DEVELOPMENT OF A CULTURAL ALGORITHM BASED-ARTIFICIAL BEE COLONY FOR IMPROVED PROPORTIONAL-INTEGRAL-DERIVATIVE CONTROLLER PARAMETER TUNING

BY

Busayo ADEBIYI, B.Sc (ABU) 2010

M.Sc/ENG/32931/2012-2013

A DISSERTATION SUBMITTED TO THE SCHOOL OF POST GRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE (M.Sc) DEGREE IN CONTROL ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

FACULTY OF ENGINEERING

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

January, 2017

DECLARATION

I, ADEBIYI Busayo, hereby declare that the work in this Dissertation entitled “Development of a Cultural Algorithm Based-Artificial Bee Colony Algorithm for Improved Proportional-Integral-Derivative Controller Parameter Tuning” has been carried out by me in the Department of Electrical and Computer Engineering. The information derived from the literature has been duly acknowledged in the text and a list if references are provided. No part of this dissertation was previously presented for another degree at this or any Institution.

Busayo ADEBIYI _____ _____

Signature Date

CERTIFICATION

This dissertation titled "DEVELOPMENT OF A CULTURAL ALGORITHM BASED-ARTIFICIAL BEE COLONY FOR IMPROVED PROPORTIONAL INTEGRAL DERIVATIVE CONTROLLER PARAMETER TUNING**"** by BUSAYO ADEBIYI meets the regulations govering the award of the award of degree of Master of Science (M.Sc) in Control Engineering of the Ahmadu Bello University, Zaria and is approved for its contribution to knowledge and literary presentation.

Prof. M. B. Mu'azu		
(Chairman Supervisory Committee)	(Signature)	Date
Dr. A. M. S. Tekanyi		
(Member Supervisory Committee)	(Signature)	Date
Dr. Y. Jibril		
(Head of Department)	(Signature)	Date
Prof. S. Z. Abubakar		
(Dean School of Postgraduate Studies)	(Signature)	Date

DEDICATION

This research work is dedicated to all the victims of the insurgency that ravaged the North-Eastern part of Nigeria, which claimed over twenty thousand lives.

ACKNOWLEDGEMENT

In the name of Allah (SWT), the Most Gracious and Most Merciful. All praise is due to Allah (SWT), the Lord of the world, who honoured mankind and favoured them markedly above a great part of His creation. May His peace and blessing be upon His noble prophet Muhammed, his entire household, companion and those who follow him till the end of time (Amin).

This is neither the time nor place for me to do justice to the memory of my parents. The opportunity for that will come in the future. But I need to acknowledge them here and mention them so that it can be distinctly clear that they are the dominant influence on the way I view reality. May Almighty Allah grant them Aljanat firdaus (Amin).

I wish to express my profound gratitude to my supervisor Prof. M. B. Mu'azu for his tireless efforts, valuable guidance, and constant supervision towards the success of this work. I really appreciate and value your tolerance, guidance, and inspiration from the beginning to the end of this dissertation. Thank you very much Sir. My appreciations also go to my co-supervisor, Dr. A. M. S. Tekanyi for his meticulous approach and always creating time out of his busy schedule to go through the manuscript during all those stages of my presentations. Many thanks to Engr. A. T. Salawudeen for having assisted me shape the problem and providing insights towards the solution. My deep appreciation goes to the Control and Computer Research Group and all those Tuesday meetings, where academic freedom and discourse are at its best.

I acknowledge with thanks the entire staff of Electrical and Computer Engineering, Ahmadu Bello University, namely: Prof. B. G. Bajoga, Prof. B. Jimoh, Dr. T. H. Sikiru, Dr. S. M. Sani, Dr. A. D. Usman and Dr. I. J. Umoh, Dr. Y. Jibril, Dr. K. A. Abu-Bilal, Dr. Gbenga Olarinoye, Engr. M. J. Mu'azu, Engr. A. I. Abdullahi, Engr. Z. M. Abdullahi, Engr. B. O. Sadiq, Engr. Adamu and most especially, those whose names could not be mentioned. My deepest

appreciation also goes to Abdullahi Tukur for all the administrative support towards me and the entire students of Department of Electrical and Computer Engineering.

My deepest appreciation goes to my course mate and beloved friends: Ndubusi, Abdulmumini, Umar Abubakar, Gbenga Adeniyi, Valentine Ikpo, Suleiman Hussein, Abdulrahman Olaniyan, Bukola Ishola, Tijesunimi Alonge, Maryam Imam, Aliyu Daniel, Stephen Soja, Atuman G. Joel, Ajayi Ore-Ofe, Wasiu Ajadi, Layi Oni, Suraj Oyewale, Mustapha Kabir, Suraj Ajibosin, Lanre Yusuf, Teslim Mohammed, Umar Ibrahim and those whose names could not be mentioned. Their continuous support and contributions towards the success of this research would never be forgotten. May God reward them and strengthen our friendship.

To my 'alter ego' Risikat Folashade O. Adebiyi, I am highly grateful for the patience, perseverance, and understanding you exhibited, especially, during those tough periods. May Allah continue to strengthen you in faith and wisdom (Amin). Also to my priceless gift; Muhammad El-Amin Adebiyi, the sky is just the beginning.

I am highly indebted to my siblings Asiwaju, Brother Tunde, Biodun, Oke, Sister Funmi, Funke and Aisha, your inestimable care and support, kindness and understanding personality will always be treasured. Thank you very much. Many thanks to Prof. L. O. Onundi's family, Alh Bashir Adeshina's family, Bar. W. K. Shittu's family, Mal Abdulaziz Nasir, Hajia Latifat Adebiyi, Mr. Rasaq Yusuf, Mr. Gbola Raji and Engr Mohammed Abdulsalam and to everyone who contributed to the success of this work. May God reward all of you abundantly (Amin).

Busayo ADEBIYI

January, 2017.

ABSTRACT

This research is aimed at the development of a cultural-algorithm based artificial bee colony algorithm (CABCA) for improved Proportional Integral Derivative (PID) controller parameters tuning. The normative and situational knowledge inherent in cultural algorithm were utilized to guide the step size as well as the direction of evolution of Artificial Bee Colony (ABC) at different configurations, in order to combat the disparity between exploration and exploitation associated with the standard ABC, which results to poor convergence and optimization efficiency. Four variants of CABCA (CABCA(Ns), CABCA(Sd), CABCA(Ns+Sd) and CABCA(Ns+Nd)) were accomplished in MATLAB R2015a using different configurations of cultural knowledge. A total of 20 standard applied mathematical optimization test functions (Ackley, Michalewicz, Quartic, Sphere etc) were employed to evaluate the performance of each CABCA variant. The results indicated that CABCA(Ns) performed best in 4 test functions (20%), CABCA(Ns+Nd) also in 4 functions (20%), while CABCA(Sd) and CABCA(Ns+Sd) performed best in 3 test cases (15%) and 2 test cases (10%) respectively. On the remaining 7 test functions (35%) of their results were similar. The CABCA(Ns) was chosen as the best performed variant based on the success ratio, which is the number of successful runs that found the solution. Hence, CABCA(Ns) was used to obtain the optimal parameters K_p, K_i and K_d of a PID controller which was employed in the speed control of a DC motor. The DC motor attained steady-state in 0.4178s with the CABCA-based PID controller as against 0.6778s and 2.2057s obtained using standard ABC and Ziegler-Nichols (Z-N) tuned PID controllers, respectively.

TABLE OF CONTENTS

COVER PAGE……i

DECLARATION……...ii

CERTIFICATION……iii

DEDICATION.....iv

ACKNOWLEDGEMENT.....v

ABSTRACT.....vii

LIST OF FIGURES xii

LIST OF TABLES.....xiv

LIST OF ABBREVIATIONS.....xv

CHAPTER ONE: INTRODUCTION

1.1 Background 1

1.2 Motivation……..…...3

1.3 Significance of Research …… 4

1.4 Statement of Problem …… 4

1.5 Aim and Objectives.....5

1.6 Dissertation Organization.....5

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction 7

2.2 Review of Fundamental Concepts.....7

2.2.1 The DC Motor 7

2.2.2 PID Controller 9

2.2.3 PID tuning.....11

2.2.4 Artificial Bee Colony Optimization 16

2.2.5 Culture algorithm.....22

2.2.6 Optimization Benchmark Functions 27
2.3 Review of Similar Works 35

CHAPTER THREE: METHODS AND MATERIALS

3.1 Introduction 42
3.2 Methodology 42
3.3 Initialization of ABC and the Parameters 43
3.4 Replication of Standard ABC Algorithm 44
3.4.1 Assigning the employed bees to the food sources 44
3.4.2 Employing of onlookers and calculation of probabilities 45
3.4.3 Scout bee exploring new food source 45
3.4.5 Stoping Condition 45
3.5 Development of Cultural Artificial Bee Colony Algorithm 46
3.5.1 CABCA Knowledge adjustment 47
3.5.2 CABCA variants 47
3.6 Performance Evaluation 52
3.7 Determination of PID Gains Using CABCA 52

CHAPTER FOUR: RESULTS AND DISCUSSIONS

4.1 INTRODUCTION 57
4.2 Initialization results of the parameters 57
4.3 Results of the standard ABC 58
4.4 Results of the developed CABCA 69
4.5 Test Functions Visualization 63
4.5.1 Ackley Function 63
4.5.2 Axis parallel Hyper-Ellipsoid Function 63
4.5.3 Cosine Mixture (CM) Function 64
4.5.4 De JongF4 Function 64
4.5.5 Exponential Function 65

4.5.6 Griewangk function.....65
4.5.7 Hyperelliptic Fuction 66
4.5.8 Michalewicz Fuction.....66
4.5.9 Neumaier3 Function.....67
4.5.10 Penalized Levy and Montalvo 1 Function 67
4.5.11 Penalized Levy and Montalvo 2 Function 68
4.5.12 Quartic Function 69
4.5.13 Rastrigin Function.....69
4.5.14 Rosenbrock Function 70
4.5.15 Sal Function 70
4.5.16 Schaffer Function.....70
4.5.17 Schwefel Function.....71
4.5.18 Sphere Function 72
4.5.19 Step Function 72
4.5.20 Zakharov Function……73
4.6 Performance, Evaluation and Comparison of ABC and CABCA 73
4.6.1 Performance evaluation of CABCA(Ns) using normative knowledge. 73
4.6.2 Performance Evaluation of CABCA(Sd) Using Situational Knowledge. 75
4.6.3 Performance Evaluation of CABCA(NsSd) using Situational and Normative.....77
4.6.4 Performance Evaluation of CABCA(Ns+Nd) using Normative Knowledge 79
4.7 Computer Specification 81
4.8 Application of the Proposed Method for Optimal Determination of PID Parameters.....81

CHAPTER FIVE: **CONCLUSION AND RECOMMENDATION**

5.1 Conclusion 84
5.2 Significant Contributions ……. 85
5.3 Limitations……..……85
5.4 Recommendation for Further Work……...85

REFERENCES……86
APPENDICES……...90
APPENDIX A
M-File Code of Artificial Bee Colony (ABC) Algorithm……....90
APPENDIX B1
M-File Code of CABCA(Ns)……94
APPENDIX B2
M-File Code of CABCA(Sd)……98
APPENDIX B3
M-File Code of CABCA(Ns+Sd)……102
APPENDIX B4
M-File Code of CABCA(Ns+Nd)……107
APPENDIX C
M-File Code of the Objective function……112
APPENDIX D
M-File Code of the 3D plots……121
APPENDIX E
M-File Code for K_p, K_i and ……126

LIST OF FIGURES

Figure 2.1: The Structure of a DC motor……8

Figure 2.2: Mathematical model of DC motor……8

Figure 2.3: PID Controlled System……..…10

Figure 2.4: Flowchart of Artificial Bee Colony (ABC) algorithm……..21

Figure 2.5: Framework of Cultural Algorithm…….....23

Figure 3.1: Flowchart of Cultural Artificial Bee Colony Algorithm (CABCA)…….….51

Figure 3.2: PID tuning system with CABCA……53

Figure 3.3: Simulink Model of the Entire System……54

Figure 3.4: PID Controller Tuning Configuration Interface…….55

Figure 3.5: Block Representation of CABCA Based PID Controller Design……...56

Figure 4.1: 3D Visualization of Ackley Function……63

Figure 4.2: 3D Visualization of Axis Parallel Function……..64

Figure 4.3: 3D Visualization of Cosine Mixture Function……..64

Figure 4.4: 3D Visualization of DejongF4 Function……65

Figure 4.5: 3D Visualization of Exponential Function……65

Figure 4.6: 3D Visualization of Greiwangk Function……..66

Figure 4.7: 3D Visualization of Hyperelliptic Function……...66

Figure 4.8: 3D Visualization of Michalewicz Function……...67

Figure 4.9: 3D Visualization of Neumaier Function……67

Figure 4.10: 3D Visualization of PM1 Function……..68

Figure 4.11: 3D Visualization of PM2 Function……..68

Figure 4.12: 3D Visualization of Quartic Function……..69

Figure 4.13: 3D Visualization of Rastrigin Function……...69

Figure 4.14: 3D Visualization of Rosenbrock Function……..70

Figure 4.15 3D Visualization of Sal Function…….70

Figure 4.16: 3D Visualization of Schaffer Function……...71

Figure 4.17: 3D Visualization of Schwefel Function……..71

Figure 4.18: 3D Visualization of Sphere Function……..72

Figure 4.19: 3D Visualization of Step Function……..72

Figure 4.20: 3D Visualization of Zakharov Function……..73

Figure 4.21: Step Response of the System……..…….82

LIST OF TABLES

Table 4.1: Parameter Setting for ABC……..……..……..57

Table 4.2: Results of ABC……. ……..…..……..58

Table 4.3: Results of CABCA(Ns)……..……59

Table 4.4: Results of CABCA(Sd)……..……60

Table 4.5: Results of CABCA(Ns+Sd)……61

Table 4.6: Results of CABCA(Ns+Nd).……..……..…62

Table 4.7: Performance Evaluation of CABCA(Ns) and ABC……74

Table 4.8: Performance Evaluation of CABCA(Sd), CABCA(Ns) and ABC……76

Table 4.9: Performance Evaluation of CABCA(NsSd), CABCA(Sd), CABCA(Ns) and ABC……...……...……78

Table 4.10: Performance Evaluation of CABCA(Ns+Nd), CABCA(Ns+Sd), CABCA(Sd), CABCA(Ns) and ABC……..79

Table 4.11: Computer system specification…….81

Table 4.12: Performance Comparison of Z-N, ABC and CABCA……..83

LIST OF ABBREVIATIONS

Acronyms	Meanings
ABC	Artificial Bee Colony Algorithm
ACO	Ant Colony Optimization
AFSA	Artificial Fish Swarm Algorithm
BFA	Bacterial Foraging Algorithm
BLF	Belief Space
CA	Cultural Algorithm
CABC	Chaotic Artificial Bee Colony Algorithm
CABCA	Cultural Artificial Bee Colony Algorithm
CABCA(Ns)	Cultural Artificial Bee Colony Algorithm (Normative Knowledge for step size)
CABCA(Sd)	Cultural Artificial Bee Colony Algorithm (Situational Knowledge for Visual distance)
CABCA(Ns+Sd)	Cultural Artificial Bee Colony Algorithm (Normative Knowledge for Step Size and Situational Knowledge for Visual Distance)
CABCA(Ns+Nd)	Cultural Artificial Bee Colony Algorithm (Normative Knowledge for Step Size and Normative Knowledge for Visual Distance)
CABCA-PID	Cultural Artificial Bee Colony Algorithm Tuned Proportional Integral Derivative.
CM	Cosine Mixture
CPU	Central Processing Unit
DC	Direct Current

DE	Differential Evolution
EP	Evolutionary Programming
FA	Firefly Algorithm
FOPID	Fractional Order Proportional Integral Derivative
GA	Genetic Algorithm
G_{min}	Global Minimum
MATLAB	Matrix Laboratory
MCN	Maximum Cycle Number
OS	Operating System
PID	Proportional Integral Derivative
POP	Population
PM	Penalized Levy and Montalvo
PS	Population Space
PSO	Particle Swarm Optimization
RAM	Read Access Memory
SA	Simulated Annealing
SMIB	Single Machine Define Bus
SN	Population Size

CHAPTER ONE

INTRODUCTION

1.1 Background

In the presence of modern controllers, the prorpotional-integral-derivative (PID) controllers are still widely used in many industrial control systems. The architectural simplicity, the availability of a large number of highly efficient, reliable, and cost-effective commercial PID control modules, and their acceptance from the operators are among the reasons for which makes PID controllers popular. The process of determining these actions to achieve consistent performance specification is known as "Controller Tuning". A proportional controller has the effect of reducing the rise time, but never eliminates the steady-state error. If the proportional gain K_p is too high, the system can become unstable whereas a small gain results in a small output response to a large input error, and a less responsive or less sensitive controller. An integral controller has the effect of eliminating the steady-state error, but it may make the transient response worse. High integral gain K_i can cause overshoot and low value will make the system sluggish. A derivative controller has the effect of increasing the stability of the system by reducing the overshoot, and improving the transient response. If the derivative gain K_d is sufficiently large it can cause a process to become unstable. These parameters $\left(K_p, K_i \,\&\, K_d\right)$ must be optimally selected such that a closed loop system has to give a desired response. Moreover, the operation of the parameter determination must be performed as fast as possible for a given process (Åström and Hägglund, 2004). In control system applications, the chosen performance criterion is often a weighted combination of various performance characteristics such as rise time, settling time, overshoot, and steady state error. The desired

system response should have minimal settling time with a small or no overshoot in the step response of the closed loop system.

The conventional tuning methods sometimes fail to achieve satisfactory performance because many industrial plants are often burdened with problems such as high orders, time delays and nonlinearities (Yan and Li, 2011). Among the conventional PID tuning methods, Ziegler–Nichols approach has been used for years, especially when little or no information about the plant under control is provided. For a wide range of practical processes, this tuning approach works quite well. However, sometimes it does not provide good tuning and tends to deviate from the desired performance. (Bassi *et. al.*, 2011). Therefore, this method usually needs re-tuning before it is applied to control industrial processes.

With advances in computational methods, several intelligent optimization algorithms have been proposed for tuning the PID control parameters in order to get an optimal performance (Yan and Li, 2011). To enhance the capabilities of traditional PID parameter tuning techniques, several intelligent approaches have been suggested. For example, Genetic Algorithm (GA) based PID tuning was proposed by (Bagis, 2007) and the use of Ant Colony optimization (ACO) for PID tuning was also proposed by (Jones and Bouffet, 2008). The Particle Swarm Optimization (PSO) based PID tuning was presented in (Solihin *et. al.*, 2011). Also the use of Artificial Bee Colony (ABC) algorithm for PID parameter tuning was presented in (Varma and Kumar, 2013). ABC uses minimum number of algorithm parameters (Population size (SN), Maximum Cycle Number (MCN) and Limit) than most heuristic algorithms. From the responses of ABC, it is observed that it performs better than other heuristic techniques such as GA, ACO and PSO (Kaliappan and Thathan, 2015).

However, due to the risk of premature convergence and fitness stagnation which usually occur as a result of imbalance between explorations and exploitations, the standard ABC tends to get

stuck in local minima and may take a long time to converge (Shapla *et. al.*, 2015). To address these shortcomings, a proper guiding parameter is required. Therefore, this research proposes an improved ABC algorithm called Cultural Artificial Bee Colony Algorithm (CABCA) for PID parameter tuning.

Artificial Bee Colony algorithm is a global optimization algorithm which is motivated by the foraging behaviour of swarms of honey bee. In this algorithm, the bee swarm mainly consists of three components: the employed bees, the onlooker bees, and the scout bees. Employed bees are responsible for exploiting the nectar sources and take the information obtained about the quality of the food source locations which they are exploiting to the waiting bees (onlooker bees). The scout bee is responsible for random searching of new food sources (Karaboga and Basturk, 2008).

Cultural Algorithm models two levels of evolution: social population level and belief space level. The belief space is the place where the information on the solution of the problem is developed and stored. The experience of the selected individuals from the population contributes to the cultural knowledge by means of acceptance function. In return, this problem solving knowledge can bias the evolution of the population components by means of the influence function (Reynolds and Jin, 1999). This research focuses on modifying the ABC, using the cultural algorithm knowledge to accelerate the problem solving process during evolution of ABC.

1.2 Motivation

The PID controllers are the most popular controllers used in industry because of their remarkable effectiveness, simplicity of implementation and broad applicability. The main problem of PID is the selection of the three coefficients such that these coefficients readily translate into the desired performance and robustness characteristics that the control system

designer has in mind. However, the classical tuning method (such as Ziegler Nichols) is tedious and generally lead to poor performance as a result of plant parameter changes. However, using intelligent method (such as ABC), if appropriate initial parameters and proper guiding conditions are not sufficient, this algorithm may get trapped in local minima and may take too long to converge. In this research, the individual experience which consists of (step size and direction of change) will be employed through cultural evolution process to propose a new variant of ABC called the CABCA. This approach has superior features, which include easy implementation, fast convergence characteristic and good computational efficiency over the classical method Z-N and the standard ABC.

1.3 Significance of Research

The goal of tuning the PID controller is to determine parameters that meet closed loop system performance specifications, and to improve the robust performance of the control loop over a wide range of operating conditions. Practically, it is often difficult to simultaneously achieve all of these desirable qualities. For example, if the PID controller is adjusted to provide better transient response to set point change, it usually results in a sluggish response when under disturbance conditions (Bassi et al., 2011). This dissertation presents a novel intelligent method of cultural artificial bee colony algorithm (CABCA) for tuning the optimal proportional-integral derivative (PID) controller parameters.

1.4 Statement of Problem

PID controller is one of the most popular controllers which are still widely used in industries. One of the major challenges faced with PID controller design is selecting the appropriate gain $\left(K_p, K_i \,\&\, K_d\right)$ such that the controller behaviour yields the optimal results with respect to

design specification. Several intelligent techniques (such as ABC) have been proposed for selecting these PID parameters with little or no designer intervention. However, if appropriate initial parameters and proper guiding conditions are not sufficient, this algorithm may get trapped in local minima and may take too long to converge. In order to combat these shortcomings, a cultural-based Artificial Bee Colony Algorithm (CABCA) is developed.

1.5 Aim and Objectives

The aim of this research is to develop a cultural-algorithm based artificial bee colony algorithm called the Cultural Artificial Bee Colony Algorithm (CABCA) for improved proportional-integral-derivative (PID) controller parameter tuning.

To accomplish the above aim, the following objectives will be employed:

1) Replication of the standard ABC algorithm and development of the variants of CABCA in MATLAB R2015a using both situational and normative knowledge.
2) Evaluation of the performances of the developed CABCA variants in 1) using 20 standard applied mathematical optimization test functions (e.g. Schwefel, Schaffer, DeJongF4, etc.) with a view to obtain the best variant.
3) Application of the best variant in the PID parameter tuning for the control of DC motor model (developed in El-Telbany (2013)) and comparing with results obtained using the standard ABC.

1.6 Dissertation Organization

The general introduction has been presented in Chapter One. The rest of the chapters are structured as follows: First, detailed review of related literature and relevant fundamental concepts about DC motor, PID Controller, and PID Tuning. Also, classical and intelligent

methods of tuning PID controller, Artificial Bee Colony (ABC) Algorithm, Cultural Algorithm (CA) and optimization test functions are carried out in Chapter Two. Second, detailed approach and relevant mathematical models describing the development of Cultural Artificial Bee Colony Algorithm (CABCA) are presented in Chapter Three. Third, the optimization test functions visualization, analysis, performance and discussion of the results are evaluated in Chapter Four. Finaly, conclusion and recommendations of further works makes up the Chapter Five. The list of cited references and Appendices are also provided at the end of this dissertation.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The literature review comprises of overview of the fundamental concepts and the review of similar works. In the review of fundamental concepts, most of the existing works and the fundamental theories of all the algorithms that will be used for the success of this research are reviewed, after which similar works will be reviewed.

2.2 Review of Fundamental Concepts

In this section, concepts fundamental to optimization of PID controller, PID tuning methods, Artificial Bee Colony and Cultural algorithm amongst others are reviewed.

2.2.1 The DC Motor

The Direct Current (DC) motor drives are widely used in applications requiring adjustable speed, good speed regulations and frequent starting, braking and reversing. For example, if we have DC motor in radio controller car, if we just apply a constant power to the motor, it is impossible to maintain the desired speed. It will go slower over rocky road, slower uphill, faster downhill and so on. DC motor plays a significant role in modern industry. It is commonly used as an actuator in control system. As an actuator, the DC motor is influenced by the rest of the components in control system, and in turn influences the behaviour of the control system (Dorf and Bishop, 2011). There are numerous applications where control of speed is required, as in rolling mills, cranes, hoists, elevators, machine tools, transit system and locomotive drives. These applications may demand high-speed control accuracy and good dynamic responses. To reduce the loading effect and minimize time delay, a PID controller is added. The position of the motor is the rotation of the motor shaft or the degree of the rotation which is to be controlled by giving the feedback to the controller which rectifies the controlled output to achieve the

desired position. The electric circuit of the armature and the free body diagram of the rotor are shown in Figure 2.1 (El-Telbany, 2013).

$$iR + L\frac{di}{dt} = V_r$$

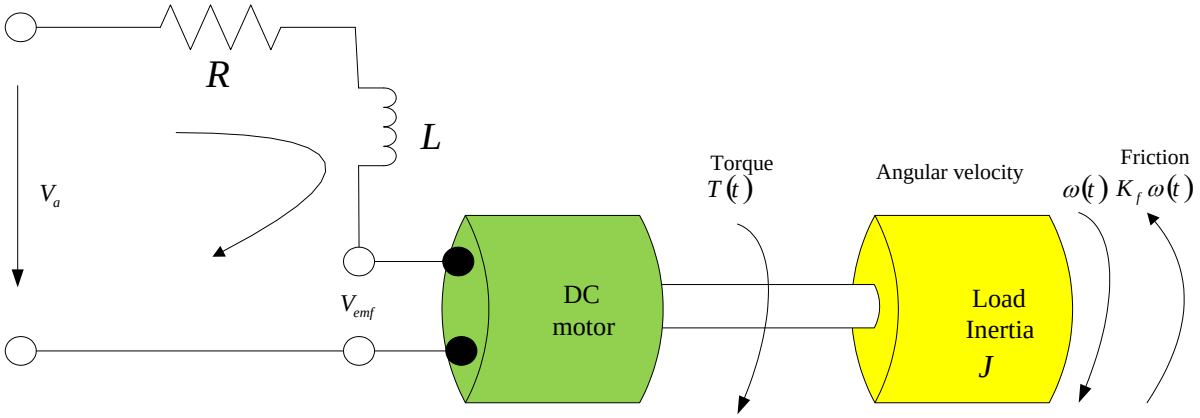


Figure 2.1: The Structure of a DC motor (El-Telbany, 2013)

The applied voltage V_a controls the angular velocity $\omega(t)$. The relationships for the armature controlled DC motor are shown schematically in Figure 2.2 (El-Telbany, 2013).

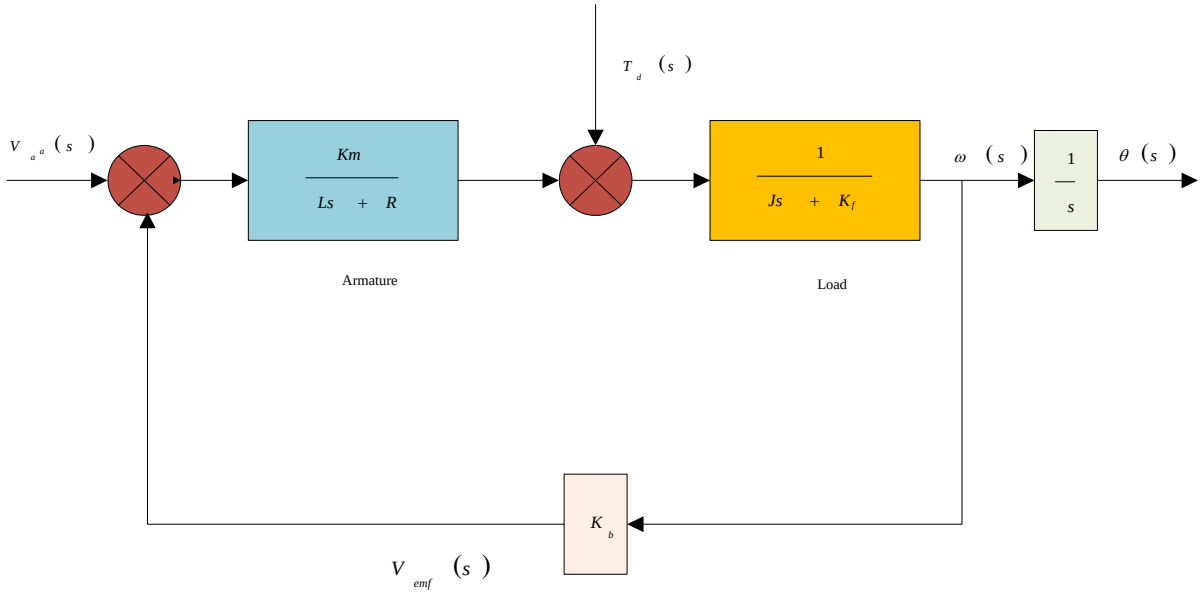


Figure 2.2: Mathematical Model of DC Motor (El-Telbany, 2013a)

The transfer function (with $T_d(s)$) has the form (Varma and Kumar, 2013):

$$G(s)=\frac{\theta(s)}{V_a(s)}=\frac{K}{(sL_a+R)(Js+K_f)+K_bK_m} \tag{2.1}$$

where $\theta(s)=$ angular position of rotor shaft (rad), $V_a(s)=$ armature voltage (V), $R=$ armature resistance (Ω), $L=$ armature inductance (H), $J_s=$ rotor inertia (kgm^2), $K_m=$ toque constant (Nm/A), $K_f=$ viscous friction coefficient (Nm/rad) and $K_b=$ black emf constant (Vs/rad)

.

