A picture containing text, queen

Description automatically generated

**UNIVERSITY OF NAIROBI**

**FACULTY OF ENGINEERING**

**DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING**

**FINAL YEAR PROJECT REPORT**

**PROJECT INDEX: PRJ 111**

A CHAOTIC MULTI-OBJECTIVE RUNGE-KUTTA OPTIMIZATION ALGORITHM FOR OPTIMIZED CIRCUIT DESIGN

STUDENT NAME: **OWEN MOGAKA NYANDIEKA**

REGISTRATION NUMBER: **F17/1360/2018**

SUPERVISOR(S): **DR. D.R. SEGERA**

EXAMINER(S): **DR. C.R. KIRUKI**

This project report is submitted in partial fulfillment of the requirement for the award of the Degree of Bachelor of Science in Electrical and Electronics Engineering from the University of Nairobi.

Submitted on the 9th of June 2023

# **DECLARATION OF ORIGINALITY**

**NAME OF STUDENT:** OWEN MOGAKA NYANDIEKA

**REGISTRATION NUMBER:** F17/1360/2018

**SCHOOL/FACULTY:** ENGINEERING

**DEPARTMENT:** ELECTRICAL AND INFORMATION ENGINEERING

**COURSE NAME**: BACHELOR OF SCIENCE IN ELECTRICAL AND ELECTRONICS ENGINEERING

**PROJECT TITLE:** A CHAOTIC MULTI-OBJECTIVE RUNGE-KUTTA OPTIMIZATION ALGORITHM FOR OPTIMIZED CIRCUIT DESIGN

1. I understand what plagiarism is, and I am aware of the university policy in this regard.
2. I declare that this final year project report is my original work and has not been submitted elsewhere for examination, award of a degree, or publication. Where other people's work has been used, this has properly been acknowledged and referenced in accordance with the University of Nairobi's requirements.
3. I have not sought or used the services of any professional agencies to produce this work.
4. I have not allowed and shall not allow anyone to copy my work with the intention of passing it off as his/her own work.
5. I understand that any false claim in respect of this work shall result in disciplinary action in accordance with the University's anti-plagiarism policy.

Signature: ………...……………………. Date: ………….……………………

Approved by:

Supervisor: Dr. D.R. Segera, University of Nairobi.

Signature: ……………………………… Date: ………………………………...

# **DEDICATION**

To my family and friends, I dedicate this work to you as a thank you for your love and support throughout my academic years.

# **ACKNOWLEDGEMENT**

I would like to thank my supervisor Dr. Davies Segera whose insights and feedback have helped me to shape the direction and quality of my work.

I would also like to express my gratitude to my project group members for their support, intellectual discussions, and encouragement during challenging times.

I am deeply indebted to my family for their unwavering moral and financial support, encouragement, and belief in me to deliver the highest quality of work in this project.

Special thanks to my friends for their unyielding support, constructive criticism, and valuable suggestions that enriched the outcomes of this study.

**TABLE OF CONTENTS**

[DECLARATION OF ORIGINALITY i](#_Toc137133128)

[DEDICATION ii](#_Toc137133129)

[ACKNOWLEDGEMENT iii](#_Toc137133130)

[LIST OF ABBREVIATIONS vi](#_Toc137133131)

[LIST OF FIGURES vii](#_Toc137133132)

[LIST OF TABLES viii](#_Toc137133133)

[ABSTRACT ix](#_Toc137133134)

[CHAPTER 1: INTRODUCTION 1](#_Toc137133135)

[1.1 Background 1](#_Toc137133136)

[1.2 Statement of the problem 3](#_Toc137133137)

[1.3 Objectives 4](#_Toc137133138)

[1.4 Research Questions 4](#_Toc137133139)

[1.5 Significance of the study 5](#_Toc137133140)

[1.6 Scope of the study 5](#_Toc137133141)

[CHAPTER 2: LITERATURE REVIEW 6](#_Toc137133142)

[2.1 Related Work 6](#_Toc137133143)

[2.2 The Runge-Kutta Optimization Algorithm (RUN) 8](#_Toc137133144)

[2.2.1 Inspiration Behind The Runge Kutta Optimization Algorithm 8](#_Toc137133145)

[2.2.2 Optimization Steps of the RUN 10](#_Toc137133146)

[2.2.3 Limitations of Runge-Kutta Optimizer 20](#_Toc137133147)

[CHAPTER 3: THEORETICAL BACKGROUND 22](#_Toc137133148)

[3.1 Chaotic Maps 22](#_Toc137133149)

[3.2 Multi-objective Optimization Techniques 24](#_Toc137133150)

[3.2.1 Weighted Sum Method 24](#_Toc137133151)

[3.2.2 Scalarizing Method 25](#_Toc137133152)

[3.2.3 Pareto Optimization 25](#_Toc137133153)

[3.2.4 Evolutionary Algorithms 25](#_Toc137133154)

[3.2.5 Multi-objective Gradient Descent Methods 26](#_Toc137133155)

[3.2.6 Decomposition-based Methods 26](#_Toc137133156)

[3.3 Optimized Circuit Design 26](#_Toc137133157)

[CHAPTER 4: METHODOLOGY 31](#_Toc137133158)

[4.1 Multi-objective Chaotic Runge-Kutta Optimization Algorithm (CMRUN) 31](#_Toc137133159)

[4.1.1 Selecting what part of the base RUN to improve by chaos 31](#_Toc137133160)

[4.1.2 Chaotic Multi-objective Runge-Kutta Optimizer (CMRUN) 34](#_Toc137133161)

[4.2 Evaluation of the algorithm’s performance 35](#_Toc137133162)

[4.3 Circuit design 37](#_Toc137133163)

[CHAPTER 5: RESULTS AND ANALYSIS 39](#_Toc137133164)

[5.1 Benchmark test results of the Chaotic Multi-objective Runge-Kutta Optimization Algorithm 39](#_Toc137133165)

[5.2 Optimized Circuit Design 43](#_Toc137133166)

[CHAPTER 6: CONCLUSION AND RECOMMENDATION 48](#_Toc137133167)

[Conclusion 48](#_Toc137133168)

[Limitations of the design 49](#_Toc137133169)

[Recommendations 49](#_Toc137133170)

[REFERENCES 50](#_Toc137133171)

[APPENDIX 53](#_Toc137133172)

[MATLAB CODES 53](#_Toc137133173)

[I. CMRUN 53](#_Toc137133174)

[II. Circuit Design Simulation 55](#_Toc137133175)

[III. CMRUN Optimization Function 56](#_Toc137133176)

# **LIST OF ABBREVIATIONS**

RUN - Runge-Kutta Optimizer

IEC - International Electro-technical Commission

MOA - Metaheuristic Optimization Algorithm

PBA - Physics-Based Algorithm

EA - Evolutionary Algorithm

SIA - Swarm Intelligence Algorithm

FDB - Fitness Distance Balance

LEO - Local Escaping Operator

CRUN - Chaotic Runge-Kutta Optimization Algorithm

SM - Searching Mechanism

ESQ - Enhanced Solution Quality

MOP - Multi-objective Optimization Problem

CMRUN - Chaotic Multi-objective Runge-Kutta Optimization Algorithm

NA - Not applicable

Avg - Average

Std - Standard Deviation

# **LIST OF FIGURES**

[Figure 1 General Flow Chart for Metaheuristic Algorithm 7](#_Toc137039590)

[Figure 2 Slopes Utilized in the Runge-Kutta Method 10](#_Toc137039591)

[Figure 3 Slopes used to determine x\_(n+1) in the RUN algorithm 14](#_Toc137039592)

[Figure 4 The Search Mechanism of the RUN 16](#_Toc137039593)

[Figure 5 Flowchart of the Run Algorithm 19](#_Toc137039594)

[Figure 6 Matlab implementation of chaotic maps 24](#_Toc137039595)

[Figure 7 Single-stage amplifier circuit 29](#_Toc137039596)

# **LIST OF TABLES**

[Table 1 RUN Pseudocode 19](#_Toc137039774)

[Table 2 List of chaotic maps 23](#_Toc137039775)

[Table 3 CMRUN Pseudocode 35](#_Toc137039776)

[Table 4 Benchmark Test Functions 37](#_Toc137039777)

[Table 5 Amplifier Circuit Parameters 38](#_Toc137039778)

[Table 6 Comparison results of the 10 chaotic maps using 15 benchmark test functions 39](#_Toc137039779)

[Table 7 Statistical results of the Benchmark test functions from RUN and 11 other multi-objective optimization algorithms 40](#_Toc137039780)

[Table 8 CMRUN Performance Rank 41](#_Toc137039781)

[Table 9 Comparison results of the 10 chaotic maps for optimized circuit design 43](#_Toc137039782)

[Table 10 Results for Optimized Circuit Design 43](#_Toc137039783)

[Table 11 Optimized design variables for a single-stage amplifier using CMRUN 44](#_Toc137039784)

# **ABSTRACT**

Circuit design is an essential skill in engineering as it leads to the creation of reliable, efficient, and cost-effective electronic devices. Circuit design involves assembling and interconnecting several circuit elements to perform a specific objective function. Over the years, there has been an increase in using multi-objective optimizers in circuit design. However, metaphor-based traditional optimization algorithms mimic animal searching trends that offer little contribution to the optimization process. These algorithms may be unable to handle complex circuit design problems as they suffer from bias and premature convergence. To address these problems, this project proposes a chaotic multi-objective Runge-Kutta algorithm for optimized circuit design based on the novel metaheuristic optimization algorithm, the Runge-Kutta optimizer (RUN). The base RUN is modified to handle multiple objectives, and 10 chaotic maps are integrated to introduce chaos in the multi-objective RUN. The proposed algorithm uses chaotic maps to avoid local optima, handles non-linear and non-convex optimization problems, and can handle multiple circuit design objectives. The chaotic multi-objective Runge-Kutta optimization algorithm is compared with 11 other metaheuristic algorithms using 15 benchmark test functions. The algorithm is then applied in the optimization of a designed circuit to determine its superiority in optimizing real-world problems.

# **CHAPTER 1: INTRODUCTION**

## **1.1** **Background**

Designing analog circuits can be frustrating due to the many constraints to be attained for a circuit to function correctly. An engineer's goal is to develop a circuit that meets certain specifications and the International Electro-technical Commission (IEC) standards [9]. For most integrated circuits, the design of the analog part of the overall circuit takes the more significant portion of the overall design time. This is because analog circuit design involves tuning the circuit parameters through trial and error until the required output is met. This process is time-consuming and may not produce the desired results.

Analog circuit design optimization refers to improving the performance of an analog circuit by adjusting its parameters or components to meet specific performance goals, such as increased accuracy, stability, power efficiency, or reduced noise[14]. This process involves using mathematical models, simulation tools, and optimization algorithms to identify the best combination of circuit components and parameters to achieve the desired performance characteristics. Optimization may be iterative, with multiple design iterations being evaluated until the desired performance criteria are met [14].

Circuit design requires the engineer to know the components used in the design. The following are the most commonly used components by students in laboratory exercises. The first are capacitors of two types: Decoupling capacitors are added in parallel with the power supply to compensate for voltage drops or spikes. Coupling Capacitors are capacitors used in amplifier circuits to filter out low-frequency noise, which is useful in blocking dc signals.The second are resistors useful when designing circuits that implement digital ICs. The ICs contain logic levels 1 and 0. For example, in a CMOS with +5V as Vcc, the range of 2.2V to 5V will be interpreted as logic 1, and the 0V to 0.8V will be logic 0. This means voltages 0.9V to 2.7V will be indeterminate. Pull-up resistors are used to fix the voltage close to Vcc, and pull-down resistors are used to pull the voltage close to GND to avoid the indeterminate regions [12].

The third are transistors which are the most used components in electronics. They are used as switches or amplifiers depending on the needs of a circuit. Sometimes transistors are interconnected to form a Darlington pair to increase the gain. Transistors get easily burnt when the voltage supplied exceeds the transistor ratings, and thus the engineer has to consider each transistor's ratings from the datasheet. The last are microcontrollers, which help engineers avoid over-killing the circuit design using too many components. Thus, microcontroller knowledge is needed to combine the circuit's suitable digital and analog parts. Several other components are widely used in circuit design, but the aforementioned components are the most common ones. Integrating the above components requires using theoretical formulas to compute their power requirements.

Generally, circuit design involves three steps: topology selection, component sizing, and layout generation. During topology selection, power loss, power gain, current, temperature, circuit stability, and noise are specified, and constraints are set. Component sizing ensures the circuit design is not bulky and uses the fewest number of components possible to avoid redundancy [9]. Many circuit simulation software, such as Autodesk Eagle, NI Multisim, and LTSpice, allow testing of the circuit parameters to see if the desired results are obtained. This is the most widely used method that involves iterative trial-and-error until the parameters fit. Since the above process is time-consuming, there have been several efforts to develop optimization tools to speed up analog circuit design. However, some optimization tools developed use gradient search methods that utilize initial guesses. Furthermore, these tools struggle with discrete variables and non-linear constraints and get stuck in local minima, unable to use the exploration phase fully [9].

Therefore, there is a need for more robust and efficient optimization techniques for circuit design. Iman Ahmadianfar et al. proposed an optimization problem that does not use metaphors but instead uses mathematics to optimize based on this Runge-Kutta mathematical method. The Runge Kutta optimizer (RUN) uses the derivative part of the 4th-order Runge Kutta equation to explore the search space. It further employs an enhanced solution quality (ESQ) to improve the convergence speed of the search and evade the best local solutions [5]. Generally, optimization problems are presented as single-objective, multi-objective, robust, memetic, or large-scale. Metaheuristic optimization algorithms are the most widely used optimizers in solving real-world problems.

