACTS and bacteria foraging algorithm," in *Parallel Problem Solving from Nature-PPSN* , Springer, pp. 222-231, 2006.

[131] S. Mishra and C. Bhende, "Bacterial foraging technique-based optimized active power filter for load compensation," *Power Delivery, IEEE Transactions On,* vol. 22, pp. 457-465, 2007.

[132] S. Mishra, M. Tripathy and J. Nanda, "Multi-machine power system stabilizer design by rule based bacteria foraging," *Electr. Power Syst. Res.,* vol. 77, pp. 1595-1607, 2007.

[133] E. Ali and S. Abd-Elazim, "Coordinated design of PSSs and TCSC via bacterial swarm optimization algorithm in a multimachine power system," *International Journal of Electrical Power & Energy Systems,* vol. 36, pp. 84-92, 2012.

[134] D. Sumina, N. Bulić and I. Erceg, "Three-dimensional power system stabilizer," *Electr. Power Syst. Res.,* vol. 80, pp. 886-892, 2010.

[135] B. Sumanbabu, S. Mishra, B. Panigrahi and G. K. Venayagamoorthy, "Robust tuning of modern power system stabilizers using bacterial foraging algorithm," *IEEE Congress* in *Evolutionary Computation, CEC 2007.* pp. 2317-2324, 2007.

[136] S. Subramanian and S. Padma, "Bacterial foraging algorithm based parameter estimation of three winding transformer," *Energy and Power Engineering,* vol. 3, pp. 135, 2011.

[137] H. Chen, Y. Zhu and K. Hu, "Multi-colony bacteria foraging optimization with cell-to-cell communication for RFID network planning," *Applied Soft Computing,* vol. 10, pp. 539-547, 2010.

[138] T. Datta, I. S. Misra, B. B. Mangaraj and S. Imtiaj, "Improved adaptive bacteria foraging algorithm in optimization of antenna array for faster convergence," *Progress in Electromagnetics Research C,* vol. 1, pp. 143-157, 2008.

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

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

[141] G. G. Gielen, H. C. Walscharts and W. Sansen, "ISAAC: A symbolic simulator for analog integrated circuits," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1587-1597, 1989.

[142] H. Walscharts, G. Gielen and W. Sansen, "Symbolic simulation of analog circuits in s-and z-domain," in *Circuits and Systems, IEEE International Symposium On*, pp. 814-817, 1989.

[143] G. Gielen, K. Swings and W. Sansen, "An intelligent design system for analogue integrated circuits," in *Proceedings of the Conference on European Design Automation*, pp. 169-173, 1990.

[144] L. Nagel and D. Pederson, "Simulation program with integrated circuit emphasis (SPICE)," *Memorandum ERL-M382.Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA, USA*, 1973.

[145] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," *ERL Memo ERL-M520*, 1975.

[146] R. J. Baker, *CMOS: Circuit Design, Layout, and Simulation.* John Wiley & Sons, 2011.

[147] S. Rubin, "ELECTRIC: An Integrated Aid for Top-Down Electrical Design," *Schlumberger Palo Alto Research*, 1987.

[148] G. W. Zobrist, *Progress in Computer-Aided VLSI Design: Implementations.* Intellect Books, 1990.

[149] *Gpsim*, http://gpsim.sourceforge.net/gpsim.html. Nov 6, 2014.

[150] *DoCircuits*, http://www.docircuits.com/ Nov 6, 2014.

[151] *PartsSim on Circuit Analysis Simulator by Aspen Lab*, http://www.partsim.com/, Nov 6, 2014.

[152] *Simone Software System Description*, http://www.simone.eu/simone-simonesoftware.asp Nov 6, 2014.

[153] *Effortless Schematics; Powerful Simulation*, https://www.circuitlab.com/, Nov 6, 2014.

[154] *An Easier EDA Experience*, http://www.easyeda.com, Nov 6, 2014.

