
Electrical Engineering Theses

Electrical Engineering

Fall 12-2011

Using a Micro-Integrator to Eliminate the Numerical Butterfly Effect in Non-Linear Chaotic Partial Differential Equations

Joshua Jemegbe

Follow this and additional works at: https://scholarworks.uttyler.edu/ee_grad



Part of the Electrical and Computer Engineering Commons

Recommended Citation

Jemegbe, Joshua, "Using a Micro-Integrator to Eliminate the Numerical Butterfly Effect in Non-Linear Chaotic Partial Differential Equations" (2011). *Electrical Engineering Theses*. Paper 2.

<http://hdl.handle.net/10950/44>

This Thesis is brought to you for free and open access by the Electrical Engineering at Scholar Works at UT Tyler. It has been accepted for inclusion in Electrical Engineering Theses by an authorized administrator of Scholar Works at UT Tyler. For more information, please contact tgullings@uttyler.edu.

**USING A MICRO-INTEGRATOR TO ELIMINATE THE
NUMERICAL BUTTERFLY EFFECT IN NON-LINEAR CHAOTIC
PARTIAL DIFFERENTIAL EQUATIONS**

by

JOSHUA MISAN JEMEGBE

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering
Department of Electrical Engineering.

Dr. Ron Pieper, Ph.D., Committee Chair

College of Engineering and Computer Science

The University of Texas at Tyler
December 2011

The University of Texas at Tyler
Tyler, Texas.

This is to certify that the Master's Thesis of

JOSHUA MISAN JEMEGBE

has been approved for the thesis requirement on

December, 2011

for the Master of Science in Electrical Engineering

Approvals:



Thesis Chair: Ron Pieper, Ph.D.



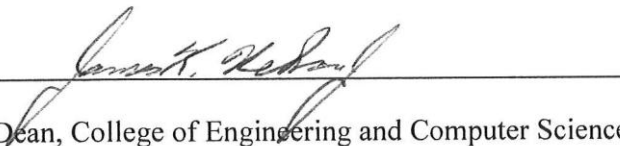
Member: Mukul Shirvaikar, Ph.D.



Member: Deborah Koslover, Ph.D.



Chair: Department of Electrical Engineering



Dean, College of Engineering and Computer Science

Table of Contents

| | |
|---|------|
| List of Tables..... | iv |
| List of Figures..... | v |
| Abstract..... | viii |
| 1. Introduction | |
| 1.1 Early Appearances of Chaos..... | 1 |
| 1.2 The Numerical Butterfly Effect..... | 1 |
| 1.3 Implementations of Chaotic Modeling..... | 2 |
| 1.3.1 Chaotic Encryption..... | 3 |
| 1.3.2 Chaotic Secure Communications..... | 3 |
| 1.4 Computing Chaotic Signals..... | 3 |
| 2. Background | |
| 2.1 The Scope of this Research..... | 5 |
| 2.2 The Sets of Chaotic Partial Differential Equations used in this Study | 5 |
| 2.2.1 The Lorenz System..... | 5 |
| 2.2.2 The Rossler System..... | 6 |
| 2.2.3 The Chen System..... | 7 |
| 2.2.4 The Chua System..... | 7 |
| 2.2.5 The Hadley System..... | 8 |
| 2.2.6 The ACT System..... | 9 |
| 2.2.7 The Diffusionless Lorenz System..... | 9 |
| 2.3 The Numerical Integration Algorithms Used..... | 10 |
| 2.3.1 The Euler's Method..... | 10 |
| 2.3.2 The Modified Euler's Method..... | 11 |
| 2.3.3 Simpson's Rule..... | 12 |
| 2.3.4 The Fourth Order Runge-Kutta Method..... | 14 |
| 2.4 The Maximum Error Obtainable in Each Numerical Integration Method.. | 16 |
| 2.5 Lyapunov Coefficients for the Chaotic Systems in this Study..... | 16 |