Metaheuristics are simply guidelines used to establish rules when solving optimization problems. These rules are referred to as heuristics and are the basic foundations of algorithms. Metaheuristic optimization algorithms (MOAs) are divided into physics-based, evolutionary, and swarm-based algorithms. Despite their high performance in producing optimal solutions, MOAs experience several limitations, such as increased sensitivity and difficulty setting control parameters. The main problem for all MOAs is balancing the exploration and exploitation phases for optimum performance [5]. The exploration phase involves obtaining the global optimal solutions, while the exploitation phase is when the algorithm searches for local optimal solutions. Algorithms stuck in local optima experience premature convergence leading to inaccurate results. Convergence is the rate at which an algorithm converges at the global optimum solution. MOAs are schostatic since they use randomly generated components in their work; hence, achieving the appropriate balance between the two phases is difficult [5]. Most researchers try to improve this balance by combining MOAs with other optimizers or choosing control parameters appropriately.

Iman Ahmadianfar et al. proposed the RUN algorithm as a more robust alternative to different optimization algorithms. This algorithm is divided into two parts that utilize slope calculations using the Runge-Kutta method to provide a more powerful search space for optimization. The first part is the search mechanism using the Runge Kutta method followed by the enhanced solution quality, which improves the algorithm's efficiency by producing a more quality solution than the initially obtained solutions [5]. However, the RUN algorithm is single-objective, and circuit design involves optimizing multiple objectives to increase efficiency and minimize cost. To address the limitation of the traditional Runge-Kutta algorithm, multi-objective optimization techniques have been proposed for circuit design which aims to optimize circuits for multiple performance metrics simultaneously [1]. Furthermore, chaos is introduced to the multi-objective RUN to improve the exploitation and exploration phases and, thus, a high convergence rate.

## **1.2 Statement of the problem**

Circuit design involves dealing with multimodal, non-linear, and non-convex problems. Due to these characteristics, traditional optimization algorithms, such as gradient-based methods, cannot efficiently optimize circuit design. Genetic algorithms have long been used as a more robust method of optimizing circuit design, but even then, there have been challenges with premature convergence. This is also true for swarm-based algorithms such as the particle swarm optimizer. These metaphor-based algorithms face difficulties in balancing the local and global search phases. Therefore, many researchers have attempted to improve their performances by hybridizing with other optimization algorithms to create an appropriate trade-off between the exploitation and exploration phases [2]. Despite this, a more consistent and robust algorithm that can work efficiently is needed to optimize complex real-world problems. Iman Ahmadianfar et al. proposed the RUN algorithm, which provided superior performance in exploitation, exploration, and convergence compared to other metaheuristic algorithms. However, the RUN algorithm can only handle a single objective. Most optimization problems are multi-objective; thus, the RUN can be made multi-objective to handle such problems. Furthermore, introducing chaos to the multi-objective RUN will improve its convergence rate.

## **1.3 Objectives**

The main objective of this project is to improve the Runge-Kutta Optimizer to become chaotic and multi-objective and apply the improved version in the optimization of circuit design. The main objective is divided into the following objectives:

* To develop a chaotic multi-objective Runge-Kutta algorithm and test its performance against the traditional optimization techniques on 15 benchmark functions.
* To design a circuit with multiple objective functions that are to be optimized.
* To apply the chaotic multi-objective algorithm to the proposed circuit design and optimize the circuit parameters based on multiple objectives.

## **1.4 Research Questions**

The objectives stated above can be achieved by asking the following questions:

* What are chaotic maps, and can they be used to enhance the performance of the RUN optimizer?
* What is multi-objective optimization, and how can the RUN algorithm be made multi-objective?
* Can a chaotic multi-objective Runge-Kutta algorithm be developed for optimized circuit design to handle multiple objectives and avoid premature convergence?
* How does the suggested algorithm perform in terms of convergence, solution diversity, and ability to optimize complex problems compared to already existing ones, such as gradient-based methods and genetic algorithms?

## **1.5 Significance of the study**

Most researchers have used the hybridization of multiple algorithms to solve multi-objective problems characterized by multimodality, nonlinearity, and non-differentiability. However, this method fails to create a proper trade-off between local and global searches. Creating a robust algorithm that provides more accurate and consistent results that are not trapped in local optima is challenging. The chaotic multi-objective Runge-Kutta algorithm is proposed to fill this gap. This new approach intends to improve circuit design optimization since it is an essential part of modern technology as it plays a vital role in developing electronics such as computers and medical equipment. Circuit design often involves multiple design objectives, such as power consumption, speed, and noise. Multi-objective optimization techniques can help to balance these objectives and find optimal solutions. Chaotic maps have shown promise as optimization tools because they generate diverse solutions and avoid local optima. The proposed project aims to combine the strengths of chaotic maps and the Runge-Kutta algorithm to develop a new optimization algorithm for circuit design. The results of this project could lead to the development of better-performing electronic devices with improved power consumption, speed, and noise.

## **1.6 Scope of the study**

The proposed project's scope includes the development, implementation, and testing of a chaotic multi-objective Runge-Kutta algorithm for optimized circuit design. The key areas in this project are:

* The development of a new optimization algorithm using chaotic maps and the RUN algorithm that can handle multiple objectives and avoid local optima.
* The proposed algorithm will be implemented by testing it on 15 benchmark functions to evaluate its ability to handle multiple objectives, diversity of solutions, and convergence speed.
* The proposed algorithm will be used to optimize circuit design to demonstrate its usefulness in solving real-world problems. The algorithm will optimize circuit parameters based on the optimal circuit performance metrics such as noise and stability.

# **CHAPTER 2: LITERATURE REVIEW**

This chapter reviews the existing literature on the Runge-Kutta optimization algorithm (RUN) and the mathematical formulation behind its creation.

This chapter also gives a review of the existing work on the improvement of the RUN algorithm.

## **2.1 Related Work**

The RUN is a fairly new optimizer created in 2021. This algorithm has presented superior performance to traditional metaphor-based algorithms, and thus several researchers have proposed ways to improve its performance even more. The RUN is a stochastic population-based algorithm [3]. At each iteration, population-based algorithms randomly generate a set of solutions as opposed to single-based optimizers to avoid being trapped in local optima. They are, therefore, superior to single-based algorithms as they have increased convergence speed. The solutions can share information hence convenient search in complex feature space sceneries.

The RUN algorithm edges nature-inspired algorithms in performance. Generally, metaheuristic algorithms are divided into three: evolutionary algorithms (EAs), swarm intelligence algorithms (SIAs), and physics-based algorithms (PBAs) [5]. These algorithms have shown a high capability at obtaining optimum solutions at considerable speed while avoiding getting trapped in local optima. However, when solving complex design problems, they tend to suffer from premature convergence and mismatch between the design variables. Over the years, researchers developed hybrid optimizers to fix the issues encountered by metaheuristics by combining the domineering features of individual algorithms. An example is the hybrid of the Whale Optimization Algorithm and the Grey Wolf Optimizer, which showed better performance than the individual algorithms [2].

The RUN algorithm was designed to be a powerful and more accurate optimizer that avoids local optima. The RUN finds the global best in solving optimization problems by using the Enhanced Solution Quality to increase the convergence speed and avoid premature convergence [5]. Although the RUN performs better than other metaheuristic algorithms, it suffers limitations when solving multimodal problems. This is because most metaheuristics tend to get trapped in local optima. Enes CENGİZ et al. proposed a method to improve the RUN algorithm using the FDB (Fitness Distance Balance) method [4]. In the RUN algorithm, local minima traps can occur during execution; hence the global optimal solutions are not attained. Thus the FDB method is proposed to enhance the exploration phase guiding the algorithm toward global solutions.

R. Manjula Devi et al. proposed another way to improve the RUN algorithm to avoid premature convergence and enhance the accuracy of solutions. They proposed integrating a Local Escaping Operator (LEO) in the RUN [11]. The LEO improves it by bypassing local minima and thus increasing its convergence. The above methods improve the performance of the RUN algorithm, but the most widely used method to improve metaheuristic algorithms is enhancing them using chaotic maps.

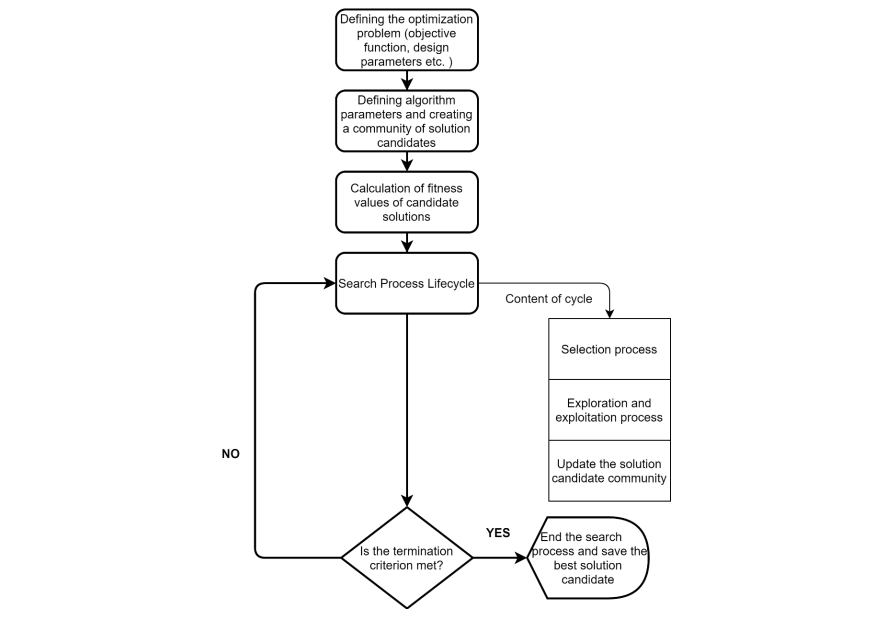


Figure 1 General Flow Chart for Metaheuristic Algorithm

Several optimizers have been improved by introducing chaos, such as the chaotic Mayfly algorithm, the chaotic whale optimizer, and the chaotic bat algorithm [10], [17],[18], [19], [20]. Several metaheuristic algorithms' performance and quality parameters have been tested by applying them to real-world design problems. For example, the chaotic Harris Hawks optimizer [17] developed by Livatyali and Gezici provided better performance and results compared to the traditional Harris Hawks optimizer. Therefore, from previous research, Betül S. Yıldız et al. proposed the use of chaos to enhance the base RUN by making it chaotic (CRUN) to increase its ability to avoid local optima [3]. The CRUN was applied to real-world design problems and showed superior performance regarding the diversity of solutions and convergence speed.

## **2.2 The Runge-Kutta Optimization Algorithm (RUN)**

### **2.2.1 Inspiration Behind The Runge Kutta Optimization Algorithm**

The Runge Kutta method is one of the techniques in numerical methods used to find solutions for first-order ordinary differential equations:

(1)

The slope of the line of best fit at point is defined as in the equation above, giving the main idea of what the algorithm will base its search space on.

The Runge Kutta method can be derived using the Taylor series and ignoring the higher-order terms, as shown below.

(2)

The equation below is the approximate solution for the Taylor series derivation of the Runge Kutta method after dropping the higher-order terms.

(3)

From equation 3, we can rewrite the equation as shown below to begin finding the formula for the Runge Kutta fourth-order method.

(4)

where

The first-order derivative, which is equivalent to as seen above, can be determined as shown in the expression below:

Therefore, we can use equation (5) to rewrite equation (4) as shown below:

The solutions using this method are formulated as shown below:

where,

From the equations above, uses to give the slope at the beginning of the interval []. is defined by the midpoint slope using and [5]. is defined by the midpoint slope using and , and lastly, is determined by the pitch at the end using and . Remember, from the Runge-Kutta method, the value is given by the initial value and the weighted averages of and . This is depicted in Figure 2.

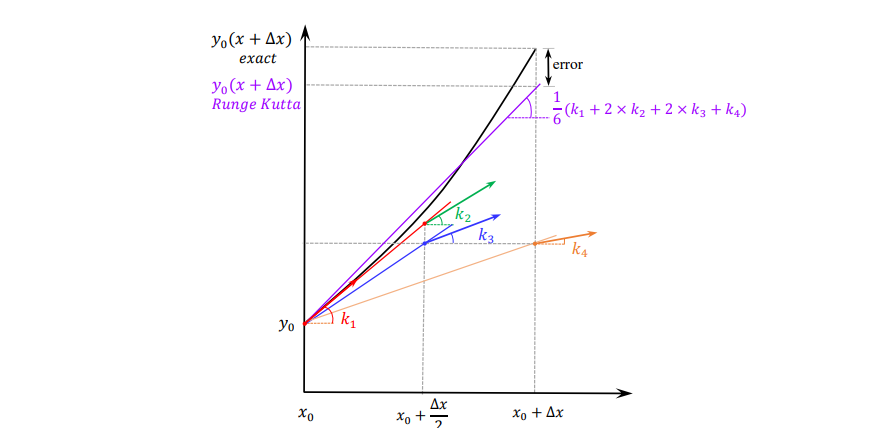


Figure 2 Slopes Utilized in the Runge-Kutta Method

### **2.2.2 Optimization Steps of the RUN**

This algorithm uses random components in a swarm-based model, which is population-based, and the Runge Kutta logic to develop ordinary differential equations used to find the slope [5]. This slope is used as the search mechanism to explore and find the best solutions while observing the rules of the evolution of the swarm-based model. The Runge Kutta Optimizer is mathematically formulated in the sections below.