[155] *RF Channels*, http://www.falstad.com/circuit/, Nov 6, 2014.

[156] *Gecko-Simulation*, http://www.gecko-simulations.com/geckocircuits.html, Nov 6, 2014.

[157] *Mixed Mode-Mixed Level Circuit Simulator*,http://ngspice.sourceforge.net/presentation.html, Nov 6, 2014.

[158] *NGSPICE Online*, http://www.ngspice.com, Nov 6, 2014.

[159] *NL5 Circuit Simulator*, http://nl5.sidelinesoft.com/index.php?lang=en, Nov 6, 2014.

[160] *Anasoft SuperSpice*, http://www.anasoft.co.uk/, Nov 6, 2014.

[161] *Simetrix Technologies*, http://www.simetrix.co.uk/ simetrix-simplis.html, Nov 6, 2014.

[162] *Maple (software)*, http://en.wikipedia.org/wikimaple_(software), Nov 6, 2014.

[163] *NI Multisim* , http://en.wikipedia.org/NI_Multisim, Nov 6, 2014.

[164] G. G. Gielen, H. C. Walscharts and W. Sansen, "Analog circuit design optimization based on symbolic simulation and simulated annealing," *Solid-State Circuits, IEEE Journal Of,* vol. 25, pp. 707-713, 1990.

[165] G. Gielen, P. Wambacq and W. M. Sansen, "Symbolic analysis methods and applications for analog circuits: A tutorial overview," *Proc IEEE,* vol. 82, pp. 287-304, 1994.

[166] P. Wambacq, F. Fernández, G. Gielen, W. Sansen and A. Rodríguez-Vázquez, "Efficient symbolic computation of approximated small-signal characteristics of analog integrated circuits," *Solid-State Circuits, IEEE Journal Of,* vol. 30, pp. 327-330, 1995.

[167] Q. Yu and C. Sechen, "A unified approach to the approximate symbolic analysis of large analog integrated circuits," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions On,* vol. 43, pp. 656-669, 1996.

[168] W. Chen and G. Shi, "Implementation of a symbolic circuit simulator for topological network analysis," *IEEE Asia Pacific Conference* in *Circuits and Systems, APCCAS 2006.* pp. 1368-1372, 2006.

[169] B. Rodanski, "Modification of the two-graph method for symbolic analysis of circuits with non-admittance elements," in *International Conference on Signals and Electronic Systems (ICSES–2002).–Wroclaw-Swieradow Zdroj*, pp. 249-254, 2002.

[170] G. Shi, W. Chen and C. R. Shi, "A graph reduction approach to symbolic circuit analysis." in *ASP-DAC,* pp. 197-202, 2007.

[171] C. Shi and X. Tan, "Canonical symbolic analysis of large analog circuits with determinant decision diagrams," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions On,* vol. 19, pp. 1-18, 2000.

[172] X. Tan and C. Shi, "Hierarchical symbolic analysis of analog integrated circuits via determinant decision diagrams," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions On,* vol. 19, pp. 401-412, 2000.

[173] R. Sommer, T. Halfmann and J. Broz, "Automated behavioral modeling and analytical model-order reduction by application of symbolic circuit analysis for multi-physical systems," *Simulation Modelling Practice and Theory,* vol. 16, pp. 1024-1039, 2008.

[174] A. Pakhira, S. Das, I. Pan and S. Das, "Symbolic representation for analog realization of a family of fractional order controller structures via continued fraction expansion," *ISA Trans.*, 2015.

[175] M. Shokouhifar and A. Jalali, "An evolutionary-based methodology for symbolic simplification of analog circuits using genetic algorithm and simulated annealing," *Expert Syst. Appl.*, vol. 42, pp. 1189-1201, 2015.

[176] Z. Erdei, L. A. Dicso, L. Neamt and O. Chiver, "Symbolic equation for linear analog electrical circuits using Matlab," *WSEAS Transactions on Circuits and Systems,* vol. 9, pp. 493-502, 2010.