| | |
|---|----|
| 3. Introduction to the Micro-Integrator scheme | 18 |
| 3.1 Overview..... | 18 |
| 3.2 Integration Without the Micro-Integrator..... | 18 |
| 3.3 The Proposal of the Micro-Integrator Solution..... | 19 |
| 3.4 Calculation of the Error..... | 22 |
| 3.4.1 Sample Error Calculation..... | 23 |
| 3.5 A more detailed description of the Micro-Integrator..... | 23 |
| 3.6 Testing the Micro-Integrator Algorithm..... | 25 |
| 4. Methodology for the Evaluation of the Performance of the Micro-Integrator | |
| 4.1 The Micro-Integrator Performance Factor..... | 26 |
| 4.1.1 The Memory Estimation Disclaimer..... | 26 |
| 4.2 Limiting the Error..... | 26 |
| 4.3 Error Control Without the Micro-Integrator..... | 27 |
| 4.3.1 Algorithm for the error control without the Micro-Integrator..... | 27 |
| 4.3.2 Alternate Depiction of the error control Algorithm without the Micro-Integrator..... | 29 |
| 4.4 Error Control With the Micro-Integrator..... | 31 |
| 4.4.1 Algorithm for the error control with the Micro-Integrator..... | 31 |
| 4.4.2 Alternate Depiction of the error control Algorithm with the Micro-Integrator | 34 |
| 4.5 Other Parameters Introduced..... | 36 |
| 4.5.1 The Performance Measures..... | 36 |
| 4.5.2 Algorithm for the error control without the Micro-Integrator..... | 36 |
| 4.5.3 The Performance Ratio..... | 36 |
| 4.6 Problems with the Modular Programming approach..... | 37 |
| 5. Test Cases and Results | 38 |
| 5.1 A Brief description of the Parameters used in the Tables..... | 38 |
| 5.2 Result Tables..... | 38 |
| 5.3 Graphical Presentation of Resultts..... | 43 |
| 5.4 Observations on The Results..... | 43 |

| | |
|--|-----|
| 5.5 Table Data Presented in Plot..... | 52 |
| 5.5.1 Illustrating the Effectiveness of the Error Control Algorithms.... | 52 |
| 5.5.2 The Performance Measures of the Chaotic Systems..... | 53 |
| 5.5.3 Percentage Reduction in CPU time..... | 57 |
| 5.5.4 Percentage Reduction in CPU time..... | 57 |
| 6. Conclusion and Future Work | 59 |
| 6.1 Conclusion..... | 59 |
| 6.2 Future Work | 59 |
| References..... | 60 |
| Appendix A Interpolation Algorithm..... | 63 |
| Appendix B The Matlab Codes | 65 |
| Appendix B-1 Lorenz Equations by Euler's Method without the Micro Integrator | 65 |
| Appendix B-2 Lorenz Equations by the Modified Euler's Method without the Micro-Integrator | 69 |
| Appendix B-3 Lorenz Equations by Simpson's Method without the Micro- Integrator..... | 74 |
| Appendix B-4 Lorenz Equations by Runge-Kutta's Method without the Micro- Integrator..... | 80 |
| Appendix B-5 Lorenz Equations by Euler's Method with the Micro-Integrator ... | 86 |
| Appendix B-6 Lorenz Equations by the Modified Euler's Method with the Micro-Integrator..... | 90 |
| Appendix B-7 Lorenz Equations by Simpson's Method with the Micro- Integrator | 94 |
| Appendix B-8 Lorenz Equations by Runge-Kutta's Method with the Micro- Integrator..... | 99 |
| Appendix C Principal Subroutines..... | 104 |
| Appendix C-1 The Interpolation Subroutine..... | 104 |
| Appendix C-2 The Error Calculation Subroutine..... | 104 |