Note: It is assumed that at equilibrium, the electrical power is equal to the power developed by the rotor, i.e. $K_b = K_m$

2.2.2 PID Controller

The PID controller is viewed as the backbone of the process industry. In spite of the presence of modern controllers, many industrial processes are controlled using (PID) controllers. The popularity of the PID controllers can be attributed to their good performance in a wide range of operating conditions, simplicity with few parameters; which allows Engineers to operate them in a simple, straightforward manner and robustness; which provides a good behaviour despite variation in the plant parameters (Bassi *et. al.*, 2011).

The PID controller has three separate parameters, and it is therefore sometimes called the three-term control: the proportional (P), the integral (I) and derivative (D) values. The proportional value determines the reaction to the current error, the integral value determines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element (Bansal *et. al.*, 2012). By tuning these three parameters, the controller can provide control action designed for specific process requirements.

In many industrial process control requirements, only proportional control is unsatisfactory since the offset cannot be tolerated. PID is satisfactory for some dynamics that are second or higher order (Åström, 2000). However, if the process has a large time delay, the derivative action does not seem to help much. Hence, the PID controller finds it difficult to control processes of this nature, and generally a more sophisticated controller such as a dead time compensator or a predictive controller is required. Processes that are highly under damped with complex conjugate poles close to the imaginary axis are also difficult to control with a PID controller. Processes with this type of dynamic characteristics are rare in the chemical processing industries.

A typical closed loop system controlled by PID controller is given in Figure 2.3 (El-Telbany, 2013).

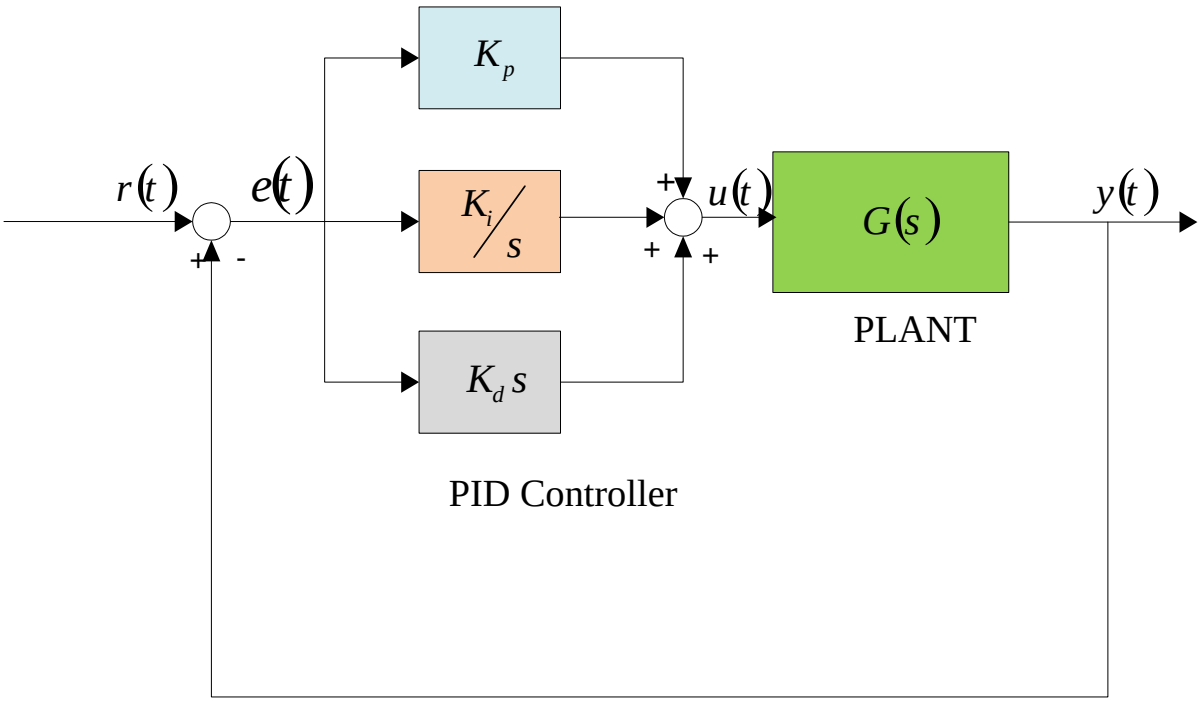


Figure 2.3: PID Controlled System (El-Telbany, 2013)

From Figure 2.3 $r(t)$ is the reference input to the controller, $u(t)$ is the controller output and $e(t)$ is a measured error (which is the difference between $r(t)$ and $u(t)$). PID is the controller

whose parameters $(K_p, K_i \ \& \ K_d)$ needed to be determined optimally. While plant is the system that needs to be controlled which will be a DC motor in the case of this research.

In practice, the output of a PID controller is given by (Bansal *et al*., 2012):

$$u(t) = K_p \times e(t) + K_i \times \int_0^t e(t)dt + K_d \times \frac{de}{dt} \tag{2.2}$$

K_i and K_d can be written as:

$$K_d = K_p \times T_d, \qquad K_i = K_p \times \frac{1}{T_i}$$

$$u(t) = K_p\left[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d \frac{de(t)}{dt} \right] \tag{2.3}$$

$$G(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{S} + K_d \tag{2.4}$$

$$G(s) = K_p\left(1 + \frac{1}{T_i S} + T_d S \right) \tag{2.5}$$

In equations (2.2) - (2.5), K_p is the proportional tuning constant, K_i is the integral tuning constant, K_d is the derivative tuning constant, T_i is the integral time, T_d is the derivative time, $G(s)$ is the transfer function of the controller and error $e(t)$ is the difference between the set-point and the process variable at a time called error signal.

2.2.3 PID tuning

Tuning of a PID controller is a process of determining the parameters ($K_p, K_i \ \& \ K_d$) for a given system. The performance of a control system may be poor and unstable if improper values of the controller tuning constants are used. So it is necessary to tune the controller parameters to achieve a good control performance with the proper choice of tuning constants (Visioli, 2006).

There have been various types of methods applied for PID tuning. These methods can be broadly classified as classical and intelligent computational techniques.

A. Classical Methods

Classical methods make certain assumptions about the plant and the desired output and try to obtain analytically, or graphically some features of the process that is then used to decide the controller settings. These methods are computationally very fast and simple to implement, and are good with few iteration (Bansal *et. al.*, 2012). But due to the assumptions made, the controller settings usually do not give the desired results directly and further tuning is required. A few classical methods are discussed as follows:

I. Direct Synthesis Method

This tuning technique is model based. The specified technique uses an identified process model in conjunction with a user specified closed-loop response characteristic. A positive aspect to this approach is that it provides insight on the role of the "model" in control system design. A limitation to this approach is that a PID controller may not be realized unless an appropriate model form is used to synthesize the control law (Kumar *et. al.*, 2015).

II. Zeigler Nichols (Z-N) method

This is regarded as the most popular tuning methodology (Ogata, 2009). There are two methods proposed by Ziegler and Nichols. The first method of Z-N tuning is based on the open-loop step response of the system. The open loop system's S-shaped response is characterized by the parameters, namely the process time constant T and L. These parameters are used to determine the controller's tuning parameters. The second method of Z-N tuning rule is based on critical gain (K_u) and critical period (P_u). In this method, the integral time (T_i) will be set to infinity

and the derivative time (T_d) to zero. In this method, only the proportional control action is used. The K_p is increased to a critical value K_u at which the system output will exhibit sustained oscillations. If the system output does not exhibit the sustained oscillations, Ziegler Nichols method does not apply (Varma and Kumar, 2013). This method is good to get the initial PID setting of the system. However, there is always a requirement for further tuning; because the controller settings derived are rather aggressive and thus result in excessive overshoot and oscillatory response (Bansal *et. al.*, 2012).

B. Intelligent Methods

In view of the aforementioned challenges encountered by the classical methods, and to enhance the capabilities of the classical PID parameter tuning techniques, intelligent methods have been suggested to obtain the PID parameters. Some intelligent methods are discussed as follows:

I. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a popular optimization algorithm that was proposed by Eberhart and Kennedy in 1995. In the algorithm, there is a population of particles which move through the solution space to find the optimal. In PSO method the system keeps a track of the best solution obtained till now and each individual particle keeps a track of its own individual best solution. Based on these two, each particle moves to a new position decided by a velocity and its current position. The velocity is dependent on the global and particle's best solution. As indicated in the work of (El-Telbany, 2007), if the i^{th} particle of the swarm is represented by the D–dimensional vector $X_i = (x_{i1}, x_{i2,}...., x_{iD})$ and the best particle in the swarm. That is the particle with the smallest function value, is denoted by the index g. The best previous position (the position giving the best function value) of the i^{th} particle is recorded and represented as

$P_i = (p_{i1}, p_{i2}, ..., p_{iD})$, and the position change (velocity) of the i^{th} particle is $V_i = (v_{i1}, v_{i2}, ..., v_{iD})$. The particles are manipulated according to the equations;

$$v_{id} = w \cdot v_{id} + c_1 \cdot r_1(p_{id} - x_{id}) + c_2 \cdot r_2(p_{gd} - x_{id}) \quad (2.6)$$

where $d = 1,2,...D$; $i = 1,2,.., N$ and N is the size of population; w is the inertia weight; c_1 and c_2 are two positive constants r_1 and r_2 are random values in the range [0, 1].

In the work of Jones and Bouffet, (2008), PSO was used in the tuning of PID. The result was compared with that of ant colony optimization (ACO) and ABC. The ABC had the best performance in terms of (rise time and settling time) and the best overshoot was in case of PSO in the three cases.

However, in (Kaliappan and Thathan, 2015) a PID controller tuning based on the enhanced Artificial Bee Colony (E-ABC) algorithm was presented. The result was compared with Genetic Algorithm, Simulated Annealing (SA), and ABC; the ABC provided a better tracking and improved closed loop accuracy.

II. Ant Colony Optimization

Ant Colony Optimization (ACO) is another meta-heuristic approach to solve optimization problems based on working relationship of ants in their colony. More precisely, it is based on the ant colony finding the shortest path to the food. Each ant tries to find the food through some random path leaving behind a trail of pheromones. The pheromone trail weakens with reducing number of ants passing through that path and strengthens with increasing number of ants passing through it. So basically it is a search algorithm which depends on a number of ants acting together moving towards the optimal solution (Dorigo and Stützle, 2010).

(Pham *et. al.*, 2006) demonstrated the use of ABC to tune a PID controller and solving complex systems. The results of ACO, PSO and ABC were compared and presented in (Jones and Bouffet, 2008).

III. Artificial Bee Colony

Artificial Bee Colony (ABC) is a global optimization algorithm which is motivated by the foraging behaviour of honey bee swarms. It was proposed by Dervis Karaboga in 2005. Standard ABC has the advantages of strong robustness, fast convergence and high flexibility with fewer setting parameters. In comparison to other greedy and local search based algorithms, ABC shows a very competitive and often better performance in comparison to many other existing evolutionary and swarm intelligence algorithms such as genetic algorithm (GA), differential evolution (DE), ant colony optimization (ACO) and particle swarm optimization (PSO) (Karaboga and Basturk, 2008).

However, it is still possible for the evolving population of candidate solutions to lose its diversity and explorative search capability too soon. This leads the candidate solutions to prematurely get trapped around the local optima of the search space. Also, another problem that is faced by the ABC algorithm is fitness stagnation, where all the candidate solutions fail to improve their fitness values for indefinitely prolonged iterations, for no obvious reason and even without any premature convergence around the local optimal points. The risk of premature convergence and fitness stagnation usually occur with reduced explorations and increased exploitations. But increasing the explorations may lead to unacceptably slow convergence speed (Shapla *et. al.*, 2015). In this work, the knowledge (situational and normative) in cultural evolution was employed to serve as guiding parameters. This will serve as an adaptive and balanced mix of explorations and exploitations which is necessary for good results and sufficient convergence speed of the algorithm.

2.2.4 Artificial Bee Colony Optimization

Similar to PSO, Artificial Bee Colony (ABC) algorithm has been proposed for PID parameters tuning (Varma and Kumar, 2013). Artificial Bee Colony algorithm is a recently introduced swarm intelligence optimization algorithm that tries to mimic the intelligent food foraging behaviour of honey bees. Since its introduction by Dervis karaboga in 2005 to solve numerical optimization problems, it has also been applied to many kinds of optimization problems (Akay & Karaboga, 2009; Tsai *et al*, 2009; Karaboga & Ozturk, 2010; Yeh & Hsei, 2010; Yan & Li, 2011; Ozkis & Babalik, 2014, Sulaiman *et al*, 2013; Sonmez *et al*, 2015). Many optimization researchers have shown their interest in ABC since it has demonstrated efficiency in solving various optimization problems as well as demonstrated excellent performance in comparison with other prominent optimization algorithms such as Genetic Algorithm (GA), Differential Evolution (DE), Particle Swarm Optimization (PSO) algorithms and few others. Apart from that, its control parameters are easy to tune and its implementation is simple and flexible (Karaboga and Akay, 2009).

The honey bee system can be classified into two components: food sources and foragers.

1) Food Sources: The value of a food source depends on many factors such as its proximity to the nest, its richness or concentration of its energy, and the ease of extracting this energy. The progress and adaptation of the bees in their colony dependents upon the speedy discovery and efficient utilization of the best food resources (Karaboga, 2005).
2) Foragers: Foraging is the swarm behaviour of a bee towards the food source. The foragers serve as carrier of information about a particular source, its distance and direction from the nest, the profitability of the source and communicate this information with a certain probability of its energy, as well as the ease of extracting this energy. There are two types of foragers; employed foragers (associated with a particular food source which they are

currently exploiting) and unemployed foragers (they are not associated with a particular food source; hence they continue to look for food to be exploited) (Karaboga, 2005).

The basic ABC Algorithm consists of three kinds of bees; employed bees, onlooker bees and scout bees. Employed bees are responsible for exploiting the nectar sources explored and take the information obtained about the quality of the food source locations which they are exploiting to the waiting bees (onlooker bees). Onlooker bees wait in the hive and decide on a food source to be exploited based on the information received from the employed bees. Scouts bees randomly search the environment in order to find a new food source depending on an internal motivation or based on possible external traces. In the ABC algorithm, the position of a food source represents a possible solution to the optimization problem, and the nectar amount of a food source corresponds to the profitability (fitness) of the associated solution (Akay and Karaboga, 2012). Each food source is exploited by only one employed bee. In other words, the number of employed bees is equal to the number of food sources existing around the hive (number of solutions in the population).

In ABC algorithm, one of the employed bees is selected and classified as the scout bee. The classification is controlled by a control parameter called ‘‘limit’’. If a solution representing a food source is not improved by a predetermined number of trials, then that food source is abandoned by the employed bee and this bee associated with that food source becomes a scout bee (TSai *et. al.*, 2009). The number of trials for releasing a food source is equal to the value of ‘‘limit’’, which is an important control parameter (stopping criteria) in the ABC algorithm.

Using the analogy between emergent intelligence in foraging of bees and the ABC algorithm, the stages of the basic ABC algorithm can be explained as follows:

I. Employed Bee Stage

An employed bee searches for food source within the neighbourhood of the food source. Suppose, an employed bee is currently positioned at a food source position x_i . During this stage, each employed bee searches in the neighbourhood of its current position x_i to produce new trial food source v_i .

$$v_{ij} = x_{ij} + \phi_{ij}\left(x_{kj} - x_{ij}\right) \tag{2.7}$$

Using equation (2.7), where $j \in \{1,2,...,D\}$ and $k \in \{1,2,..SN\}$ are randomly picked indices, D is dimension of the problem, SN is the number of food positions and ϕ_{ij} is a uniform random value chosen by [-1, 1] (Karaboga and Basturk, 2007).

Thus, the new solution v_i is produced from x_i by perturbing its randomly picked j^{th} parameter and using the information of x_i and another randomly picked solution x_k . If v_i has better fitness than the old food position x_i , then x_i is replaced by v_i . Otherwise the previous position x_i is retained.

$$fitness(x_i) = \begin{cases} \dfrac{1}{(1+f(x_i))} & if\ f(x_i) \geq 0 \\ 1+\left|f(x_i)\right| & if\ f(x_i) \prec 0 \quad (otherwise) \end{cases}$$

(2.8)

For the problem of function optimization, where f is the function to be minimized, ABC computes the fitness of a candidate solution x_i using equation (2.8) (Karaboga and Basturk, 2007).

where $f_i =$ function to be minimized

II. Onlooker Bee Stage

After all employed bees complete their search, they share their information related to the nectar amounts and the positions of their sources with the onlooker bees on the dance area. The dance area is where exchange of information about the quality of food sources occurs. An onlooker bee evaluates the nectar information taken from all employed bees and chooses a food source location with a probability related to its nectar amount.

$$P_i = \frac{fitness(x_i)}{\sum_{n=0}^{SN} fitness(x_n)} \tag{2.9}$$

The probability P_i that the employed bee with food source x_i would be picked by an onlooker bee is computed using equation (2.9), which makes the probability P_i to be proportional to $fitness(x_i)$ (Karaboga and Basturk, 2007).

Like the employed bees, each onlooker bee also employs (2.8) to produce trial food source v_i in the vicinity of its current food source position x_i. If v_i has better fitness then x_i is replaced by v_i. Otherwise, v_i is discarded.

III. Scout Bee Stage

A scout bee is created only when a particular food source x_i cannot be improved through a predetermined number of trials limit.

$$x_{i_j} = \min_j + rand(0,1)(\max_j - \min_j) \tag{2.10}$$

The employed bee now becomes a scout bee and its food source is positioned randomly across the search space using equation (2.10), where $j = 1,2,...,D$ and $\lfloor \min_j, \max_j \rfloor$ is the search space along the j^{th} dimension (Alam *et. al.*, 2015).

In a robust search process, exploration and exploitation processes must be carried out together in equal proportion. The onlooker and employed bees carried out the exploitation process in the search space, while the scouts are responsible for the exploration process. In the case of real honeybees, the recruitment rate represents a measure of how quickly the bee colony finds and exploits a newly discovered food source (Karaboga and Basturk, 2008). This recruiting could also represent the measurement of the speed with which the feasible solutions of the difficult optimization problems can be discovered. The flowchart of ABC is shown in Figure 2.4 (Karaboga and Akay, 2011).

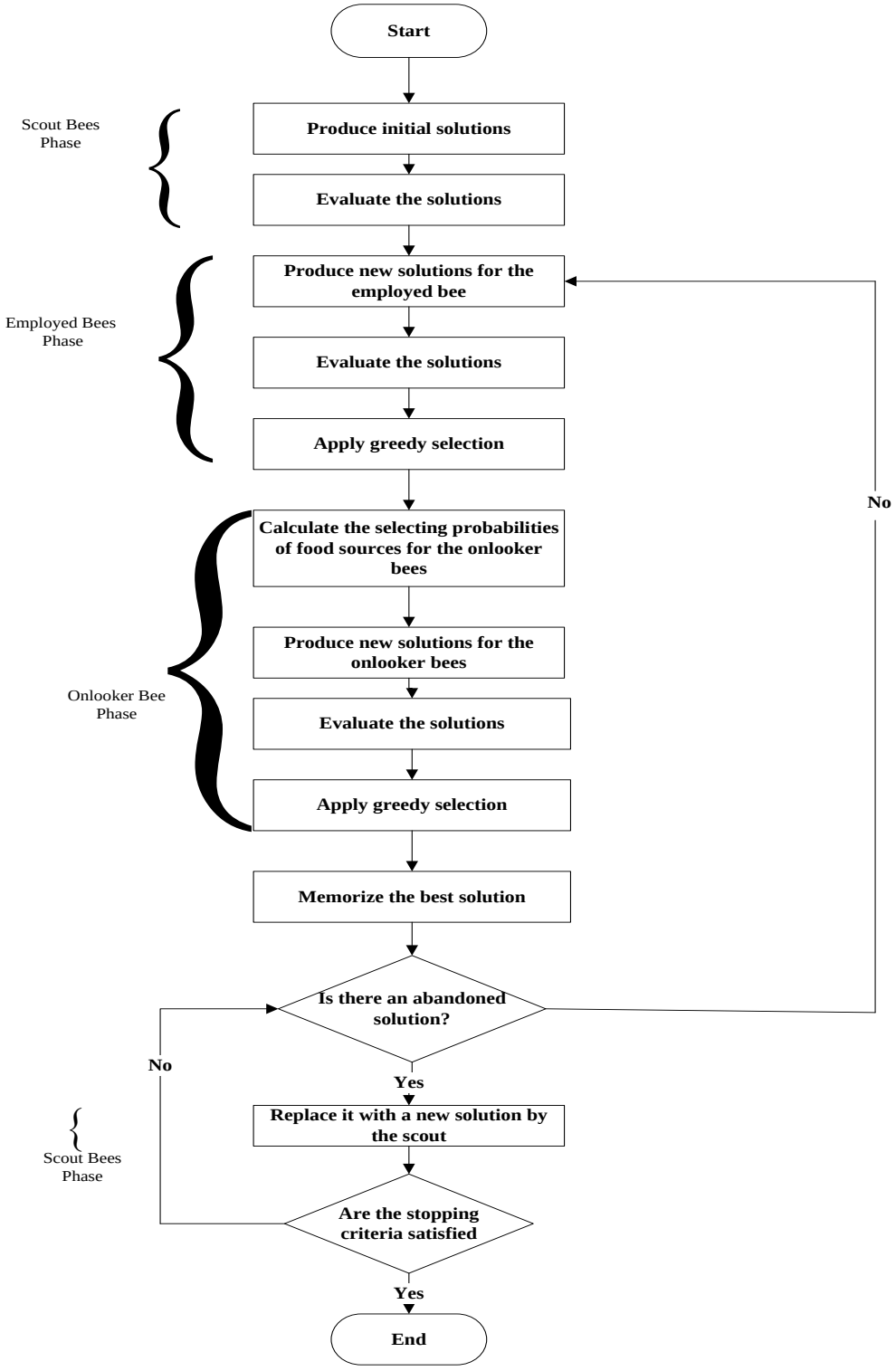


Figure 2.4: Flowchart of Artificial Bee Colony (ABC) Algorithm (Karaboga and Akay, 2011)

2.2.5 Culture algorithm

Cultural Algorithms (CAs), is an evolutionary model which are inspired by model of cultural evolution process. Cultural Algorithms are based on knowledge of an evolutionary system and were introduced by Reynolds as a means of simulating cultural evolution (Reynolds, 1994). Cultural Algorithm is a dual mechanism of inheritance where inherited characteristics of both the level of the population, as well as the belief space (culture). Algorithms that use social learning are higher than those using individual learning, because they present a better and faster convergence in the search for solutions (Reynolds, 1994). The belief space is the place where the information about the knowledge on the solution of the problem is developed and stored. This is acquired through the population space over the evolutionary process. The belief space has the goal to guide individuals in search of better regions. In CAs the characteristics and behaviours of individuals are represented in the Population Space (PS). This representation can support any population-based computational model such as Artificial Bee Colony (ABC), Particle Swarm Optimization (PSO), Artificial Fish Swarm Algorithm (AFSA), Genetic Algorithms (GA) and Evolutionary Programming etc (Peng and Reynolds, 2004).

In this research, cultural algorithm will be used to propose an improvement on Artificial Bee Colony (ABC) Algorithm. The population level of the cultural algorithm component will be the ABC algorithm. The global knowledge that has been learned by the population will be expressed in terms of both normative and situational knowledge.

The basic aspects of the Cultural Algorithm structure are (Salawudeen, 2015):

1) The level of self-adaptation.
2) The various types of knowledge in the belief space.
3) Methods to acquire knowledge (acceptance function).
4) Methods to exploit the knowledge to guide the population (influence function).

5) Communication protocols that describe how knowledge is exchange between the belief space and population.

The framework of Cultural Algorithm is shown in Figure 2.5 (Reynolds, 1994).

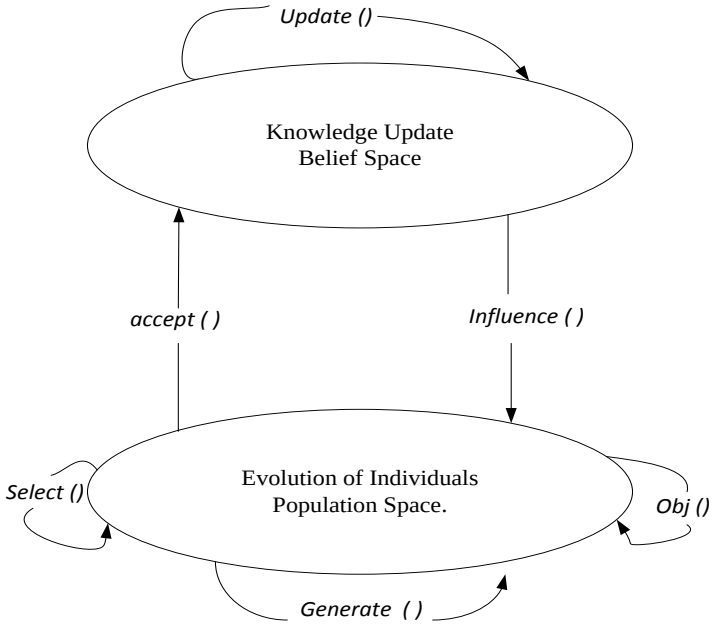


Figure 2.5: Framework of Cultural Algorithm (Reynolds, 1994)

A cultural algorithm is generally initialized with a randomly generated population of individuals. In this research, initialization will be done using ABC algorithm.

As stated earlier CA consist of a social population and a belief space. Selected individuals from the population space contribute to the cultural knowledge by means of the acceptance function. The knowledge resides in the belief space where it is stored and employed based on individual experiences and their successes or failures. In turn, the knowledge controls the evolution of the population by means of an influence function. The knowledge in the belief space can influence the evolution in two ways (Reynolds and Chung, 1997):

1) Determination of the size of the changes.
2) Determination of the direction of change, for example increase in the current value.

These two operations will be applied to the solution search equation for the ABC as given in the equation (2.7) which will serve as adaptive parameters in order to balance explorations and exploitations. Consequently, four variants of CABCA will be developed.

A Cultural Algorithm therefore provides a framework in which to accumulate and communicate knowledge within a population so as to allow self-adaptation in an evolving model (Kobti *et. al.*, 2004).

The pseudo-code description of culture algorithm is given as follows (Peng & Reynolds, 2004):

Begin
  t = 0;
  Initialize POP(t);
  Initialize BLF(t);
  repeat
    Evaluate POP(t);
    Vote(BLF(t), *Accept*(POP(t)));
    Adjust(BLF(t));
    Evolve(POP(t), *Influence*(BLF(t)));
    t = t + 1;
    Select POP(t) from POP(t-1);
  until (termination condition achieved)
End

A. Knowledge Sources in the Belief Space

The belief space in Cultural Algorithm is where the cultural knowledge is made and conserved. Five basic categories of cultural knowledge have been identified: Normative, Situational, Domain, History, and Topographical Knowledge (Reynolds and Peng, 2004). For the purpose of this work, (situational and normative) knowledge will be used to guide the individuals in

search for better region. This is because they are deemed more fundamental to the operation of cultural system (Reynolds and Chung, 1997).

The formal syntax in the belief space B is defined as a pair of structure. That is, $B = \langle S, N_{(1,..n)} \rangle$, where S is set of exemplars, in this case best individuals that constitute the situational knowledge.

A. Situational Knowledge

Situational knowledge represents the best individuals found at a certain time of evolution and it contains a number of individuals considered as a set of exemplars to the rest of the population. The situational knowledge is updated when the best individual of the population is found, that is at iteration t, $S = \prec S^t \left| S^t = \{ s_1^t, s_2^t, ..., s_n^t \} \right. \succ$. This situational knowledge equation can be represented as (Reynolds and Chung, 1997):

$$S_j^{t+1} = \begin{cases} X_{best,j}^{t+1} & if \quad f\left(X_{best,j}^{t+1}\right) \prec f\left(s^t\right) \\ s_j^t & otherwise \end{cases} \tag{2.11}$$

B. Normative knowledge

Normative knowledge describes how individual should act in terms of ranges of acceptable behaviour. In other words, normative knowledge defines a standard or ideal way that can be used to judge which behaviour is desirable or undesirable (Kobti *et. al.*, 2004). Normative knowledge provides standards for interpreting and determining individual behaviours through guidelines within which individual adjustments can be made. The normative component N is a set of interval information for each n parameter. Each of the N intervals in the belief space is represented as a triple. That is defined as (Reynolds and Chung, 1997):

$$N = \langle I, L, U \rangle, \tag{2.12}$$

where I, L and U, are $n-$dimensional vectors, and I_j denotes the closed interval for variable j that is a continuous set of real numbers x represented as a number pair:

$$I_j = [l_j, u_j] = \{x | l_j \le x \le u_j\} \tag{2.13}$$

n is the number of the variables, l_j and u_j are the lower and upper bounds for the j^{th} variable, respectively, L_j and U_j are the values of the fitness function associated with the bound l_j and u_j are usually initialized with positive infinity.