[177] E. Cheever, *"Symbolic Circuit Analysis in MATLAB (SCAM)," Swarthmore College*. http://www.swarthmore.edu/NatSci/echeeve1/Ref/mna/MNA_All.html, Feb. 2nd, 2014.

[178] J. O. Attia, *Electronics and Circuit Analysis using MATLAB.* CRC press, 2004.

[179] A. Waters, *Active Filter Design.* McGraw-Hill, 1991.

[180] W. G. Jung, *Op Amp Applications Handbook.* Newnes, 2005.

[181] R. L. Geiger and E. Sanchez-Sinencio, "Active filter design using operational transconductance amplifiers: a tutorial," *Circuits and Devices Magazine, IEEE,* vol. 1, pp. 20-32, 1985.

[182] J. Tow, "A step-by-step active-filter design," *Spectrum, IEEE,* vol. 6, pp. 64-68, 1969.

[183] Matlab Basics and Using the Symbolic Editor. Available: https://rophoenixmakerevolution.files.wordpress.com/2015/08/matlab_basics_and_the_symbolic_editor.pdf. (01/08/2015).

[184] (). *Minimizing Component-Variation Sensitivity in Single Op Amp Filters*. Available: http://www.maximintegrated.com/app-notes/index.mvp/id/738, May 22, 2014.

[185] O. J. USHIE, M. ABBOD, E. C. ASHIGWUIKE and S. LAWAN, "Constrained Nonlinear Optimization of Unity Gain Operational Amplifier Filters Using PSO, GA and Nelder-Mead," *Int. J. Intell. Control Syst.,* vol. 20, pp. 26-34, 2015.

[186] G. T. A. Kovacs, " EE113 Course Notes Electronic Circuits, Stanford University, Department of Electrical Engineering," pp. 1-161, 1997.

[187] B. CARTER and R. MANCINI, "Op Amps for Everyone. [Sl]: Newnes," 2009.

[188] O. J. Ushie, M. F. Abbod, and B. E. Usibe, "Genetic Folding/Programming Toolbox: Analogue Circuit Design Case Study," *Journal of Automation & Systems Engineering,* vol. 10, pp. 40-40-64, 2016.

[189] 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.

[190] 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.

[191] 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.

[192] 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.

[193] 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.

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

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

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

[197] 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.

[198] 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.

[199] 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.

[200] 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, .

[201] 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.

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

[203] 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," *IEEE Transactions On Evolutionary Computation,* vol. 1, pp. 109-128, 1997.

[204] 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*, Springer, pp. 185-200, 2008.

Appendix

A Netlist from PSpice and the modified Netlist for MSCAM

PSpice source file for Common-Emitter SSA	Modified file version for MSCAM
V_V1 1 0 DC 0Vdc AC 1Vac	V1 1 0 1
R_R111 1 2 100 TC=0,0	R111 1 2 100
R_R2 0 3 9.6k TC=0,0	R2 0 3 9.6e3
R_R1 0 3 51.2k TC=0,0	R1 0 3 51.2e3
R_Rpi 3 0 1.51k TC=0,0	Rpi 3 0 1.51e3
R_Ro 4 0 58.3k TC=0,0	Ro 4 0 58.3e3
R_RL 5 0 3k TC=0,0	RL 5 0 3e3
C_C111 2 3 1u TC=0,0	C111 2 3 1e-6
C_Cpi 0 3 10p TC=0,0	Cpi 0 3 10e-12
C_Cmu 4 3 2p TC=0,0	Cmu 4 3 2e-12
C_C2 5 4 1u TC=0,0	C2 5 4 1e-6
R_Rc 0 4 2k TC=0,0	Rc 0 4 2e3
C_CL 0 5 3p TC=0,0	CL 0 5 3e-12
I_I4 4 0 DC 69.6e-3*0..713 AC 1Aac	I4 4 0 69.6e-3*0..713