#### **2.2.2.1 Initialization of the Run Algorithm**

An initial swarm will undergo evolution for several iterations to start the algorithm. Therefore, initializing a size N population means N random positions are generated [3], [5]. The solutions of the optimization problem are the members of the population with a dimension D. The idea below is used in generating the random initial solutions in the RUN algorithm.

- The lower limit of the lth parameter and - Upper limit of parameter where

- Random number in [0,1]

#### **2.2.2.2 Development of a searching mechanism using the Runge-Kutta Method**

The RUN algorithm’s searching mechanism is centered on the Runge-Kutta method and uses random solutions to perform searches in the search space, both globally and locally [3], [5]. Like every other optimizer, this is the core of the program. From equation (5) of the RK method, the neighbors of are:

- best position

- worst position

We can use equation (5) to come up with the first coefficient, defining it as:

- worst solution at each iteration

- best solution at each iteration

and are determined by selecting three random solutions from the population. These three solutions are:

and

From the equation of , we can see that it lacks stochastic behavior. Therefore, to improve the exploration search phase, we introduce randomness and rewrite the equation as follows:

- random number in the range [0,1]

The best solution () is essential in enhancing the global search to find the global best solution. Thus parameter is introduced to grow the best solution’s importance [5].

The range between adjacent positions is given by:

The step size ( is given by the equation:

- is the scale factor determined by:

The scale factor decreases exponentially during optimization and is determined by the solution space size. The randomness of the numbers in equations (11-1) to (11-3) provides diversification in the searches [5].

The rest of the coefficients , , and are as written below:

The numbers and are random numbers in the range [0,1]. The program below determines the worst and best solutions and.

(15)

is the best random solution from random selections *(, , )*

The searching mechanism of the RUN algorithm is given by:

in which

#### **2.2.2.3 Modernizing the Solutions**

During optimization, the RUN algorithm updates the positions of the solutions at each iteration using the Runge Kutta method since the initial solutions were random selections [5]. The following Pseudocode shows how the solutions are updated in the exploitation and exploration phases.

where

where is a random number, and is a random number with a normal distribution.

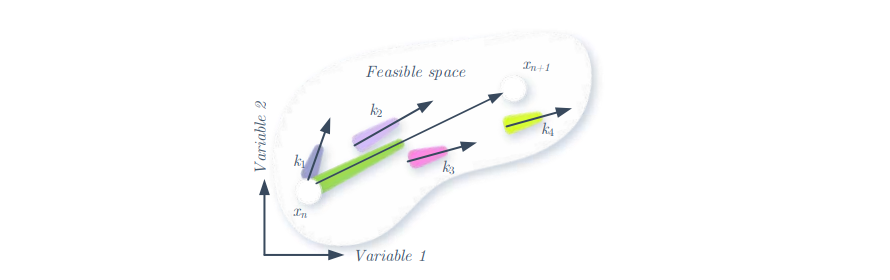
****

Figure 3 Slopes used to determine x\_(n+1) in the RUN algorithm

The expressions for and are:

-) (

The expressions for and are as follows:

(17-3)

(17-4)

where is a random number in [0 1]. is the current best solution. is the best solution at each iteration.

|  |  |
| --- | --- |
|  | (17-5) |
|  | (17-6) |

The adaptive factor, SF, is the one that provides the balance needed between exploitation and exploration [3], [5]. Initially, SF is large to enhance exploration by increasing diversity, and it decreases in value afterward, thus increasing the number of iterations and enhancing exploitation.

and are the main control parameters of the SF.

and are constants

- number of iterations

– maximum number of iterations

From equation 17, when , in the solution space, the algorithm conducts a global search and simultaneously, around solution , it does a local search. The exploration phase ensures all high-quality solutions are searched. When the RUN applies a local search around . Exploitation increases the speed of convergence, focusing on promising solutions. This is done by rewriting equation 17 as below.

|  |
| --- |
|  |
| **(exploration phase)** |
| (18) |
| **(exploitation phase)** |
|  |

- integer number(either 1 or – 1)

- random number in [0,2]

increases diversity by changing the search directions, and with each iteration, exploitation around reduces.

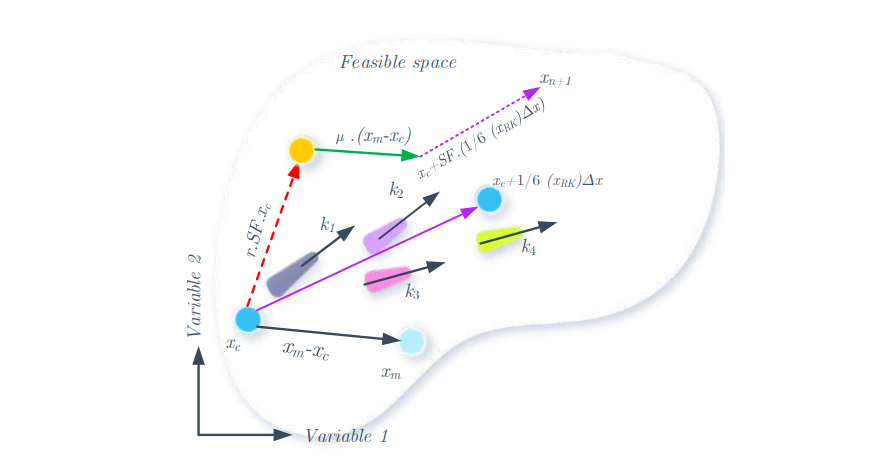


Figure 4 The Search Mechanism of the RUN

#### **2.2.2.4 Enhanced Solution Quality**

This ensures solutions after each iteration head towards better positions. It operates using the following Pseudocode:

|  |  |
| --- | --- |
|  |  |
|  | (19) |
|  |
|  |
|  |
|  |
|  |  |

in which

|  |  |
| --- | --- |
|  |  |
|  | (19-1) |
|  | (19-2) |

-is a random number in [0,1]

- random number is equal to

- random number (decreases with increase in iterations)

- integer number(either 1 or 0 or – 1)

The solution ()may not be better than the current solution; hence the following equation is used to give the algorithm another chance to find a better solution than the two [5].

|  |  |
| --- | --- |
| **if** |  |
|  | (20) |
| **end** |  |

where = .

The implementation of () occurs when < to move () to a better position. The parameter enhances the importance of the best solution and to find ; and respectively become and since the fitness of is lower than that of .

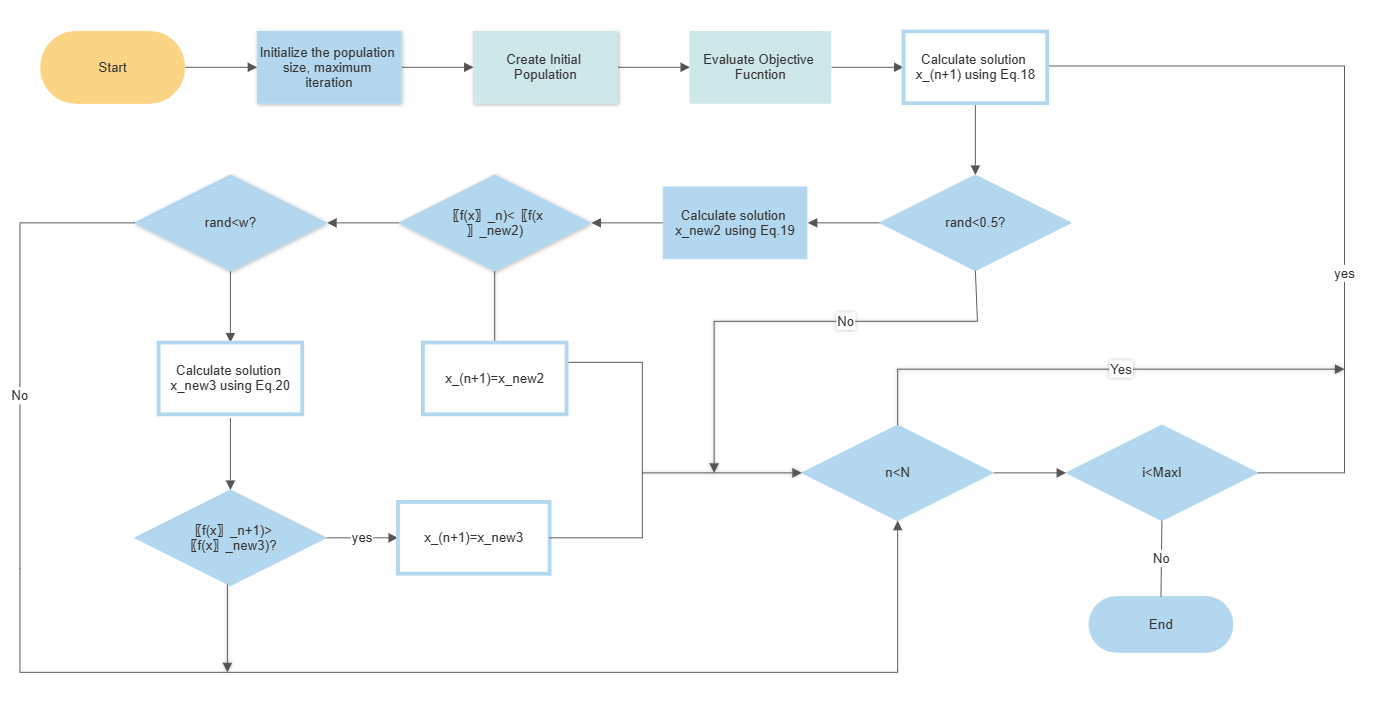
****

Figure 5 Flowchart of the Run Algorithm

#### **2.2.2.5 The Pseudocode for the RUN algorithm**

Table 1 RUN Pseudocode

|  |
| --- |
| **Algorithm 1.** The pseudocode of RUN |
| **Part 1. Initialization** |
| Initialize |
| Randomly generate the initial population of the RUN |
| Evaluate the objective function for each population member |
| Obtain , , and |
| **Part 2. RUN operators** |
| **for** *it= 1: MaxIt* |
| **for** *i* = 1 : *np* |
| **for** *j* = 1 : *dim* |
| Evaluate position (Equation 18) |
| **end for** |
| **ESQ** |
| **if** |
| Evaluate position (Equation 19)  **if**  **if** rand<  Evaluate position (Equation 20)  **end**  **end** |
| **end**  Modernize positions and |
| **end for**  Modernize position |
| *it*=*it*+1 |
| **end** |
| **Part 3.** return |

### **2.2.3 Limitations of Runge-Kutta Optimizer**

Despite the RUN algorithm having better convergence speeds due to appropriate balancing between the local and global searches, it still suffers some limitations similar to the metaphor-based optimizers mentioned below.

•         The algorithm is single-objective based and cannot solve multi-objective optimization problems [5].

•         During ESQ, the algorithm can be trapped in the local search hence confining it to only locally optimal solutions.

The proposed RUN algorithm can be enhanced by using chaotic maps when running ESQ in each iteration to improve the convergence speed and avoid converging at local solutions. Chaotic maps introduce chaos into the algorithm, thus promoting its search abilities to provide better solutions [3]. The algorithm can further be made multi-objective to handle complex problems that have multiple objectives [1].

# **CHAPTER 3: THEORETICAL BACKGROUND**

This chapter provides the theoretical background for the abstract ideas used in this project. These concepts include: using chaotic maps in optimization algorithms, multi-objective optimization techniques, and circuit design optimization.

## **3.1 Chaotic Maps**

In essence, chaotic maps are mathematical functions that behave chaotically, such that slight variations in the original conditions have radically diverse effects. To broaden the range of potential solutions and avoid local minima traps, chaotic maps are typically utilized in control and optimization algorithms [10]. The logistic, tent, circular, Gaussian, and Chebyshev maps are some examples of chaotic maps.

Due to their tendency to get stuck in local optima, traditional algorithms struggle to optimize non-convex and multimodal objective functions. By adding unpredictability to the optimization process, chaotic maps offer a solution to this issue [17], [18], [19], [20]. The optimization algorithm can better traverse the search space and avoid local optima by using chaotic maps to produce random solutions or perturb the present solution. This is especially helpful in high-dimensional search spaces, where many local optima may exist [3].

Chaotic maps can be incorporated in several ways into optimization algorithms. One method is to use chaotic maps to produce the initial population of solutions in an evolutionary algorithm. A group of random solutions can be produced using the chaotic map, and these solutions can then be evolved by crossover, mutation, and selection processes [23].

Another approach is when a local search algorithm like simulated annealing or tabu search uses chaotic maps to alter the existing solution. The chaotic map can produce a minor perturbation to the present solution, which is accepted or rejected according to a probability distribution [10].

Finally, the parameters of a chaotic system itself can also be optimized using chaotic maps. The chaotic map changes the system's parameters and alters the behavior of the objective function as it determines the system's complexity or randomness [3].

Overall, chaotic maps provide an effective tool for exploring complex, non-linear search spaces and avoiding local optima in optimization algorithms. Chaotic maps can enhance the effectiveness and efficiency of optimization algorithms in various applications by bringing randomness and diversity into the optimization process. Randomization helps in balancing the exploitation and exploration phases for optimal performance [20].