Usually the normative knowledge leads individuals "to jump into the good range" if they are not there yet (Reynolds and Peng, 2004).

The normative knowledge is updated as follows (Reynolds and Chung, 1997)

$$l_j^{t+1} = \begin{cases} x_{i,j}^t & \text{if } x_{i,j}^t \le l_j^t \quad \text{or} \quad f(x_i^t) \prec L_j^t \\ l_j^t & \text{otherwise} \end{cases} \tag{2.14}$$

$$u_j^{t+1} = \begin{cases} x_{k,j} & \text{if } x_{k,j} \ge u_j^t \quad \text{or} \quad f(x_k) \prec U_j^t \\ u_j^t & \text{otherwise} \end{cases} \tag{2.15}$$

$$L_j^{t+1} = \begin{cases} f(X_i) & \text{if } x_{i,j}^t \le l_j^t \text{ or } f(X_i) \prec L_j^t \\ L_j^t & \text{otherwise} \end{cases} \tag{2.16}$$

$$U_j^{t+1} = \begin{cases} f(X_k) & \text{if } x_{k,j} \ge u_j^t \text{ or } f(X_k) \prec U_j^t \\ U_j^t & \text{otherwise} \end{cases} \tag{2.17}$$

Where the i^{th} individual affects the lower bound for variable j, and the k^{th} individual affect the upper bound for variable j. It should be noted that t denotes the current generation of individuals.

C. Acceptance Function

An acceptance function is used to control which members of the population are permitted to impact the belief space. They determine which individuals in the current generation of the

population will contribute with their knowledge to belief space (Reynolds and Peng, 2004). The experiences of the selected individuals will be used to update the knowledge of the current belief space. In turns, the knowledge in the beliefs will be used to guide and influence the evolution of the population. In this research, the acceptance function for component N selects the best range of the 20% performing individuals in each generation is calculated and successive generations will be randomly generated in this promising range. This will increase the convergence and will preserve wasting time in discovering the good regions. The number of the individuals accepted for the update of the belief space is obtained according to the following function (Reynolds and Chung, 1997):

$$\alpha(N,t) = N.\beta + \lfloor N.\beta/t \rfloor \tag{2.18}$$

where N is the size of the population, t is the current number of generation and β is the top 20% of the population (usually chosen to be 0.2).

2.2.6. Optimization Benchmark Functions

In order to validate any new optimization algorithm, one has to validate it against standard test functions so as to compare its performance with well-established or existing algorithms. There are many test functions, so there is no standard list or set of test functions one has to follow. However, various test functions do exist in literature, so new algorithms should be tested using at least a subset of functions with diverse properties so as to make sure whether or not the tested algorithm can solve certain type of optimization efficiently. In this research, a collection of twenty (20) unconstrained optimization test functions which is used to evaluate the performance of the proposed algorithm. The purpose of this collection is to give the CABCA a large number of general test functions to be used in testing the optimization algorithm and

comparison studies. For each function, its algebraic expression and the standard initial point as well as optimal values are given.

i. Sphere Function

This is a unimodal, convex and separable test function. It is expressed mathematically as follows (Kang *et. al.*, 2011):

$$f(x)=\sum_{i=1}^{n} x_i^2, \tag{2.19}$$

subject to $-100 \le x_i \le 100,\ i=1,...,n$

The global minimum is located at $x^*=(0,0,...,0)$ with $f(x^*)=0$

ii. Schwefel Function

This is a multimodal, separable function. The global minimum is geometrically distant, over the parameter space, from the next best local minima, hence it is deceptive. Therefore, the search algorithms are potentially prone to convergence in the wrong direction. The mathematical definition of this function is as follows (Molga & Smutnicki, 2005):

$$f(x)\sum_{i=1}^{n} -x_i \sin\left(\sqrt{|x_i|}\right), \tag{2.20}$$

subject to $-500 \le x_i \le 500,\ i=1,..,n$

The function has the global minimum at $x^* \approx (420.9687,...,420.9687)$ with $f(x^*) \approx 0$

iii. Ackley Function

Ackley function is widely used multimodal function. It has the following definition (Kang *et. al.*, 2011):

$$f(x)=20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)-\exp\left(\frac{1}{n}n\sum_{i=1}\cos(2\pi x_i)\right)+20+e \tag{2.21}$$

where $a=20$, $b=0.2$ and $c=2\pi$

subject to $-32 \le x_i \le 32,\ i = 1,..., n$

The global minimum is located at $x^* = (0,..,0)$ with $f(x^*) = 0$

iv. Rastrigin Function

Rastrigin's function is a multimodal, separable function. The locations of the local minima are regularly distributed. It can be expressed mathematically as (Kang et al., 2011):

$$f(x) = 10n + n\sum_{i=1}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right) \tag{2.22}$$

subject to $-5.12 \le x_i \le 5.12,\ i = 1,.., n$

The global minimum is $x^* = (0,..,0)$ with $f(x^*) = 0$

v. Griewangk Function

Griewangk's function is similar to the function of Rastrigin. It has many widespread local minima regularly distributed. This function becomes simpler and smoother as the dimension ability of the search space is increased. The function has the following definition (Molga & Smutnicki, 2005):

$$f(x) = \sum_{i=1}^{n}\frac{x_i^{\,2}}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{2.23}$$

subject to $-600 \le x_i \le 600,\ i = 1,..., n$

The global minimum is located at $x^* = (0,..,0)$ with $f(x^*) = 0$

vi. Rosenbrock Function

This is a unimodal function, yet due to a saddle point it is very difficult to locate the minimum $x^* = (1,...,1)$ with $f(x^*) = 0$. The global minimum lies inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial, however convergence to the global optimum is difficult (Kang *et. al.*, 2011).

$$f(x)=\sum_{i=1}^{n-1}\left[100\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2\right] \tag{2.24}$$

subject to $-30\le x_i\le 30,\ i=1,..,n$

vii. Schaffer Function

This is a multimodal, non-convex optimization test function. It can be express mathematically as follows (Kang et al., 2011):

$$f(x)=\sum_{i=1}^{30}\left(x_i^2+x_{i+1}^2\right)^{0.25}\Big/\left\{\left[\sin 50\left(x_i^2+x_{i=1}^2\right)^{0.1}\right]^2+1\right\} \tag{2.25}$$

subject to $-100\le x_i\le 100,\ i=1,...,n$

The function has the local minimum $x^*=(0,..,0)$ and $f\left(x^*\right)=0$

viii. Michalewicz Function

The Michalewicz function is a multimodal test function. The parameter m defines the "steepness" of the valleys or edges. Larger m leads to more difficult search. For very large m the function behaves like a needle in the haystack (the function values for points in the space outside the narrow peaks give very little information on the location of the global optimum). Function has the following definition (Kang *et. al.*, 2011):

$$f(x)=-\sum_{i=1}^{n}\sin(x_i)\left(\sin\left(\frac{ix_i^2}{\pi}\right)\right)^{2m} \tag{2.26}$$

where $m=10,$ subject to $0\le x_i\le\pi,\ i=1,...,n.$

There are $n!$ local optima and the global minimum value for $n=10$ is $f\left(x^*\right)=-9.660152$

ix. Penalized Levy and Montalvo 1

Penalized levy and Montalvo 1 (PM1) is a multimodal, non-separable function, which has been widely used to verify the performance of optimization function (Kang *et. al.*, 2011)

$$f(x)=\frac{\pi}{n}\left\{10\sin^2(\pi y_i)+\sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right]+(y_n-1)^2\right\}+\sum_{i=1}^{n-1}u(x_i,10,100,4) \tag{2.27}$$

subject to $-50\le x_i\le 50, i=1,...,n$

The global minimum of PM1 function is $f(x^*)=0$ at $x^*=(0,0,..,0)$,

x. Penalized Levy and Montalvo 2

Penalized Levy and Montalvo 2 (PM2) is also a multimodal, non-separable function. It can be expressed by the following mathematical definition (Kang *et. al.*, 2011):

$$f(x)=0.1\left\{\sin^2(3\pi x_i)+\sum_{i=1}^{n-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]+(x_n-1)^2\left[1+\sin^2(2\pi x_n)\right]\right\}+\sum_{i=1}^{n-1}u(x_i,5,100,4) \tag{2.28}$$

subject to $-50\le x_i\le 50$, $i=1,..,n$

The global minimum is known to be $x^*=(0,0,..,0)$ with $f(x^*)=0$

xi. Quartic function

The Quartic function is padded with noise. The random noise makes sure that the algorithm never gets the same value at the same point. It is expressed mathematically as: (Kang *et. al.*, 2011).

$$f(x)=\sum_{i=1}^{n}ix_i+rand[0,1) \tag{2.29}$$

subject to $-1.28\le x_i\le 1.28,\ i=1,....,n.$

The global minimum is $x^* = (0,...,0)$ with $f(x^*) = 0$

xii. Zakharov

This is a unimodal function which can be represented mathematically as (Kang *et. al.*, 2011):

$$f(x) = \left(\sum_{i=1}^{n} x_j^2\right) + \left(\sum_{i=1}^{n} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^4 \quad (2.30)$$

subject to $-5 \le x_i \le 10,\ i = 1,..., n.$

The global minimum is $x^* = (0,..0)$ with $f(x^*) = 0$

xiii. Step Function

The step function is a unimodal, separable function define by (Kang *et. al.*, 2011):

$$f(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2 \quad (2.31)$$

subject to $-100 \le x_i \le 100,\ i = 1,..., n$.

The global minimum is located at $-0.5 \le x_i^* \le 0.5$ with $f(x^*) = 0$

xiv. De Jong F4 Function

De Jong 4th order function is a unimodal function defined as (Molga & Smutnicki, 2005).

$$f(x) = \sum_{i=1}^{n} ix_i^4 \quad (2.32)$$

subject to $-5.12 \le x_i \le 5.12,\ i = 1,..., n$.

The global minimum is $x^* = (0,...,0)$ with $f(x^*) = 0$

xv. Hyperelliptic Function

The axis parallel is similar to De Jong function. It is described by the following expression (Salawudeen, 2015b):

$$f(x)=\sum_{i=1}^{n} i^2 x_i^2 \tag{2.33}$$

subject to $-5.12 \le x_i \le 5.12,\ i=1,...,n$.

The global minimum is $x^*=(0,...,0)$ with $f(x^*)=0$

xvi. Expfun Function

Expfun function is regarded as the simplest form of De Jong function. It is given by the following expression (Li *et. al.*, 2013):

$$f(x)=\exp\left(-0.5\sum_{i=1}^{n} x_i^2\right) \tag{2.34}$$

subject to $-5.12 \le x_i \le 5.12,\ i=1,...,n$

The global minimum is $x^*=(0,..0)$ with $f(x^*)=1$

xvii. Cosine Mixture Function

The cosine Mixture (CM) function is a multimodal optimization test function. It can be represented by the expression (Li *et. al.*, 2013):

$$f(x)=(-0.1)\sum_{i=1}^{n}\cos(5\pi x_i)+\sum_{i=1}^{n} x_i^2 \tag{2.35}$$

Subject to $-1 \le x_i \le 1,\ i=1,2,....,n$

The global minimum is of $x^*=(0,..,0)$ with $f(x^*)=-3$

xviii. Sal Function

Sal function can be expressed as follows (Li et al., 2013):

$$f(x)=1-\cos\left(2\pi\sqrt{\sum_{i=1}^{n}x_i^2}\right)+0.1\sqrt{\sum_{i=1}^{n}x_i^2} \tag{2.36}$$

subject to $-2\pi \le x_i \le 2\pi,\ i=1,...,n$

The global minimum of Sal function is located at $x^*=(0,00)$ with $f(x^*)=0$

xix. Neumaier3 Function

Neumaier3 function can be represented mathematically as (Li et al., 2013):

$$f(x)=\sum_{i=1}^{n}(x_i-1)^2-\sum_{i=2}^{n}x_i x_{i-1} \tag{2.37}$$

Nuemaier3 has a lot of local minima around the global minima with a global solution given as

$f(x^*)=-4930$ at $x^*=(0,..,0)$

xx. Axis Parallel Hyper-Ellipsoid function

The axis parallel hyper-ellipsoid is similar to the function of De Jong. It is also known as the weighted sphere model. The function is continuous, convex and unimodal. It has the following general definition (Molga and Smutnicki, 2005):

$$f(x)=\sum_{i=1}^{n}ix_i^2 \tag{2.38}$$

subject to $-5.12 \le x_i \le 5.12,\ i=1,..,n$ with global minimum $x^*=(0,00)$ at $f(x^*)=0$

2.3 Review of Similar Works

Some literature related to ABC algorithm, its improvement and limitations, and application in PID tuning is discussed in this section.

Godze *et. al.,* (2010) studied a new artificial intelligence based optimization method to optimize the gains of PID controller for Automatic Voltage Regulator (AVR) system. The dynamic performance of the controller which was optimized by ABC algorithm was compared with Particle Swarm Optimization (PSO). The settling time and the maximum overshoots of the voltage variation curves were measured with transient response analysis in order to determine the performances of the AVR system. It was observed that the maximum overshoot obtained through the ABC algorithm was smaller than the results obtained through PSO algorithm. In addition to this, the settling time obtained through the ABC algorithm was approximately half of the times obtained through PSO algorithm. Consequently, ABC algorithm obtained better performance than the other optimization method according to the transient response analysis. However, the risk of premature convergence and fitness stagnation as a result of imbalance between explorations and exploitations associated with the ABC was not considered.

Gao *et. al.*, (2010) implemented a novel optimal PID tuning, an online tuning based on artificial bee colony algorithm. The ABC algorithm approach was introduced in PID tuning and online tuning as a novel technique for optimum adaptive control in a non-Lyapunov way. The PID parameters $\left(K_p, K_i \,\&\, K_d\right)$ were converted to a series of multi-modal non-negative functions minimization and whose minimum values were optimally determined by ABC. In order to gain satisfactory dynamic properties of process, integral time of the absolute error (ITAE) was

chosen as the objective function to be minimized the steady-state erorr. At the end, the K_p, K_i and K_d contributed to obtain a fast rise time and minimum overshoot. However, the convergence rate and quality of solutions deteriorated with the increase in the problem complexity.

Abachizadeh *et. al.*, (2010) employed Artificial Bee Colony (ABC) algorithm to tune PID controllers for plants of high order and with time delay. The method optimized PID gains as design variables in both single and multi-objective approaches. The objective functions taken from literature were important performance indices of integral time square erorr (ITSE), integral square erorr (ISE) and integral absolute error (IAE) as well as overshoot and settling time. The results clearly expressed that the utilized method has been successful in comparison to Genetic Algorithm (GA), Evolutionary Programming, fuzzy rule-based approaches and Ziegler-Nichols (Z-N). In addition, it was observed that the optimal gains obtained by ABC were not in the neighborhood of the ones reported by GA. This proved that ABC has been well equipped not to trap in local optima though the optimization problem is not constrained.

Ercin and Coban, (2011) presented the performance of the Artificial Bee Colony (ABC) and the Bees Algorithm (BA), and were compared for (PID) controller tuning. The results obtained showed good stability, set-point tracking performance and robustness. However, performance of the ABC was better than BA for PID controller tuning. To indicate the effectiveness and efficiency of the developed optimization methods, the step responses of closed loop systems were compared with the ABC and the BA and it was found that ABC has a minimum values than BA in respect to overshoot, settling time and rise time. It was observed that the ABC even with its less control parameters had a better tuning performance than the BA which consisted

of many control parameters. However, the poor convergence associated with the ABC algorithm was not addressed.

Rajasekhar *et. al.*, (2011) developed a new technique for designing feedback control of a DC motor speed using fractional order proportional-integral-derivative (FOPID) controller. The controller was formulated as a single objective optimization problem and based on integral time absolute error (ITAE) criterion; finally, a Levy mutated ABC algorithm called (L-ABC), which is a modified ABC was used to tune the FOPID controller parameters. The developed approach was demonstrated by comparing it with conventional and the ABC algorithm. However, the convergence rate and quality of solutions deteriorated with the increase in the problem complexity.

Yan and Li, (2011) developed an effective refinement artificial bee colony optimization algorithm based on chaotic search (CABC) and application for PID controller tuning. The chaotic search was based on ABC algorithm for solving the accuracy of global optimal value. In their approach, the premature convergence problem of standard ABC was solved by increasing the number of scouts and also using chaotic search. The developed algorithm was applied to the objective function (ITAE), to evaluate the fitness of each food source. Though results showed that CABC had a high accuracy and the PID control systems adopted optimal parameters, the quality of solutions declined with the increase in the problem complexity.

Shayanfar *et. al.*, (2012) developed a hybrid Proportional Integral Derivative (PID) controller and Power System Stabilizer (PSS) controller based Artificial Bee Colony (ABC) techniques to damp low frequency oscillation. The developed control strategy in this contribution

combined the advantage of the PID control system and PSS controller by ABC techniques for achieving the desired level of robust performance under different operating conditions and load disturbances. The effectiveness of the proposed ABC-PID was tested on a Single Machine Definite Bus (SMIB) power system under different operating conditions which was compared with the other variant of Particle Swarm Optimization (PSO) through nonlinear time simulation and different performance indices. Also, beside the parameters of PSS, the eigenvalues of the power system were considered to test the flexibility of the proposed technique in different situation. The presented results in this work, demonstrated that in a SMIB system, fixed-structure damping controllers can be tuned to provide satisfactory damping performance over a pre-specified set of operating conditions. The poor balance between exploration and exploitation associated with the standard ABC which results in poor convergence was not addressed.

El-Telbany, (2013) applied Artificial Bees Colony (ABC) optimization algorithm to determine the optimal values of the PID controller parameters of DC motors. During the optimization process, the reference input and closed loop response of the process was used by the ABC. Using the changed closed loop control performance according to the adjusted controller parameters $\left(K_p, K_i \,\&\, K_d\right)$ at each generation, the tuning algorithm searches the optimal gains of the PID controller to satisfy the desired system specifications. To illustrate the effectiveness of the developed method, the time domain performance of the ABC-PID controller was compared with other population based (PSO and GA) and classical Ziegler-Nichols method and it was found that the ABC-tuned PID controller outperforms the GA, PSO and Z-N tuned controller in terms of settling time and overshoot. However, the individual behaviour of the bee population as it affected optimization efficiency of the algorithm was not addressed.

Mishra *et. al.*, (2013) designed a speed controller of a DC motor by selecting of PID parameters using the ABC technique. The application of ABC to the PID controller enhanced its ability of tuning itself automatically in an on-line process while the application of optimization algorithm to the PID controller made it to give an optimum output by searching for the best set of solutions for the PID parameters. In order to verify the effectiveness of the bio-inspired ABC technique, a conventional Ziegler-Nichols (Z-N) based PID was used to control the same speed for same experimental cases. From the results, while using ABC, the overshoot obtained was zero as compared to the case when the PID controller was tuned using Z-N methods. The settling time was also lesser in the case of ABC, also the rise time was reduced. The poor balance between exploration and exploitation associated with the standard ABC resulting in poor convergence was not addressed.

Varma and Kumar, (2013) studied the position control of DC motor using Artificial Bee Colony Algorithm. In order to solve this problem, a PID controller under Artificial Bee Colony Algorithm with self–tuning was applied, which performed high efficiency position control. The efficiency of control algorithm was presented through a simulation and compared with the quality of PID controller. The developed method was compared with Ziegler Nichols method and it was found that the overall system gave better response (in terms of rise time, settling time and overshoot) when tuned by Artificial Bee Colony Algorithm compared to classical Ziegler Nichols Method. The individual behaviour of bees which could result into fitness stagnation and poor convergence of ABC algorithm was not considered in this research.

Sonmez *et. al.*, (2015) developed ABC algorithm to determine gain parameters of the PID controller for controlling a designed buck converter. In order to verify the effectiveness of the

proposed algorithm, GA based PID was used to control the same converter for same experimental cases. Experiments with different cases were done via simulations, in the simulations, settling time, steady-state error and re-adjustment time of output voltage in cases of load variation were analyzed in order to determine the performance of the proposed algorithm. Results obtained from simulations shows that ABC-PID controlled converter has superior performance than GA-PID controlled buck converter. The individual behaviour of bee which could cause the ABC to fall into local minima easily was not considered in this research.

Kesarkar and Selvaganesan, (2015) explored the usefulness of ABC algorithm in solving high-dimensional and multi-modal optimization problem while tuning a fractional-order PID $\left(PI^{\lambda}D^{\mu}\right)$ controller. The controller parameters $\left(K_p, K_i, K_d, \lambda, \mu\right)$ were tuned by minimizing the selected performance index. It was interesting to note that ABC outperformed the deterministic Nelder–Mead algorithm. This is due to the in-built randomness which helps ABC to come out of local minima which is not possible with the deterministic Nelder–Mead algorithm. However, the convergence rate of the ABC and quality of solutions deteriorates with the increase in the problem complexity.

Kaliappan and Thathan, (2015) developed a nonlinear PID (N-PID) controller tuning based on the Enhanced Artificial Bee Colony (E-ABC) algorithm. The ABC algorithm used the foraging behaviour of honey bee swarm to find the optimal PID parameters K_p, K_i and K_d. In the proposed E-ABC, the Particle Swarm Optimization (PSO) was inherited to ABC scout bee to get proper selection of food source. The convergence characteristics of the E-ABC based optimization showed that the proposed method produce a better controller settings with minimum iteration. To show the effectiveness of the proposed method, it was presented to the

nonlinear Continuous Stirred Tank Reactor (CSTR) process and the results obtained were compared with the conventional Internal Model Control (IMC) tuning method. It was evident that the proposed E-ABC delivered better tracking. The individual behaviour of bee which could cause the E-ABC to fall into local minima easily was not considered in this research.

From the similar works reviewed, it is evident that ABC optimization algorithm has captured a wide area of applications in tuning the parameter of PID controller. Several efforts have been made by researchers to address some of the problems associated with the ABC. However, experience of individuals and the knowledge derived from the parent as a result of evolution has not been employed to address some of the challenges associated with the standard ABC, to the best of my knowledge. In this research, this individual experience which consists of (step size and direction of change) was employed through cultural evolution process to propose a new variant of ABC called the CABCA. This is expected to improve the original ABC algorithm while overcoming some of the highlighted shortcomings. The modified ABC algorithm is used to improve the PID parameters tuning process which is then used in the PID control of DC motor.

CHAPTER THREE

METHODS AND MATERIALS

3.1 Introduction

In this section, the detailed procedure and all the relevant equations considered for the successful completion of this research are discussed. Four variants of improved ABC was developed and the systematic approach and essential mathematical model describing these variants are explained in this chapter.

3.2 Methodology

The step by step procedure that was employed in this research work is as follows:

1) Initialization of the population of ABC and its parameters.
2) Replication of the standard ABC which is done as follows:
    a) The employed bees are moved onto their food sources and determine their nectar amounts.
    b) The onlooker bees are moved onto their food sources and determine their nectar amounts.
    c) The scouts were saddled with searching for new food sources
    d) The best food source found is memorized until the requirements are met.
3) Development of four variants of Cultural Artificial Bee Colony Algorithm (CABCA) are developed using MATLAB R2015a Control/ Optimization tool box, these include:
    (a) CABCA(Ns)
    (b) CABCA(Sd)
    (c) CABCA(Ns+Sd)

(d) CABCA(Ns+Nd)

4) Evaluation of the performance of the developed algorithm in 2 & 3 using a collection of twenty (20) reported applied mathematical optimization test functions (such as: Ackley, Rastrigin etc.)

5) Compare the results of the four variants with the one obtained using the standard ABC with reference to the global minima.

6) Finally, the best performing variant in 5) is used in finding the optimal tuning parameters of PID controller which is then employed in the speed control of DC motor.

3.3 Initialization of ABC and the Parameters

All the vectors of the population of food sources are initialized by scout bees and control parameters are set. The ABC algorithm has three control parameters: the number of food sources (population), the number of test after which a food source is treated to be discarded (limit) and the termination criteria (maximum number of cycle). In the standard ABC, the number of food sources is equal to the employed bees or onlooker bees. Initially it considered an evenly dealt swarm of food sources (SN), where each food source $x_i(i = 1,2,..., SN)$ is a dimensional vector corresponded to the number of PID parameters optimized. Each food source was generated using following method:

$$x_{i_j} = \min_j + rand[0,1](\max_j - \min_j) \quad (3.1)$$

Here $rand[0.1]$ is a function that generates an evenly distributed random number in range $[0,1]$, while $\max_j$ and $\min_j$ are the maximum and minimum limits of the parameters to be optimized.

3.4 Replication of Standard ABC Algorithm

As discussed in section 2.2.4, ABC is a robust optimization method. It is based on foraging behavior of honey bee colonies searching for food source, the artificial optimization procedure consists of employed bees, onlooker and scout bees searching for food source (optimized values) with high nectar amount (fitness).

The steps involved in design of PID using ABC algorithm are expressed as

3.4.1 Assigning the employed bees to the food sources

Employed bees explore new food sources v_{ij} (new PID gain values) having high nectar values (optimal values) contained by the proximity of the food source in their remembrance. They found the PID gain values and then estimated its optimality. Employed bees phase informed the present solution based on the information and individual experiences and the optimum value of the recently found solution. The new gain values with elevated optimum value replaced the existing one. The position update equation for j^{th} dimension of i^{th} candidate during this phase is as follow:

$$v_{ij} = x_{ij} + \phi_{ij}\left(x_{kj} - x_{ij}\right) \tag{3.2}$$

Here $\phi_{ij}\left(x_{kj} - x_{ij}\right)$ is known as step size, $k \in \{1,2,....,SN\}$, $j \in \{1,2,....,D\}$ are two randomly chosen indices. $k \neq i$ ensure that step size has some indicative perfection.

As can be seen from (3.2), as the difference between the parameters of the x_{kj} and x_{ij} decreases, the perturbation on the position x_{ij} gets decrease, too. Thus, as the search approaches to the optimum solution in the search space, the step size is adaptively reduced. If a parameter value produced by this operation exceeds its predetermined limit, the parameter can be set to an acceptable value. In this work, the value of the parameter exceeding its limit is set to its limit

value. The gains of which the nectar is abandoned by the bees is replaced with a new gains by the scout.

3.4.2 Employing of onlookers and calculation of probabilities

The onlooker bee selected the gains based on the fitness function which was derived from the employed bee's information. The gains which is having high nectar value has high probability for the selection by onlooker bees and it is explained in Equation (3.2).

$$P_i = \frac{fitness(x_i)}{\sum_{n=0}^{SN} fitness(x_n)} \tag{3.3}$$

3.4.3 Scout bee exploring new food source

At this step, if a food source is not improved further, that source is abandoned and it will be replaced with a new one by scout bee generated evenly at random within the search space using equation (3.1). This process is done according to the "limit" parameter which is predetermined number of cycles for abandoning the food source.

3.4.4 Stoping Condition

If maximum iteration (Maximum Cycle Number) is reached then stop search process and return the best food source obtained so far, otherwise, the employed bees is re-assigned to the food sources.

The pseudo-code of the standard ABC algorithm is given as follows:

1) ***Begin***
2) ***Initialize the solution population,*** $i = 1,...., SN$
3) ***Evaluate population***
4) ***cycle = 1***

5) ***Repeat***

6) ***Generate new solutions*** v_{ij} ***for the employed bees using (3.2) and evaluate them.Keep the best solution between current and candidate***

7) ***Select the visited solution for onlooker bees by their fitness***

8) ***Generate new solutions*** v_{ij} ***for the Onlooker bees using (3.2) and evaluate them***

9) ***Keep the best solution between current and candidate***

10) ***Determine if there exist an abandoned food source and replace it using a scout bee***

11) ***Save in memory the best solution so far***

12) ***cycle = cycle + 1***

13) ***Until cycle = M C N***

The complete program for the standard ABC algorithm can be found in Appendix A.

3.5 Development of Cultural Artificial Bee Colony Algorithm

For the CABCA, all the processes listed in methodology 1) and 2) of this work were repeated. Then, the cultural knowledge that exists in cultural evolutional process is added at the employee phase, to improve the ABC algorithm. The experience of the selected individuals from the population (bee) contributes to the cultural knowledge by means of acceptance function. In turn, the knowledge controls the evolution of the population by means of influence function. Consequently, four variants of CABCA (CABCA(Ns), CABCA(Sd), CABCA(Ns+Sd) and CABCA(Ns+Nd) were developed using the different influence function.

Below is the pseudo-code for the "cultured" ABC algorithm.

1) *Begin*
2) *Initialize the solution population,*
3) *Evaluate population*
4) *cycle = 1*
5) *Repeat*
6) *Generate new solutions for the employed bees using (3.2) and evaluate them.Keep the best solution between current and candidate*
7) ***Update the belief space with the given problem domain and candidate solutions. The structure of the belief space is given in Figure 2.5***
8) ***Apply acceptance function (top 20%)***

9) Select the visited solution for onlooker bees by their fitness
10) Generate new solutions for the Onlooker bees using (3.2) and evaluate them
11) Keep the best solution between current and candidate
12) Determine if there exist an abandoned food source and replace it using a scout bee
13) Save in memory the best solution so far
14) cycle = cycle + 1
15) Until cycle = M C N

3.5.1 CABCA Knowledge adjustment

Each of the knowledge in the belief space has its own rules and approach of operations. This rule which is also referred to as update function as described in sub-section 2.2.5/B and 2.2.5/C for situational and normative knowledge respectively.