PSpice source file for Example 3 Circuit	Modified file version for MSCAM
R_R111 1 2 1 TC=0,0	OAmp 5 3 4
C_Cpi 0 6 10p TC=0,0	Cmu 6 7 2e-12
R_R5 0 4 3.9k TC=0,0	I7 7 0 VALUE
V_V1 1 0 DC 0Vdc AC 1Vac	Rpi 0 6 1.51e3
G_G1 7 0 VALUE { V(6)*69.6m }	Cpi 0 6 10e-12
R_Rpi 6 0 1.51k TC=0,0	Ro 0 7 58.3e3
R_RL 8 0 1k TC=0,0	R2 3 2 12e3
R_R2 3 2 12k TC=0,0	R3 2 7 100
C_CL 8 7 1n TC=0,0	V1 1 0 1
C_Cmu 7 6 2p TC=0,0	R5 0 4 3.9e3
R_Ro 7 0 58.3k TC=0,0	R4 4 7 8.2e3
R_R4 7 4 8.2k TC=0,0	RL 0 8 1e3
E_U1 5 0 VALUE {LIMIT(V(3,4)*1E6,-15V,+15V)}	R111 1 2 1
C_Cb 5 6 1n TC=0,0	Cb 5 6 1e-9
R_R3 2 7 100 TC=0,0	CL 7 8 1e-9

PSpice source file for Common-Source SSA	Modified file version for MSCAM
C_Cgs 4 3 1p TC=0,0	Cgs 4 3 1e-12
C_C111 2 3 1u TC=0,0	C111 2 3 1e-6
R_Rg1 0 3 1M TC=0,0	Rg1 0 3 1e6
V_V1 1 0 DC 0Vdc AC 1Vac	V1 1 0 1
C_C2 5 4 1u TC=0,0	C2 5 4 1e-6
R_R111 1 2 50K TC=0,0	R111 1 2 50e3
R_Rd 4 0 2.2k TC=0,0	Rd 4 0 2.2e3
R_Rds 4 0 90.086k TC=0,0	Rds 4 0 90.086e3
R_RL 5 0 3k TC=0,0	RL 5 0 3e3
R_Rg2 0 3 2.2M TC=0,0	Rg2 0 3 2.2e6
C_Cpi 0 3 10p TC=0,0	Cpi 0 3 10e-12
C_Cds 0 4 1p TC=0,0	Cds 0 4 1e-12
I_I4 4 0 DC 0.863e-3*2.486 AC 1Aac	I4 4 0 0.863e-3*2.486
C_CL 0 5 3p TC=0,0	CL 0 5 3e-12