In this project, 10 chaotic maps are integrated to determine the best chaotic map in circuit design. They are obtained from [3] and are shown below:

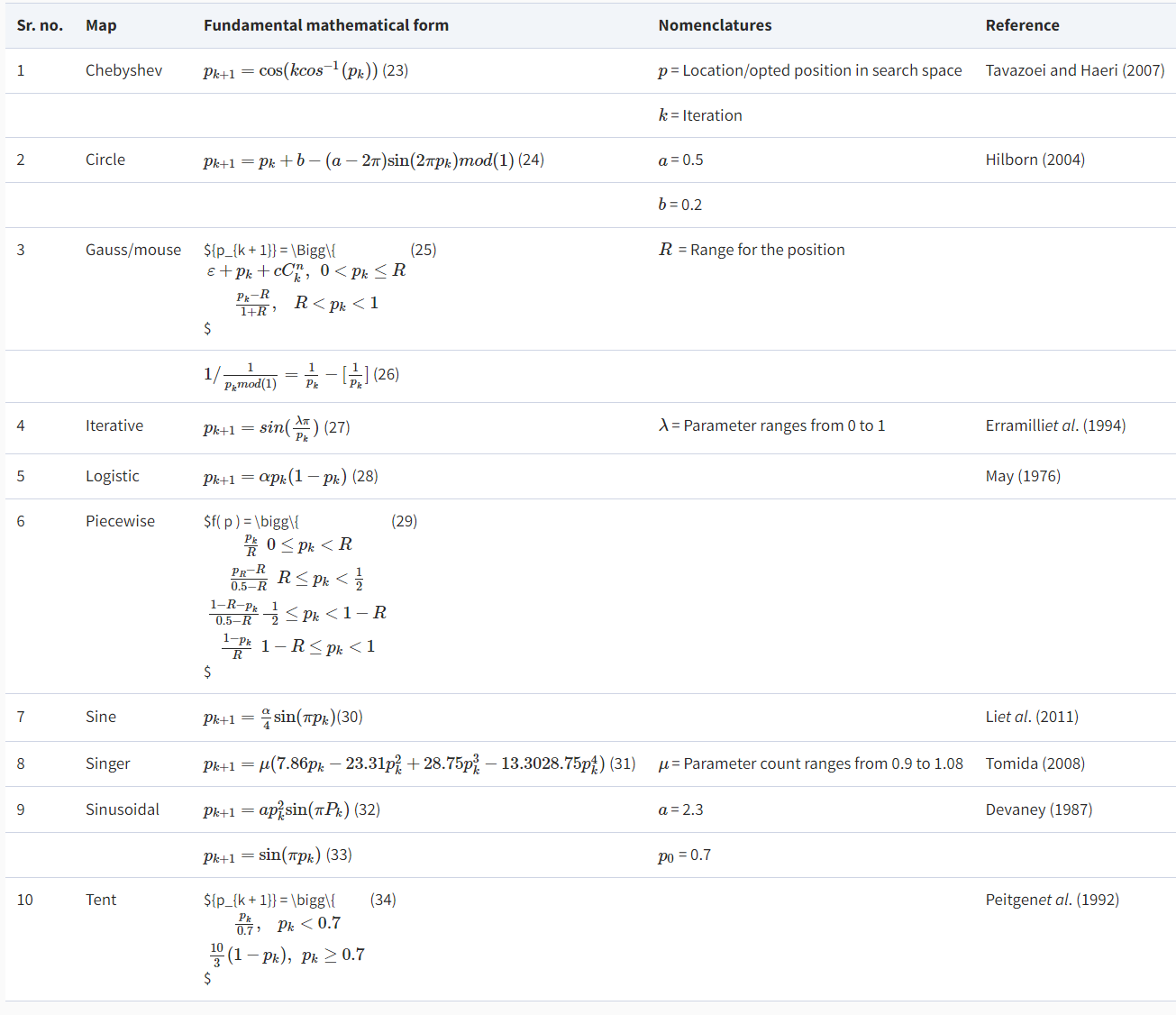


Table 2 List of chaotic maps

In the MATLAB implementation of these maps, the following figures are obtained that show the characteristics of each map.

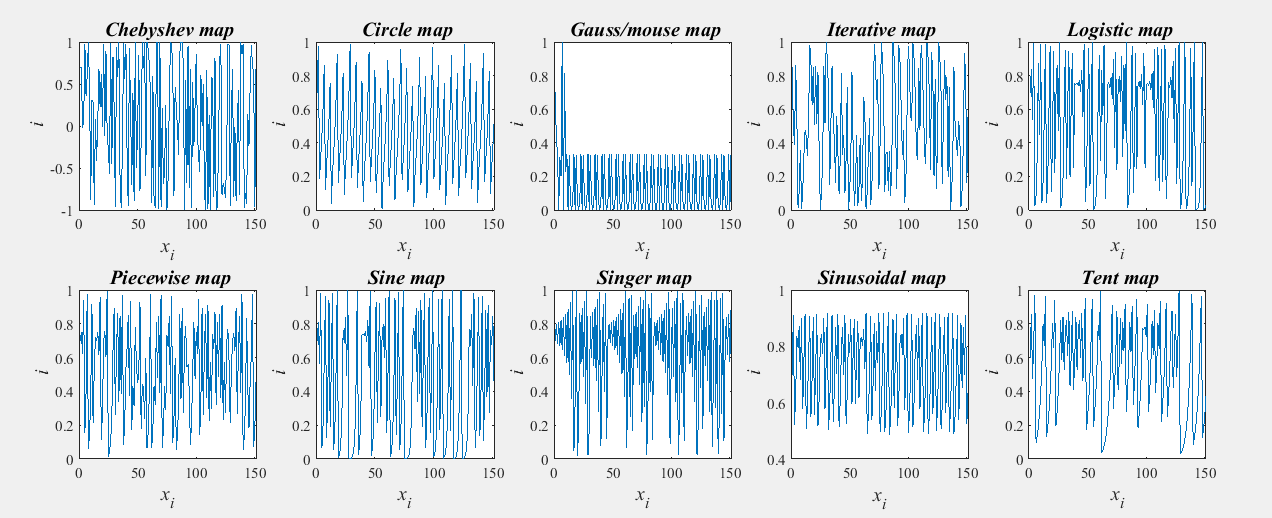


Figure 6 Matlab implementation of chaotic maps

## **3.2 Multi-objective Optimization Techniques**

In the engineering world, engineers often encounter several problems with multiple objectives that must be optimized simultaneously. In most cases, these objectives have some sort of conflict, such as improving one objective deteriorates another. Additionally, these objectives have different units of measurement and are called Multi-objective Optimization Problems (MOPs). Unlike in single-objective problems where a single optimal result is obtained by determining which solution is better, MOPs do not have a clear-cut method of determining which solution is better due to conflicts and different measurement units [1]. Circuit design often involves the optimization of multiple objectives, such as the gain output of a system and the cost of components. Thus in this project, the RUN algorithm will be made multi-objective to handle several objectives simultaneously.

There are several ways of making an algorithm multi-objective for solving MOPs. These methods include:

### **3.2.1 Weighted Sum Method**

This technique involves the combination of several objectives into one objective in MOPs. Each objective is assigned a weight, then applying linear combination to all weighted objectives [1]. The weights reflect the relative importance of each objective and hence can be used to balance the trade-offs between objectives. For example, consider a MOP with two objectives, and . The objectives can be combined using the weighted sum method as shown below:

=

where and are the weights of and respectively.

The optimal solution can be found by minimizing and will depend on the values of and , which can be attuned to give the relative importance of each objective.

### **3.2.2 Scalarizing Method**

This technique involves converting a MOP into a single-objective optimization problem by defining a scalarizing function that maps several objectives into one objective. For example, consider a MOP with two objectives, and . The problem is changed from having multiple objectives to having one objective, as shown below:

where is the scalarizing function.

Based on the specified needs of the problem, the scalarizing function is defined in several ways. For example, it could be a function that reflects the trade-offs between objectives or as a weighted sum of objectives [1]. The function is then mapped back to the original MOP once an optimal solution is found.

### **3.2.3 Pareto Optimization**

This technique involves finding a set of non-dominated solutions which means that improving one objective has the effect of deteriorating one or more of the other objectives. If there exists no other solution that is better in at least one or all objectives, then the solution obtained is considered Pareto optimal [7]. These non-dominated solutions form the Pareto front, which is a representation of all Pareto optimal solutions.

### **3.2.4 Evolutionary Algorithms**

These algorithms are inspired by natural evolution processes and come in handy in solving complex MOPs that have conflicting objectives. The operation of evolutionary algorithms involves the iterative generation of populations of solutions, and the fitness function searches for the optimal solution by evaluating the quality of each solution [1]. Modification of the solutions selected occurs through crossover and mutation that creates a fresh set of solutions. The processes are repeated until the optimal solution is attained or a set condition is achieved. Evolutionary algorithms are best suited to solve problems with wide search spaces and complex non-linear objectives.

### **3.2.5 Multi-objective Gradient Descent Methods**

This technique optimizes multiple objectives simultaneously using modified gradient descent algorithms. Here, a multi-objective loss function is defined, which combines all the objectives into a single objective. The loss function reflects the trade-offs between objectives and should be differentiable to make the computation of the gradient possible. The multi-objective gradient descent algorithm then adjusts the model parameters to minimize the loss function iteratively, whereby at each iteration, the gradient of the loss function is computed, and then the model parameters are updated in a way that minimizes loss. This process is repeated until a preset condition is achieved or the loss function converges to a minimum.

### **3.2.6 Decomposition-based Methods**

These are algorithms that divide MOPs into a series of sub-problems and optimize each sub-problem separately. After obtaining individual solutions to each sub-problems, these separate solutions are combined into one final solution to the original MOP.

The choice of a multi-objective optimization technique typically depends on the unique aspects of the problem and the optimization's objectives. The choice of technique can significantly affect the effectiveness and quality of the solution since each technique has advantages and disadvantages.

## **3.3 Optimized Circuit Design**

Circuit design is designing electronic circuits by choosing parameters that meet specific performance criteria based on theoretical expectations. The performance of a circuit is measured using criteria such as noise, stability, speed, and power consumption. Optimized circuit design aims to achieve the desired performance while minimizing the cost, size, and complexity of the circuit [9].

There are several approaches to optimized circuit design, each with its advantages and limitations. Here are some of the most commonly used techniques:

* Top-down design: This approach involves starting with the overall system requirements and working downwards to the circuit level. The design is broken down into a hierarchy of sub-circuits, each designed to meet specific requirements. The advantage of this approach is that it ensures that the design meets the overall system requirements, but it can be time-consuming and may not always result in the most efficient circuit.
* Bottom-up design: This approach involves starting with individual circuit components and building the overall system. The advantage of this approach is that it can be faster and more efficient than top-down design, but it may not always result in the best overall performance.
* Simulation-based design: This approach uses computer simulations to model the circuit's behavior and optimize its performance. Simulation-based design can be very effective at identifying design flaws and optimizing circuit performance, but it can be time-consuming and computationally expensive.
* Optimization algorithms: Optimization algorithms such as genetic algorithms can optimize circuit designs by generating a population of candidate solutions and evaluating their performance using simulations [14].

Of all the methods, optimization algorithms help to save time by giving the optimal solution based on the range of input parameters eliminating the need for trial-and-error, which is time-consuming.

The circuit design proposed in this project is a single-stage amplifier. Amplifiers are used to improve the strength of weak signals. Multi-stage amplifiers provide superior performance by allowing more flexibility for the input and output impedances. It is crucial to understand the working of a single-stage amplifier since they are connected in a cascade to form multi-stage amplifiers to provide even higher gain [12].

A single-stage amplifier comprises a single transistor with a bias circuit and other components to facilitate the desired gain output for quantities such as current, voltage, and power [12]. The following components and parts are included for a typical single-stage amplifier design.

* Transistor: The transistor is the active device in the amplifier circuit and is responsible for amplifying the input signal. There are different types of transistors that can be used depending on the specific requirements of the circuit.
* Biasing circuit: The biasing circuit sets the operating point of the transistor, which is important for ensuring that the amplifier operates in the linear region and avoids distortion [12]. The biasing circuit typically consists of resistors and/or capacitors connected to the transistor.
* Input coupling capacitor: used to isolate the input signal from the DC bias of the amplifier circuit.
* Output coupling capacitor: The output coupling capacitor isolates the amplifier circuit from the load or the next stage in the circuit by blocking DC signals [12].
* Load resistor: converts the output current into an output voltage. It determines the gain of the amplifier and is typically chosen to match the load's impedance.
* Feedback network: The feedback network is used to provide negative feedback to the amplifier, which helps to reduce distortion and improve stability. The feedback network typically consists of resistors and/or capacitors that are connected between the output and input of the amplifier.
* Power supply: It powers the amplifier circuit. It typically consists of a DC voltage source and filtering capacitors to remove any AC noise or ripple from the supply voltage.

The circuit for a simple, practical single-stage amplifier is given below:

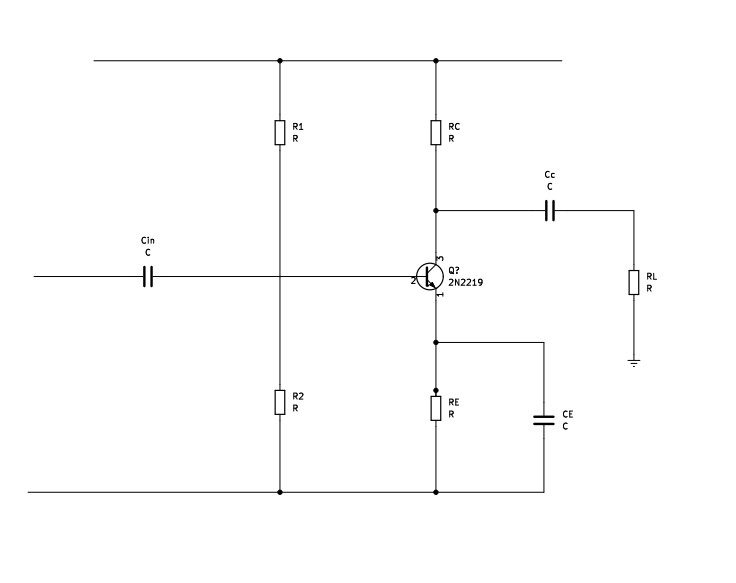


Figure 7 Single-stage amplifier circuit

In designing a single-stage amplifier, several parameters can be optimized to produce optimum performance. These are:

* Gain: Increasing the gain of the amplifier can improve its sensitivity and ability to amplify weak signals.
* Bandwidth: Increasing the bandwidth of the amplifier can improve its ability to amplify high-frequency signals.
* Input and output impedance: Optimizing the input and output impedance can improve the matching between the amplifier and the signal source and load, respectively.
* Distortion: Distortion is any unwanted modification of the input signal, and it can be caused by nonlinearity in the amplifier circuit. Minimizing distortion can improve the fidelity and accuracy of the output signal.
* Noise: Noise is any unwanted signal that is added to the output signal by the amplifier circuit. Minimizing noise can improve the signal-to-noise ratio (SNR) of the output signal.
* Power consumption: Minimizing power consumption can improve the amplifier's efficiency and prolong the battery life in portable devices.
* Stability: Stability refers to the ability of the amplifier to operate reliably and predictably over time without oscillations or instability. An unstable amplifier can produce unwanted oscillations, which can cause distortion, noise, and other performance issues.