List of Tables

| | | |
|------------|--|----|
| Table 2.1 | The error bound in each numerical method by Step Size order proportionality | 16 |
| Table 2.2 | Lyapunov Coefficients for the chaotic systems tested..... | 17 |
| Table 5.1 | Results from the Lorenz Equations with and without the Micro-Integrator | 39 |
| Table 5.2 | Results from the Rossler equations with and without the Micro-Integrator..... | 39 |
| Table 5.3 | Results from the Chen equations with and without the Micro-Integrator..... | 39 |
| Table 5.4 | Results from the Chua equations with and without the Micro-Integrator..... | 40 |
| Table 5.5 | Results from the Hadley equations with and without the Micro-Integrator..... | 40 |
| Table 5.6 | Results from the ACT equations with and without the Micro-Integrator..... | 40 |
| Table 5.7 | Results from the Diffusionless Lorenz with and without the Micro-Integrator..... | 41 |
| Table 5.8 | Results from computing all the listed chaotic sets of partial differential equations with Euler's method with and without the Micro-Integrator.. | 41 |
| Table 5.9 | Results from computing all the listed chaotic sets of partial differential equations with Euler's method with and without the Micro-Integrator on an increased time window | 42 |
| Table 5.10 | Demonstrating the accuracy of the error control algorithm by computations with and without it for the Lorenz and Rossler systems with the Micro-Integrator | 42 |

List of Figures

| | | |
|-------------|---|----|
| Figure 1.1 | Plot of signal strength vs. time for the Lorenz Equations | 2 |
| Figure 2.1 | Poincare Chaotic Attractor Map of the Lorenz System | 6 |
| Figure 2.2 | Poincare Chaotic Attractor Map of the Rossler System | 6 |
| Figure 2.3 | Poincare Chaotic Attractor Map of the Chen System..... | 7 |
| Figure 2.4 | Poincare Chaotic Attractor Map of the Chua System | 8 |
| Figure 2.5 | Poincare Chaotic Attractor Map of the Hadley System..... | 8 |
| Figure 2.6 | Poincare Chaotic Attractor Map of the ACT System.. | 9 |
| Figure 2.7 | Poincare Chaotic Attractor Map of the Diffusionless Lorenz System... | 10 |
| Figure 2.8 | Visualization Sketch for Euler’s Method..... | 10 |
| Figure 2.9 | Visualization Sketch for the Modified Euler’s Method..... | 11 |
| Figure 2.10 | Visualization Sketch for Simpson’s Method | 13 |
| Figure 2.11 | Visualization Sketch for Runge-Kutta’s Method..... | 14 |
| Figure 3.1 | Integration without the Micro-Integrator..... | 18 |
| Figure 3.2 | Plot of the x variable of the Lorenz equations against time for $N_A = 5000$ and $T=15s$ | 19 |
| Figure 3.3 | Schematic for the Micro-Integrator Concept | 20 |
| Figure 3.4 | Automated Adaptive Extended Micro-Integrator Algorithm, typical $N_{HRo} = 2, m = 2, E = 0.01$ | 24 |
| Figure 3.5 | Convergence demonstration with Micro-Integrator algorithm, $N_A = 5000, N_{HRo} = 2, m = 2, E = 0.01, T=15seconds$ | 25 |
| Figure 4.1 | (a) Predictor Checker Algorithm Without the Micro-Integrator, (b) Error Control Algorithm Without the Micro-Integrator using the Predictor Checker Algorithm..... | 28 |
| Figure 4.2 | Alternate Depiction of the Error Control Algorithm without the Micro-Integrator..... | 30 |

| | | |
|------------|---|----|
| Figure 4.3 | (a) First Predictor Checker Algorithm With the Micro-Integrator, (b) Second Predictor Checker Algorithm With the Micro-Integrator, (c) Error Control Algorithm With the Micro-Integrator using the Predictor Checker Algorithm..... | 32 |
| Figure 4.4 | Alternate Depiction of the Error Control Algorithm with the Micro-Integrator..... | 35 |
| Figure 5.1 | x vs t for the Lorenz System (a) Without the Micro-Integrator $N_A = 2000$ (b) Without the Micro-Integrator $N_A = 3626132$ (c) With the Micro-Integrator $N_A = 1024$ | 44 |
| Figure 5.2 | x vs t for the Rossler System (a) Without the Micro-Integrator $N_A = 2000$ (b) Without the Micro-Integrator $N_A = 345984$ (c) With the Micro-Integrator $N_A = 1024$ | 45 |
| Figure 5.3 | x vs t for the Chen System (a) Without the Micro-Integrator $N_A = 2000$ (b) Without the Micro-Integrator $N_A = 1839110$ (c) With the Micro-Integrator $N_A = 1024$ | 46 |
| Figure 5.4 | x vs t for the Chua System (a) Without the Micro-Integrator $N_A = 2000$ (b) Without the Micro-Integrator $N_A = 80551$ (c) With the Micro-Integrator $N_A = 1024$ | 47 |
| Figure 5.5 | x vs t for the Hadley System (a) Without the Micro-Integrator $N_A = 2000$ (b) Without the Micro-Integrator $N_A = 1946160$ (c) With the Micro-Integrator $N_A = 1024$ | 48 |
| Figure 5.6 | x vs t for the ACT System (a) Without the Micro-Integrator $N_A = 2000$ (b) Without the Micro-Integrator $N_A = 2059814$ (c) With the Micro-Integrator $N_A = 1024$ | 49 |