PSpice source file for 7thorderchebyshev	Modified file version for MSCAM
E_U1 3 0 VALUE {LIMIT(V(0,2)*1E6,-15V,+15V)}	OAmp1 3 2 0
E_U2 5 0 VALUE {LIMIT(V(0,4)*1E6,-15V,+15V)}	OAmp2 5 4 0
E_U3 7 0 VALUE {LIMIT(V(0,6)*1E6,-15V,+15V)}	OAmp3 7 6 0
R_R111 1 2 16.9k TC=0,0	OAmp4 9 8 0
R_R2 3 4 16.9k TC=0,0	OAmp5 11 10 0
R_R3 5 6 10k TC=0,0	OAmp6 13 12 0
R_R4 2 3 95.3k TC=0,0	OAmp7 15 14 0
R_R5 6 7 10k TC=0,0	OAmp8 17 16 0
R_R6 2 7 16.9k TC=0,0	OAmp9 19 18 0
V_V1 1 0 DC 0Vdc AC 1Vac	OAmp10 21 20 0
C_C1 2 3 1200p TC=0,0	R111 1 2 16.9e3
C_C2 4 5 1200p TC=0,0	R2 3 4 16.9e3
C_C4 8 9 1500p TC=0,0	R3 5 6 10e3
R_R10 8 9 26.7k TC=0,0	R4 2 3 95.3e3
R_R9 11 12 10k TC=0,0	R5 6 7 10e3
C_C3 10 11 1500p TC=0,0	C1 2 3 1200e-12
R_R12 8 13 16.2k TC=0,0	C2 4 5 1200e-12
E_U4 9 0 VALUE {LIMIT(V(0,8)*1E6,-15V,+15V)}	R6 2 7 16.9e3
R_R7 5 8 16.2k TC=0,0	C3 8 9 1500e-12
R_R11 12 13 10k TC=0,0	R10 8 9 26.7e3
E_U5 11 0 VALUE {LIMIT(V(0,10)*1E6,-15V,+15V)}	R11 12 13 10e3
R_R8 9 10 16.2k TC=0,0	R12 8 13 16.2e3
E_U6 13 0 VALUE {LIMIT(V(0,12)*1E6,-15V,+15V)}	R9 11 12 10e3
E_U7 15 0 VALUE {LIMIT(V(0,14)*1E6,-15V,+15V)}	R8 9 10 16.2e3
R_R16 14 15 12.7k TC=0,0	C4 10 11 1500e-12
R_R14 15 16 16.2k TC=0,0	R7 5 8 16.2e3
C_C5 16 17 2200p TC=0,0	R18 14 19 16.2e3
E_U9 19 0 VALUE {LIMIT(V(0,18)*1E6,-15V,+15V)}	R17 18 19 10e3
R_R15 17 18 10k TC=0,0	C5 14 15 2200e-12
E_U8 17 0 VALUE {LIMIT(V(0,16)*1E6,-15V,+15V)}	R14 15 16 16.2e3
R_R18 14 19 16.2k TC=0,0	R16 14 15 12.7e3
R_R17 18 19 10k TC=0,0	C6 16 17 2200e-12
R_R13 14 11 16.2k TC=0,0	R15 17 18 10e3
C_C6 14 15 2200p TC=0,0	R13 11 14 16.2e3
E_U10 21 0 VALUE {LIMIT(V(0,20)*1E6,-15V,+15V)}	R20 20 21 42.2e3
R_R20 20 21 42.2k TC=0,0	C7 20 21 1200e-12
R_R19 20 17 42.2k TC=0,0	R19 17 20 42.2e3
C_C7 20 21 1200p TC=0,0	V1 1 0 1