3.5.2 CABCA variants

The belief space knowledge can influence the evolutional operation, in two ways (Reynolds and Chung, 1997):

1) Determining the step size of the evolution
2) Determine the direction of evolution

These two operations are applied to the solution search equation for the ABC as given in equation (3.2) as adaptive parameters in order to balance exploration and exploitation which is often necessary for good results and sufficient convergence speed of the algorithm, especially for complex, high dimensional, multimodal problems with many locally optimal points. The variants were developed using the following configurations:

A. CABCA(Ns): CABCA using normative knowledge

This is the simplest form of cultural influence. This variant uses only the normative knowledge to determine the step size of evolution. For all components $i = 1,...,t$ and $j = 1,...,n.$ This can be expressed mathematically as follows:

$$x_{ij}^{t+1} = x_{ij}^{t} + \left| size(I_j) \right| \cdot \phi_{ij}\left(x_{ij} - x_{kj}\right) \tag{3.4}$$

where $size(I_j) = u_i - l_i$ is the size of the belief space interval for parameter $i,$ which is decided by the normative knowledge for i^{th} variable (Chung, 1997).

B. CABCA(Sd): CABCA using situational knowledge

In this variant only situational knowledge of the current exemplar or best solution found so far, is used to decide the direction of evolution. The variation is influenced as follows:

$$x_{ij}^{t+1} = \begin{cases} x_{ij}^{t} + \left| \phi_{ij}\left(x_{ij} - x_{kj}\right) \right| & if \quad x_{ij}^{t} \prec s_j^{t} \\ x_{ij}^{t} - \left| \phi_{ij}\left(x_{ij} - x_{kj}\right) \right| & if \quad x_{ij}^{t} \succ s_j^{t} \\ x_{ij}^{t} + \phi_{ij}\left(x_{ij} - x_{kj}\right) & otherwise \end{cases} \tag{3.5}$$

C. CABCA(Ns+Sd): CABCA using normative and situational knowledge

This variant uses both normative knowledge to determine the step size, and situational knowledge to determine the direction as shown in the following, for all components $i = 1,..,t$ and $j = 1,...,n$

$$x_{ij}^{t+1} = \begin{cases} x_{ij}^{t} + \left|\delta \times \phi_{ij}\left(x_{ij} - x_{kj}\right)\right| & if \quad x_{ij}^{t} \prec s_{j}^{t} \\ x_{ij}^{t} - \left|\delta \times \phi_{ij}\left(x_{ij} - x_{kj}\right)\right| & if \quad x_{ij}^{t} \succ s_{j}^{t} \\ x_{ij}^{t} + \delta \times \phi_{ij}\left(x_{ij} - x_{kj}\right) & otherwise \end{cases} \tag{3.6}$$

where $\delta = size(I_j)$ is the size of the belief space interval which is $u_i - l_i$

If an individual's parameter value is less than that of the current best, then the absolute value of the calculated step size, $size(I_j)$, is added to the current parameter value. If an individual's parameter value is greater than that of the current best, then the absolute value of the step size, $size(I_j)$, is subtracted from the current parameter value. If an individual's parameter value is equal to that of the current best, then $size(I_j) \cdot \phi_{ij}\left(x_{ij} - x_{kj}\right)$ is just added to the current parameter value. In this case the direction of evolution will be random.

D. CABCA(Ns+Nd): CABCA using normative for step size and direction of evolution

This variant uses only normative knowledge to influence the evolution of both step size and direction. The basic idea behind this variant is to perturb small in a random direction when the parameter value of a parent is in the acceptable range; otherwise perturb according to the current belief range found toward the left or right boundary of the current range in the belief space. For all components $i = 1,...,t$ and $j = 1,...,n$.

$$x_{ij}^{t+1} = \begin{cases} x_{ij}^{t} + \left|\omega \times \phi_{ij}\left(x_{ij} - x_{kj}\right)\right| & if \quad x_{ij}^{t} \prec l_{j} \\ x_{ij}^{t} - \left|\omega \times \phi_{ij}\left(x_{ij} - x_{kj}\right)\right| & if \quad x_{ij}^{t} \succ l_{j} \\ x_{ij}^{t} + \left|\omega\right| \times \phi_{ij}\left(x_{ij} - x_{kj}\right) & otherwise \end{cases} \tag{3.7}$$

where $|\omega| = size(I_k) = |u_k - l_k|$ which represents the size of the current upper limit and lower limit in the belief space for parameter j respectively.

Also if an individual's parameter value is less than that of the current best, then the absolute value of the calculated step size, $size(I_k)$, is added to the current parameter value. If an individual's parameter value is greater than that of the current best, then the absolute value of the step size, $size(I_k)$, is subtracted from the current parameter value. If an individual's parameter value is equal to that of the current best, then $size(I_k) \cdot \phi_{ij}(x_{ij} - x_{kj})$ is just added to the current parameter value. In this case the direction of evolution will be random.

The Flowchart of cultural artificial bee colony algorithim is shown in Figure 3.1

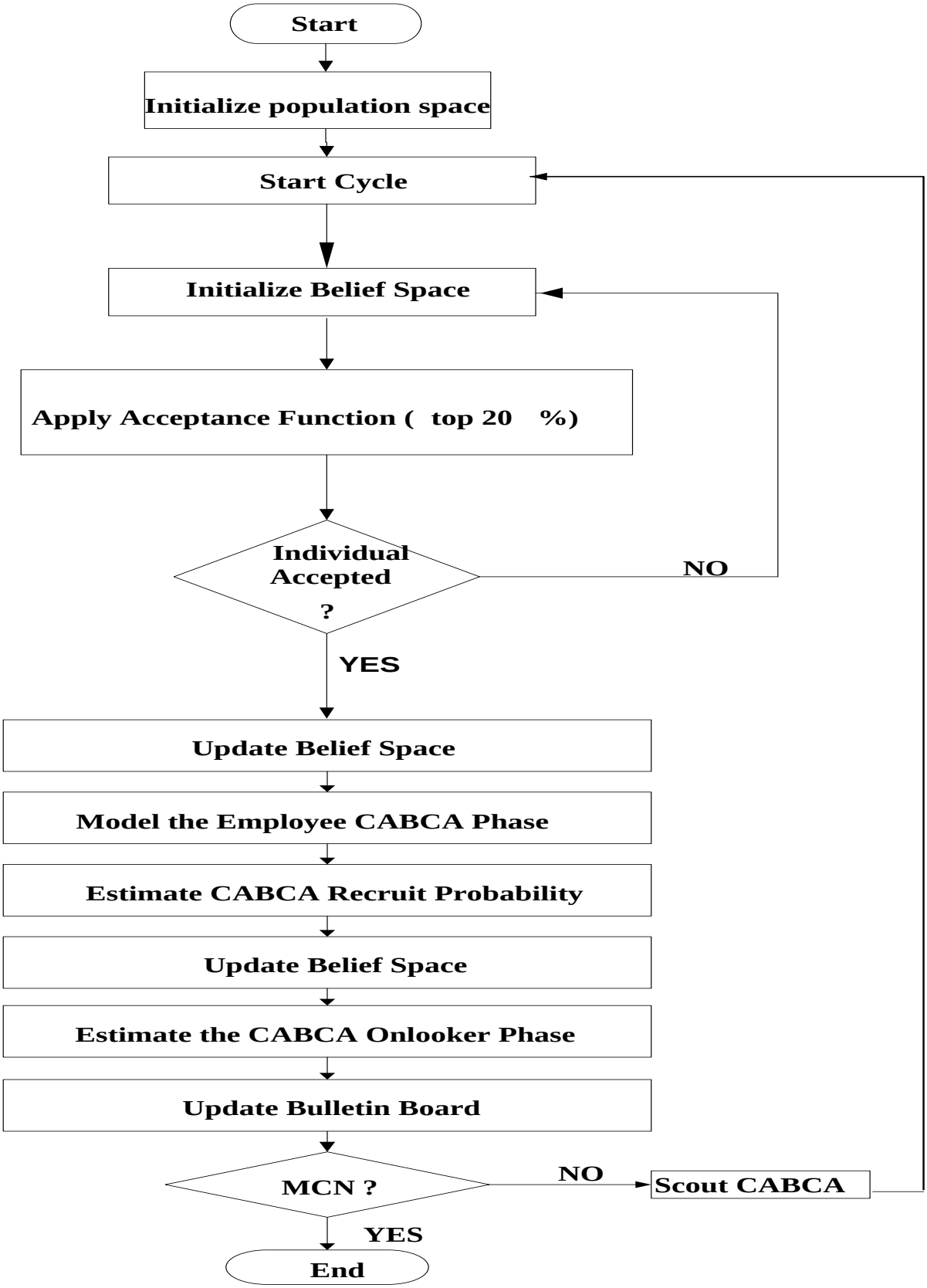


Figure 3.1: Flowchart of Cultural Artificial Bee Colony Algorithm (CABCA)

3.6 Performance Evaluation

In this subsection, the performance of the proposed CABCA variants on the twenty (20) optimization test functions presented in subsection 2.2.6 is evaluated and the result is reported and compared with the replicated standard ABC algorithm.

3.7 Determination of PID Gains Using CABCA

The variant of CABCA with superior performance over all the other variants developed in this research is used for the determination of optimal values of the parameter gains $(K_p, K_i \ \& \ K_d)$.

During the optimization process, the reference input and closed loop response of the process is used by the CABCA. Using the changed closed loop control performance according to the adjusted controller parameters at the each generation, the tuning algorithm searches the optimal parameters for the PID controller in an on-line mode to satisfy the desired system specifications. To illustrate the effectiveness of the presented method, compared the closed loop response to a step change of simulated system of (El-Telbany, 2013) with the following transfer function.

$$G(s) = \frac{\theta(s)}{V_a(s)} = \frac{K_m}{s\left[(Ls+R)(Js+K_f)+K_bK_m\right]} \tag{3.8}$$

It is of interest that to note that $K_b = K_m$. For our Dc motor, the physical constants are:

$R = 1\Omega$, $L = 0.5H$, $K_m = K_b = 0.01$, $K_f = 0.1Nms$ and $J = 0.02kgm^2/s^2$.

For these motor constants the transfer function (3.9) of the DC motor has the form:

$$G(s) = \frac{0.01}{0.005s^2 + 0.06s + 0.1} \tag{3.9}$$

The goal of PID controller is to show how each of K_p, K_i and K_d contributes to obtain: fast rise time, minimal settling time with small or no overshoot in the step response of the closed

loop system. Therefore, the objective function J is defined using the performance indices integral of the square of the error (ISE) as:

$$J = \sum_{i=1}^{N} f(r_i - y_i) = \sum_{i=1}^{N} f(e_i) \tag{3.9}$$

where r_i is the reference input, y_i is controlled output, e_i is control error and N is the number of samples. The PID tuning scheme with CABCA is shown in Figure 3.2

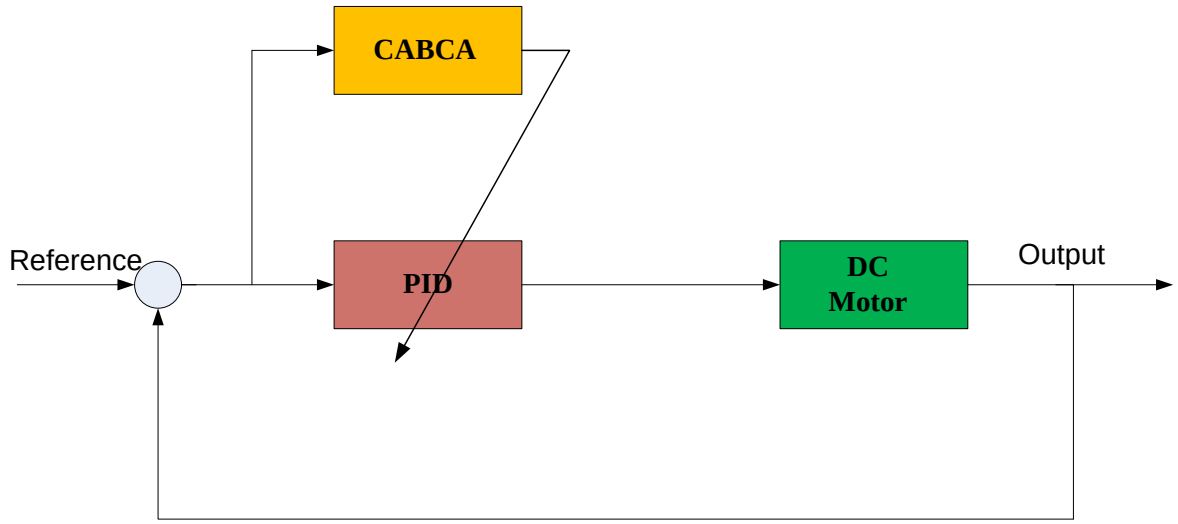


Figure 3.2: PID tuning system with CABCA

From Figure 3.2, a step input actuating force is appied at the reference as input signal. The entire system is developed into a Simulink model given in Figure 3.3.

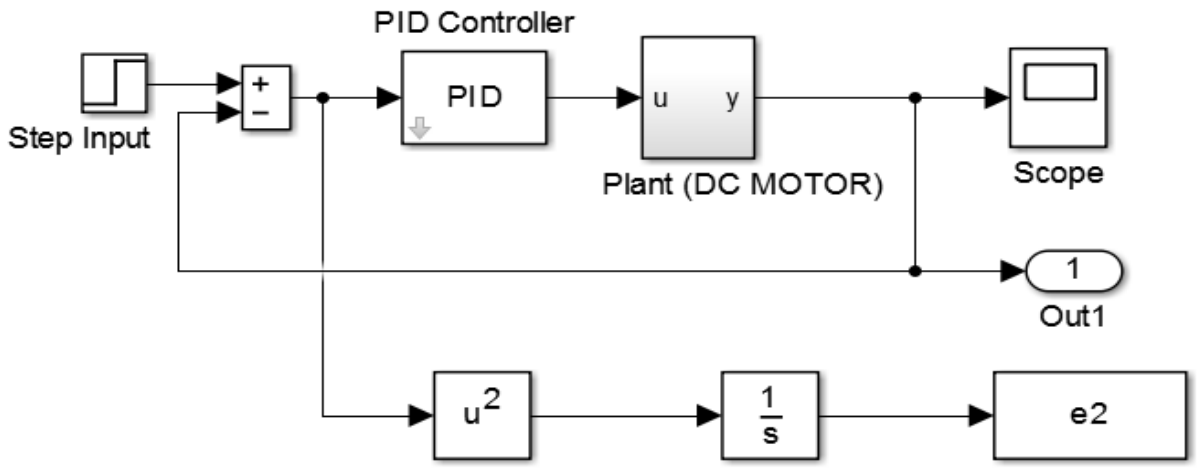


Figure 3.3: Simulink Model of the Entire System

Notice from Figure 3.3 that, the DC motor model is compressed into a subsystem called the plant and the PID controller is commpresed into another subsystem called the PID Controller. This is to reduce the complexity of the Simulink design and provide easy simulation. The output of the system is feedback into the input via a feedback loop in order to determine the error.

Since the objective of this research is to determine the optimized values of PID controller gain capable of controlling the DC motor, the parameters in the PID controller given in Figure 3.3 were initially configured as K_p, K_i and K_d. The configuration interface of the PID controller is given in Figure 3.4

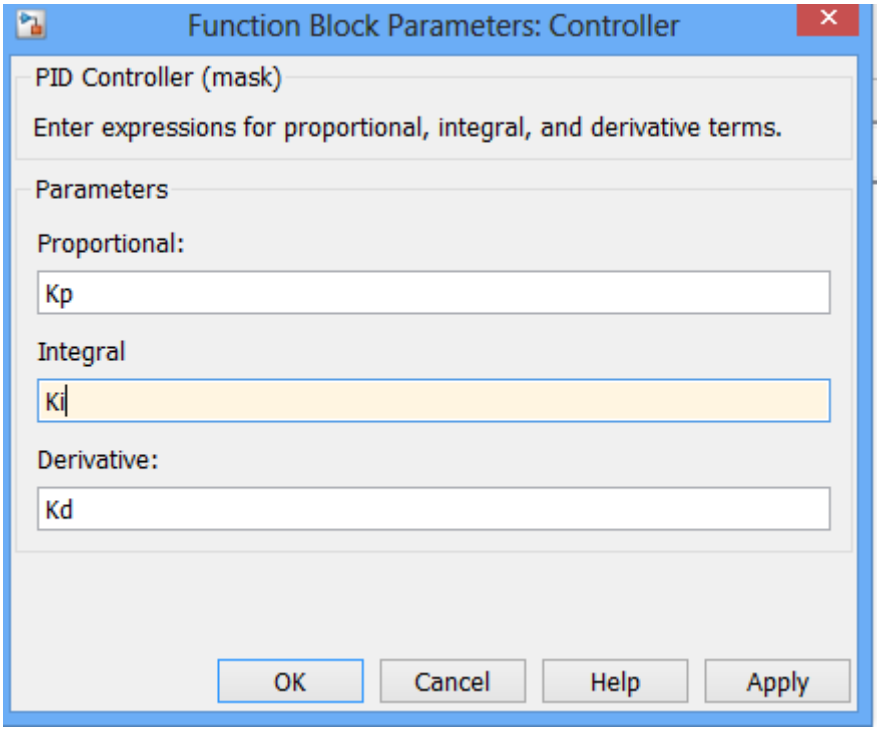


Figure 3.4: PID Controller Tuning Configuration Interface

Based on the discussion so far, a function file called the *objpid.m* which was formulated based on the objective function was programmed. The objective function file contains a MATLAB command called the *simset*, which was used to set the simulation work space and defined the solver. Another MATLAB command called the *sim*, was used to import the Simulink model into the objective function file, so that the developed CABCA algorithm can extract the necessary information for the PID controller parameter tuning. The implementation ideology is presented in Figure 3.4.

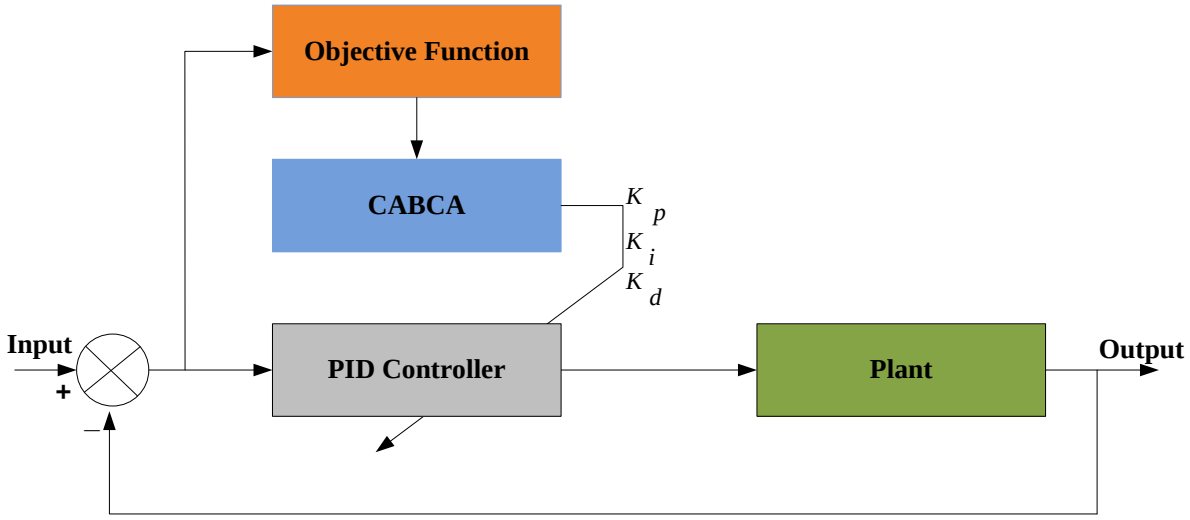


Figure 3.5: Block Representation of CABCA Based PID Controller Design

Figure 3.5 shows the simplified model representation of this research. The figure shows the pictorial view of how the developed CABCA tuned the PID controller and how the tuned controller is used for the DC motor position control.

The MATLAB program for the implementation of the proposed algorithm can be found in Appendix E.

CHAPTER FOUR

RESULTS AND DISCUSSIONS

4.1 INTRODUCTION

In this Chapter, the performances of the CABCA(Ns), CABCA(Sd), CABCA(Ns+Sd) and CABCA(Ns+Nd) are evaluated using the optimization test functions discussed in subsection 2.2.4. 3D plots were presented to enhance the understanding of all the test function. The performances of the proposed algorithms were compared with the performance of the replicated artificial bee colony (ABC) algorithm with reference to the global minima. The PID tuning parameter gains determined using CABCA and the responses in the speed control of a DC motor were also presented in this chapter.

4.2 Initialization Results of ABC Parameters

During initializations, control parameters SN, D, L and MCN of each solution x_i and v_{ij} values were initialized to random values from $\lfloor \max_j, \min_j \rfloor$ and [-1, 1], respectively. The performance of the ABC algorithm depends largely on the choice of the control parameters. In this research, the values of the parameters which give an acceptable optimal result are given in the following table 4.1.

Table 4.1: Parameter Setting for ABC

S/N	Parameters	Values
1	SN	100
2	D	50
3	MCN	5000
4	L	50

From the Table 4.5 SN (population size) is the number of food sources (solutions); D is the dimension of the problem that was optimized, MCN (Maximum Cycle Number) maximum number of generations (stopping criteria) and L is the limit which was used to diversify the

search to determine the number of allowable generations for which each non-improved food source is to be abandoned.

4.3 Result of Standard ABC Algorithm

Having employed all the processes listed in methodology (1) and (2) of this work, the result of the standard ABC on twenty (20) standard benchmark functions was computed using MATLAB R2015a. Meanwhile, the statistical results of each algorithm are presented in Table 4.2. The results show the average of 20 runs for each test function and their global minimal values.

Table 4.2: Results of ABC algorithm

Test Function	G_{min}	ABC
Ackley	0.000E+00	2.1157E-13
Axis parallel	0.0000E+00	9.5846E-16
CM	-3.0000E+00	-3.0000E+00
DeJongf4	0.0000E+00	5.4743E-16
ExpFun	1.0000E+00	1.0000E+00
Griewangk	0.0000E+00	0.0000E+00
Hyperelliptic	0.0000E+00	1.0000E-15
Michalewicz	-9.6602E+00	-26.5754E+00
Neumaier3	-4.9300E+03	-2.8331E+03
PM1	0.0000E+00	1.2349E-15
PM2	0.0000E+00	1.55319E-11
Quartic	0.0000E+00	10.9770E+00
Rastrigin	0.0000E+00	2.3628E-12
Rosenbrock	0.0000E+00	1.1236E+00
Sal	0.0000E+00	9.9873E-02
Schaffer	0.0000E+00	5.7452E-15
Schwefel	0.0000E+00	3.5536E+03
Sphere	0.0000E+00	1.0875E-15
Step	0.0000E+00	0.0000E+00
Zakharov	0.0000E+00	6.8857E-13

The next step presents the results when the introduction of knowledge which served as adaptive parameters in order to balance exploration and exploitation for better results.

4.4 Result of the Performance of the Developed CABCA

Cultural algorithm based ABC technique was developed and its performance was tested on some 20 different minimization functions. The results show the average of 20 runs for each test function and their global minimal values. The results of the performance of the developed CABCA variants are shown in Table 4.3 to 4.6.

Table 4.3: Results of CABCA(Ns) algorithm

Test Function	**G_{min}**	**CABCA(N_s)**
Ackley	0.000E+00	4.1466e-15
Axis parallel	0.0000E+00	1.0472E-16
CM	-3.0000E+00	-3.0000E+00
DeJongf4	0.0000E+00	4.3421E-16
ExpFun	1.0000E+00	1.0000E+00
Griewangk	0.0000E+00	0.0000E+00
Hyperelliptic	0.0000E+00	8.8185E-43
Michalewicz	-9.6602E+00	-2.3832E-13
Neumaier3	-4.9300E+03	-2.9000E-04
PM1	0.0000E+00	3.7832E-27
PM2	0.0000E+00	2.5959E-27
Quartic	0.0000E+00	8.4591E-01
Rastrigin	0.0000E+00	1.1806E-13
Rosenbrock	0.0000E+00	5.9490E-02
Sal	0.0000E+00	9.9873E-02
Schaffer	0.0000E+00	7.4541E-19
Schwefel	0.0000E+00	3.5542E+03
Sphere	0.0000E+00	9.8378E-16
Step	0.0000E+00	0.0000E+00
Zakharov	0.0000E+00	1.1469E-15

Table 4.4: Results of CABCA(Sd) algorithm

Test Function	**G_{min}**	**CABCA(N_s)**
Ackley	0.000E+00	1.9851E-13
Axis parallel	0.0000E+00	8.7434E-16
CM	-3.0000E+00	-3.0000E+00
DeJongf4	0.0000E+00	5.1616E-16
ExpFun	1.0000E+00	1.0000E+00
Griewangk	0.0000E+00	0.0000E+00
Hyperelliptic	0.0000E+00	9.1751E-16
Michalewicz	-9.6602E+00	-24.3579E+00
Neumaier3	-4.9300E+03	-2.9523E+03
PM1	0.0000E+00	1.2310E-15
PM2	0.0000E+00	4.4354E-12
Quartic	0.0000E+00	8.3048E+00
Rastrigin	0.0000E+00	2.1652E-12
Rosenbrock	0.0000E+00	1.1179E+00
Sal	0.0000E+00	9.9873E-02
Schaffer	0.0000E+00	6.5000E-15
Schwefel	0.0000E+00	3.5535E+03
Sphere	0.0000E+00	9.1795E-16
Step	0.0000E+00	0.0000E+00
Zakharov	0.0000E+00	1.3933E-15

Table 4.5: Results of CABCA(Ns+Sd) algorithm

Test Function	G_{min}	CABCA(Ns+Sd)
Ackley	0.000E+00	5.7676E-15
Axis parallel	0.0000E+00	5.4133E-16
CM	-3.0000E+00	-3.0000E+00
DeJongf4	0.0000E+00	4.5198E-16
ExpFun	1.0000E+00	0.0000E+00
Griewangk	0.0000E+00	1.0000E+00
Hyperelliptic	0.0000E+00	1.51583-42
Michalewicz	-9.6602E+00	-6.0315E-15
Neumaier3	-4.9300E+03	-2.9000E-04
PM1	0.0000E+00	4.0409E-30
PM2	0.0000E+00	2.5959E-27
Quartic	0.0000E+00	2.4158E-02
Rastrigin	0.0000E+00	3.5527E-15
Rosenbrock	0.0000E+00	1.4246E-02
Sal	0.0000E+00	9.9873E-02
Schaffer	0.0000E+00	7.2048E-19
Schwefel	0.0000E+00	3.5537E+03
Sphere	0.0000E+00	9.2855E-16
Step	0.0000E+00	0.0000E+00
Zakharov	0.0000E+00	1.6837E-19

Table 4.6: Results of CABCA(Ns+Nd) algorithm

Test Function	G_{min}	CABCA(Ns+Nd)
Ackley	0.000E+00	4.3750E-15
Axis parallel	0.0000E+00	8.5830E-16
CM	-3.0000E+00	-3.0000E+00
DeJongf4	0.0000E+00	4.7587E-16
ExpFun	1.0000E+00	0.0000E+00
Griewangk	0.0000E+00	1.0000E+00
Hyperelliptic	0.0000E+00	1.2385E-42
Michalewicz	-9.6602E+00	-2.4091E-13
Neumaier3	-4.9300E+03	-2.9000E-04
PM1	0.0000E+00	1.3401E-45
PM2	0.0000E+00	2.5959E-27
Quartic	0.0000E+00	4.2308E-21
Rastrigin	0.0000E+00	7.1059E-15
Rosenbrock	0.0000E+00	2.0832E-13
Sal	0.0000E+00	9.9873E-02
Schaffer	0.0000E+00	7.6013E-19
Schwefel	0.0000E+00	3.5533E+03
Sphere	0.0000E+00	1.0886E-15
Step	0.0000E+00	0.0000E+00
Zakharov	0.0000E+00	4.3144E-15

The results in general demonstrated that the use of a cultural framework for self-adaptation in ABC can produce substantial performance improvements as expressed in terms of CPU time. Thus, in the end, the proposed algorithm, CABCA, has enhanced and balanced exploration and exploitation processes. With this, it is expected to converge faster and to be able to reach global optimum efficiently.

4.5 Test Functions Visualization

The power of an algorithm remains its implementation in computer codes and its testing on large classes of test problems of different structures and characteristics. Consequently, to enhance the understanding and visualization of the local minimal point of most of the optimization test functions described in section 2.2.6, Figures (4.2 - 4.21) were generated in MATLAB-R2015a to show the surfaces and shapes of the test functions.

4.5.1 Ackley Function

The 3D visualization of Ackley function is shown in Figure 4.1

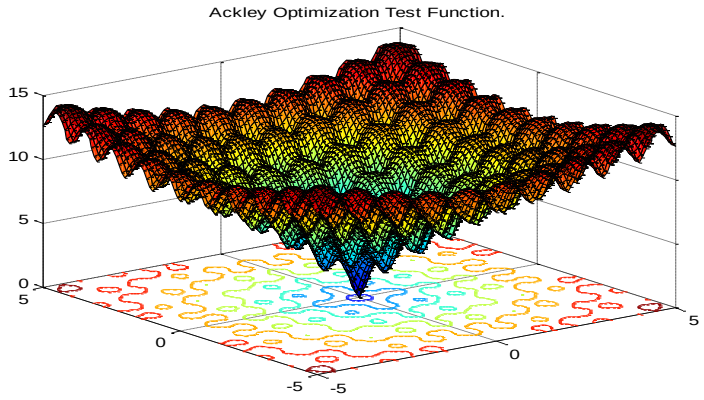


Figure 4.1: 3D Visualization of Ackley Function

From Figure 4.1, it is obvious that in Ackley function, the exact number of local minima is not known as there exist a lot, but the global minimum is located at the origin (0, 0).