| | | |
|-------------|--|----|
| Figure 5.7 | x vs t for the Diffusionless Lorenz (a) Without the Micro-Integrator $N_A = 2000$ (b) Without the Micro-Integrator $N_A = 633032$ (c) With the Micro-Integrator $N_A = 1024$ | 50 |
| Figure 5.8 | x vs. t for the Chen equations Without the Micro Integrator (a) Using Euler's Method, $N_A = 2000$ (b) Using the modified Euler's Method, $N_A = 2000$ (c) Using Simpson's Method, $N_A = 2000$ (d) Using Runge-Kutta's Method, $N_A = 2000$ | 51 |
| Figure 5.9 | Illustrating the Effectiveness of the Error Control Algorithm using the Lorenz Equations..... | 52 |
| Figure 5.10 | Illustrating the Effectiveness of the Error Control Algorithm using the Rossler Equations..... | 53 |
| Figure 5.11 | Performance Measures of the Lorenz System..... | 54 |
| Figure 5.12 | Performance Measures of the Rossler System..... | 54 |
| Figure 5.13 | Performance Measures of the Chen System..... | 55 |
| Figure 5.14 | Performance Measures of the Chua System..... | 55 |
| Figure 5.15 | Performance Measures of the Hadley System..... | 56 |
| Figure 5.16 | Performance Measures of the ACT System..... | 56 |
| Figure 5.17 | Performance Measures of the Diffusionless Lorenz System..... | 57 |
| Figure 5.18 | Percentage Reduction in CPU time..... | 58 |
| Figure 5.19 | Performance Ratios for the chaotic systems..... | 58 |
| Figure A.1 | Graphical representation of the Interpolation concept..... | 63 |
| Figure A.2 | Alternate Graphical representation of the Interpolation concept..... | 63 |
| Figure A.3 | The Interpolation algorithm..... | 64 |

Abstract

USING A MICRO-INTEGRATOR TO ELIMINATE THE NUMERICAL BUTTERFLY EFFECT IN NON-LINEAR CHAOTIC PARTIAL DIFFERENTIAL EQUATIONS

Joshua Misan Jemegbe

Thesis Chair: Ron Pieper, Ph. D.

The University of Texas at Tyler
December 2011

Chaos theory is a relatively new scientific paradigm for the analysis, simulation and prediction of non-linear phenomena whose initial conditions determine the behavior of their entire time series representation. It finds many applications in mathematics, science, and engineering. These include, but are not limited, to data encryption and decryption, designing secure communication systems, predicting weather patterns, noise fluctuations on data lines, understanding turbulence in fluid flow, and analyzing quantum wells. Systems that exhibit chaos are called chaotic systems. In computing solutions to non-linear chaotic partial differential equation sets, slight deviations in step size could lead to completely diverging trajectories as the system's time series progresses. This is called the numerical butterfly effect. Smaller step sizes produce arrays closer to the desired continuous time solution, but they require more sampling points and as a result more memory. The Micro-Integrator produces results with a high level of accuracy while using only a fraction of the amount of memory required by conventional numerical integration methods. The reduction in memory requirements by the Micro-Integrator was quantified by introducing a performance factor ' η ' that was mathematically equal to the ratio of the amount of memory required for computing without the Micro-Integrator to that required for computing with it. Recorded values of the performance factor from the tests ranged from 5 to 10^4 , out of which 75% were above 10^3 . The performance factor was also found to depend on the type of chaotic system, the numerical method, and the time window for computation. Less computationally efficient numerical methods resulted in higher performance factors than the more efficient ones.