B Code for transforming Netlist to matrices.

```
%This program takes a netlist (similar to SPICE), parses it to derive the

%circuit equations, then solves them symbolically.
syms V1 I2 R111 R112 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16 R17 R18 R19 R20
        R21 R22 R23 C1 C2 C3 C4 C5 C6 C7 Av s numO J C111 L111 R999 ...
    v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_10 v_11 v_12 v_13 v_14 v_15 v_16 v_17 v_18 v_19 v_20 v_21
fname='7thorderchebyshev.cir';
[Name, N1, N2, arg3]=textread(fname,'%s %s %s %s ');
```

```
tic % start a stopwatch timer
%Initialize
numElem=0;  %Number of passive elements.
numV=0;     %Number of independent voltage sources
numO=0;     %Number of op amps
numI=0;     %Number of independent current sources
numI2=0;
numNode=0;  %Number of nodes, not including ground (node 0).

%Parse the input file
for i=1:length(Name),
   switch(Name{i}(1)),
      case {'R','L','C' 'R111','L111','C111'},
         numElem=numElem+1;
         Element(numElem).Name=Name{i};
         Element(numElem).Node1=str2num(N1{i});
         Element(numElem).Node2=str2num(N2{i});
         try
            Element(numElem).Value=str2num(arg3{i});
         catch
            Element(numElem).Value=nan;
         end
      case 'V',
         numV=numV+1;
         Vsource(numV).Name=Name{i};
         Vsource(numV).Node1=str2num(N1{i});
         Vsource(numV).Node2=str2num(N2{i});
         try
            Vsource(numV).Value=str2num(arg3{i});
         catch
            Vsource(numV).Value=nan;
         end
         case 'O',
         numO=numO+1;
         Opamp(numO).Name=Name{i};
         Opamp(numO).Node1=str2num(N1{i});
         Opamp(numO).Node2=str2num(N2{i});
         Opamp(numO).Node3=str2num(arg3{i});
      case 'I'
         numI=numI+1;
         Isource(numI).Name=Name{i};
         Isource(numI).Node1=str2num(N1{i});
         Isource(numI).Node2=str2num(N2{i});
         try
            Isource(numI).Value=str2num(arg3{i});
         catch
            Isource(numI).Value=nan;
         end
   end
   numNode=max(str2num(N1{i}),max(str2num(N2{i}),numNode));
end

%Preallocate all of the cell arrays ##################################
G=cell(numNode,numNode);
V=cell(numNode,1);
I=cell(numNode,1);
I2=cell(numNode,1);
if ((numV)~=0),
   E=cell(numNode,1);
```

```
end

%Fill the G matrix ###################################################
%Initially, make the G Matrix all zeros.
[G{:}]=deal('0');

%Now fill the G matrix with conductances from netlist
for i=1:numElem,
   n1=Element(i).Node1;
   n2=Element(i).Node2;
   %Make up a string with the conductance of current element.
   switch(Element(i).Name(1)),
      case 'R',
         g = ['1/' Element(i).Name];

      case 'L',
         g = ['1/s/' Element(i).Name];
      case 'C',
         g = ['s*' Element(i).Name];
         case 'R111',
         g = ['1/' Element(i).Name];

      case 'L111',
         g = ['1/s/' Element(i).Name];
      case 'C111',
         g = ['s*' Element(i).Name];
    end

   %If neither side of the element is connected to ground
   %then subtract it from appropriate location in matrix.
   if (n1~=0) & (n2~=0),
      G{n1,n2}=[ G{n1,n2} '-' g];
      G{n2,n1}=[ G{n2,n1} '-' g];
   end

   %If node 1 is connected to graound, add element to diagonal
   %of matrix.
   if (n1~=0),
      G{n1,n1}=[ G{n1,n1} '-' g];
   end
   %Ditto for node 2.
   if (n2~=0),
      G{n2,n2}=[ G{n2,n2} '+' g];
   end

   %Go to next element.
   %     i=i+4;
end
%The G matrix is finished ---------------------------------------

%Fill the V matrix ###################################################
for i=1:numNode,
   V{i}=['v_' num2str(i)];
end
%The V matrix is finished ---------------------------------------

%Add each opamp output to the list of symbolic variables.
% for i=1:numO,
```

```
%     SymString=[SymString J{i+numV} ' '];
% end

%Fill the I matrix ##################################################
[I{:}]=deal('0');
for j=1:numNode,
   for i=1:numI,
      if (Isource(i).Node1==j),