4.5.2 Axis parallel Hyper-Ellipsoid Function

The global minimal of Axis parallel hyper-ellipsoid function is 0 which can be observed in the 3D visualization of the function in Figure 4.2. This function is described in sub-section 2.2.6/xx.

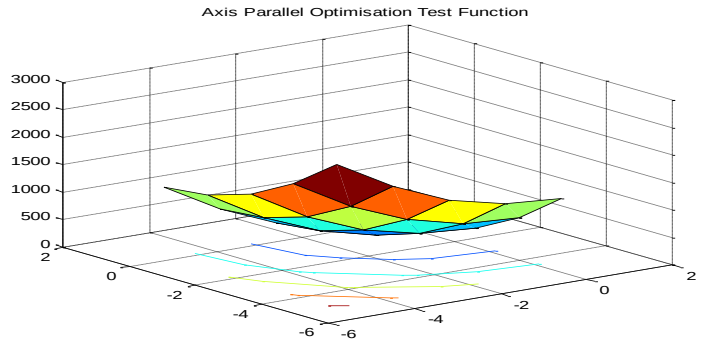


Figure 4.2: 3D Visualization of Axis Parallel Function

4.5.3 Cosine Mixture (CM) Function

CM function is similar to Ackley function with a lot of local minimal around a global minimal. The global minimum of this test function is at -3.0. The 3D visualization of CM function is shown in Figure 4.3.

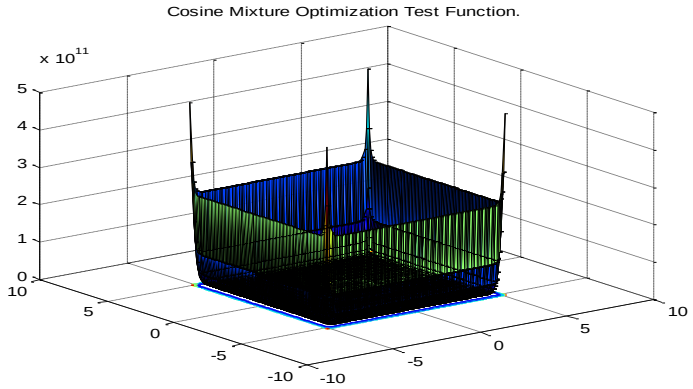


Figure 4.3: 3D Visualization of Cosine Mixture Function

4.5.4 De JongF4 Function

The global minima of De JongF4 function is 0 which can be observed in the 3D visualization of the function in Figure 4.4. This function is described in sub-section 2.2.6/xiv.

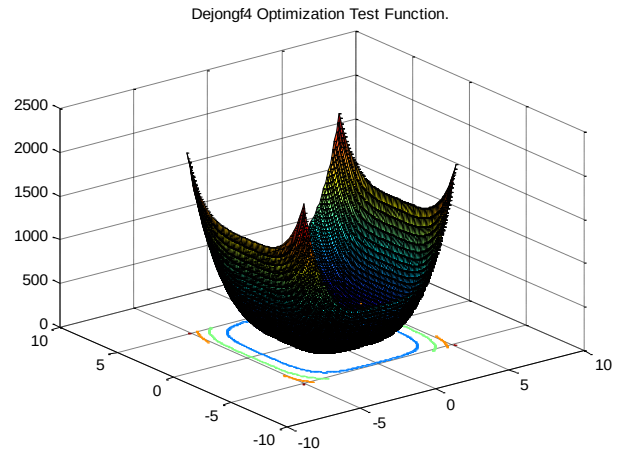


Figure 4.4: 3D Visualization of De Jongf4 Function

4.5.5 Exponential Function

The global minimal of Expfun function is 0 which can be observed in the 3D visualization of the function in Figure 4.5. This function is described in sub-section 2.2.6/xvi

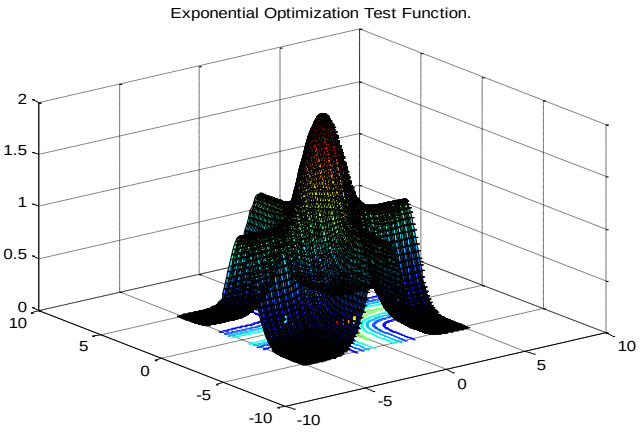


Figure 4.5: 3D Visualization of Exponential Function

From Figure 4.5, it is obvious that in Exponential function has a small number of significant local optima.

4.5.6 Griewangk function

This function has many widespread local minima regularly distributed as shown in the 3D visualization of Griewangk function in Figure 4.6

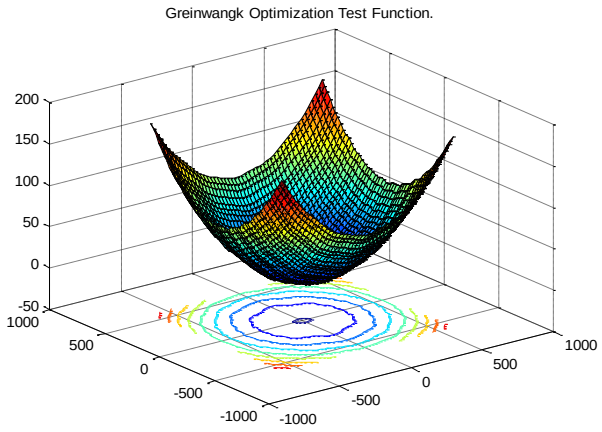


Figure 4.6: 3D Visualization of Greiwangk Function

4.5.7 Hyperelliptic Fuction

The Hyperelliptic function is similar to De JongF4 function with global optimal at point 0.

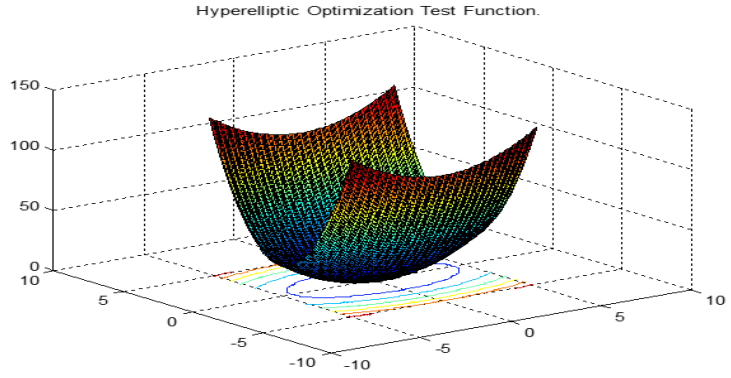


Figure 4.7: 3D Visualization of Hyperelliptic Function

4.5.8 Michalewicz Fuction

The 3D Michalewicz function visualization is as shown in Figure 4.8 with the global optimal point is at -9.660152.

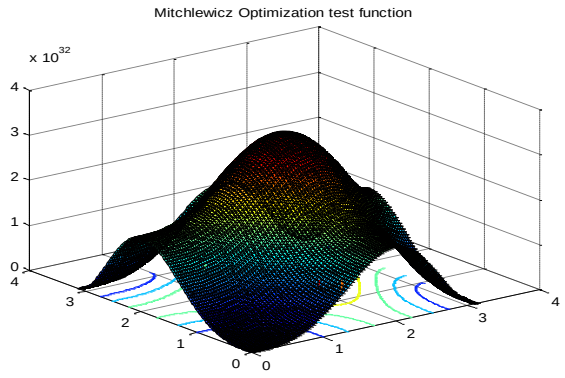


Figure 4.8: 3D Visualization of Michalewicz Function

4.5.9 Neumaier3 Function

The 3D Neumaier3 function visualization is as shown in Figure 4.9 with the global optimal point is at 0.

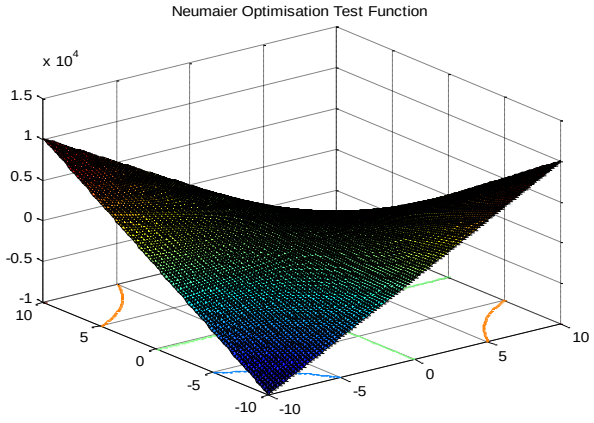


Figure 4.9: 3D Visualization of Neumaier Function

4.5.10 Penalized Levy and Montalvo 1 Function

The global minimal of PM1 function is 0 which can be observed in the 3D visualization of the function in Figure 4.10. This function is described in sub-section 2.2.6/ix.

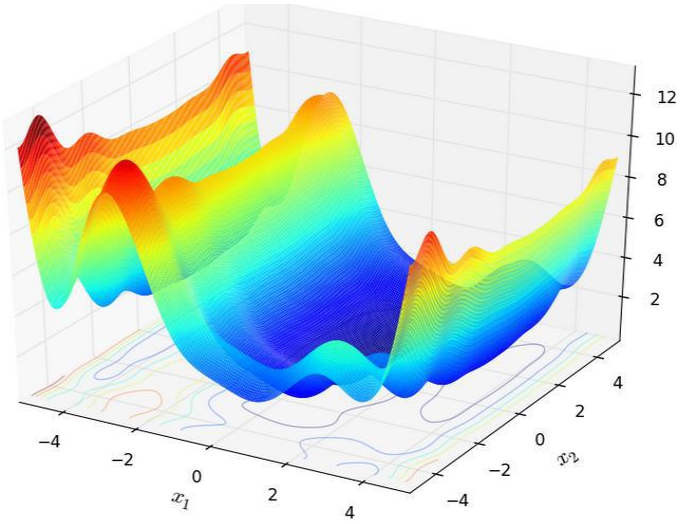


Figure 4.10: 3D Visualization of PM1 Function

4.5.11 Penalized Levy and Montalvo 2 Function

Just like the PM1 function, the LM2 function has its local minimal at 0 which can be observed in the 3D visualization of the function in Figure 4.11. The detail of this function is described sub-section 2.2.6/x

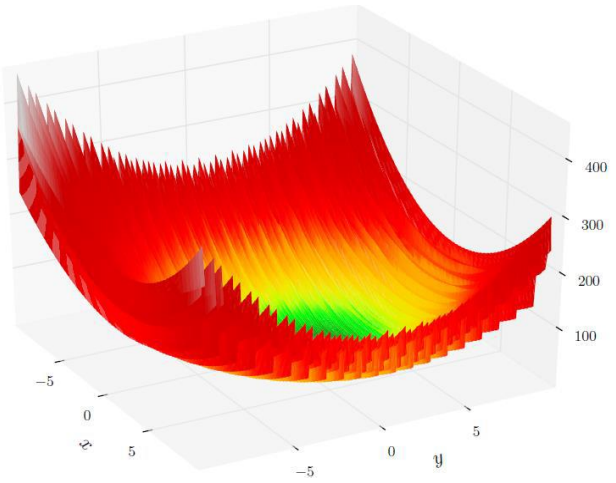


Figure 4.11: 3D Visualization of PM2 Function

4.5.12 Quartic Function

The Quartic Function is padded with noise as described in sub-section 2.2.6/xi. The 3D visualization of this function as shown in Figure 4.12 looks like a square shape with global optimal point at 0.

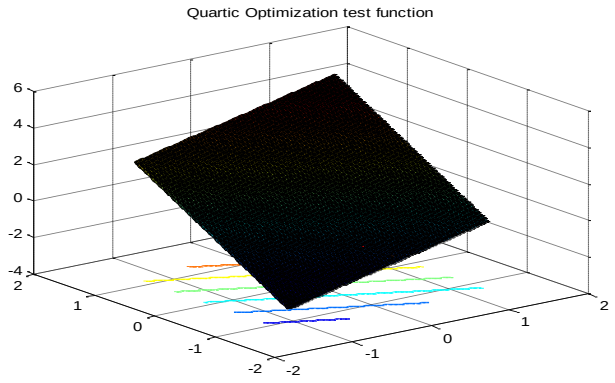


Figure 4.12: 3D Visualization of Quartic Function

4.5.13 Rastrigin Function

Rastrigin function is based on the function of De Jong with the addition of cosine modulation in order to produce many local minima. Conversely, the location of these local minima is regularly distributed. The 3D visualization of Rastrigin function is as shown in Figure 4.13 with the global minimal located at point zero.

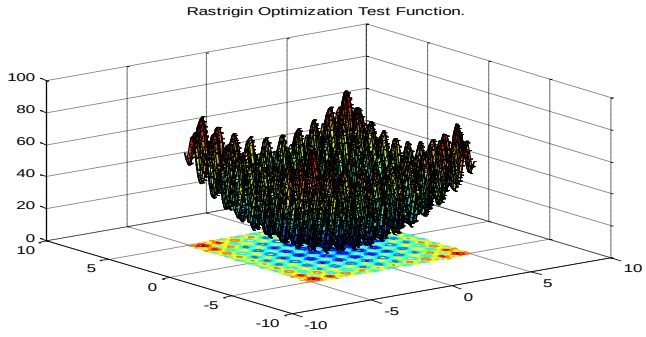


Figure 4.13: 3D Visualization of Rastrigin Function

From Figure 4.13, it is obvious that in Rastrigin function posses a huge number of significant local optima.

4.5.14 Rosenbrock Function

Rosenbrock function as described in sub-section 2.2.6/vi has its global minimum lies inside a long, narrow, parabolic shaped flat valley. To obtain convergence to the global optimum is often difficult to obtain. The 3D visualization of Rosenbrock function is as shown in Figure 4.14 with the global optimal located at point zero.

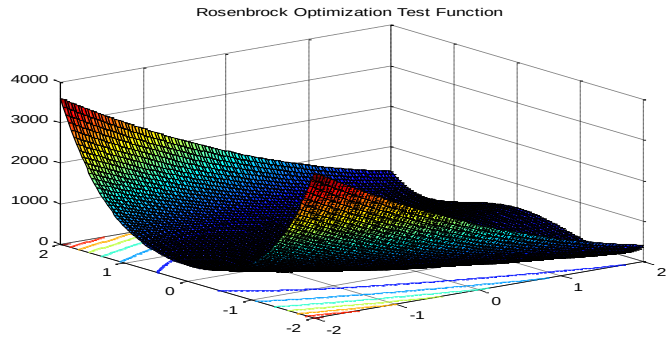


Figure 4.14: 3D Visualization of Rosenbrock Function

4.5.15 Sal Function

The 3D visualization of Sal function is as shown in Figure 4.15 with the global optima located at point zero.

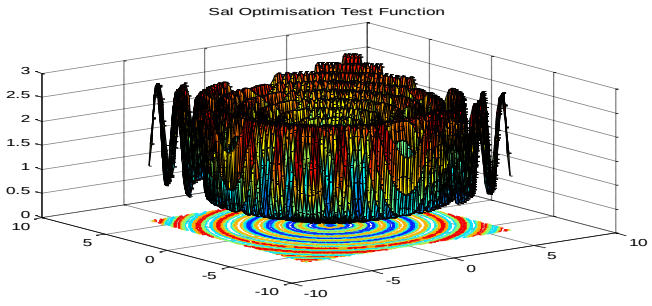


Figure 3.16 3D Visualization of Sal Function

4.5.16 Schaffer Function

Schaffer is a multimodal non-convex optimization test function described in sub-section 2.2.6/vii. Figure 4.16 shows the 3D visualization of Schaffer optimization test function and it can be seen that the global minimal of the Schaffer function is zero.

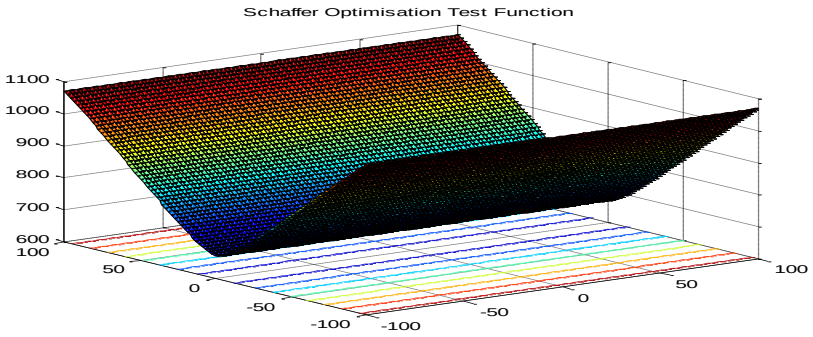


Figure 4.16: 3D Visualization of Schaffer Function

4.5.17 Schwefel Function

The global optima of this function are at 0, which is shown in the 3D visualization of Schwefel function is as shown in Figure 4.17. The detailed description can be found in sub-section 2.2.6/ii.

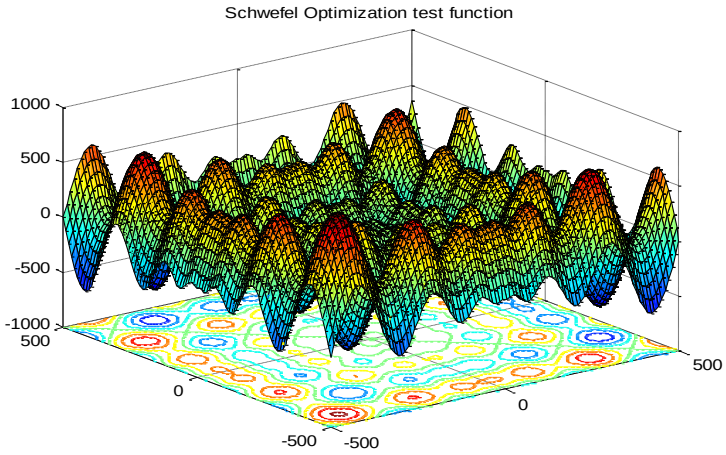


Figure 4.17: 3D Visualization of Schwefel Function

4.5.18 Sphere Function

The 3D visualization of Sphere function is as shown in Figure 4.18 with the global minimal points located at 0.

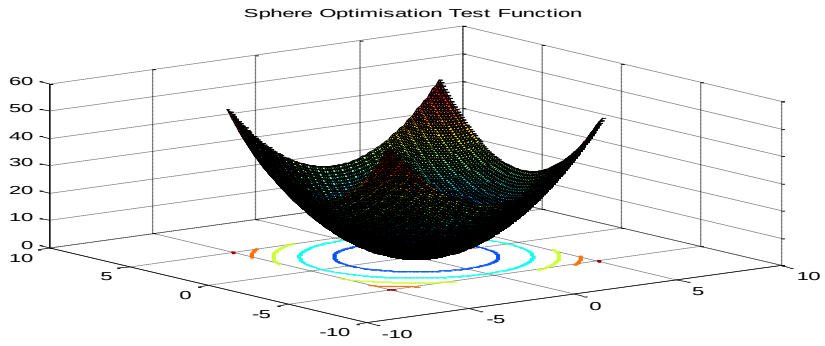


Figure 4.18: 3D Visualization of Sphere Function

4.5.19 Step Function

The Step function is a unimodal function. The 3D visualization of this function is as shown in Figure 4.19 with the global minima located at point 0.

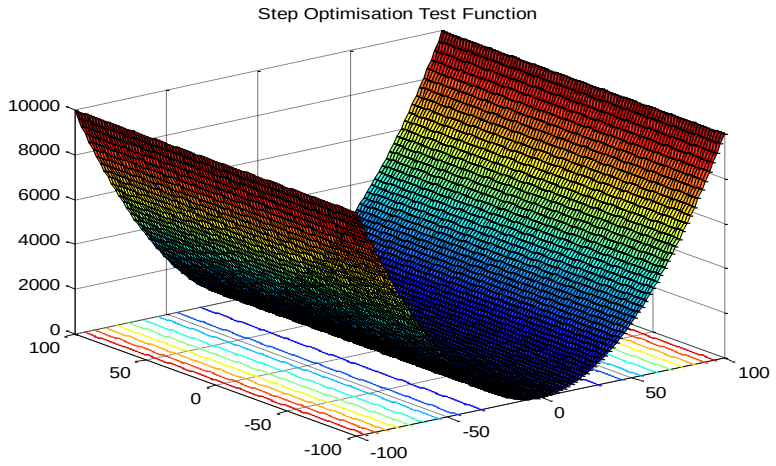


Figure 4.19: 3D Visualization of Step Function

4.5.20 Zakharov Function

The 3D visualization of Zakharov function is as shown in Figure 4.20 with the global minima at point 0.

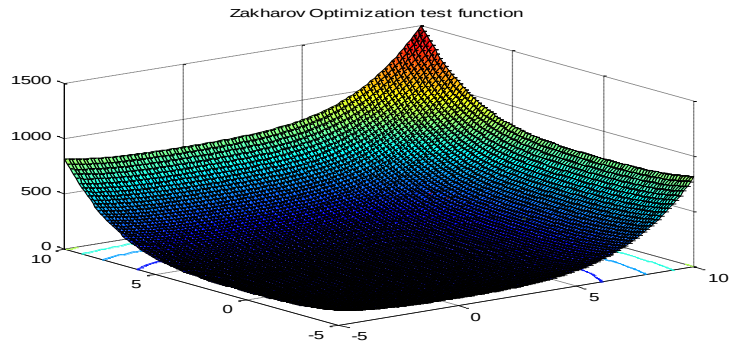


Figure 4.20: 3D Visualization of Zakharov Function

The MATLAB program for all the test functions (objective function) and the programme used to plot all the 3D graphs can be found in Appendix C and D respectively.

4.6 Performance, Evaluation and Comparison of ABC and CABCA

In this section, the simulation results in section 4.3 and 4.4 are evaluated and compared. The t-test introduced in the last column presented in Table 4.6 to 4.9, clearly indicate the statistical significance of the algorithms.

4.6.1 Performance evaluation of CABCA(Ns) using normative knowledge.

The standard Artificial Bee Colony Algorithm was first replicated. The performance of the replicated ABC was evaluated using twenty most commonly used optimization test functions as discussed in section 2.2.6. The performance of CABCA(Ns) when the normative knowledge

was used to determine the step size of the evolution. The result obtained from CABCA(Ns) is compared with that of ABC. The t-test is used to assess and compare the effectiveness and efficiency of the algorithms. The results are tabulated as in Table 4.7.

Table 4.7: Performance Evaluation of CABCA(Ns) and ABC

Test Function	**G_{min}**	**ABC**	**CABCA(N_s)**	**Better Performance (t-Test with λ =0.99)**
Ackley	0.000E+00	2.1157E-13	***4.1466e-15***	CABCA(Ns)
Axis parallel	0.0000E+00	9.5846E-16	***1.0472E-16***	CABCA(Ns)
CM	-3.0000E+00	-3.0000E+00	-3.0000E+00	Similar
DeJongf4	0.0000E+00	5.4743E-16	***4.3421E-16***	CABCA(Ns)
ExpFun	1.0000E+00	1.0000E+00	1.0000E+00	Similar
Griewangk	0.0000E+00	0.0000E+00	0.0000E+00	Similar
Hyperelliptic	0.0000E+00	1.0000E-15	***8.8185E-43***	CABCA(Ns)
Michalewicz	-9.6602E+00	-26.5754E+00	***-2.3832E-13***	CABCA(Ns)
Neumaier3	-4.9300E+03	***-2.8331E+03***	-2.9000E-04	ABC
PM1	0.0000E+00	1.2349E-15	***3.7832E-27***	CABCA(Ns)
PM2	0.0000E+00	1.55319E-11	***2.5959E-27***	CABCA(Ns)
Quartic	0.0000E+00	10.9770E+00	***8.4591E-01***	CABCA(Ns)
Rastrigin	0.0000E+00	2.3628E-12	***1.1806E-13***	CABCA(Ns)
Rosenbrock	0.0000E+00	1.1236E+00	***5.9490E-02***	CABCA(Ns)
Sal	0.0000E+00	9.9873E-02	9.9873E-02	Similar
Schaffer	0.0000E+00	5.7452E-15	***7.4541E-19***	CABCA(Ns)
Schwefel	0.0000E+00	3.5536E+03	3.5542E+03	Similar
Sphere	0.0000E+00	1.0875E-15	***9.8378E-16***	CABCA(Ns)
Step	0.0000E+00	0.0000E+00	0.0000E+00	Similar
Zakharov	0.0000E+00	6.8857E-13	***1.1469E-15***	CABCA(Ns)

From Table 4.7, out of the 20 functions, CABCA(Ns) performs better than ABC on 13 functions (65%); while ABC performs better only on one (1) function (5%). On the remaining six (6) functions (30%), their results are similar (i.e., the performance difference is not

statistically significant in t-tests with at least 99% degree of confidence). Thus the overall performance of CABCA(Ns) is better than standard ABC.

Note that the algorithms with the best performance with respect to the global solution are shown in bold and italic characters in the tables respectively.

4.6.2 Performance Evaluation of CABCA(Sd) Using Situational Knowledge.

The performance of the CABCA when the situational knowledge of the current exemplar or best solution found so far was used to determine the step size of evolution is evaluated in this section. The result obtained is compared with the results obtained from ABC and CABCA(Ns). The results are tabulated as in Table 4.8.

Table 4.8: Performance Evaluation of CABCA(Sd), CABCA(Ns) and ABC

Test Function	**G_{min}**	**ABC**	**CABCA(N_s)**	**CABCA(S_d)**	**Best Performance (t-Test with λ =0.99)**
Ackley	0.000E+00	2.1157E-13	***4.1466e-15***	1.9851E-13	CABCA(Ns)
Axis parallel	0.0000E+00	9.5846E-16	***1.0472E-16***	8.7434E-16	CABCA(Ns)
CM	-3.0000E+00	-3.0000E+00	-3.0000E+00	-3.0000E+00	Similar
DeJongf4	0.0000E+00	5.4743E-16	***4.3421E-16***	5.1616E-16	CABCA(Ns)
ExpFun	1.0000E+00	1.0000E+00	0.0000E+00	1.0000E+00	Similar
Griewangk	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	Similar
Hyperelliptic	0.0000E+00	1.0000E-15	***8.8185E-43***	9.1751E-16	CABCA(Ns)
Michalewicz	-9.6602E+00	-26.5754E+00	-2.3832E-13	***-24.3579E+00***	CABCA(Sd)
Neumaier3	-4.9300E+03	-2.8331E+03	-2.9000E-04	***-2.9523E+03***	CABCA(Sd)
PM1	0.0000E+00	1.2349E-15	***3.7832E-27***	1.2310E-15	CABCA(Ns)
PM2	0.0000E+00	1.55319E-11	***2.5959E-27***	4.4354E-12	CABCA(Ns)
Quartic	0.0000E+00	10.9770E+00	***8.4591E-01***	8.3048E+00	CABCA(Ns)
Rastrigin	0.0000E+00	2.3628E-12	***1.1806E-13***	2.1652E-12	CABCA(Ns)
Rosenbrock	0.0000E+00	1.1236E+00	***5.9490E-02***	1.1179E+00	CABCA(Ns)
Sal	0.0000E+00	9.9873E-02	9.9873E-02	9.9873E-02	Similar
Schaffer	0.0000E+00	5.7452E-15	***7.4541E-19***	6.5000E-15	CABCA(Ns)
Schwefel	0.0000E+00	3.5536E+03	3.5542E+03	3.5535E+03	Similar
Sphere	0.0000E+00	1.0875E-15	9.8378E-16	***9.1795E-16***	CABCA(Sd)
Step	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	Similar
Zakharov	0.0000E+00	6.8857E-13	***1.1469E-15***	1.3933E-15	CABCA(Ns)

It can be observed from Table 4.8 that the variant of CABCA that considered normative knowledge produced the best result for 11 (55%) out of twenty optimization test cases

considered while the variant that employed the situational knowledge produced the best result in 3 (15%) and similar in 6 (30%). Comparing ABC with CABCA(Sd) shows that CABCA(Sd) excelled in 11 (55%), ABC in 2 (10%) and similar results in 7 (35%).

4.6.3 Performance Evaluation of CABCA(Ns+Sd) using Situational and Normative Knowledge

The performance of the CABCA when the normative knowledge was used to guide the step size and the situational knowledge of the current exemplar or best solution found so far was used to determine the direction of evolution is evaluated in this section. The result obtained is compared with the results obtained from ABC, CABCA(Ns) and CABCA(Sd). The results are tabulated as in Table 4.9.