Chapter One

Introduction

1.1 Early Appearances of Chaos

In 1956, Benoit Mandelbrot published a paper [1] about self similar clusters of apparently random electrical signals that he had observed in phone wires. This brought about a new inquiry as to the nature of these signals and the reasons for their strange patterns. Later, in 1960 Edward N. Lorenz noticed a similar pattern of fluctuations in the non-linear system he was using in weather prediction calculations, and he was able to trace its source to the fact that he had approximated the results from the previous computation stage from the sixth decimal place to the third decimal place [2]. This dependence of the behavior of the entire time series solution of certain non-linear systems on initial conditions and integration step size formed the basis of a new paradigm for their analysis and prediction called 'chaos theory'. Signals that exhibit this dependence are called 'chaotic signals', and the systems that generate them are called 'chaotic systems'. Mitchell Feigenbaum in the mid-70s demonstrated that the Reynolds numbers for turbulence were predictable from chaos theory [3]. In the mid 80s, it was demonstrated that chaos theory is applicable in solving the equations of pipeline networks [4]. More recent investigations have demonstrated that chaotic signals could be used to achieve secure communications [5,6].

1.2 The Numerical Butterfly Effect

Numerical computation of non-linear chaotic partial differential equation sets are not only sensitive to initial conditions but also to integration step size [7-9]. Integrating the same set of non-linear chaotic partial differential equations with different initial conditions or different integration time steps produce entirely different trajectories as the system's time series progresses [2,7,8]. As shown in Figure 1.1. This is known as the numerical butterfly effect [2]. In Figure 1.1, the different plots were obtained by integrating the Lorenz non-linear chaotic partial differential equation set using two different integration step sizes of 0.011 and 0.0055 for the solid green and dotted blue lines respectively.

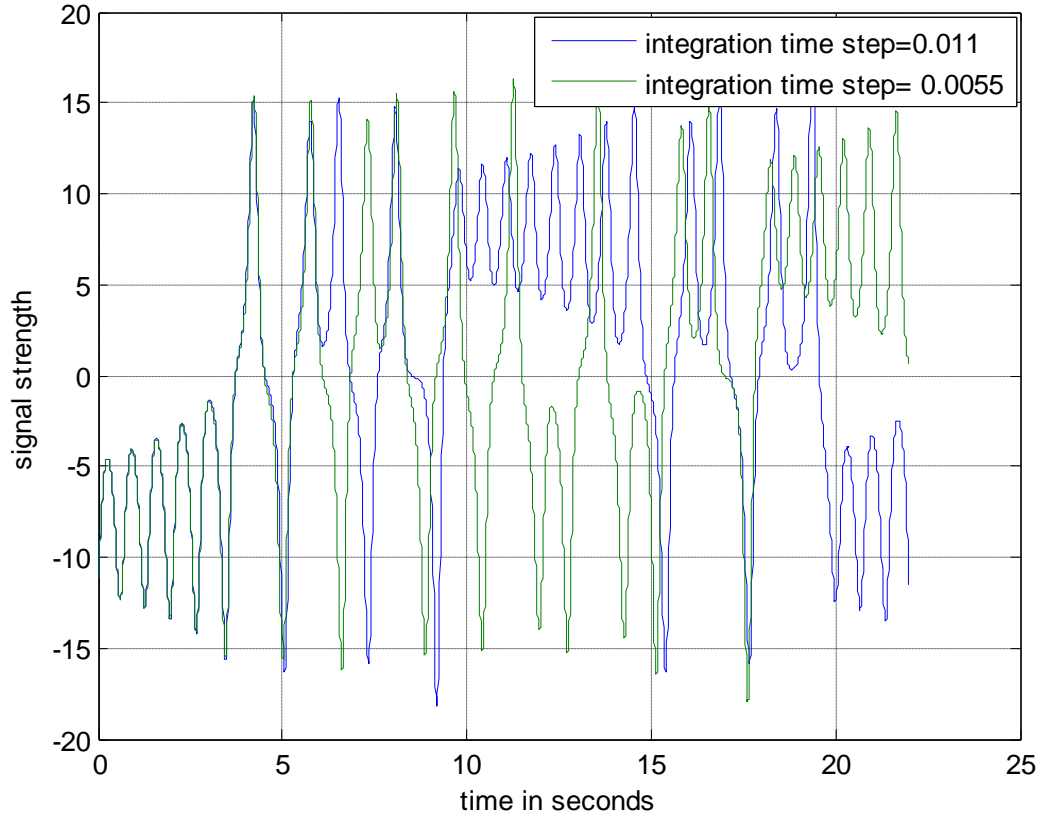


Figure 1.1 Plot of signal strength vs. time for the Lorenz Equations.