         I{j}=[I{j} '-' Isource(i).Name];
      elseif (Isource(i).Node2==j),
         I{j}=[I{j} '+' Isource(i).Name];
      end
   end
end
%The I matrix is done ----------------------------------------------

[E{:}]=deal('0');
for j=1:numNode,
   for i=1:numV,
      if (Vsource(i).Node1==j),
         E{j}= [E{j} '+' Vsource(i).Name];
      elseif (Vsource(i).Node2==j),
         E{j}= [E{j} '-' -Vsource(i).Name];
      end
   end
end

%The I matrix is done ----------------------------------------------

[I2{:}]=deal('0');
   for j = 1: numNode
   for i=1:length(Name),
        % ismember('R111',Name) mean R111 is member of name while ~ attached to ismember mean not
          member
         if ismember('R111',Name) & ~ismember('C111',Name) & ~ismember('L111',Name)
         R999=R111;
         elseif ~ismember('R111',Name) & ismember('C111',Name) & ~ismember('L111',Name)
         R999=1/(s*C11i);
         elseif ~ismember('R111',Name) & ~ismember('C111',Name) & ismember('L111',Name)
         R999=(s*L111);
         elseif ismember('R111',Name) & ismember('C111',Name) & ~ismember('L111',Name)
         R999=(R111 + 1/(s*C111));
         elseif ismember('R111',Name) & ~ismember('C111',Name) & ismember('L111',Name)
         R999=(R111+(s*L111));
         elseif ~ismember('R111',Name) & ismember('C111',Name) & ismember('L111',Name)
         R999=((s*L111)+(1/(s*C111)));
         elseif ismember('R111',Name) & ismember('C111',Name) & ismember('L111',Name)
         R999=(R111+(s*L111)+(1/(s*C111)));
         elseif ~ismember('R111',Name) & ~ismember('C111',Name) & ~ismember('L111',Name)
         R999=0;
         end
   end
  I2=E/R999;
   end

%      I3 = char(I) + str(I2);
```

```
%Form the A, X, and Z matrices (As cell arrays of strings).
   Acell=[deal(G)];
   Xcell=[deal(V)];
   Zcell=[deal(I)];
%     Z2cell=[deal(I2)];

   %Create assignments for three arrays
Astring='A=[';
Xstring='X=[';
Zstring='Z=[';
Z2string='Z2=[';

for i=1:length(Acell),     %for each row in the arrays.
   for j=1:length(Acell),      %for each column in matrix A.
      Astring=[Astring ' ' Acell{i,j}]; %Get element from Acell
   end
   Astring=[Astring ';'];          %Mark end of row with semicolon
   Xstring=[Xstring Xcell{i} ';'];    %Enter element into array X;
   Zstring=[Zstring Zcell{i} ';'];    %Enter element into array Z;
%     Z2string=[Z2string Z2cell{j} ';']; %Enter element into array Z2;
end
Astring=[Astring '];'];  %Close array assignment.
Xstring=[Xstring '];'];
Zstring=[Zstring '];'];
% Z2string=[Z2string '];'];

%Evaluate strings with array assignments.
eval([Astring ' ' Xstring ' ' Zstring])

% A(4,4) =  A(4,4)+1;
% A(4,3) =  A(4,3)+Av;
% A(4,2) =  A(4,2)-Av;
%
% A(8,8) =  A(8,8)+1;
% A(8,7) =  A(8,7)+Av;
% A(8,6) =  A(8,6)-Av;
% if numO ~=0,

if numO ~=0,
   for i=1:numO,
         ss=char(N1(i,1)); % conversion to character (string)and opamp is being arranged fist from first to last
         numNode=str2num(ss); % conversion from character (string) to number
         A(numNode,numNode)= A(numNode,numNode)+1; % add 1 to o/p node voltage of opamp (ie Vout of
           opamp has coefficient of 1)
         yy=char(N2(i,1));
         zz=str2num(yy);
         uu=char(arg3(i,1)); % Negative i/p of opamp (N2, N2,arg3 respectively arrangement for o/p, +ve, and -
           ve i/p of opamp)
         vv=str2num(uu);
         if zz==0 | zz==numNode; % zz=numNode means if the is direct feedback from opamp o/p to any i/p of
           opamp
         A(numNode,numNode-1) =  A(numNode,numNode-1)+Av; % add Av to negative i/p voltage (V-ve) of
           opamp
         elseif vv==0 | vv==numNode;
         A(numNode,numNode-1) =  A(numNode,numNode-1)-Av; % add Av to positive i/p voltage (V+ve) of
           opamp
         else
         A(numNode,numNode-1) =  A(numNode,numNode-1)+Av;
```

```
            A(numNode,numNode-2) =  A(numNode,numNode-2)-Av; % add Av to positive i/p voltage (V+ve) of
              opamp
            end
        end
    end
I3 = Z + I2;
    disp(A);
    disp(X);
    disp(I3);
```