Table 4.9: Performance Evaluation of CABCA(Ns+Sd), CABCA(Sd), and CABCA(Ns) and ABC

Test Function	G_{min}	ABC	CABCA(N_s)	CABCA(S_d)	CABCA(Ns+Sd)	Best Performance (t-Test with λ =0.99)
Ackley	0.000E+00	2.1157E-13	***4.1466e-15***	1.9851E-13	5.7676E-15	CABCA(Ns)
Axis parallel	0.0000E+00	9.5846E-16	***1.0472E-16***	8.7434E-16	5.4133E-16	CABCA(Ns)
CM	-3.0000E+00	-3.0000E+00	-3.0000E+00	-3.0000E+00	-3.0000E+00	Similar
DeJongf4	0.0000E+00	5.4743E-16	***4.3421E-16***	5.1616E-16	4.5198E-16	CABCA(Ns)
ExpFun	1.0000E+00	1.0000E+00	0.0000E+00	1.0000E+00	0.0000E+00	Similar
Griewangk	0.0000E+00	0.0000E+00	1.0000E+00	0.0000E+00	1.0000E+00	Similar
Hyperelliptic	0.0000E+00	1.0000E-15	***8.8185E-43***	9.1751E-16	1.51583-42	CABCA(Ns)
Michalewicz	-9.6602E+00	-26.5754E+00	-2.3832E-13	***-24.3579E+00***	-6.0315E-15	CABCA(Sd)
Neumaier3	-4.9300E+03	-2.8331E+03	-2.9000E-04	***-2.9523E+03***	-2.9000E-04	CABCA(Sd)
PM1	0.0000E+00	1.2349E-15	3.7832E-27	1.2310E-15	***4.0409E-30***	CABCA(Ns+Sd)
PM2	0.0000E+00	1.55319E-11	2.5959E-27	4.4354E-12	2.5959E-27	Similar
Quartic	0.0000E+00	10.9770E+00	8.4591E-01	8.3048E+00	***2.4158E-02***	CABCA(Ns+Sd)
Rastrigin	0.0000E+00	2.3628E-12	1.1806E-13	2.1652E-12	***3.5527E-15***	CABCA(Ns+Sd)
Rosenbrock	0.0000E+00	1.1236E+00	5.9490E-02	1.1179E+00	***1.4246E-02***	CABCA(Ns+Sd)
Sal	0.0000E+00	9.9873E-02	9.9873E-02	9.9873E-02	9.9873E-02	Similar
Schaffer	0.0000E+00	5.7452E-15	7.4541E-19	6.5000E-15	***7.2048E-19***	CABCA(Ns+Sd)
Schwefel	0.0000E+00	3.5536E+03	3.5542E+03	3.5535E+03	3.5537E+03	Similar
Sphere	0.0000E+00	1.0875E-15	9.8378E-16	***9.1795E-16***	9.2855E-16	CABCA(Sd)
Step	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	Similar
Zakharov	0.0000E+00	6.8857E-13	1.1469E-15	1.3933E-15	***1.6837E-19***	CABCA(Ns+Sd)

From the Table 4.9, it could be observed that the variant of CABCA that employ both the normative and situational knowledge to guide the evolution produced the best result in 6 out of twenty (20). While the variant that employed the normative and situational knowledge produced the best results in four (4) and three (3) cases respectively. On the remaining seven (7) functions, their results are similar. In overall, CABCA(Ns+Sd) outperformed the ABC in 11 (55%) and 2 (10%) cases respectively and similar in 7 (35%).

4.6.4 Performance Evaluation of CABCA(Ns+Nd) using Normative Knowledge

The performance of the CABCA when only the normative knowledge was used to guide the step size and direction of evolution is evaluated in this section. The result obtained is compared with the results obtained from ABC, CABCA(Ns), CABCA(Sd) and CABCA(Ns+Nd). The results are tabulated as in Table 4.10.

Table 4.10 Performance Evaluation of CABCA(Ns+Nd), CABCA(Ns+Sd), CABCA(Sd), CABCA(Ns) and ABC

Test Function	G_{min}	ABC	CABCA(N_s)	CABCA(S_d)	CABCA(Ns+Sd)	CABCA(Ns+Nd)	Best Performance (t-Test with λ =0.99)
Ackley	0.000E+00	2.1157E-13	***4.1466e-15***	1.9851E-13	5.7676E-15	4.3750E-15	CABCA(Ns)
Axis parallel	0.0000E+00	9.5846E-16	***1.0472E-16***	8.7434E-16	5.4133E-16	8.5830E-16	CABCA(Ns)
CM	-3.0000E+00	-3.0000E+00	-3.0000E+00	-3.0000E+00	-3.0000E+00	-3.0000E+00	Similar
DeJongf4	0.0000E+00	5.4743E-16	***4.3421E-16***	5.1616E-16	4.5198E-16	4.7587E-16	CABCA(Ns)
ExpFun	1.0000E+00	1.0000E+00	0.0000E+00	1.0000E+00	0.0000E+00	0.0000E+00	Similar
Griewangk	0.0000E+00	0.0000E+00	1.0000E+00	0.0000E+00	1.0000E+00	1.0000E+00	Similar
Hyperelliptic	0.0000E+00	1.0000E-15	***8.8185E-43***	9.1751E-16	1.51583-42	1.2385E-42	CABCA(Ns)
Michalewicz	-9.6602E+00	-26.5754E+00	-2.3832E-13	***-24.3579E+00***	-6.0315E-15	-2.4091E-13	CABCA(Sd)
Neumaier3	-4.9300E+03	-2.8331E+03	-2.9000E-04	***-2.9523E+03***	-2.9000E-04	-2.9000E-04	CABCA(Sd)
PM1	0.0000E+00	1.2349E-15	3.7832E-27	1.2310E-15	4.0409E-30	***1.3401E-45***	CABCA(Ns+Nd)
PM2	0.0000E+00	1.55319E-11	2.5959E-27	4.4354E-12	2.5959E-27	2.5959E-27	Similar
Quartic	0.0000E+00	10.9770E+00	8.4591E-01	8.3048E+00	2.4158E-02	***4.2308E-21***	CABCA(Ns+Nd)
Rastrigin	0.0000E+00	2.3628E-12	1.1806E-13	2.1652E-12	3.5527E-15	***7.1059E-15***	CABCA(Ns+Nd)
Rosenbrock	0.0000E+00	1.1236E+00	5.9490E-02	1.1179E+00	1.4246E-02	***2.0832E-13***	CABCA(Ns+Nd)
Sal	0.0000E+00	9.9873E-02	9.9873E-02	9.9873E-02	9.9873E-02	9.9873E-02	Similar
Schaffer	0.0000E+00	5.7452E-15	7.4541E-19	6.5000E-15	***7.2048E-19***	7.6013E-19	CABCA(Ns+Sd)
Schwefel	0.0000E+00	3.5536E+03	3.5542E+03	3.5535E+03	3.5537E+03	3.5533E+03	Similar
Sphere	0.0000E+00	1.0875E-15	9.8378E-16	***9.1795E-16***	9.2855E-16	1.0886E-15	CABCA(Sd)
Step	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	Similar
Zakharov	0.0000E+00	6.8857E-13	1.1469E-15	1.3933E-15	***1.6837E-19***	4.3144E-15	CABCA(Ns+Sd)

Notice that CABCA(Ns+Nd) outpaced the ABC in 10 (50%) and 2 (10%) cases respectively while they provide similar result in 8 (40%) cases.

In total, the variant of CABCA (i.e. CABCA(Ns), CABCA(Sd), CABCA(Ns+Sd) and CABCA(Ns+Nd) outperformed the standard ABC in all the optimization test functions. The superiority of CABCA justifies the essence of knowledge introduction in the belief space for self-adaptation. This has substantially improved the performance of the standard ABC in the search for global solution. In overall, out of the twenty (20) test cases, CABCA(Ns) and CABCA(Ns+Nd) each produced best result in four (4) test cases, CABCA(Sd) in three (3) cases and CABCA(Ns+Sd) in two (2) test cases.

The following points summarize the observations on the results.

1) Out of the 20 functions CABCA(Ns) performs best in 4 (20%) functions, CABCA(Ns+Nd) also in 4 (20%) functions, while CABCA(Sd) and CABCA(Ns+Sd) performed best in 3 (15%) and 2 (10%) functions respectively. On the remaining 7 (35%) functions, their results are similar (i.e., the performance difference is not statistically significant in t-tests with at least 99% degree of confidence). Thus the overall performance of CABCA is better than ABC.
2) Variants that employed normative knowledge triumphed in most of the test cases. Hence, normative knowledge seems to be the prevailing knowledge source for the optimization functions.
3) The unimodal and seperable functions are relatively easier to optimize. On these functions, the performance of ABC and CABCA variants are mostly similar.
4) The best performed variant is selected based on the success ratio, which is the number of successful runs that found the solution. In case of a tie as it exists between

CABCA(Ns) and CABCA(Ns+Nd), the mean best solution is used to break the tie. Consequently, the average mean performance of CABCA(Ns) is **1.1714e-15** and CABCA(Ns+Nd) is **5.3856e-14**. Hencefort CABCA(Ns) is taken as the best performed.

4.7 Computer Specification

All experiments were conducted on the average of 20 times, independently for each test function using MATLAB R2015a. The simulations were performed on HP-Pavilion g7-2270us with the following specifications.

Table 4.11: Computer system specification

S/N	Items	Specification
1	Processor	Intel Core i3
2	Random Access Memory (RAM)	6.00GB
3	Storage Capacity	750GB
4	Operating System (OS)	64-bit Window 8 pro
5	Rating	5.3

4.8 Application of the Developed Method for Optimal Determination of PID Parameters

As discussed in section 3.7, in order to demonstrate the effectiveness of the developed CABCA, the step responses of the closed system were compared with that of the existing methods in the literature. Hence, the results obtained by CABCA and ABC are compared with that of Ziegler Nichols (Z-N) method as presented in (El-Telbany, 2013). For CABCA and ABC, all bees start at a random position in the range $[0.0, 500.0]$ for each dimension. The colony size is 50, control parameter in order to abandon the food source is 100.

Therefore, in order to evaluate the performance of the optimized PID gains obtained using CABCA, the step response of the system was superimposed against the PID parameters obtained using the ABC and Z-N method. This can be demonstrated in Figure 4.21.

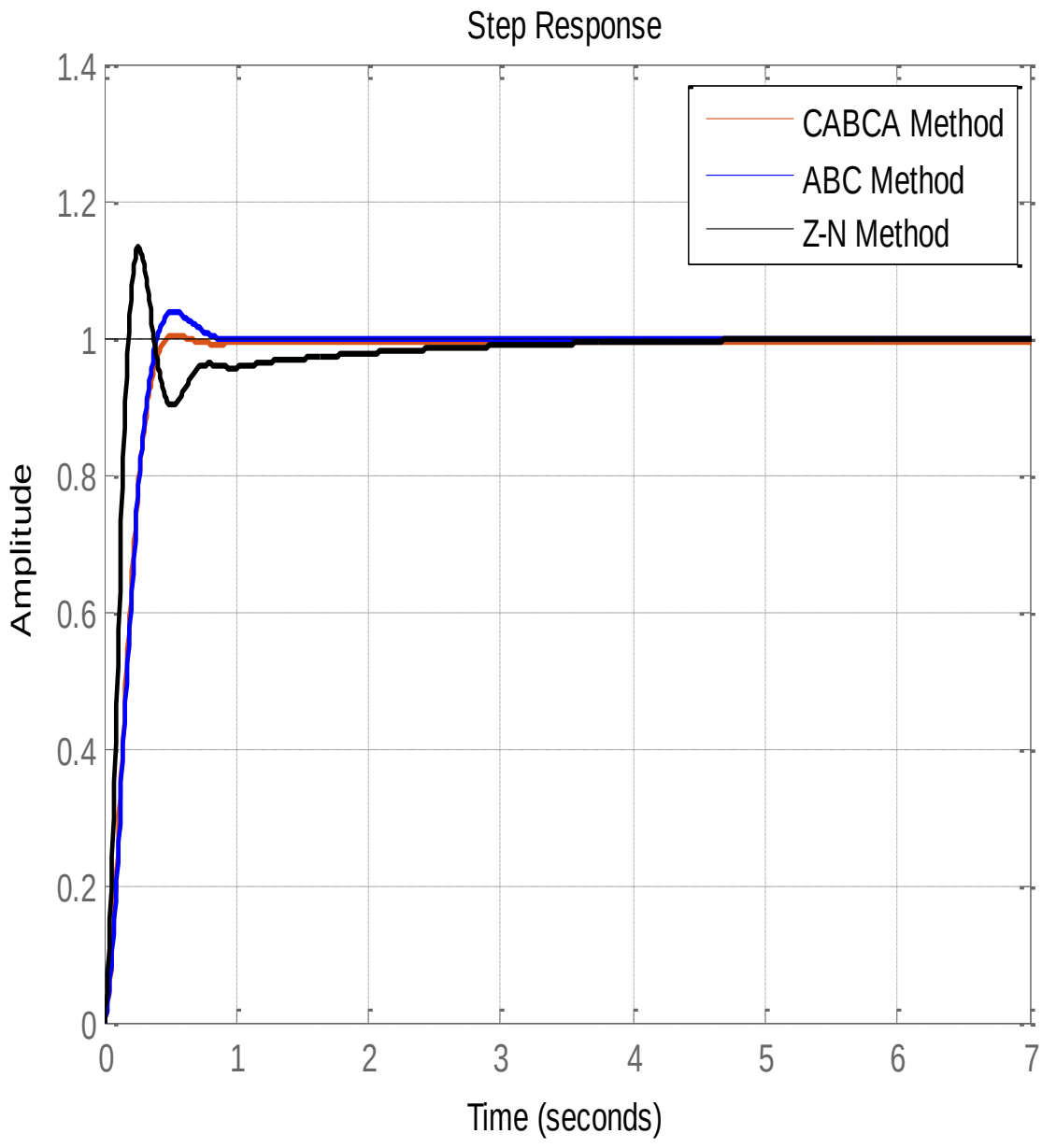


Figure 4.21: Step Response of the System

From Figure 4.21, it can be observed that all the tuning method attained the steady state of the system efficiently, which is a comfirmation of the validity of the proposed method. However, the system have a much better steady state performance when the PID controller gains were optimized using CABCA. The steady state performance of all the system is given in Table 4.12

Table 4.12: Performance Comparison of Z-N, ABC and CABCA

SN	Parameters/Method	Z-N	ABC	CABCA
1	K_p	8.042	3.164	3.042
2	K_i	5.351	5.742	5.351
3	K_p	0.039	0.03895	0.07895
4	Rise Time	0.1241	0.2584	0.2685
5	Settling Time	2.2057	0.6778	0.4178
6	Overshoot (%)	13.0722	3.7684	0.2869
7	SS Error	0	0	0

The following points summarize the observatios on the validation of the algorithm.

1) The CABCA based optimized PID controller obtained an overshoot of 0.2869% as compared with 3.7684% obtained using ABC and 13.0722% obtained using the Z-N approach.
2) Similarly, the CABCA tuned PID obtained a settling time of 0.4178 seconds as compared with 0.6778 seconds and 13.0722 seconds obtained using the standard ABC and the Z-N methods respectively.
3) As expected, all the three methods converge towards the same design specification (which is less than 20% overshoot and less 20 seconds settling time). But however, using the intelligent method, the time to computation of the PID gains has been significantly reduced to within 12 seconds.

The program used to plot the responses of the CABCA-PID can be found in Appendix E.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

PID controller is one of the most popular controllers which are still widely used in industries. One of the major challenges faced with PID controller design is selecting the appropriate gain such that the controller behaviour yields the optimal results with respect to design specification. Several intelligent techniques (such as ABC) have been proposed for selecting these PID parameters with little or no designer intervention. In order to overcome the common problems (local minima and poor convergence) associated with ABC, an improved method based on a modified Artificial Bee Colony (ABC) algorithm for optimal determination of gain parameters technique has been developed. The modified ABC algorithm is called the Cultural Artificial Bee Colony Algorithm (CABCA). Four new variants (CABCA(Ns), CABCA(Sd), CABCA(Ns+Sd) and CABA(Ns+Nd)) were developed in MATLAB R2015a and the performance of the developed varaints were evaluated using a total of twenty applied mathematical optimization test functions (such as Ackley, Michalewizc, Step, etc). Simulation result obtained shows that CABCA(Ns) has a better precession accuracy and higher ability of escaping local minimal when compared with the standard ABC and all the other variations of CABCA. The CABCA(Ns) was then employed to tune PID controller for DC motor drive system. It optimized the PID gain parameters $\left(K_p, K_i \text{ and } K_d\right)$ with a reduced settling time and low overshoot. Results clearly expressed that this method has been successful in comparison to classical Z-N approach and optimal PID tuning using ABC and can also be considered as a powerful tuning scheme for controllers. This is an indication of the validity of the CABCA.

5.2 Significant Contributions

The significant contributions of this research work are as follows:

1) Development of four cultural algorithm modified variants of the ABC (CABCA(Ns), CABCA(Sd), CABCA(Ns+Sd) and CABCA(Ns+Nd)) which outperformed the standard ABC when tested on 20 standard test functions by 65%, 60%, 50% and 55% respectively.
2) The DC motor attained steady-state in 0.4178s with the CABCA-based PID controller as against 0.6778s and 2.2057s obtained using standard ABC and Ziegler-Nichols (Z-N) tuned PID controllers respectively.

5.3 Limitations

The limitations of this research work are highlighted as follows:

1) The evolving population of the candidate solutions looses its diversity
2) The development of a model system for DC motor was not considered.

5.4 Recommendation for Further Works

The following possible areas of further work are recommended for consideration for future research:

1) The modeling of a DC motor should be considered for speed and position control system.
2) Other types of Cultural knowledge such as Historical, Topographical or Domain knowledge can be considered as alternative source of belief space structure instead of the Normative and/or Situational.

REFERENCES

Abachizadeh, M., Yazdi, M. R. H., and Yousefi-Koma, A. (2010). *Optimal tuning of pid controllers using artificial bee colony algorithm.* Paper presented at the 2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.

Akay, B., and Karaboga, D. (2012). A modified artificial bee colony algorithm for real-parameter optimization. *Information sciences, 192*, 120-142.

Alam, M. S., Islam, M. M., and Murase, K. (2015). Artificial Bee Colony Algorithm with Adaptive Explorations and Exploitations: A Novel Approach for Continuous Optimization. *Foundation of Computer Science, 8*(2), 32-43.

Åström, K., and Hägglund, T. (2004). Revisiting the Ziegler–Nichols step response method for PID control. *Journal of process control, 14*(6), 635-650.

Åström, K. J. (2000). Limitations on control system performance. *European Journal of Control, 6*(1), 2-20.

Bagis, A. (2007). Determination of the PID controller parameters by modified genetic algorithm for improved performance. *Journal of Information Science and Engineering, 23*(5), 1469-1480.

Bansal, H. O., Sharma, R., and Shreeraman, P. (2012). PID controller tuning techniques: a review. *Journal of Control Engineering and Technology, 2*(4).

Bassi, S., Mishra, M., and Omizegba, E. (2011). Automatic tuning of proportional-integral-derivative (PID) controller using particle swarm optimization (PSO) algorithm. *International Journal of Artificial Intelligence & Applications (IJAIA), 2*(4), 25-34.

Chung, C. (1997). *Knowledge-based approaches to self-adaptation in cultural algorithms.* (Ph.D Ph.D Desertation), Wayne State University, Wayne, U.S.A.

Dorf, R. C., and Bishop, R. H. (2011). *Modern control systems* (Twelfth Edition ed.): Prentice Hall.

Dorigo, M., and Stützle, T. (2010). Ant colony optimization: overview and recent advances *Handbook of metaheuristics* (pp. 227-263): Springer.

El-Telbany, M. E.-S. (2007). Employing particle swarm optimizer and genetic algorithms for optimal tuning of PID controllers: A comparative study. *ICGST-ACSE Journal, 7*(2), 49-54.

El-Telbany, M. E. (2013). Tuning PID controller for DC motor: An artificial bees optimization approach. *International Journal of Computer Applications, 77*(15), 18-21.

Ercin, O., and Coban, R. (2011). *Comparison of the artificial bee colony and the bees algorithm for pid controller tuning.* Paper presented at the 2011 International Symposium on Innovations in Intelligent Systems and Applications.

Gao, F., Qi, Y., Yin, Q., and Xiao, J. (2010). *An novel optimal pid tuning and on-line tuning based on artificial bee colony algorithm.* Paper presented at the Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on.

Gozde. H, M. C. Taplamacioglu and I. Kocaarslan, "Applications of Artificial Bees Colony Algorithm in an Automatic Voltage Regulator (AVR) System", International Journal on Technical and Physical Problems of Engineering (IJTPE), Volume 2 Number 3, pp. 88-92, 2010.

Jones, K. O., and Bouffet, A. (2008). *Comparison of bees algorithm, ant colony optimisation and particle swarm optimisation for PID controller tuning.* Paper presented at the Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing.

Kaliappan, V., and Thathan, M. (2015). Enhanced ABC based PID controller for nonlinear control systems. *Indian Journal of Science and Technology, 8*(S7), 48-56.

Kang, F., Li, J., and Ma, Z. (2011). Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions. *Information Sciences, 181*(16), 3508-3531.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization: Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

Karaboga, D., and Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation, 214*(1), 108-132.

Karaboga, D., and Akay, B. (2011). A modified artificial bee colony (ABC) algorithm for constrained optimization problems. *Applied soft computing, 11*(3), 3021-3031.

Karaboga, D., and Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of global optimization, 39*(3), 459-471.

Karaboga, D., and Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied soft computing, 8*(1), 687-697.

Kesarkar, A. A., and Selvaganesan, N. (2015). Tuning of optimal fractional-order pid controller using an artificial bee colony algorithm. *Systems Science & Control Engineering, 3*(1), 99-105.

Kobti, Z., Reynolds, R. G., and Kohler, T. (2004). Agent-based modeling of cultural change in swarm using cultural algorithms. *Funded by NSF grant BCS-0119981*, 1988-1995.

Kumar, R., Singla, S., and Chopra, V. (2015). Comparison among some well known control schemes with different tuning methods. *Journal of Applied Research and Technology, 13*(3), 409-415.

Li, X., Tang, K., Omidvar, M. N., Yang, Z., Qin, K., and China, H. (2013). Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization. *gene, 7*, 33.

Mishra, A. K., Tiwari, V. K., Kumar, R., and Verma, T. (2013). *Speed control of DC motor using artificial bee colony optimization technique.* Paper presented at the Control, Automation, Robotics and Embedded Systems (CARE), 2013 International Conference on.

Molga, M., and Smutnicki, C. (2005). Test functions for optimization needs. *Test functions for optimization needs*, 1-43.

Ogata, K. (2009). modern control systems, university of Minnesota (5th ed.). New Delhi: prentice hall.

Peng, B., and Reynolds, R. G. (2004). *Cultural algorithms: Knowledge learning in dynamic environments.* Paper presented at the Evolutionary Computation, 2004. CEC2004. Congress on Tools with Artificial Intelligence.

Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M. (2006). *The Bees Algorithm–A Novel Tool for Complex Optimisation.* Paper presented at the Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference 3-14 July 2006.

Rajasekhar, A., Kunathi, P., Abraham, A., and Pant, M. (2011). *Fractinal order speed control of DC motor using levy mutated Artificial Bee Colony Algorithm.* Paper presented at the Information and Communication Technologies (WICT), 2011 World Congress on.

Reynolds, R. G. (1994). *An introduction to cultural algorithms.* Paper presented at the Proceedings of the third annual conference on evolutionary programming.

Reynolds, R. G., and Chung, C. (1997). *Fuzzy approaches to acquiring experimental knowledge in cultural algorithms.* Paper presented at the Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on.

Reynolds, R. G., and Jin, X. (1999). *Using Region-Schema to Solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach.* Paper presented at the Festschrift Conference in Honor of John H. Holland:[sn].

Reynolds, R. G., and Peng, B. (2004). *Cultural algorithms: modeling of how cultures learn to solve problems.* Paper presented at the Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on.

Salawudeen, A. T. (2015). *Developement of an Improved Cultural Artificial Fish Swarm Algorithm with Crossover.* (Master of Science Desertation), Ahmadu Bello University Zaria, Nigeria, Unpublished. (25)

Shapla, S. S., Haque, H., and Alam, M. S. (2015). Explorative Artificial Bee Colony Algorithm: A Novel Swarm Intelligence Based Algorithm for Continuous Function Optimization. *International Journal of Science and Research (IJSR), 4*(7), 1339-1344.

Shayanfar, H., Ghasemi, A., Abedinia, O., Izadfar, H., and Amjady, N. (2012). Optimal PID power system stabilizer tuning via artificial bee colony. *Technical and Physical Problems of Engineering"(IJTPE)*(12), 75-82.

Solihin, M. I., Tack, L. F., and Kean, M. L. (2011). Tuning of PID controller using particle swarm optimization (PSO). *International Journal on Advanced Science, Engineering and Information Technology, 1*(4), 458-461.

Sonmez, Y., Ayyildiz, O., Kahraman, H. T., Guvenc, U., and Duman, S. (2015). Improvement of Buck Converter Performance Using Artificial Bee Colony Optimized-PID Controller. *Journal of Automation and Control Engineering Vol, 3*(4).

TSai, P.-W., Pan, J.-S., Liao, B.-Y., and Chu, S.-C. (2009). Enhanced artificial bee colony optimization. *International Journal of Innovative Computing, Information and Control, 5*(12), 5081-5092.

Varma, P., and Kumar, B. A. (2013). Control of DC motor using artificial bee colony based PID controller. *International Journal Digital Applications Contemporary Research, 2*, 1-9.

Visioli, A. (2006). *Practical PID control*: Springer Science & Business Media.

Yan, G., and Li, C. (2011). An effective refinement artificial bee colony optimization algorithm based on chaotic search and application for pid control tuning. *J Comput Inf Syst, 7*(9), 3309-3316.

Yan, G., and Li, C. (2011). An effective refinement artificial bee colony optimization algorithm based on chaotic search and application for pid control tuning. *Journal of Computational Information Systems, 7*(9), 3309-3316.