Single-stage amplifiers are used in tape recorders, radio, television receivers, CD players, and public address systems. Therefore, the design of optimal single-stage amplifiers is essential in electronics.

# **CHAPTER 4: METHODOLOGY**

In this chapter, the steps leading to the development of a chaotic multi-objective RUN algorithm are described together with the performance of the developed algorithm.

To conduct this project, an HP Probook G3 with 8GB RAM and Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40 GHz processor was used to run and test the algorithms using MATLAB R2021a.

## **4.1 Multi-objective Chaotic Runge-Kutta Optimization Algorithm (CMRUN)**

The following steps were undertaken to improve the base RUN algorithm to handle multiple objectives while using chaos to improve exploration searches.

### **4.1.1 Selecting what part of the base RUN to improve by chaos**

Many optimization algorithms fall short due to local minima traps, especially when solving non-convex and multimodal problems. Researchers, therefore, came up with a solution to enhance their performance by increasing the diversity of solutions and thus avoiding local minima traps [19], [20], [21], [22]. As mentioned earlier in Chapter 3, there are several ways to implement chaos in optimization algorithms. One way is to use chaotic maps to generate the initial population of solutions, which undergoes selection, crossover, and mutation operations.

Another way is the chaotic map can be used to generate a small perturbation to the current solution, which is then accepted or rejected based on a probability distribution. The final way is chaotic maps can also be used to optimize the parameters of a chaotic system itself [17]. The implementation of the RUN can take all the above approaches, but in this project, the approach of creating a perturbation to current solutions is used.

As discussed in Chapter 2, the best solution () is essential in enhancing the global search space to find the optimal solution in the original RUN algorithm. The best and worst solutions obtained at each iteration determine the searching mechanism of the RUN algorithm. Iman Ahmadianfar et al. Used the rand parameter to introduce randomness and enhance the exploration search. Furthermore, equations 11-1 to 11-3 provide search diversification to avoid local optima traps.

The range between adjacent positions:

The step size:

The scale factor:

By their contribution to the global search of the RUN algorithm, these equations were selected to be improved by chaos, thus improving the overall performance of the original RUN.

In this project, the 10 chaotic maps are represented as a vector, and each map is selected by inputting its index value as in Table 2. The initial value chosen for all the maps is 0.7. The final values of all the maps should lie in the range [0 1], and thus all values in [-1 1] are normalized so that they are within the acceptable [0 1] range. Usually, chaotic parameters are used in the parts of algorithms that require random parameters. In this project, the parameter for equations 11-1, 11-2, and 11-3 was replaced with a chaotic parameter .

The chaos vector is first calculated, and then the index of the chaotic map to be integrated determines which vector values will be selected to introduce chaos in the RUN algorithm. The parameters selected are improved at each iteration to ensure new random values are produced with each iteration. These values are then used to introduce chaos in the RUN through equations 11-1 to 11-3, which are modified as follows:

The range between adjacent positions:

The step size:

The scale factor:

where is the current position at each iteration.

The above parameters are used in the searching mechanism, and thus they enhance the quality of solutions by increasing the search space. Furthermore, the algorithm begins optimization using a set of random initial solutions, which get updated after each iteration using the Runge-Kutta method, which employs the searching mechanism. This implies that the solutions after each iteration have chaotic behavior, and thus the global search is enhanced. Equations (18) are further improved using the chaotic maps to enhance the exploration and exploitation phases by replacing with , as shown below:

|  |
| --- |
|  |
| **(exploration phase)** |
| (25) |
| **(exploitation phase)** |
|  |

The same is done to the Enhanced Solution Quality (ESQ) section of the code, as shown below.

|  |  |
| --- | --- |
|  |  |
|  | (26) |
|  |
|  |
|  |
|  |
|  |  |

The original RUN is changed to utilize chaos in updating solutions and converging to the global optimum solution by performing these changes.

### **4.1.2 Chaotic Multi-objective Runge-Kutta Optimizer (CMRUN)**

The original RUN and the chaotic RUN described above can only optimize single-objective problems. The first step to making the chaotic RUN multi-objective is modifying the objective function to take more than one objective. In this case, the objective function was modified to take two objectives and generate a set of solutions for each objective. The fitness function was also modified to record the solutions for two objectives, generating two convergence curves that form the Pareto front [1].

**The Pseudocode for the CMRUN algorithm**

|  |
| --- |
| **Algorithm 2.**  The pseudo-code of CMRUN |
| **Part 1. Initialization** |
| Define the number of objective functions (nObj = 2) |
| Initialize the fitness function for both objective functions |
| Randomly generate the initial population for the CMRUN |
| Evaluate the objective function values of each population member |
| Sort the costs obtained from the objective function |
| Initialize the chaos parameters |
| Update the convergence curves of both objectives with the first best costs |
| **Part 2. CMRUN operations** |
| **for** *it= 1: MaxIt* |
| Update the chaotic parameters |
| **for** *n* = 1: *N* |
| Apply chaotic parameters in updating the algorithm’s equations |
| Determine the solutions , , and for each objective function |
| Perform operations to improve and update the solutions |
| Update best costs for the objective functions |
| Check if solutions go outside the search space and bring them back |
| Update chaos parameters for ESQ |
| **Enhance the solution quality** |
| **for** *j* = 1 : *dim* |
| Determine from equation 18 |
| **end for** |
| Perform boundary check for solutions again |
| **if** |
| Evaluate position  **if**  **if** rand<  Determine position  **end**  **end** |
| **end**  Modernize positions and |
| **end for**  Modernize position |
| *it*=*it*+1 |
| **end** |
| **Part 3.** Return and best costs |
| Update Convergence Curves |

Table 3 CMRUN Pseudocode

## **4.2 Evaluation of the algorithm’s performance**

All the 10 chaotic maps were integrated one by one into the CMRUN, and their performances were evaluated using 15 selected benchmark functions. The results of the best version of the CMRUN were compared with those of 11 known multi-objective optimization algorithms.

The benchmark test functions are of two kinds [5]:

* **Unimodal Functions**- they have one global optimum and test for an algorithm's exploitation and convergence to the global optimum.
* **Multimodal Functions**- they have multiple local optima and are used to test for an algorithm's exploration and ability to avoid premature convergence.

The following parameters were kept constant to test the CMRUN’s performance and the other 11 optimizers.

* Population number = 50
* Dimension = 30
* Maximum iterations = 200
* Number of runs = 10

The benchmark functions used are given in Table 4. The following 11 algorithms were tested to compare their performance with the CMRUN:

1. Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D)
2. Multi-Objective Particle Swarm Optimization (MOPSO)
3. Non-dominated Sorting Genetic Algorithm II (NSGA-II)
4. Multi-Objective Firefly Algorithm (MOFA)
5. Multi-Objective Bat Algorithm (MOBA)
6. Strength Pareto Evolutionary Algorithm 2 (SPEA2)
7. Multi-Objective Cuckoo Search (MOCS)
8. Multi-Objective Flower Pollination Algorithm (MOFPA)
9. Multi-Objective Mayfly Optimization Algorithm
10. Hybrid NSGAII-MOPSO Algorithm
11. Multi-Objective Non-Sorted Moth FLAME (MOMFO) (NSMFO)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Name | Type | Dimension | Range |
| F1 | SCHAFFER | U | 30 | [-100 100] |
| F2 | POLONI | U | 30 | [-100 100] |
| F3 | VIENNET2 | U | 30 | [-4 4] |
| F4 | VIENNET3 | U | 30 | [-3 3] |
| F5 | TANAKA | U | 30 | [0 pi] |
| F6 | ZDT2 | M | 30 | [-100 100] |
| F7 | ZDT3 | M | 30 | [-100 100] |
| F8 | ZDT4 | M | 30 | [0 1] |
| F9 | ZDT5 | M | 30 | [0 1] |
| F10 | ZDT6 | M | 30 | [0 1] |
| F11 | DTLZ1 | M | 30 | [-100 100] |
| F12 | DTLZ2 | M | 30 | [-100 100] |
| F13 | DTLZ4 | M | 30 | [-100 100] |
| F14 | KURSAWE | M | 30 | [-5 5] |
| F15 | FONSECA-FLEMING | M | 30 | [-4 4] |
| **U-Unimodal M-Multimodal** | | | | |
|

Table 4 Benchmark Test Functions

To evaluate the performance, the minimum score of each optimizer for each test function was 10 runs. These scores' averages were then calculated with their standard deviations as the criteria for measuring performance.

## **4.3 Circuit design**

In this project, a simple circuit for a single-stage amplifier was designed. As discussed in Chapter 3, a single-stage amplifier has many practical applications in the real world. Circuit design is always an essential skill in the electronics world. Designers are always trying to develop new circuit design methods that will lead to minimal power consumption, minimal power losses, low noise, low costs, and peak performance [14].

This project was proposed to create an algorithm that could be used to optimize circuit parameters. The circuit chosen is a simple circuit to test the CMRUN’s convergence capability. Optimizing a simple circuit and giving out the best parameters would mean the algorithm is powerful enough to be applied to more complex circuit design problems.

The objective functions chosen in this project are:

* Noise
* Distortion

Noise is any unwanted signal that arises at the output of a circuit. Noise reduces the performance of an electronic device due to errors and reduced sensitivity. Distortion, on the other hand, is when the output signal has been changed by nonlinearity; hence, the circuit does not produce an output consistent with the input.

For the circuit selected, the following parameter is to be optimized:

|  |  |  |
| --- | --- | --- |
| Variable | Symbol | Range |
| Input Resistor | Rin | [5000, 15000] |
| Collector resistor | Rc | [500, 1500] |
| Load resistor | RL | [5000, 15000] |
| Emitter resistor | Re | [500, 15000] |
| Lower frequency | fL | [10K, 50K] |
| Upper frequency | fH | [1M, 20M] |

Table 5 Amplifier Circuit Parameters

From the values of resistances and frequencies obtained, the values of capacitances can easily be calculated from the standard formulae. This part of the project aims to minimize noise and distortion and obtain the values of frequencies and resistances where the two objective functions are minimum and hence find the optimum performance of the circuit from the given ranges of inputs.

# **CHAPTER 5: RESULTS AND ANALYSIS**

This chapter will analyze the results from the benchmark tests for the CMRUN and the other 11 multi-objective algorithms to determine the CMRUN's performance.

The results and analysis of the circuit design parameters are also included in this section.

## **5.1 Benchmark test results of the Chaotic Multi-objective Runge-Kutta Optimization Algorithm**

The CMRUN was implemented using 10 chaotic maps and ran 10 times for each benchmark function. The mean scores were recorded to determine the best chaotic map out of the ten. The best CMRUN version was selected for comparison with the other 11 multi-objective algorithms. The other algorithms were also run 10 times, and their average scores and standard deviations were recorded.

**Functions F1-F5 are Unimodal Test Functions.**

**Functions F6-F15 are Multimodal Test Functions.**

Table 6 Comparison results of the 10 chaotic maps using 15 benchmark test functions

From the comparison results, the performances of all the 10 chaotic maps are close, and it was difficult to differentiate the best chaotic map. The Chebyshev map was chosen as it provided the most consistent results and better average performance. Therefore the version of CMRUN used to determine its performance against the 11 multi-objective algorithms incorporates the Chebyshev map.

Table 7 Statistical results of the Benchmark test functions from RUN and 11 other multi-objective optimization algorithms

From the comparison of the average scores, the CMRUN ranks as shown below:

|  |  |
| --- | --- |
| Function | Rank |
| F1 | 9 |
| F2 | 1 |
| F3 | 1 |
| F4 | 2 |
| F5 | 1 |
| F6 | 11 |
| F7 | 10 |
| F8 | 10 |
| F9 | 7 |
| F10 | 11 |
| F11 | 9 |
| F12 | 7 |
| F13 | 9 |
| F14 | 8 |
| F15 | 1 |

Table 8 CMRUN Performance Rank

From Table 7, the averages for each algorithm were compared for every benchmark function. The CMRUN performs averagely well compared to several of the algorithms, and its rank for all the benchmark test functions is shown in Table 8. Averaging the rank of the CMRUN, it is found to be 6.4667, which indicates that the algorithm achieved better results compared to a significant portion of the other algorithms on average.

For the unimodal functions, CMRUN’s best performance is in F2, F3, and F5, where it ranks joint first with other algorithms with a very low standard deviation. The CMRUN has the best performance and edges other algorithms in only one of the ten multimodal functions (F15).

Furthermore, the CMRUN attains the global optimum in three benchmark functions: F2, F3, and F5, and the standard deviation for F5 is zero.

The standard deviation of the CMRUN is generally low for all the benchmark functions. The low standard deviation could indicate the following about the CMRUN:

* The CMRUN is close to convergence or has converged; thus, it has found the optimal or stable solutions; hence, further iterations are unnecessary.
* The CMRUN is robust and is not affected by fluctuations during the optimization process, and thus it produces high-quality solutions.
* The CMRUN is trapped in local optima and is not finding other better solutions.

An algorithm's convergence rate and solutions quality should be considered to determine whether the algorithm is stuck in local optima. The CMRUN struggles in optimizing multimodal functions and ranks position 11 twice. It defeats at least three optimizers for most of the multimodal functions. This might indicate the CMRUN has difficulties in maintaining the balance between exploration and exploitation. The success of an algorithm is linked to its balance between the two phases.

Exploitation focuses on exploiting the current knowledge by concentrating the algorithm’s search around areas likely to contain the best solutions. It refines the search around high-quality solutions and hence finds the local optimum. On the other hand, exploration ventures towards unexplored areas in the search space to find potentially better solutions, thus maintaining diversity. As the optimization progresses, exploration finds potential global solutions. Then the search space focus shifts towards exploitation which refines and improves these solutions to find the best global solutions.

The unimodal functions test for an algorithm's exploitative behavior, and the CMRUN shows good exploitation capabilities from the results. The multimodal functions test an algorithm's explorative behavior, and the CMRUN seems to struggle in exploring the global search space depending on the problem at hand.

## **5.2 Optimized Circuit Design**

After testing the CMRUN against other algorithms, its performance was deemed acceptable for optimizing a real-world problem. Furthermore, the circuit design objective functions were optimized using the other 11 algorithms to determine the CMRUN’s performance in optimizing circuit design. The following parameters were kept constant to conduct this part of the project.

* Population number = 100
* Dimension = 50
* Maximum iterations = 1000
* Number of runs = 10

The table below shows the results of optimizing circuit parameters for all 10 chaotic maps.



Table 9 Comparison results of the 10 chaotic maps for optimized circuit design

The Iterative map provides the lowest noise value, followed closely by the Chebyshev map. The Circle map provides the lowest distortion value. The Chebyshev map offers the best performance as it provides the lowest Pareto Sum for all the maps showing a balanced performance between the two objectives.



Table 10 Results for Optimized Circuit Design

From the above results, the CMRUN outperforms the other algorithms in optimizing the designed circuit. It has the lowest Pareto sum fitness of all the algorithms with weights of [0.5 0.5] for the objective functions. The equal weights mean the objective functions are considered equally important. From the results, CMRUN has the lowest noise value of 227.202, followed closely by MOMA. The other algorithms have slightly higher noise values, with MOPSO having the highest noise value. For the distortion values, the algorithms MOFA, MOBA, SPEA-2, MOCS, MOFPA, NSGA-II/MOPSO, and NSMFO have achieved the lowest distortion values of 0.0002 or less. CMRUN has a slightly higher value for distortion, although it's not the highest. Based on the Pareto Sum, CMRUN has the lowest Pareto Sum, followed closely by MOMA. Therefore CMRUN performs generally well compared to other algorithms despite having a higher distortion value than several of the other algorithms because it offers the lowest noise value and lowest Pareto Sum showing better balance in optimizing both objectives.

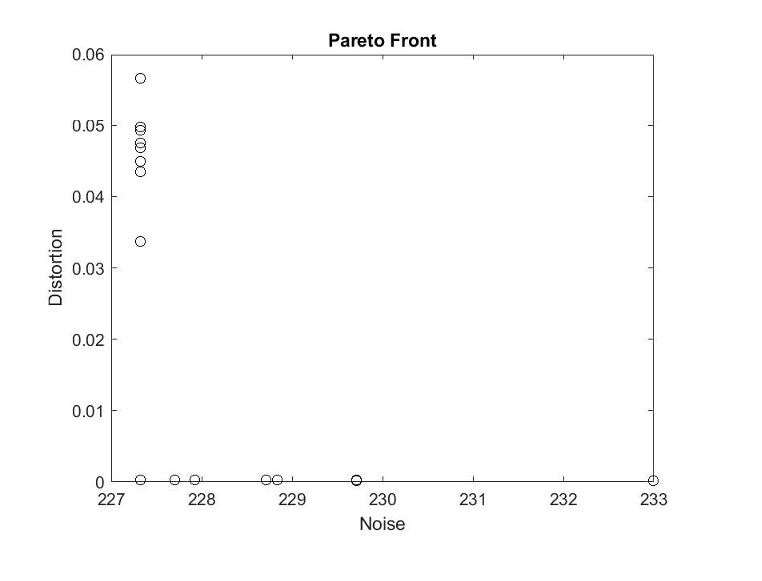
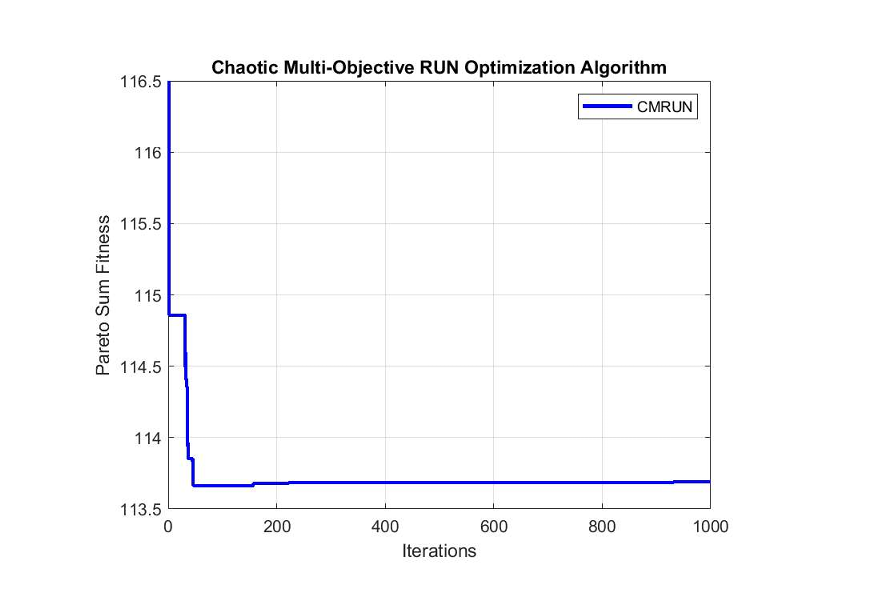
The CMRUN's performance in circuit optimization indicates that it performs well in minimizing the fluctuations in the noise values, but for distortion values, there are deviations in achieving the lowest value possible. This might be due to conflicts or trade-offs between distortion and noise. Although CMRUN shows excellence in reducing noise, it sacrifices distortion as improving noise performance deteriorates distortion. Overall, the performance of CMRUN in circuit optimization is acceptable as it defeats all the algorithms in Pareto Sum and offers the lowest noise value. The individual circuit parameters were also recorded and tabulated.

The best parameters obtained are shown in the table below.

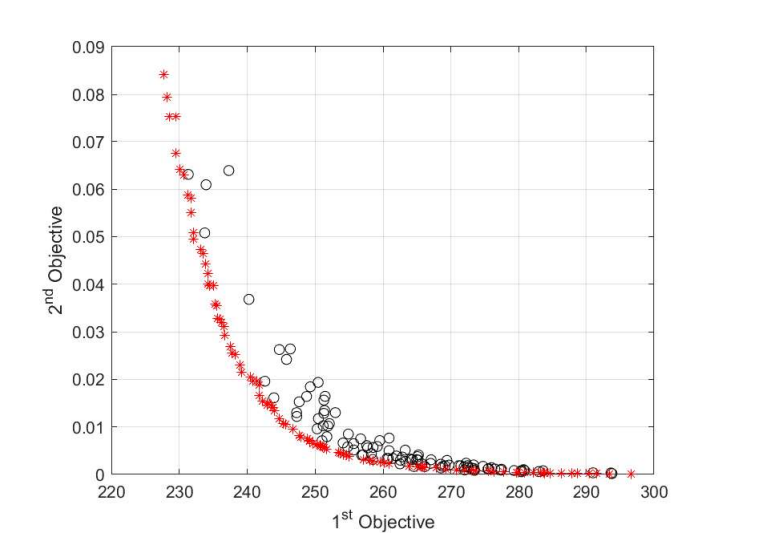
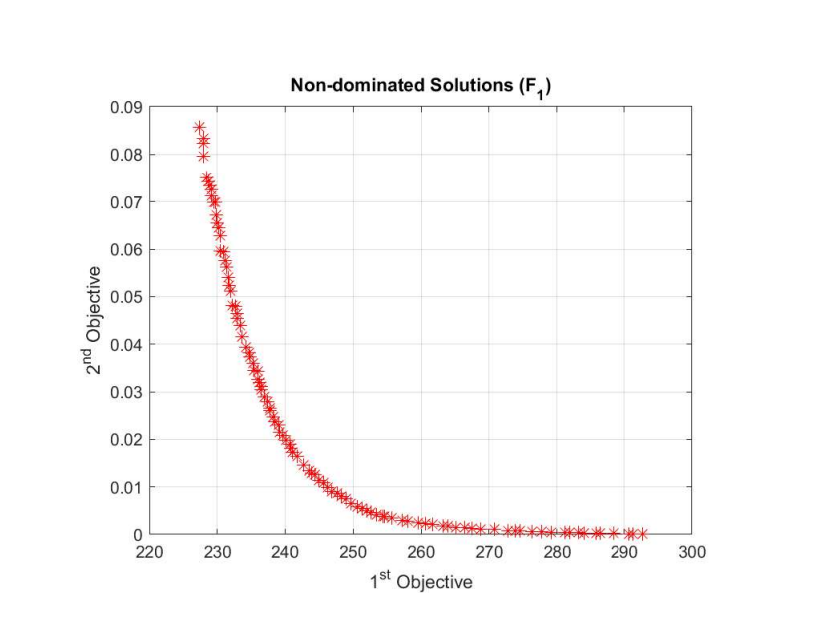
|  |  |  |
| --- | --- | --- |
| Variable | Symbol | Range |
| Input Resistor | Rin | 11300 |
| Collector resistor | Rc | 800 |
| Load resistor | RL | 150 |
| Emitter resistor | Re | 8740 |
| Lower frequency | fL | 20600 |
| Upper frequency | fH | 14400000 |

Table 11 Optimized design variables for a single-stage amplifier using CMRUN

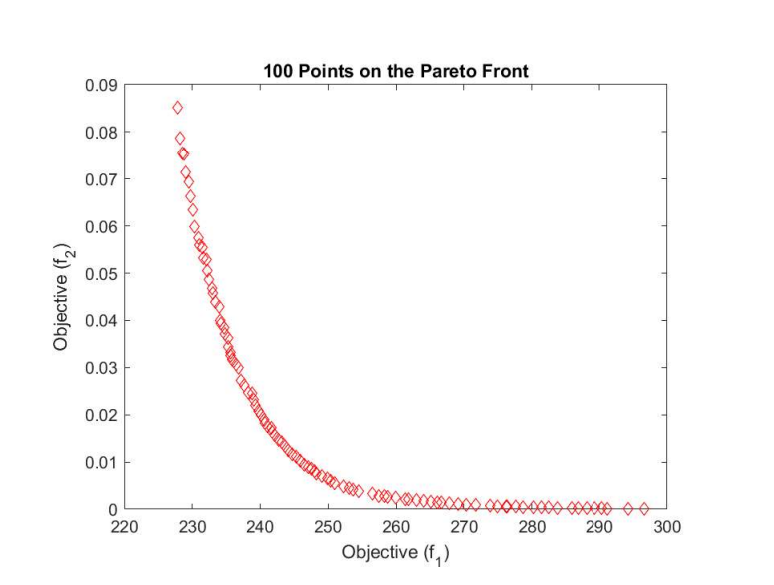
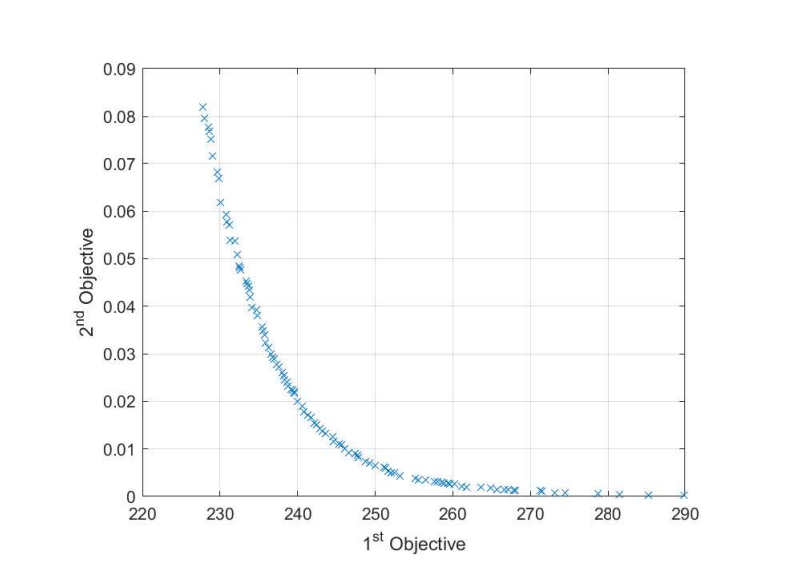
The convergence curves for some of the algorithms are attached. These curves show the algorithms' convergence rate when optimizing circuit parameters.