APPENDICES

APPENDIX A

M-File Code of Artificial Bee Colony (ABC) Algorithm

```
function ABC
clc
%/* Control Parameters of ABC algorithm*/
SN=input('Please enter the population of Bee: '); %/* The number of colony
size (employed bees+onlooker bees)*/
clc
D=input('Please Enter the Dimension(D) of the problem(Problem to be
optimize): ');%/*The number of parameters of the problem to be optimized*/
clc
FoodNumber=SN/2; %/*The number of food sources equals the half of the colony
size*/
clc
limit=input('Please Enter the trial limit of ABC before it gave up: '); %/*A
food source which could not be improved through "limit" trials is abandoned
by its employed bee*/
clc
maxCycle=input('Please Enter the maximum cycle for foraging (or stopping
criterial): '); %/*The number of cycles for foraging {a stopping criteria}*/
clc
BEE=input('PLEASE ENTER THE INITIAL BEE COLONY MATRIX OR \nPRESS ZERO(0)FOR
MATLAB TO CREATE THEM\nBEE:= ');
clc
fitness=input('Select the test function:\nPress 1 to Select Ackley\nPress 2
to Select CM\nPress 3 to Select DeJongf4\nPress 4 to Select Expfun\nPress 5
to Select Griewank\nPress 6 to Select Hyperelliptic\nPress 7 to Select
LM1\nPress 8 to Select LM2\nPress 9 to Select Neumaier3\nPress 10 to Select
Rastrigin\nPress 11 to Select Rosenbrock\nPress 12 to Select Sal\nPress 13
to Select Schwefel\nPress 14 to Select Schaffer\nPress 15 to Select
Sphere\nPress 16 to select Kursawe\nPress \nPress 20 to select Axis
Parallel\nPress 21 to select Zakharov\nPress 22 to select Step\nPress 23 to
select Quartic \nPress 24 to Select Michalewicz: ');
clc
disp('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
disp('ITERATED BEEs WILL BE RELEASE BECAREFULL NOT TO GET STINGED.!!!')
disp('PRESS ANY KEY IF YOU ARE READY')
pause
ub=ones(1,D)*100; %/*lower bounds of the parameters.*/
lb=ones(1,D)*(-100);%/*upper bound of the parameters.*/
runtime=1;
GlobalMins=zeros(1,runtime);
global popBee
tic
if BEE==0
popBee = rand(FoodNumber,D);%/*All food sources are initialized */
```

```
popBee=sort(popBee);
else
    popBee=BEE;
end
ObjVal=objfunc(fitness,popBee);
Fitness=calculateFitness(ObjVal);
trial=0;
%format long
for r=1:runtime
%reset trial counters
trial=zeros(1,FoodNumber);
%/*The best food source is memorized*/
BestInd=find(ObjVal==min(ObjVal));
BestInd=BestInd(end);
GlobalMin=ObjVal(BestInd);
GlobalParams=popBee(BestInd,:);
iter=0;
while((iter <= maxCycle)),
    iter=iter+1;
%*%*%*%*%*%*%*% PROGRAMMING THE EMPLOYED BEE *%*%*%*%*%*%*%*%*%*%*%*
    for i=1:(FoodNumber)
        P_C=fix(rand*D)+1;
        neighbour=fix(rand*(FoodNumber))+1;%The mutant solution i*/
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
       sol=popBee(i,:);
       %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
       sol(P_C)=popBee(i,P_C)+(popBee(i,P_C)-popBee(neighbour,P_C))*(rand-
0.5)*2;
       %  Conditions to determine if the generated solution is out of
       %  boundary
        ind=find(sol<lb);
        sol(Mishra et. al.)=lb(Mishra et. al.);
        ind=find(sol>ub);
        sol(Mishra et. al.)=ub(Mishra et. al.);
        %evaluate new solution
        ObjValSol=objfunc(fitness,sol);
        FitnessSol=calculateFitness(ObjValSol);
       % /* greedy selection is applied between the current solution i and
its mutant*/
       if (FitnessSol>Fitness(i)) %/*If the mutant solution is better than
the current solution i, replace the solution with the mutant and reset the
trial counter of solution i*/
            popBee(i,:)=sol;
            Fitness(i)=FitnessSol;
            ObjVal(i)=ObjValSol;
            trial(i)=0;
        else
            trial(i)=trial(i)+1; %/*if the solution i cannot be improved,
increase its trial counter*/
```

```
       end;
    end;
%/* A food source is chosen with the probability which is proportional to
its quality*/
prob=(0.9.*Fitness./max(Fitness))+0.1;
%*%*%*%*%*%*%*% PROGRAMMING THE ONLOOKER BEE *%*%*%*%*%*%*%*%*%*%*%*%*
i=1;
t=0;
while(t<FoodNumber)
    if(rand<prob(i))
        t=t+1;
        %/*The parameter to be changed is determined randomly*/
        P_C=fix(rand*D)+1;
        %/*A randomly chosen solution is used in producing a mutant solution
of the solution i*/
        neighbour=fix(rand*(FoodNumber))+1;
        %/*Randomly selected solution must be different from the solution
i*/
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
       sol=popBee(i,:);
       %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
       sol(P_C)=popBee(i,P_C)+(popBee(i,P_C)-popBee(neighbour,P_C))*(rand-
0.5)*2;
       %  /*if generated parameter value is out of boundaries, it is shifted
onto the boundaries*/
        ind=find(sol<lb);
        sol(Mishra et. al.)=lb(Mishra et. al.);
        ind=find(sol>ub);
        sol(Mishra et. al.)=ub(Mishra et. al.);
        %evaluate new solution
        ObjValSol=objfunc(fitness,sol);
        FitnessSol=calculateFitness(ObjValSol)
       % /*a greedy selection is applied between the current solution i and
its mutant*/
       if (FitnessSol>Fitness(i)) %/*If the mutant solution is better than
the current solution i, replace the solution with the mutant and reset the
trial counter of solution i*/
            popBee(i,:)=sol;
            Fitness(i)=FitnessSol;
            ObjVal(i)=ObjValSol;
            trial(i)=0;
        else
            trial(i)=trial(i)+1; %/*if the solution i can not be improved,
increase its trial counter*/
       end;
    end;
    i=i+1;
    if (i==(FoodNumber)+1)
        i=1;
```

```
    end;
end;
%/*The best food source is memorized*/
         ind=find(ObjVal==min(ObjVal));
         ind=ind(end);
         if (ObjVal(ind)<GlobalMin)
         GlobalMin=ObjVal(Mishra et. al.);
         GlobalParams=popBee(ind,:);
         end;
%*%*%*%*%*%*%*%*%*%*% EMPLOYED BEE BECOMES THE SCOUT BEE *%*%*%*%*%*%*%*%*%*%*
%/*Determine the food sources whose trial counter exceeds the "limit" value.
ind=find(trial==max(trial));
ind=ind(end);
if (trial(ind)>limit)
    trial(Mishra et. al.)=0;
    sol=(ub-lb).*rand(1,D)+lb;
    ObjValSol=objfunc(fitness,sol);
    FitnessSol=calculateFitness(ObjValSol);
    popBee(ind,:)=sol;
    Fitness(Mishra et. al.)=FitnessSol;
    ObjVal(Mishra et. al.)=ObjValSol;
end;
fprintf('Iter = %d ObjVal = %g\n',iter,GlobalMin);
end % End of ABC
GlobalMins(r)=GlobalMin;
end; %end of runs
% x=0:0.5:20;
% % plot(x,GlobalMin)
% grid on
disp('@Author: ')
disp('BUSAYO Adebiyi(MSc/ENG/32931/2012-2013')
disp('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
disp('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
disp('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
disp('*******************************************************************')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%')
disp('BEST SOLUTION FOR THIS ITERATION')
disp(GlobalMins(r))
toc
```

APPENDIX B1

M-File Code of CABCA(Ns)

```
%Start of Cultural Artificial Bee Colony Algorithm CABCA
population=[];
for i=1:SN
    population=[population;GlobalParams];
end
popBee_Ns=sort(population);
Beta=input('ENTER THE VALUE FOR B (% of the accepted Bee)\n or press zero
(0) for default value.\nBeta =  ');
if Beta==0
Beta=0.2;
end
ObjVal_Ns=objfunc_Ns(fitness,popBee_Ns);
Fitness_Ns=calculateFitness(ObjVal_Ns);
trial=0;
for r=1:runtime%Start of Run
%reset trial counters
trial=zeros(1,FoodNumber);
%/*The best food source is memorized*/
BestInd_Ns=find(ObjVal_Ns==min(ObjVal_Ns));
BestInd_Ns=BestInd_Ns(end);
GlobalMin_Ns=ObjVal_Ns(BestInd_Ns);
GlobalParams_Ns=popBee_Ns(BestInd_Ns,:);
iter=maxCycle;
    aceptance=N*Beta+floor(N*Beta/iter);
    popnew_Ns=popBee_Ns;
    fitpopnew_Ns=objfunc_Ns(fitness,popnew_Ns);
    fitp_Ns=[fitpopnew_Ns,popnew_Ns];
    fitp_Ns=sortc(fitp_Ns,1);
    L=fitp_Ns(1);
    U=fitp_Ns(N,1);
    ub=fitp_Ns(N,(2:(D+1)));
    lb=fitp_Ns(1,(2:(D+1)));
    try_num=1;
    for Try=1:try_num
        popnewii=popnew_Ns;
        hh=[];
        popnewi=popnew_Ns;
        for vv=1:N
            hh=[hh;(ub-lb)];
        end
%           popnewii=popnewii+rand*hh;
          for i=1:N
              if
objfunc_Ns(fitness,popnewii(i,:))>objfunc_Ns(fitness,popnewi(i,:))
                  popnewpkk(i,:)=popnewi(i,:);
              end
    fitpopnew_Ns=objfunc_Ns(fitness,popnew_Ns);
```

```
    fitp_Ns=[fitpopnew_Ns,popnew_Ns];
    fitp_Ns=sortc(fitp_Ns,1);
    L=fitp_Ns(1);
    U=fitp_Ns(N,1);
    ub=fitp_Ns(N,(2:(D+1)));
    lb=fitp_Ns(1,(2:(D+1)));
    end
    popnew_Ns=popBee_Ns;
%     popnewpj=popnew;
%     popnewsj=popnew;
%       popnewcj=popnew;
%      popnewpi=popnew;
%       popnewsi=popnew;
%        popnewci=popnew;
    fitpopnew_Ns=objfunc_Ns(fitness,popnew_Ns);
    I=[];
for i=1:N
    if objfunc_Ns(fitness,popnew_Ns)>=L
        if objfunc_Ns(fitness,popnew_Ns)<=U
            I=[I;popnew_Ns(i,:)];
        end
    end
end
fitI=objfunc_Ns(fitness,I);
FITI=[fitI,I];
FITI=sortc(FITI,1);
lk=FITI(1,(1:(D+1)));
uk=FITI(end,(1:(D+1)));
iter=0;
while((iter <= maxCycle))%Start of ABC
    iter=iter+1;
%*%*%*%*%*%*%*% PROGRAMMING THE EMPLOYED BEE %*%*%*%*%*%*%*%*%*%*%*%*%*%*%*
    for i=1:(FoodNumber)
        P_C_Ns=fix(rand*D)+1;
        neighbour=fix(rand*(FoodNumber))+1;%The mutant solution i*/
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
       sol_Ns=popBee_Ns(i,:);
       clear i j
       for i=1:aceptance
           for j=1:P_C_Ns
       % /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
       sol_Ns(P_C_Ns)=popBee_Ns(i,P_C_Ns)+abs(uk(j)-
lk(j))*(popBee_Ns(i,P_C_Ns)-popBee_Ns(neighbour,P_C_Ns))*(rand-0.5)*2;
           end
       end
       ind_Ns=find(sol_Ns<lbi);
        sol_Ns(ind_Ns)=lbi(ind_Ns);
        ind_Ns=find(sol_Ns>ubi);
        sol_Ns(ind_Ns)=ubi(ind_Ns);
```

```
        %evaluate new solution
        ObjValSol_Ns=objfunc_Ns(fitness,sol_Ns);
        FitnessSol_Ns=calculateFitness(ObjValSol_Ns);
       % /* greedy selection is applied between the current solution i and
its mutant*/
       if (FitnessSol_Ns>Fitness_Ns(i)) %/*If the mutant solution is better
than the current solution i, replace the solution with the mutant and reset
the trial counter of solution i*/
            popBee_Ns(i,:)=sol_Ns;
            Fitness_Ns(i)=FitnessSol_Ns;
            ObjVal_Ns(i)=ObjValSol_Ns;
            trial(i)=0;
        else
            trial(i)=trial(i)+1; %/*if the solution i cannot be improved,
increase its trial counter*/
       end;
    end%End of Employee
    %/* A food source is chosen with the probability which is proportional
to its quality*/
prob_Ns=(0.9.*Fitness_Ns./max(Fitness_Ns))+0.1;
%*%*%*%*%*%*%*% PROGRAMMING THE ONLOOKER BEE %*%*%*%*%*%*%*%*%*%*%*%*%*%*%*
i=1;
t=0;
while(t<FoodNumber)
    if(rand<prob_Ns(i))
        t=t+1;
        %/*The parameter to be changed is determined randomly*/
        P_C_Ns=fix(rand*D)+1;
 %/*A randomly chosen solution is used in producing a mutant solution of
the solution i*/
        neighbour=fix(rand*(FoodNumber))+1;
      %/*Randomly selected solution must be different from the solution i*/
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
       sol_Ns=popBee_Ns(i,:);
       %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
       clear i j
       for i=1:aceptance
           for j=1:D
       sol_Ns(P_C_Ns)=popBee_Ns(i,P_C_Ns)+abs(uk(j)-
lk(j))*(popBee_Ns(i,P_C_Ns)-popBee_Ns(neighbour,P_C_Ns))*(rand-0.5)*2;
           end
       end
       %  /*if generated parameter value is out of boundaries, it is
shifted onto the boundaries*/
        ind_Ns=find(sol_Ns<lbi);
        sol_Ns(ind_Ns)=lbi(ind_Ns);
        ind_Ns=find(sol_Ns>ubi);
        sol_Ns(ind_Ns)=ubi(ind_Ns);
        %evaluate new solution
```

```
        ObjValSol_Ns=objfunc_Ns(fitness,sol_Ns);
        FitnessSol_Ns=calculateFitness(ObjValSol_Ns);
       % /*a greedy selection is applied between the current solution i and
its mutant*/
       if (FitnessSol_Ns>Fitness_Ns(i)) %/*If the mutant solution is better
than the current solution i, replace the solution with the mutant and reset
the trial counter of solution i*/
            popBee_Ns(i,:)=sol_Ns;
            Fitness_Ns(i)=FitnessSol_Ns;
            ObjVal_Ns(i)=ObjValSol_Ns;
            trial(i)=0;
        else
            trial(i)=trial(i)+1; %/*if the solution i cannot be improved,
increase its trial counter*/
       end;
    end;
    i=i+1;
    if (i==(FoodNumber)+1)
        i=1;
    end;
end;%End the ONLOCKER
%/*The best food source is memorized*/
         ind_Ns=find(ObjVal_Ns==min(ObjVal_Ns));
         ind_Ns=ind_Ns(end);
         if (ObjVal_Ns(ind_Ns)<GlobalMin_Ns)
         GlobalMin_Ns=ObjVal_Ns(ind_Ns);
         GlobalParams_Ns=popBee_Ns(ind_Ns,:);
         end;
%*%*%*%*%*%*%*% ONLOOKER BEE BECOMES THE SCOUT *%*%*%*%*%*%*%*%*%*%*%*%*
%/*Determine the food sources whose trial counter exceeds the "limit"
value.
ind_Ns=find(trial==max(trial));
ind_Ns=ind_Ns(end);
if (trial(ind_Ns)>limit)
    trial(ind_Ns)=0;
    sol_Ns=(ubi-lbi).*rand(1,D)+lbi;
    ObjValSol_Ns=objfunc_Ns(fitness,sol_Ns);
    FitnessSol_Ns=calculateFitness(ObjValSol_Ns);
    popBee_Ns(ind_Ns,:)=sol_Ns;
    Fitness_Ns(ind_Ns)=FitnessSol_Ns;
    ObjVal_Ns(ind_Ns)=ObjValSol_Ns;
end;
fprintf('CABCA_Ns = %d ObjVal_Ns = %g\n',iter,GlobalMin_Ns);
end%End of ABC
GlobalMins_Ns(r)=GlobalMin_Ns;
end%end of Run
clc
help 'CABCA
_Ns.m'
disp('The Optimal Value Based on CABCA_Ns')
Best_Ns=min(GlobalMins_Ns(r))
```

APPENDIX B2

M-File Code of CABCA(Sd)

```
% The Situational Based Cultural Artificial Bee Colony Algorithm (CABCA)
begins here.
population=[];
for i=1:SN
    population=[population;GlobalParams];
end
popBee_Sd=sort(population);
Beta=input('ENTER THE VALUE FOR B (% of the accepted Bee)\n or press zero
(0) for default value.\nBeta =  ');
if Beta==0
Beta=0.2;
end
ObjVal_Sd=objfunc(fitness,popBee_Sd);
Fitness_Sd=calculateFitness(ObjVal_Sd);
trial=0;
for r=1:runtime%Start of Run
%reset trial counters
trial=zeros(1,FoodNumber);
%/*The best food source is memorized*/
BestInd_Sd=find(ObjVal_Sd==min(ObjVal_Sd));
BestInd_Sd=BestInd_Sd(end);
GlobalMin_Sd=ObjVal_Sd(BestInd_Sd);
GlobalParams_Sd=popBee_Sd(BestInd_Sd,:);
iter=maxCycle;
popnew_Sd=popBee_Sd;
    fitpopnew_Sd=objfunc(fitness,popnew_Sd);
    fitp_Sd=[fitpopnew_Sd,popnew_Sd];
    fitp_Sd=sortc(fitp_Sd,1);
    S=fitp_Sd(1,(2:(D+1)));
Itr=0;
while((Itr <= maxCycle))%Start of CABCA_Sd
    Itr=Itr+1;
    aceptance=N*Beta+floor(N*Beta/iter);
%*%*%*%*%*%*%*%*% PROGRAMMING THE EMPLOYED BEE %*%*%*%*%*%*%*%*%*%*%*%*%*%*%*
    for i=1:(FoodNumber)
        P_C_Sd=fix(rand*D)+1;
        neighbour=fix(rand*(FoodNumber))+1;%The mutant solution i*/
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
       sol_Sd=popBee_Sd(i,:);
       clear i j
       for i=1:aceptance
           for j=1:P_C_Sd
               if popnew_Sd(i,j)<S(j)
       %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
```

```
       sol_Sd(P_C_Sd)=popBee_Sd(i,P_C_Sd)+abs((popBee_Sd(i,P_C_Sd)-
popBee_Sd(neighbour,P_C_Sd))*(rand-0.5))*2;
               elseif popnew_Sd(i,j)>S(j)
       sol_Sd(P_C_Sd)=popBee_Sd(i,P_C_Sd)-abs((popBee_Sd(i,P_C_Sd)-
popBee_Sd(neighbour,P_C_Sd))*(rand-0.5)*2);
               else
       sol_Sd(P_C_Sd)=popBee_Sd(i,P_C_Sd)+(popBee_Sd(i,P_C_Sd)-
popBee_Sd(neighbour,P_C_Sd))*(rand-0.5)*2;
               end
           end
       end
       ind_Sd=find(sol_Sd<lbi);
        sol_Sd(ind_Sd)=lbi(ind_Sd);
        ind_Sd=find(sol_Sd>ubi);
        sol_Sd(ind_Sd)=ubi(ind_Sd);
        %evaluate new solution
        ObjValSol_Sd=objfunc(fitness,sol_Sd);
        FitnessSol_Sd=calculateFitness(ObjValSol_Sd);
       % /* greedy selection is applied between the current solution i and
its mutant*/
       if (FitnessSol_Sd>Fitness_Sd(i)) %/*If the mutant solution is better
than the current solution i, replace the solution with the mutant and reset
the trial counter of solution i*/
            popBee_Sd(i,:)=sol_Sd;
            Fitness_Sd(i)=FitnessSol_Sd;
            ObjVal_Sd(i)=ObjValSol_Sd;
            trial(i)=0;
        else
            trial(i)=trial(i)+1; %/*if the solution i can not be improved,
increase its trial counter*/
       end;
    end%End of Employee
    %/* A food source is chosen with the probability which is proportional
to its quality*/
prob_Sd=(0.9.*Fitness_Sd./max(Fitness_Sd))+0.1;
%*%*%*%*%*%*%*% PROGRAMMING THE ONLOOKER BEE %*%*%*%*%*%*%*%*%*%*%*%*%*%*%*
i=1;
t=0;
while(t<FoodNumber)
    if(rand<prob_Sd(i))
        t=t+1;
        %/*The parameter to be changed is determined randomly*/
        P_C_Sd=fix(rand*D)+1;
        %/*A randomly chosen solution is used in producing a mutant solution
of the solution i*/
        neighbour=fix(rand*(FoodNumber))+1;

      %/*Randomly selected solution must be different from the solution i*/
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
```

```
            end;
        sol_Sd=popBee_Sd(i,:);
        %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
        clear i j
        for i=1:aceptance
            for j=1:D
                if popnew_Sd(i,j)<S(j)
        %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
        sol_Sd(P_C_Sd)=popBee_Sd(i,P_C_Sd)+abs((popBee_Sd(i,P_C_Sd)-
popBee_Sd(neighbour,P_C_Sd))*(rand-0.5))*2;
                elseif popnew_Sd(i,j)>S(j)
        sol_Sd(P_C_Sd)=popBee_Sd(i,P_C_Sd)-abs((popBee_Sd(i,P_C_Sd)-
popBee_Sd(neighbour,P_C_Sd))*(rand-0.5)*2);
                else
        sol_Sd(P_C_Sd)=popBee_Sd(i,P_C_Sd)+(popBee_Sd(i,P_C_Sd)-
popBee_Sd(neighbour,P_C_Sd))*(rand-0.5)*2;
                end
            end
        end
        %  /*if generated parameter value is out of boundaries, it is shifted
onto the boundaries*/
         ind_Sd=find(sol_Sd<lbi);
         sol_Sd(ind_Sd)=lbi(ind_Sd);
         ind_Sd=find(sol_Sd>ubi);
         sol_Sd(ind_Sd)=ubi(ind_Sd);
         %evaluate new solution
         ObjValSol_Sd=objfunc(fitness,sol_Sd);
         FitnessSol_Sd=calculateFitness(ObjValSol_Sd);
 % /*a greedy selection is applied between the current solution i and its
mutant*/
        if (FitnessSol_Sd>Fitness_Sd(i)) %/*If the mutant solution is better
than the current solution i, replace the solution with the mutant and reset
the trial counter of solution i*/
             popBee_Sd(i,:)=sol_Sd;
             Fitness_Sd(i)=FitnessSol_Sd;
             ObjVal_Sd(i)=ObjValSol_Sd;
             trial(i)=0;
         else
 trial(i)=trial(i)+1; %/*if the solution i cannot be improved, increase its
trial counter*/
        end;
    end;
    i=i+1;
    if (i==(FoodNumber)+1)
        i=1;
    end;
end;%End the ONLOCKER
%/*The best food source is memorized*/
          ind_Sd=find(ObjVal_Sd==min(ObjVal_Sd));
          ind_Sd=ind_Sd(end);
          if (ObjVal_Sd(ind_Sd)<GlobalMin_Sd)
```

```
          GlobalMin_Sd=ObjVal_Sd(ind_Sd);
          GlobalParams_Sd=popBee_Sd(ind_Sd,:);
          end;
%*%*%*%*%*%*%*%*% EMPLOYED BEE BECOMES THE SCOUT BEE %*%*%*%*%*%*%*%*%*%*
%/*Determine the food sources whose trial counter exceeds the "limit" value.
ind_Sd=find(trial==max(trial));
ind_Sd=ind_Sd(end);
if (trial(ind_Sd)>limit)
    trial(ind_Sd)=0;
    sol_Sd=(ubi-lbi).*rand(1,D)+lbi;
    ObjValSol_Sd=objfunc(fitness,sol_Sd);
    FitnessSol_Sd=calculateFitness(ObjValSol_Sd);
    popBee_Sd(ind_Sd,:)=sol_Sd;
    Fitness_Sd(ind_Sd)=FitnessSol_Sd;
    ObjVal_Sd(ind_Sd)=ObjValSol_Sd;
end;
fprintf('CABCA_Sd = %d ObjVal_Sd = %g\n',iter,GlobalMin_Sd);
end%End of ABC
GlobalMins_Sd(r)=GlobalMin_Sd;
end%end of Run
help 'CABCA_Sd.m'
disp('The Optimal Value Based on CABCA_Sd')
Best_Sd=min(GlobalMins_Sd(r))
```

APPENDIX B3

M-File Code of CABCA(Ns+Sd)

```
% Start of CABCA_NsSd
population=[];
for i=1:SN
    population=[population;GlobalParams];
end
popBee_NsSd=sort(population);
Beta=input('ENTER THE VALUE FOR B (% of the accepted Bee)\n or press zero
(0) for default value.\nBeta =  ');
if Beta==0
Beta=0.2;
end
ObjVal_NsSd=objfunc_NsSd(fitness,popBee_NsSd);
Fitness_NsSd=calculateFitness(ObjVal_NsSd);
trial=0;
for r=1:runtime%Start of Run
%reset trial counters
trial=zeros(1,FoodNumber);
%/*The best food source is memorized*/
BestInd_NsSd=find(ObjVal_NsSd==min(ObjVal_NsSd));
BestInd_NsSd=BestInd_NsSd(end);
GlobalMin_NsSd=ObjVal_NsSd(BestInd_NsSd);
GlobalParams_NsSd=popBee_NsSd(BestInd_NsSd,:);
iter=maxCycle;
    aceptance=N*Beta+floor(N*Beta/iter);
    popnew_NsSd=popBee_NsSd;
    fitpopnew_NsSd=objfunc_NsSd(fitness,popnew_NsSd);
    fitp_NsSd=[fitpopnew_NsSd,popnew_NsSd];
    fitp_NsSd=sortc(fitp_NsSd,1);
    L=fitp_NsSd(1);
    U=fitp_NsSd(N,1);
    ub=fitp_NsSd(N,(2:(D+1)));
    lb=fitp_NsSd(1,(2:(D+1)));
    try_num=1;
    for Try=1:try_num
        popnewii=popnew_NsSd;
        hh=[];
        popnewi=popnew_NsSd;
        for vv=1:N
            hh=[hh;(ub-lb)];
        end
%           popnewii=popnewii+rand*hh;
          for i=1:N
              if
objfunc_NsSd(fitness,popnewii(i,:))>objfunc_NsSd(fitness,popnewi(i,:))
                  popnewpkk(i,:)=popnewi(i,:);
              end
          end
```

```
    fitpopnew_NsSd=objfunc_NsSd(fitness,popnew_NsSd);
    fitp_NsSd=[fitpopnew_NsSd,popnew_NsSd];
    fitp_NsSd=sortc(fitp_NsSd,1);
    L=fitp_NsSd(1);
    U=fitp_NsSd(N,1);
    ub=fitp_NsSd(N,(2:(D+1)));
    lb=fitp_NsSd(1,(2:(D+1)));
    end
    popnew_NsSd=popBee_NsSd;
    popnew_NsSd=popBee_NsSd;
    fitpopnew_NsSd=objfunc_NsSd(fitness,popnew_NsSd);
    fitp_NsSd=[fitpopnew_NsSd,popnew_NsSd];
    fitp_NsSd=sortc(fitp_NsSd,1);
    S=fitp_NsSd(1,(2:(D+1)));
%     popnewpj=popnew;
%     popnewsj=popnew;
%       popnewcj=popnew;
%      popnewpi=popnew;
%       popnewsi=popnew;
%        popnewci=popnew;
    fitpopnew_NsSd=objfunc_NsSd(fitness,popnew_NsSd);
    I=[];
for i=1:N
    if objfunc_NsSd(fitness,popnew_NsSd)>=L
        if objfunc_NsSd(fitness,popnew_NsSd)<=U
            I=[I;popnew_NsSd(i,:)];
        end
    end
end
fitI=objfunc_NsSd(fitness,I);
FITI=[fitI,I];
FITI=sortc(FITI,1);
lk=FITI(1,(1:(D+1)));
uk=FITI(end,(1:(D+1)));
iter=0;
while((iter <= maxCycle))%Start of ABC
    iter=iter+1;
%*%*%*%*%*%*%*% PROGRAMMING THE EMPLOYED BEE %*%*%*%*%*%*%*%*%*%*%*%*%*%*%*
    for i=1:(FoodNumber)
        P_C_NsSd=fix(rand*D)+1;
        neighbour=fix(rand*(FoodNumber))+1;%The mutant solution i*/
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
       sol_NsSd=popBee_NsSd(i,:);
       clear i j
       for i=1:aceptance
           for j=1:P_C_NsSd
       %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) *
                if popnew_NsSd(i,j)<S(j)
```

```
                    sol_NsSd(P_C_NsSd)=popBee_NsSd(i,P_C_NsSd)+abs(uk(j)-
lk(j)*(popBee_NsSd(i,P_C_NsSd)-popBee_NsSd(neighbour,P_C_NsSd))*(rand-
0.5)*2);
                elseif popnew_NsSd(i,j)>S(j)
                    sol_NsSd(P_C_NsSd)=popBee_NsSd(i,P_C_NsSd)-abs(uk(j)-
lk(j)*(popBee_NsSd(i,P_C_NsSd)-popBee_NsSd(neighbour,P_C_NsSd))*(rand-
0.5)*2);
                else
                    sol_NsSd(P_C_NsSd)=popBee_NsSd(i,P_C_NsSd)+(uk(j)-
lk(j))*(popBee_NsSd(i,P_C_NsSd)-popBee_NsSd(neighbour,P_C_NsSd))*(rand-
0.5)*2;
                end
            end
        end
        ind_NsSd=find(sol_NsSd<lbi);
         sol_NsSd(ind_NsSd)=lbi(ind_NsSd);
         ind_NsSd=find(sol_NsSd>ubi);
         sol_NsSd(ind_NsSd)=ubi(ind_NsSd);
         %evaluate new solution
         ObjValSol_NsSd=objfunc_NsSd(fitness,sol_NsSd);
         FitnessSol_NsSd=calculateFitness(ObjValSol_NsSd);
        % /* greedy selection is applied between the current solution i and
its mutant*/
        if (FitnessSol_NsSd>Fitness_NsSd(i)) %/*If the mutant solution is
better than the current solution i, replace the solution with the mutant
and reset the trial counter of solution i*/
            popBee_NsSd(i,:)=sol_NsSd;
            Fitness_NsSd(i)=FitnessSol_NsSd;
            ObjVal_NsSd(i)=ObjValSol_NsSd;
            trial(i)=0;
         else
            trial(i)=trial(i)+1; %/*if the solution i can not be improved,
increase its trial counter*/
        end;
    end%End of Employee
    %/* A food source is chosen with the probability which is proportional
to its quality*/
prob_NsSd=(0.9.*Fitness_NsSd./max(Fitness_NsSd))+0.1;
%*%*%*%*%*%*%*% PROGRAMMING THE ONLOOKER BEE %*%*%*%*%*%*%*%*%*%*%*%*%*%*%*
i=1;
t=0;
while(t<FoodNumber)
    if(rand<prob_NsSd(i))
        t=t+1;
        %/*The parameter to be changed is determined randomly*/
        P_C_NsSd=fix(rand*D)+1;
        %/*A randomly chosen solution is used in producing a mutant
solution of the solution i*/
        neighbour=fix(rand*(FoodNumber))+1;
        %/*Randomly selected solution must be different from the solution
i*/
```

```
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
        sol_NsSd=popBee_NsSd(i,:);
        %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
        clear i j
        for i=1:aceptance
            for j=1:P_C_NsSd
        %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) *
                if popnew_NsSd(i,j)<S(j)
                    sol_NsSd(P_C_NsSd)=popBee_NsSd(i,P_C_NsSd)+abs(uk(j)-
lk(j)*(popBee_NsSd(i,P_C_NsSd)-popBee_NsSd(neighbour,P_C_NsSd))*(rand-
0.5)*2);
                elseif popnew_NsSd(i,j)>S(j)
                    sol_NsSd(P_C_NsSd)=popBee_NsSd(i,P_C_NsSd)-abs(uk(j)-
lk(j)*(popBee_NsSd(i,P_C_NsSd)-popBee_NsSd(neighbour,P_C_NsSd))*(rand-
0.5)*2);
                else
                    sol_NsSd(P_C_NsSd)=popBee_NsSd(i,P_C_NsSd)+(uk(j)-
lk(j))*(popBee_NsSd(i,P_C_NsSd)-popBee_NsSd(neighbour,P_C_NsSd))*(rand-
0.5)*2;
                end
            end
        end
        %  /*if generated parameter value is out of boundaries, it is
shifted onto the boundaries*/
         ind_NsSd=find(sol_NsSd<lbi);
         sol_NsSd(ind_NsSd)=lbi(ind_NsSd);
         ind_NsSd=find(sol_NsSd>ubi);
         sol_NsSd(ind_NsSd)=ubi(ind_NsSd);
         %evaluate new solution
         ObjValSol_NsSd=objfunc_NsSd(fitness,sol_NsSd);
         FitnessSol_NsSd=calculateFitness(ObjValSol_NsSd);
        % /*a greedy selection is applied between the current solution i and
its mutant*/
        if (FitnessSol_NsSd>Fitness_NsSd(i)) %/*If the mutant solution is
better than the current solution i, replace the solution with the mutant
and reset the trial counter of solution i*/
            popBee_NsSd(i,:)=sol_NsSd;
            Fitness_NsSd(i)=FitnessSol_NsSd;
            ObjVal_NsSd(i)=ObjValSol_NsSd;
            trial(i)=0;
         else
            trial(i)=trial(i)+1; %/*if the solution i can not be improved,
increase its trial counter*/
        end;
    end;
    i=i+1;
    if (i==(FoodNumber)+1)
        i=1;
    end;
```

```
end;%End the ONLOCKER
%/*The best food source is memorized*/
          ind_NsSd=find(ObjVal_NsSd==min(ObjVal_NsSd));
          ind_NsSd=ind_NsSd(end);
          if (ObjVal_NsSd(ind_NsSd)<GlobalMin_NsSd)
          GlobalMin_NsSd=ObjVal_NsSd(ind_NsSd);
          GlobalParams_NsSd=popBee_NsSd(ind_NsSd,:);
          end;
%*%*%*%*%*%*%*%*% EMPLOYED BEE BECOMES THE SCOUT BEE %*%*%*%*%*%*%*%*%*%*%*%
%/*Determine the food sources whose trial counter exceeds the "limit"
value.
ind_NsSd=find(trial==max(trial));
ind_NsSd=ind_NsSd(end);
if (trial(ind_NsSd)>limit)
    trial(ind_NsSd)=0;
    sol_NsSd=(ubi-lbi).*rand(1,D)+lbi;
    ObjValSol_NsSd=objfunc_NsSd(fitness,sol_NsSd);
    FitnessSol_NsSd=calculateFitness(ObjValSol_NsSd);
    popBee_NsSd(ind_NsSd,:)=sol_NsSd;
    Fitness_NsSd(ind_NsSd)=FitnessSol_NsSd;
    ObjVal_NsSd(ind_NsSd)=ObjValSol_NsSd;
end;
fprintf('CABCA_NsSd = %d ObjVal_NsSd = %g\n',iter,GlobalMin_NsSd);
end%End of ABC
GlobalMins_NsSd(r)=GlobalMin_NsSd;
end%end of Run
clc
help 'CABCA_NsSd.m'
disp('The Optimal Value Based on CABCA_NsSd')
Best_NsSd=min(GlobalMins_NsSd(r))
```