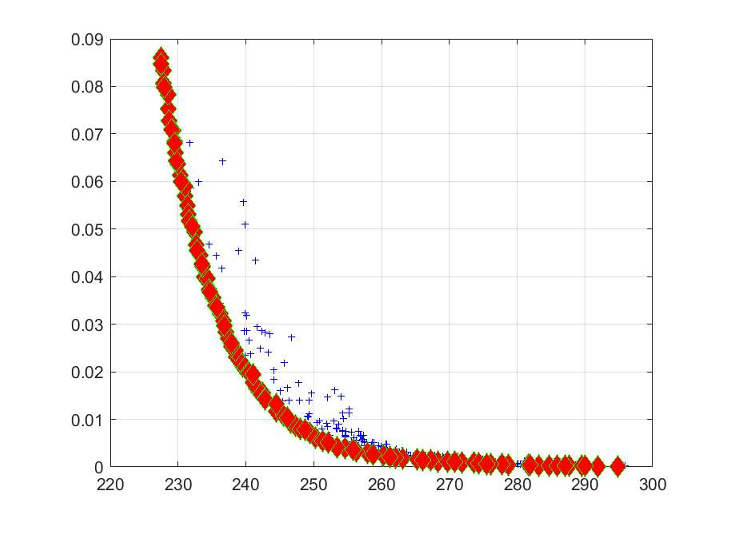
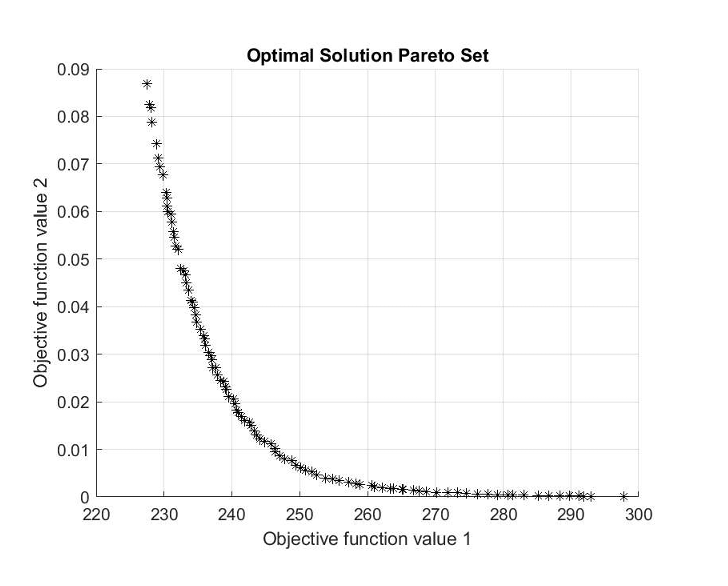
CMRUN CMRUN Pareto Fitness Plot



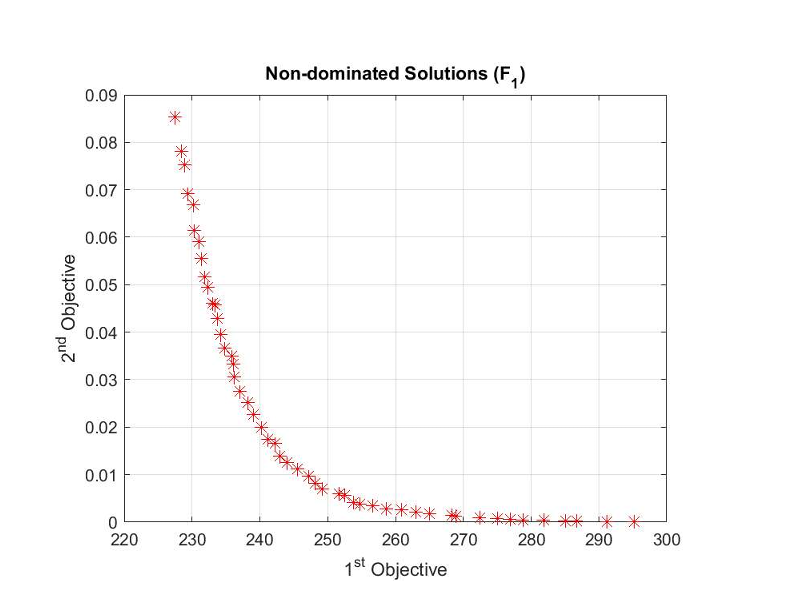
NSGA-II MOPSO

MOBA SPEA2

MOMA NSMFO



NSGA-II/MOPSO

The convergence curves show that the CMRUN has a very fast convergence rate. This indicates the algorithm’s superiority in obtaining optimal solutions using very few iterations. The CMRUN, therefore, seems to outperform the other algorithms in search exploration, narrowing the feasible region and converging to the global solution quickly when optimizing circuit parameters.

Although a fast convergence could also indicate the algorithm has converged prematurely, and the solutions are suboptimal, this is not the case for the CMRUN, according to the results. Therefore, it is important to consider both the convergence rate and the quality of solutions. From the solutions obtained by CMRUN, the solutions are of quality, and thus its fast convergence rate is desirable, for it indicates the algorithm requires fewer iterations to reach a satisfactory solution. This can lead to significant time savings and computational efficiency, especially in complex optimization problems with a large search space or computationally expensive objective functions.

# **CHAPTER 6: CONCLUSION AND RECOMMENDATION**

## **Conclusion**

This project proposed a chaotic multi-objective Runge-Kutta optimization algorithm (CMRUN) to optimize circuit design. The CMRUN is based on the original Runge-Kutta optimizer (RUN). Chaotic maps enhanced the RUN to improve its exploration capability and then made multi-objective to handle multiple objectives. The CMRUN showed excellence in exploitation but struggled in exploration searches when optimizing multimodal functions.

Obtaining the right balance between exploration and exploitation searches determines an algorithm's optimization capabilities and performance. This is because overemphasis on exploitation may lead to premature convergence to local optima, while excessive exploration may result in inefficient search and slow convergence. The best balance depends on the problem's characteristics, the objective function, and the available computational resources. The CMRUN still performed averagely well in exploration, and thus it applied circuit design optimization to test its ability to solve real-world problems.

When employed in circuit optimization, the CMRUN showed excellent performance despite struggling to optimize multimodal functions. The circuit designed was a simple single-stage amplifier. The chosen parameters to be optimized were noise and distortion to assess the performance of the CMRUN in real-world applications compared to the other already established multi-objective algorithms. The CMRUN provided the lowest noise value but had slightly higher distortion values compared to 8 of the other algorithms. A misbalance between the exploitation and exploration searches may have caused this.

Despite losing to some algorithms in optimizing benchmark test functions, the CMRUN had the lowest Pareto Sum of all the other algorithms and the lowest noise value. This is because real-world applications are typically more complex and may exhibit different characteristics than benchmark functions. Real-world problems often involve noise, uncertainty, nonlinearity, and other factors that make optimizing them more challenging. Additionally, the performance of an optimization algorithm can be influenced by the specific problem's structure and constraints, and the CMRUN was designed to handle circuit optimization.

An algorithm can perform well on benchmark functions by exploiting certain characteristics of those functions that do not translate well to real-world scenarios. It could be overly specialized or sensitive to specific problem features not present in practical applications. Conversely, an algorithm designed to be more robust and adaptive to real-world complexities might sacrifice performance on benchmark functions with simpler structures. Therefore, it's important to evaluate optimization algorithms using benchmark functions and real-world applications to comprehensively understand their capabilities and limitations.

## **Limitations of the design**

* CMRUN is susceptible to getting stuck in local optima and thus converges prematurely when optimizing multimodal functions.
* The balance between the exploitation and exploration searches cannot be quantified due to the randomization of the chaotic maps, which may not offer the best performance for some optimization problems.

## **Recommendations**

Further research should be done on this project’s objectives to develop a more practical implementation of the CMRUN that provides a better balance in optimizing circuit parameters. This could be done by:

* Implementing standard operators such as levy walks (LWs), crossover operators, mutation operators, or the opposite-based learning method to better the solution quality [5].
* Using other chaotic mapping strategies could offer better randomization to improve exploration searches and a better balance between exploitation and exploration.
* Using the Pareto Archived Evolution Strategy to store and improve non-dominated solutions [8]

# **REFERENCES**

1. Antonio L´opez Jaimes, S. Z. (2011, January). AN INTRODUCTION TO MULTI-OBJECTIVE OPTIMIZATION TECHNIQUES. Optimization in Polymer Processing.
2. Benhala, B. (2014, November). Hybridization Approaches of Metaheuristics for Optimal Analog Circuit Design. International Journal of Microwave and Optical Technology, 421-428.
3. Betül S. Yıldız, P. M. (2022, November). A Novel Chaotic Runge Kutta Optimization Algorithm. Journal of Computational Design and Engineering. doi:10.1093/jcde/qwac113
4. Enes CENGİZ, C. Y. (2021, September). Improved Runge Kutta Optimizer with Fitness Distance Balance-Based Guiding Mechanism for Global Optimization of High-Dimensional Problems. Journal of Science & Technology, 135-139. doi: 10.29130/dubited.1014947
5. Iman Ahmadianfar, A. A. (2021, April). RUN Beyond the Metaphor: An Efficient Optimization Algorithm Based on Runge Kutta Method. Expert Systems with Applications. doi: 10.1016/j.eswa.2021.115079
6. Iman Ahmadianfar, B. H.-K. (2023, January). An Enhanced Multioperator Runge-Kutta Algorithm for Optimizing Complex Water Engineering Problems. Sustainability, 15(3). doi:10.3390/su15031825
7. Ir. M., F. F. (2018, June). Using Genetic Algorithm to Generate Pareto-Front in Multi-Objective Problem. International Journal of Computer Applications, 180(51), 21-25. doi:10.5120/ijca2018917346
8. Joshua Damian Knowles, D. C. (1999, January). The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multi-objective Optimisation. ResearchGate. doi:10.1109/CEC.1999.781913
9. Miri Weiss Cohen, M. A. (2015). Genetic Algorithm Software System for Analog Circuit Design . ScienceDirect, 17-22. doi: 10.1016/j.procir.2015.01.033
10. Nassim . Aslimani, E.-G. T. (2020, August). A New Chaotic-Based Approach for Multi-Objective Optimization. Algorithms, 13(9). doi:10.3390/a13090204
11. R. Manjula Devi, P. M. (2021, October). IRKO: An Improved Runge-Kutta Optimization Algorithm for Global Optimization Problems. Computers, Materials and Continua, 70(3), 4803-4827. doi:10.32604/cmc.2022.020847
12. SASMITA. (2020, January 9). Single Stage Transistor Amplifier. Electronics Post.
13. Sundaram, A. (2020). Combined Heat and Power Economic Emission Dispatch Using Hybrid NSGA II-MOPSO Algorithm Incorporating an Effective Constraint Handling Mechanism. IEEE Access, 8, 13748-13768. doi:10.1109/ACCESS.2020.2963887
14. Wei Jer Lim, G. L. (2012, February). GA-based Optimization for Circuit Design Assistance. ResearchGate. doi:10.1109/ISMS.2012.46
15. Xiaoli Shu, Y. L. (2022, December). Multi-objective Particle Swarm Optimization with Dynamic Population Size. ResearchGate. doi:10.1093/jcde/qwac139
16. Dhawale, D., Kamboj, V. K., & Anand, P. (2021). An improved Chaotic Harris Hawks Optimizer for solving numerical and engineering optimization problems. Engineering with Computers, 44(22), 4897-4914. <https://doi.org/10.1007/s00366-021-01487-4>
17. Gezici, H., & Livatyalı, H. (2022). Chaotic Harris hawks optimization algorithm. Journal of Computational Design and Engineering, 9(1), 216-245. <https://doi.org/10.1093/jcde/qwab082>
18. Jordehi, A. R. (2015). Chaotic bat swarm optimisation (CBSO). Applied Soft Computing, 26, 523-530. <https://doi.org/10.1016/j.asoc.2014.10.010>
19. Kohli, M., & Arora, S. (2018). Chaotic grey wolf optimization algorithm for constrained optimization problems. Journal of computational design and engineering, 5(4), 458-472. <https://doi.org/10.1016/j.jcde.2017.02.005>
20. S. Saremi, S. Mirjalili, A. Lewis, Biogeography-based optimisation with chaos, Neural Computing and Applications, Vol. 25, pp. 1077 - 1097, 2014, Springer DOI: <http://dx.doi.org/10.1007/s00521-014-1597-x>
21. Yarpiz. (2023). Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) [Software]. MATLAB Central File Exchange. Retrieved June 4, 2023, from <https://www.mathworks.com/matlabcentral/fileexchange/52873-multi-objective-evolutionary-algorithm-based-on-decomposition-moea-d>
22. Yarpiz. (2023). Multi-Objective Particle Swarm Optimization (MOPSO) [Software]. MATLAB Central File Exchange. Retrieved June 4, 2023, from <https://www.mathworks.com/matlabcentral/fileexchange/52870-multi-objective-particle-swarm-optimization-mopso>
23. Yarpiz. (2023). Non-dominated Sorting Genetic Algorithm II (NSGA-II) [Software]. MATLAB Central File Exchange. Retrieved June 4, 2023, from <https://www.mathworks.com/matlabcentral/fileexchange/52869-non-dominated-sorting-genetic-algorithm-ii-nsga-ii>
24. Yang, X.-S. (2012). Multi-objective Firefly Algorithm for Continuous Optimization. Engineering with Computers, 29(2), 175–184. <https://doi.org/10.1007/s00366-012-0254-1>
25. Yang, X. S. (2011). Bat Algorithm for Multi-Objective Optimization. International Journal of Bio-Inspired Computation, 3(5), 267. <https://doi.org/10.1504/ijbic.2011.042259>
26. Yarpiz. (2023). Strength Pareto Evolutionary Algorithm 2 (SPEA2) [Software]. MATLAB Central File Exchange. Retrieved June 4, 2023, from <https://www.mathworks.com/matlabcentral/fileexchange/52871-strength-pareto-evolutionary-algorithm-2-spea2>
27. Yang, X.-S., & Deb, S. (2013). Multi-objective Cuckoo Search for Design Optimization. Computers & Operations Research, 40(6), 1616–1624. <https://doi.org/10.1016/j.cor.2011.09.026>
28. Yang, X.-S., Deb, S., Fong, S., & Fong, K. (2013). Flower Pollination Algorithm: A Novel Approach for Multi-objective Optimization. Engineering Optimization, 46(9), 1222–1237. <https://doi.org/10.1080/0305215x.2013.832237>
29. Zervoudakis, K., & Tsafarakis, S. (2020). A mayfly optimization algorithm. Computers & Industrial Engineering, 145, 106559. <https://doi.org/10.1016/j.cie.2020.106559>
30. Jangir, P. (2023). MULTI OBJECTIVE NON SORTED MOTH FLAME (MOMFO) (NSMFO) matlab [Software]. MATLAB Central File Exchange. Retrieved June 4, 2023, from <https://www.mathworks.com/matlabcentral/fileexchange/75260-multi-objective-non-sorted-moth-flame-momfo-nsmfo-matlab>