APPENDIX B4

M-File Code of CABCA(Ns+Sd)

```
%Start of CABCA_NsSd
population=[];
for i=1:SN
    population=[population;GlobalParams];
end
popBee_NsNd=sort(population);
Beta=input('ENTER THE VALUE FOR B (% of the accepted Bee)\n or press zero
(0) for default value.\nBeta =  ');
if Beta==0
Beta=0.2;
end
ObjVal_NsNd=objfunc_NsNd(fitness,popBee_NsNd);
Fitness_NsNd=calculateFitness(ObjVal_NsNd);
trial=0;
for r=1:runtime%Start of Run
%reset trial counters
trial=zeros(1,FoodNumber);
%/*The best food source is memorized*/
BestInd_NsNd=find(ObjVal_NsNd==min(ObjVal_NsNd));
BestInd_NsNd=BestInd_NsNd(end);
GlobalMin_NsNd=ObjVal_NsNd(BestInd_NsNd);
GlobalParams_NsNd=popBee_NsNd(BestInd_NsNd,:);
iter=maxCycle;
    aceptance=N*Beta+floor(N*Beta/iter);
    popnew_NsNd=popBee_NsNd;
    fitpopnew_NsNd=objfunc_NsNd(fitness,popnew_NsNd);
    fitp_NsNd=[fitpopnew_NsNd,popnew_NsNd];
    fitp_NsNd=sortc(fitp_NsNd,1);
    L=fitp_NsNd(1);
    U=fitp_NsNd(N,1);
    ub=fitp_NsNd(N,(2:(D+1)));
    lb=fitp_NsNd(1,(2:(D+1)));
    try_num=1;
    for Try=1:try_num
        popnewii=popnew_NsNd;
        hh=[];
        popnewi=popnew_NsNd;
        for vv=1:N
            hh=[hh;(ub-lb)];
        end
%           popnewii=popnewii+rand*hh;
          for i=1:N
              if
objfunc_NsNd(fitness,popnewii(i,:))>objfunc_NsNd(fitness,popnewi(i,:))
                  popnewpkk(i,:)=popnewi(i,:);
              end
          end
```

```
    fitpopnew_NsNd=objfunc_NsNd(fitness,popnew_NsNd);
    fitp_NsNd=[fitpopnew_NsNd,popnew_NsNd];
    fitp_NsNd=sortc(fitp_NsNd,1);
    L=fitp_NsNd(1);
    U=fitp_NsNd(N,1);
    ub=fitp_NsNd(N,(2:(D+1)));
    lb=fitp_NsNd(1,(2:(D+1)));
    end
    popnew_NsNd=popBee_NsNd;
    popnew_NsNd=popBee_NsNd;
    fitpopnew_NsNd=objfunc_NsNd(fitness,popnew_NsNd);
    fitp_NsNd=[fitpopnew_NsNd,popnew_NsNd];
    fitp_NsNd=sortc(fitp_NsNd,1);
    S=fitp_NsNd(1,(2:(D+1)));
%     popnewpj=popnew;
%     popnewsj=popnew;
%       popnewcj=popnew;
%      popnewpi=popnew;
%       popnewsi=popnew;
%        popnewci=popnew;
    fitpopnew_NsNd=objfunc_NsNd(fitness,popnew_NsNd);
    I=[];
for i=1:N
    if objfunc_NsNd(fitness,popnew_NsNd)>=L
        if objfunc_NsNd(fitness,popnew_NsNd)<=U
            I=[I;popnew_NsNd(i,:)];
        end
    end
end
fitI=objfunc_NsNd(fitness,I);
FITI=[fitI,I];
FITI=sortc(FITI,1);
lk=FITI(1,(1:(D+1)));
uk=FITI(end,(1:(D+1)));
iter=0;
while((iter <= maxCycle))%Start of ABC
    iter=iter+1;
%*%*%*%*%*%*%*% PROGRAMMING THE EMPLOYED BEE %*%*%*%*%*%*%*%*%*%*%*%*%*%*%*
    for i=1:(FoodNumber)
        P_C_NsNd=fix(rand*D)+1;
        neighbour=fix(rand*(FoodNumber))+1;%The mutant solution i*/
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
       sol_NsNd=popBee_NsNd(i,:);
       clear i j
       for i=1:aceptance
           for j=1:P_C_NsNd
       %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) *
                if popnew_NsNd(i,j)<S(j)
```

```
                    sol_NsNd(P_C_NsNd)=popBee_NsNd(i,P_C_NsNd)+abs(uk(j)-
lk(j)*(popBee_NsNd(i,P_C_NsNd)-popBee_NsNd(neighbour,P_C_NsNd))*(rand-
0.5)*2);
                elseif popnew_NsNd(i,j)>S(j)
                    sol_NsNd(P_C_NsNd)=popBee_NsNd(i,P_C_NsNd)-abs(uk(j)-
lk(j)*(popBee_NsNd(i,P_C_NsNd)-popBee_NsNd(neighbour,P_C_NsNd))*(rand-
0.5)*2);
                else
                    sol_NsNd(P_C_NsNd)=popBee_NsNd(i,P_C_NsNd)+(uk(j)-
lk(j))*(popBee_NsNd(i,P_C_NsNd)-popBee_NsNd(neighbour,P_C_NsNd))*(rand-
0.5)*2;
                end
            end
        end
        ind_NsNd=find(sol_NsNd<lbi);
         sol_NsNd(ind_NsNd)=lbi(ind_NsNd);
         ind_NsNd=find(sol_NsNd>ubi);
         sol_NsNd(ind_NsNd)=ubi(ind_NsNd);
         %evaluate new solution
         ObjValSol_NsNd=objfunc_NsNd(fitness,sol_NsNd);
         FitnessSol_NsNd=calculateFitness(ObjValSol_NsNd);
        % /* greedy selection is applied between the current solution i and
its mutant*/
        if (FitnessSol_NsNd>Fitness_NsNd(i)) %/*If the mutant solution is
better than the current solution i, replace the solution with the mutant
and reset the trial counter of solution i*/
            popBee_NsNd(i,:)=sol_NsNd;
            Fitness_NsNd(i)=FitnessSol_NsNd;
            ObjVal_NsNd(i)=ObjValSol_NsNd;
            trial(i)=0;
         else
            trial(i)=trial(i)+1; %/*if the solution i can not be improved,
increase its trial counter*/
        end;
    end%End of Employee
    %/* A food source is chosen with the probability which is proportional
to its quality*/
prob_NsNd=(0.9.*Fitness_NsNd./max(Fitness_NsNd))+0.1;
%*%*%*%*%*%*%*% PROGRAMMING THE ONLOOKER BEE %*%*%*%*%*%*%*%*%*%*%*%*%*%*%*
i=1;
t=0;
while(t<FoodNumber)
    if(rand<prob_NsNd(i))
        t=t+1;
        %/*The parameter to be changed is determined randomly*/
        P_C_NsNd=fix(rand*D)+1;
        %/*A randomly chosen solution is used in producing a mutant
solution of the solution i*/
        neighbour=fix(rand*(FoodNumber))+1;
        %/*Randomly selected solution must be different from the solution
i*/
```

```
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;
        sol_NsNd=popBee_NsNd(i,:);
        %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) */
        clear i j
        for i=1:aceptance
            for j=1:P_C_NsNd
        %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_(K. Åström and Hägglund)-x_{ij}) *
                if popnew_NsNd(i,j)<lb(j)
                    sol_NsNd(P_C_NsNd)=popBee_NsNd(i,P_C_NsNd)+abs(uk(j)-
lk(j)*(popBee_NsNd(i,P_C_NsNd)-popBee_NsNd(neighbour,P_C_NsNd))*(rand-
0.5)*2);
                elseif popnew_NsNd(i,j)>ub(j)
                    sol_NsNd(P_C_NsNd)=popBee_NsNd(i,P_C_NsNd)-abs(uk(j)-
lk(j)*(popBee_NsNd(i,P_C_NsNd)-popBee_NsNd(neighbour,P_C_NsNd))*(rand-
0.5)*2);
                else
                    sol_NsNd(P_C_NsNd)=popBee_NsNd(i,P_C_NsNd)+(uk(j)-
lk(j))*(popBee_NsNd(i,P_C_NsNd)-popBee_NsNd(neighbour,P_C_NsNd))*(rand-
0.5)*2;
                end
        %  /*if generated parameter value is out of boundaries, it is
shifted onto the boundaries*/
         ind_NsNd=find(sol_NsNd<lbi);
         sol_NsNd(ind_NsNd)=lbi(ind_NsNd);
         ind_NsNd=find(sol_NsNd>ubi);
         sol_NsNd(ind_NsNd)=ubi(ind_NsNd);
         %evaluate new solution
         ObjValSol_NsNd=objfunc_NsNd(fitness,sol_NsNd);
         FitnessSol_NsNd=calculateFitness(ObjValSol_NsNd);
        % /*a greedy selection is applied between the current solution i and
its mutant*/
        if (FitnessSol_NsNd>Fitness_NsNd(i)) %/*If the mutant solution is
better than the current solution i, replace the solution with the mutant
and reset the trial counter of solution i*/
            popBee_NsNd(i,:)=sol_NsNd;
            Fitness_NsNd(i)=FitnessSol_NsNd;
            ObjVal_NsNd(i)=ObjValSol_NsNd;
            trial(i)=0;
         else
            trial(i)=trial(i)+1; %/*if the solution i can not be improved,
increase its trial counter*/
        end;
    end;
    i=i+1;
    if (i==(FoodNumber)+1)
        i=1;
    end;
end;%End the ONLOCKER
%/*The best food source is memorized*/
```

```
          ind_NsNd=find(ObjVal_NsNd==min(ObjVal_NsNd));
          ind_NsNd=ind_NsNd(end);
          if (ObjVal_NsNd(ind_NsNd)<GlobalMin_NsNd)
          GlobalMin_NsNd=ObjVal_NsNd(ind_NsNd);
          GlobalParams_NsNd=popBee_NsNd(ind_NsNd,:);
          end;
%*%*%*%*%*%*%*% EMPLOYED BEE BECOMES THE SCOUT BEE %*%*%*%*%*%*%*%*%*%*%*
%/*Determine the food sources whose trial counter exceeds the "limit"
value.
ind_NsNd=find(trial==max(trial));
ind_NsNd=ind_NsNd(end);
if (trial(ind_NsNd)>limit)
    trial(ind_NsNd)=0;
    sol_NsNd=(ubi-lbi).*rand(1,D)+lbi;
    ObjValSol_NsNd=objfunc_NsNd(fitness,sol_NsNd);
    FitnessSol_NsNd=calculateFitness(ObjValSol_NsNd);
    popBee_NsNd(ind_NsNd,:)=sol_NsNd;
    Fitness_NsNd(ind_NsNd)=FitnessSol_NsNd;
    ObjVal_NsNd(ind_NsNd)=ObjValSol_NsNd;
end;
fprintf('CABCA_NsNd = %d ObjVal_NsNd = %g\n',iter,GlobalMin_NsNd);
end%End of ABC
GlobalMins_NsNd(r)=GlobalMin_NsNd;
end%end of Run
clc
help 'CABCA_NsNd.m'
disp('The Optimal Value Based on CABCA_NsNd')
disp('************************************')
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
disp('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
Best_NsNd=min(GlobalMins_NsNd(r))
```

APPENDIX C

M-File Code of the Objective Function

```
% Ackley Function
if n==1
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
    if l>=31
        X(:,(31:l))=[];
    end
    end
    x=zeros(k,1);
    z=zeros(k,1);
    for i=1:30
        for h=1:k
        x(h)=x(h)+cos(2*pi*X(h,i));
        z(h)=z(h)+(X(h,i)^2);
        end
    end
y=zeros(k,1);
    for ll=1:k
        y(ll)=-20*exp(-0.2*sqrt(z(ll)/30))-exp(x(ll)/30)+20+exp(1);
    end
    Y=y;
End

% Cosine Mixture Function CM
if n==2
    x=zeros(k,1);
    z=zeros(k,1);
    for i=1:l
        for h=1:k
        x(h)=x(h)+cos(5*pi*X(h,i));
        z(h)=z(h)+(X(h,i)^2);
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=-0.1*x(ll)+z(ll);
    end
    Y=y;
End

% DejongF4 Function
if n==3
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
```

```
    else
    if l>=31
        X(:,(31:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
        x(h)=x(h)+i*X(h,i)^4;
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll);
    end
    Y=y;
End

%Exponential Function
if n==5
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
    if l>=31
        X(:,(31:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:l
        for h=1:k
        x(h)=x(h)+X(h,i)^2;
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=exp(-0.5*sum(x(ll)));
    end
    Y=y;
End

% Griewanggk Function
if n==4
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
    if l>=31
        X(:,(31:l))=[];
    end
```

```
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
        x(h)=x(h)+X(h,i)^2;
        end
    end
    z=zeros(k,1);
    for i=1:30
        for h=1:k
        z(h)=z(h)*cos((X(h,i))/sqrt(i));
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll)/4000-z(ll)+1;
    end
    Y=y;
End

% Hyperelliptic Function
if n==6
    if l<20
        m=20-l;
        X=[X,zeros(k,m)];

    else
    if l>=21
        X(:,(21:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:20
        for h=1:k
        x(h)=x(h)+(i^2)*X(h,i)^2;
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll);
    end
    Y=y;
End

% Penalized Levy and Montolvo 1 PM1 Function
if n==7
    x=zeros(k,1);
    for i=1:l-1
        for h=1:k
        x(h)=x(h)+((X(h,i)-1)^2)*(1+(sin(pi*X(h,i+1)))^2)+(X(h,l)-1)^2;
```

```
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=(pi/l)*(10*((sin(pi*X(h,1)))^2)+x(ll));
    end
    Y=y;
End

% Penalized Levy and Montolvo 2 PM2 Function
if n==8
    x=zeros(k,1);
    for i=1:l-1
        for h=1:k
        x(h)=x(h)+((X(h,i)-1)^2)*(1+(sin(3*pi*X(h,i+1)))^2)+((X(h,l)-
1)^2)*(1+(sin(2*pi*X(h,i)))^2);
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=0.1*(10*((sin(3*pi*X(h,1)))^2)+x(ll));
    end
    Y=y;
End

% Neumaire Function
if n==9
    x=zeros(k,1);
    z=zeros(k,1);
    for i=1:l
        for h=1:k
        x(h)=x(h)+(X(h,i)-1)^2;
        if i>1
        z(h)=z(h)+(X(h,i)*X(h,i-1));
        end
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll)-z(ll);
    end
    Y=y;
End

% Rastrigin Function
if n==10
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
```

```
    else
    if l>=31
        X(:,(31:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
        x(h)=x(h)+((X(h,i)^2)-10*cos(2*pi*X(h,i))+10);
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll);
    end
    Y=y;
End

% Rosenbrock Fuction
if n==11
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];

    else
    if l>=31
        X(:,(31:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:29
        for h=1:k
        x(h)=x(h)+100*(X(h,i+1)-X(h,i)^2)^2+(X(h,i)-1)^2;
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll);
    end
    Y=y;
End

% Sal Function
if n==12
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
    if l>=31
        X(:,(31:l))=[];
```

```
    end
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
        x(h)=x(h)+X(h,i)^2;
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=1-cos(2*pi*sqrt(x(ll)))+0.1*sqrt(x(ll));
    end
    Y=y;
End

% Schwefel Function
if n==13
    if l<10
        m=10-l;
        X=[X,zeros(k,m)];
    else
    if l>=11
        X(:,(11:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:10
        for h=1:k
        x(h)=x(h)+ X(h,i)*sin(sqrt(abs(X(h,i))));
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=418.9828872724339*10-x(ll);
    end
    Y=y;
End

% Schaffer Function
if n==14
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
    if l>=31
        X(:,(31:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:29
```

```
        for h=1:k
x(h)=x(h)+(((X(h,i)^2)+(X(h,i+1)^2)^0.25)*(abs((sin(50*(((X(h,i)^2)+(X(h,i+
1)^2)^0.1))))^2)+1));
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll);
    end
    Y=y;
End

% Sphere Function
if n==15
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
    if l>=31
        X(:,(31:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
        x(h)=x(h)+X(h,i)^2;
        end
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll);
    end
    Y=y;
end

% Axis Parallel Hyper-Ellipsoid Function
if n==20
    if l<20
        m=20-l;
        X=[X,zeros(k,m)];
    else
    if l>=21
        X(:,(21:l))=[];
    end
    end
    x=zeros(k,1);
    for i=1:20
        for h=1:k
        x(h)=x(h)+(i)*X(h,i)^2;
        end
```

```
    end
    y=zeros(k,1);
    for ll=1:k
        y(ll)=x(ll);
    end
    Y=y;
End

% Zakharov Function
if n==21
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
        if l>=31
            X(:,(31:l))=[];
        end
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
            x(h)=x(h)+X(h,i)^2+(0.5*i*X(h,i))^2+(0.5*i*X(h,i))^4;
        end
        y=zeros(k,1);
        for ll=1:k
           y(ll)=x(ll);
        end
        Y=y;
    end

end

% Step Function
if n==22
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
        if l>=31
            X(:,(31:l))=[];
        end
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
            x(h)=x(h)+(floor(X(h,i)+0.5))^2;
        end
        y=zeros(k,1);
        for ll=1:k
           y(ll)=x(ll);
```

```
        end
        Y=y;
    end
end

% Quartic Function
if n==23
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
        if l>=31
            X(:,(31:l))=[];
        end
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
            x(h)=x(h)+i*X(h,i)^4+rand;
        end
        y=zeros(k,1);
        for ll=1:k
           y(ll)=x(ll);
        end
        Y=y;
    end
end

% Michalewicz Function
if n==24
    if l<30
        m=30-l;
        X=[X,zeros(k,m)];
    else
        if l>=31
            X(:,(31:l))=[];
        end
    end
    x=zeros(k,1);
    for i=1:30
        for h=1:k
            x(h)=x(h)-(sin(X(h,i))*(sin(i*X(h,i)^2/pi))^20)    ;
        end
        y=zeros(k,1);
        for ll=1:k
           y(ll)=x(ll);
        end
        Y=y;
    end
end
```

APPENDIX D

M-File Code of the 3D plots

```
clear all
clc
% Ackley Function
str1='-20*exp(-0.2*sqrt(0.5*(x^2+y^2)))';
str2='-exp(0.5*(cos(2*pi*x)+cos(2*pi*y)))+20+exp(1)';
funstr=strcat(str1,str2);
% Converting to an inline function
f=vectorize(inline(funstr));
% range=[xmin xmax ymin ymax];
range=[-5 5 -5 5];
Ngrid=100;
dx=(range(2)-range(1))/Ngrid;
dy=(range(4)-range(3))/Ngrid;
[x,y]=meshgrid(range(1):dx:range(2),...
               range(3):dy:range(4));
z=f(x,y);
% Display the shape of the objective function
figure(1);    surfc(x,y,z);
title('Ackley Optimization Test Function.')
pause
% 3D Visualization of Cosine Mixture Test function
x1 = linspace(-5.12,5.12,100);
x2 = linspace(-5.12,5.12,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluateing the function
Z =(-0.5)*cos(5*pi*XX1)+exp(XX1.^2)+(-0.5)*cos(5*pi*XX2)+exp(XX2.^2);
globalmin=min(Z)
%The 3D plot
surfc(x1,x2,Z')
title('Cosine Mixture Optimization Test Function.')
pause
% 3D Visualization of Dejong 4th-order function
x1 = linspace(-5.12,5.12,100);
x2 = linspace(-5.12,5.12,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluateing the function
Z = 1*XX1.^4+2*XX2.^4;
%The 3D plot
surfc(x1,x2,Z')
title('Dejongf4 Optimization Test Function.')
pause
% 3D Visualization of Exponential function
x1 = linspace(-5.12,5.12,100);
x2 = linspace(-5.12,5.12,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluateing the function
Z = exp(-0.5*XX1.^2)+exp(-0.5*XX2.^2);
```

```
%The 3D plot
surfc(x1,x2,Z')
title('Exponential Optimization Test Function.')
pause
% 3D Visualization of Greiwangk function
x1 = linspace(-600,600,50);
x2 = linspace(-600,600,50);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluateing the function
Z=(1/4000)*XX1.^2-cos(XX1/sqrt(1))+(1/4000)*XX2.^2-
cos(XX2/sqrt(2))*cos(XX2/sqrt(1))
%The 3D plot
surfc(x1,x2,Z)
title('Greinwangk Optimization Test Function.')
pause
% 3D Visualization of Hyperelliptic function
x1 = linspace(-5.12,5.12,100);
x2 = linspace(-5.12,5.12,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluateing the function
Z=XX1.^2+2^2*XX2.^2
%The 3D plot
surfc(x1,x2,Z)
title('Hyperelliptic Optimization Test Function.')
pause
% Mitchalewicz Function
m=10;
x1 = linspace(0,pi);
x2 = linspace(0,pi);
[X1,X2] = meshgrid(x1,x2);
%Evaluating the function
%Z=(sin(X1)*(sin(X1^2/pi))^(2*m))+sin(X2)*(sin(X2^2/pi))^(2*m);
%the 3D plot
%surfc(x1,x2,Z);
title('Mitchlewicz Optimization test function')
% pause
% disp('Press any key for the second Mitchelwitcz plot')
Z1=(sin(X1).*(sin(X1.^2/pi))^(2*m))+sin(X2).*(sin(X2.^2./pi))^(2*m);
%the 3D plot
surfc(x1,x2,Z1);
title('Mitchlewicz Optimization test function')
pause
% Neumaier Third Optimization Test Function
% initialize the variables
x1=linspace(-10,10,100);
x2=linspace(-10,10,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluate the function
Z=(XX2-1).^2-(XX2*XX1);
```

```
%ploting the 3D
surfc(x1,x2,Z')
title('Neumaier Optimisation Test Function');
pause
%Levy and Montalvo Optimization First Test Function
%initialize the variables
x1=linspace(-2.12,5.12,10);
x2=linspace(-5.12,5.12,10);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluate the function
Z=(pi/2)*(10*(sin(pi*XX1))^2+((XX1-1)^2)*(1+(sin(XX1*pi)))+(XX1-
1)^2)+(pi/2)*(10*(sin(pi*XX2))^2+((XX2-1)^2)*(1+(sin(XX2*pi)))+(XX2-1)^2);
%ploting the 3D
surfc(x1,x2,Z)
title('Levey and Motalvo(LM1) Optimisation Test Function')
pause
%Levy and Montalvo Optimization Second Test Function
%initialize the variables
x=linspace(-1,1,50);
y=linspace(-1,1,50);
[X,Y]=meshgrid(x,y);
%Evaluate the function
Z=(sin(3*pi*X))^2+((X-1)^2)*(1+(sin(3*pi*Y)^2))+((Y-1)^2)*(1+(sin(2*pi*Y)))
%ploting the 3D
surfc(x,y,Z')
title('Levey and Motalvo(LM2) Optimisation Test Function');
pause
% Quartic Function
x1 = linspace(-1.28,1.28);
x2 = linspace(-1.28,1.28);
[X1,X2] = meshgrid(x1,x2);
%Evaluating the function
Z = (X1+rand)+(2*X2+rand);
%the 3D plot
surfc(x1,x2,Z);
title('Quartic Optimization test function')
pause
% 3D Visualization of Rastrigrin function
x1 = linspace(-5.12,5.12,100);
x2 = linspace(-5.12,5.12,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluateing the function
Z = (XX1.^2-10*cos(2*pi*XX1)) + (XX2.^2-10*cos(2*pi*XX2))+20
%The 3D plot
surfc(x1,x2,Z')
title('Rastrigin Optimization Test Function.')
pause
%3D Visualization of Rosenbrock function
x1 = linspace(-2,2,100);
x2 = linspace(-2,2,100);
```

```
[X1,X2]=meshgrid(x1,x2);
%Evaluating the function
Z=100*(X2-X1.^2).^2+(1-X1).^2;
surfc(x1,x2,Z');
title('Rosenbrock Optimization Test Function');
pause
% Sal Optimization Test Function
%initialize the variables
x1=linspace(-2*pi,2*pi,100);
x2=linspace(-2*pi,2*pi,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluate the function
Z=1-cos(2*pi*(sqrt((XX1.^2)+(XX2.^2))))+0.1*sqrt(XX1.^2+XX2.^2);
%ploting the 3D
surfc(x1,x2,Z')
title('Sal Optimisation Test Function');
pause
% Schaffer Optimization Test Function
%initialize the variables
x1=linspace(-100,100,100);
x2=linspace(-100,100,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluate the function
Z=abs(((XX1.^2+XX2.^2).^(0.25))*(((sin(50*(XX1.^2+XX2.^2))^0.1).^2+1)));
%ploting the 3D
surfc(x1,x2,Z)
title('Schaffer Optimisation Test Function');
pause
% Schwefel Function
x1 = linspace(-500,500,100);
x2 = linspace(-500,500,100);
[X1,X2] = meshgrid(x1,x2);
%Evaluating the function
Z = (-1*X1.*sin(sqrt(abs(X1))))+(-1*X2.*sin(sqrt(abs(X2))));
%the 3D plot
surfc(x1,x2,Z);
title('Schwefel Optimization test function')
pause
% Sphere Function
% initialize the variables
x1=linspace(-5.12,5.12,100);
x2=linspace(-5.12,5.12,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluate the function
Z=XX1.^2+XX2.^2;
%ploting the 3D
surfc(x1,x2,Z)
title('Sphere Optimisation Test Function');
pause
clear all
```

```
% 3D Penalized Levy and Montalvo Optimization First Test
Function
%initialize the variables
x1=linspace(-100,100,100);
x2=linspace(-100,100,100);
[XX1,XX2]=meshgrid(x1,x2);
%Evaluate the function
for i=1:25
Z=floor(XX1.^2+0.5);
end
%ploting the 3D
surfc(x1,x2,Z)
title('Step Optimisation Test Function');

% Zakharov Optimization Test Function
x1 = linspace(-5,10);
x2 = linspace(-5,10);
[X1,X2] = meshgrid(x1,x2);
% Evaluating the function
Z =
((X1.^2)+(0.5.*X1).^2+(0.5.*X1).^4)+((X2.^2)+(0.5.*X2).^2+(0.5.*X2).^4);
%the 3D plot
surfc(x1,x2,Z);
title('Zakharov Optimization test function')
```

APPENDIX E

M-File Code for K_p, K_i and K_d

```
%  clc
 close all
 clear all
% this program required that you provide your PID gain and then the plant
% transfer function for it to compute control the system
    s=tf('s'); %Make 's' a tansfer function symbol
G=0.01/(0.05*s^2+0.06*s+0.1);
    kp=      0.3144;%input('Please provide the proportional gain:=  ');
%     clc
    ki=   0;%input('Please provide the integral gain:= ');
%     clc
    kd=      0.1235;%input('Please provide the derivate gain:= ');
%     clc
    Tf=input('Please provide the filter time constant (Note: 0.1 is a good
value):=');
    K=pid(kp,ki,kd,Tf);
    T=feedback(K*G,1);
    step(G)
hold on
step(T)
    stepinfo(T)
    grid on
    legend('Uncontroled Response','CABCA Based PID Controled Response')
```