# **APPENDIX**

## **MATLAB CODES**

The codes and results of this project are in the attached github repository: [mogakaowen/Final-Year-Project (github.com)](https://github.com/mogakaowen/Final-Year-Project)

The main codes are shown below:

### **CMRUN**

%CMRUN MATLAB Code

%Owen Mogaka Nyandieka, Department of Electrical and Information Engineering University Of Nairobi

function [Best\_Cost,Best\_X,Convergence\_curve]=CMRUN(nP,MaxIt,lb,ub,dim,fobj,ChaosVec)

% Define the number of objectives

nObj = 2;

X=initialization(nP,dim,ub,lb); % Initialize the set of random solutions

Cost=zeros(nP,nObj); % Record the Fitness of all Solutions

Convergence\_curve=zeros(nObj,MaxIt);

for i=1:nP

Cost(i,:) = fobj(X(i,:)); % Calculate the Value of Objective Function

end

cost=sort(Cost);

[~,ind] = min(cost(:,1)); % Determine the Best Solution

[Best\_Cost,~] = min(cost); % Determine the Best Cost

Best\_X = X(ind,:);

Convergence\_curve(1) = Best\_Cost(1); % Update the convergence curve with the minimum cost of the first objective

Convergence\_curve(2) = Best\_Cost(2); % Update the convergence curve with the minimum cost of the second objective

%% Main Loop of RUN

it=1;%Number of iterations

while it<MaxIt

it=it+1;

f=20.\*exp(-(12.\*(it/MaxIt))); % (Eq.17.6)

Xavg = mean(X); % Determine the Average of Solutions

SF=2.\*(0.5-rand(1,nP)).\*f; % Determine the Adaptive Factor (Eq.17.5)

%Update chaos parameter

chaotic\_param=ChaosVec(it);

for i=1:nP

[~,ind\_l] = min(cost(:,1));

lBest = X(ind\_l,:);

[A,B,C]=RndX(nP,i); % Determine Three Random Indices of Solutions

[~,ind1] = min(cost([A B C]));

Best\_Cost(1) = min(cost(:,1));

Best\_Cost(2) = min(cost(:,2));

% Determine Delta X (Eqs. 11.1 to 11.3)

gama = chaotic\_param.\*(X(i,:)-chaotic\_param.\*(ub-lb)).\*exp(-4\*it/MaxIt); % Modified using chaotic map

Stp= chaotic\_param.\*((Best\_X-chaotic\_param.\*Xavg)+gama); % Modified using chaotic map

DelX = chaotic\_param.\*(2\*chaotic\_param-1).\*abs(Stp); % Modified using chaotic map

% Determine Xb and Xw for using in Runge Kutta method

if cost(i,1)<cost(ind1,1)

Xb = X(i,:);

Xw = X(ind1,:);

else

Xb = X(ind1,:);

Xw = X(i,:);

end

SM = RungeKutta(Xb,Xw,DelX); % Search Mechanism (SM) of RUN based on Runge Kutta Method

L=rand<0.5;

Xc = L.\*X(i,:)+(1-L).\*X(A,:); % (Eq. 17.3)

Xm = L.\*Best\_X+(1-L).\*lBest; % (Eq. 17.4)

vec=[1,-1];

flag = floor(2\*rand(1,dim)+1);

r=vec(flag); % An Interger number

g = 2\*rand;

mu = 0.5+.1\*randn(1,dim);

% Improve the solutions

Xnew1 = X(i,:)+r.\*Xc.\*abs(Xm-X(i,:));

Xnew2 = Xnew1+rand(1,dim).\*SF(i).\*(Best\_X-X(i,:));

% Update the Solution

Xnew2 = max(Xnew2,lb);

Xnew2 = min(Xnew2,ub);

X(i,:) = Xnew2;

% Determine New Solution Based on Runge Kutta Method (Eq.18)

if chaotic\_param<0.5

Xnew = (Xc+r.\*SF(i).\*g.\*Xc) + SF(i).\*(SM) + mu.\*(Xm-Xc);

else

Xnew = (Xm+r.\*SF(i).\*g.\*Xm) + SF(i).\*(SM)+ mu.\*(X(A,:)-X(B,:));

end

% Check if solutions go outside the search space and bring them back

FU=Xnew>ub;FL=Xnew<lb;Xnew=(Xnew.\*(~(FU+FL)))+ub.\*FU+lb.\*FL;

CostNew=fobj(Xnew);

if CostNew<cost(i,1)

X(i,:)=Xnew;

cost(i,:)=CostNew;

end

%% Enhanced solution quality (ESQ) (Eq. 19)

if chaotic\_param<0.5

EXP=exp(-5\*chaotic\_param\*it/MaxIt);

r = floor(Unifrnd(-1,2,1,1));

u=2\*rand(1,dim);

w=Unifrnd(0,2,1,dim).\*EXP;

[A,B,C]=RndX(nP,i);

Xavg=(X(A,:)+X(B,:)+X(C,:))/3; %(Eq.19-1)

beta=rand(1,dim);

Xnew1 = beta.\*(Best\_X)+(1-beta).\*(Xavg); %(Eq.19-2)

for j=1:dim

if w(j)<1

Xnew2(j) = Xnew1(j)+r\*w(j)\*abs((Xnew1(j)-Xavg(j))+randn);

else

Xnew2(j) = (Xnew1(j)-Xavg(j))+r\*w(j)\*abs((u(j).\*Xnew1(j)-Xavg(j))+randn);

end

end

FU=Xnew2>ub;FL=Xnew2<lb;Xnew2=(Xnew2.\*(~(FU+FL)))+ub.\*FU+lb.\*FL;

CostNew=fobj(Xnew2);

if CostNew<cost(i)

X(i,:)=Xnew2;

cost(i,:)=CostNew;

else

if rand<w(randi(dim))

SM = RungeKutta(X(i,:),Xnew2,DelX);

Xnew = (Xnew2-rand.\*Xnew2)+ SF(i)\*(SM+(2\*rand(1,dim).\*Best\_X-Xnew2)); % (Eq. 20)

FU=Xnew>ub;FL=Xnew<lb;Xnew=(Xnew.\*(~(FU+FL)))+ub.\*FU+lb.\*FL;

CostNew=fobj(Xnew);

if CostNew<cost(i)

X(i,:)=Xnew;

cost(i,:)=CostNew;

end

end

end

end

% End of ESQ

%% Determine the Best Solution

if cost(i)<Best\_Cost

Best\_X=X(i,:);

Best\_Cost=cost(i);

end

end

% Save Best Solution at each iteration

Convergence\_curve(:,it) = Best\_Cost;

disp(['it : ' num2str(it) ', Best Cost 1 = ' num2str(Convergence\_curve(1,it)) ', Best Cost 2 = ' num2str(Convergence\_curve(2,it))]);

end

disp(['The best optimal values are Minimum Noise: ',num2str(Convergence\_curve(1,MaxIt)) ' Minimum Distortion: ' num2str(Convergence\_curve(2,MaxIt))]);

end

% A funtion to determine a random number

%with uniform distribution (unifrnd function in Matlab)

function z=Unifrnd(a,b,c,dim)

a2 = a/2;

b2 = b/3;

mu = a2+b2;

sig = b2-a2;

z = mu + sig .\* (2\*rand(c,dim)-1);

end

% A function to determine three random indices of solutions

function [A,B,C]=RndX(nP,i)

Qi=randperm(nP);Qi(Qi==i)=[];

A=Qi(1);

B=Qi(2);

C=Qi(3);

end

### **Circuit Design Simulation**

%Circuit Design MATLAB Code

%Owen Mogaka Nyandieka, Department of Electrical and Information Engineering University Of Nairobi

function CCTDesign=circuit(~)

% Define the component value ranges

Rin\_range = [5000, 15000]; % Input impedance (ohm)

Rc\_range = [500, 1500]; % Collector resistor (ohm)

RL\_range = [5000, 15000]; % Load resistor (ohm)

Re\_range = [500, 15000]; % Emitter resistor

fL\_range = [10000, 50000]; % Lower cutoff frequency (Hz)

fH\_range = [1000000, 20000000]; % Upper cutoff frequency (Hz)

Av\_range = [50, 200]; % Voltage gain

num\_tests = 100; % Number of random tests to perform

for test = 1:num\_tests

% Randomly select component values within the specified ranges

Rin = randi(Rin\_range);

Rc = randi(Rc\_range);

RL = randi(RL\_range);

Re = randi(Re\_range);

fL = randi(fL\_range);

fH = randi(fH\_range);

Av = randi(Av\_range);

% Calculate the biasing circuit

Vin = 0.1; % Input voltage (V)

Vcc = 12; % Power supply voltage (V)

Vb = Vcc\*Rc/(Rc+RL); % Bias voltage (V)

IcQ = Vcc/(2\*Rc); % Quiescent collector current (A)

Rb = (Vcc-Vb)/(Vin/IcQ - 1); % Base resistor (ohm)

% Calculate the component values

C1 = 1/(2\*pi\*fL\*Rin); % Input coupling capacitor (F)

C2 = 1/(2\*pi\*fH\*RL); % Output coupling capacitor (F)

Ce = (1 / (2 \* pi \* fL \* Re)); % Emitter capacitance (F)

Re = Vb/IcQ; % Emitter resistor (ohm)

beta = 100; % DC current gain

R1 = Rb/(beta+1); % Biasing resistor (ohm)

R2 = Rb-R1; % Biasing resistor (ohm)

% Create the circuit model

gm = IcQ/(26\*Re); % Transconductance (S)

rpi = beta/gm; % Input resistance (ohm)

Ro = RL; % Output resistance (ohm)

Cpi = 2.6e-12; % Input capacitance (F)

Cmu = 1.3e-12; % Miller capacitance (F)

Hfe = beta/(1+beta); % Small signal current gain

Av\_calc = -gm\*RL\*Ro/(1+gm\*Ro\*(Cpi+Cmu)\*Hfe); % Calculated voltage gain

% Calculating gain difference

desired\_gain = 20\*log10(Av); % desired gain in dB

actual\_gain = abs(20\*log10(Av\_calc));

%Bandwidth

Bw=fH-fL;

% Calculate distortion

Distortion = abs(100\*(1-Hfe)/(2\*Av\_calc));

% Calculate noise

kTq = 1.38e-23\*293; % Thermal voltage noise density at room temperature

In = 2\*beta\*kTq/IcQ; % Input noise density (A/sqrt(Hz))

En = In\*Rin; % Input-referred noise (V/sqrt(Hz))

Noise = abs(20\*log10(En/(0.1\*Av\_calc)));

CCTDesign=[Noise,Distortion];

end

%fprintf('Rin = %d ohm\n', Rin);

%fprintf('Rc = %d ohm\n', Rc);

%fprintf('Re = %d ohm\n', Re);

%fprintf('RL = %d ohm\n', RL);

%fprintf('fL = %d Hz\n', fL);

%fprintf('fH = %d Hz\n', fH);

end

### **CMRUN Optimization Function**

%CMRUN main Code

%Owen Mogaka Nyandieka, Department of Electrical and Information Engineering University Of Nairobi

clear

close all

clc

nP = 100; % Number of population

MaxIt = 1000; % Maximum number of iterations

lb = 0; % Lower bound of decision variables

ub = 100; % Upper bound of decision variables

dim = 50; % Number of decision variables

ChaosVec=zeros(10,MaxIt);

%Calculate chaos vector

for i=1:10

ChaosVec(i,:)=chaos(i,MaxIt,1);

end

fobj=@circuit;

[Best\_Cost,Best\_X,Convergence\_curve] = CMRUN(nP,MaxIt,lb,ub,dim,fobj,ChaosVec(1,:));

%% Draw objective space

%Pareto Front

figure()

plot(Convergence\_curve(1,:), Convergence\_curve(2,:), 'ko')

title('Pareto Front')

xlabel('Noise');

ylabel('Distortion');

weights = [0.5,0.5];

Fitness = ones(1,MaxIt);

for i=1:MaxIt

Fitness(1,:) = weights(1,1)\*Convergence\_curve(1,:) + weights(1,2)\*Convergence\_curve(2,:);

end

%Pareto Sum Fitness

figure()

plot(Fitness,'Color','b','LineWidth',2)

title('Chaotic Multi-Objective RUN Optimization Algorithm')

xlabel('Iterations');

ylabel('Pareto Sum Fitness');

grid on

box on

legend('CMRUN')