1.3 Implementations of Chaotic Modeling

Chaotic equations have been implemented in analysis and simulation of a wide range of phenomena and prove to have been inherent in certain existing theories without being discovered. One of such is computing the quantum trajectory of an individual subatomic particle [10], the dependence of the entire trajectory of the particle on initial conditions and integration time step demonstrate the butterfly effect. Quantum wells in tilted magnetic fields have also been found to make the enclosed electrons move in certain chaotic patterns [11]. Research is also being carried out on how the wave functions of electrons in ballistic motion progress from quantum disorder to chaos [12].

It has also been demonstrated that chaos is present in the transmission line oscillator in an equivalence to the continuum limit of the generalized Rossler system [13]. These are only a few of the applications of chaotic modelling. The next two subsections describe its applications to chaotic encryption and secure communication respectively.

1.3.1 Chaotic Encryption

The erratic tendencies of chaotic signals make them excellent tools for message encryption. This has led to the proposal of numerous schemes for the implementation of chaos based signal encryption systems [14]. Specific attributes have uniquely accompanied certain chaos based encryption algorithms which have placed them at an advantage over other algorithms in their class. A few of these attributes are: significantly reduced complexity [15], higher resistance to attack [16], universal range of implementation [17], and superior encryption efficiency [18]. While a significant number of simulations demonstrate the use of chaotic signals to encrypt signals, some research has been done on using chaotic signals to amplify message signals that have been corrupted by noise [19].

1.3.2 Chaotic Secure Communications

Secure digital communication schemes have also been proposed where chaotic signals are used to encrypt the message at the transmitter, after which it is synchronized with the receiver. Then by means of a decrypting procedure which relies on its synchronism with the transmitter's chaotic signal generator, the original message is recovered [5,6]. It is noteworthy that chaos synchronization helps modulation systems overcome errors in computed chaotic arrays which would otherwise be expected to interfere with the decrypting procedure. So even if the chaotic signal used in the encryption phase at the transmitter has a varying trajectory from the standard array of its values, due to the synchronization, the message signal can still be recovered successfully without any damage done to it.

Apart from using numerical integration algorithms, chaotic signals can also be generated by certain circuit implementations [20-22]. Understanding these could determine the threshold where linear or non-linear systems transit to a chaotic state, and help avert unforeseen problems that may arise in critical systems.

1.4 Computing Chaotic Signals

Ideally, solving the partial differential equations that model a particular class of chaotic signals should produce an array of values that, when represented graphically,

follow the specific trajectory associated with that particular class. But there are other factors that come to play in the integration process. Because there are no analytical solutions available for these chaos modeling equations, numerical integration techniques are the only way to solve them [7]. The sensitive dependence of numerical integration techniques on integration step size brings new considerations [5]. The pseudorandom appearance of chaotic signals make it easy to mistake computed errors for actual chaotic signals, but precision is still a factor here because there is order in chaos. This has led to research for more accurate means of computing chaotic signals such as the method proposed in this research.

A very direct approach to reducing the error in the computed chaotic signal would be reducing the integration step size [7]. This leads to a reduction in the error, but increases the amount of memory used by the computer in executing the integration process. Due to the complexity of these computations, memory limited computers are frequently at a loss. This problem inspired the design of the Micro-Integrator algorithm [7]. In this research it was demonstrated on six different systems of non-linear chaotic partial differential equations to achieve low error levels while using only a fraction of the amount of memory required by conventional integration schemes.

Chapter Two

Background

2.1 The Scope of this Research

This research focuses on applying the proposed Micro-Integrator [7] to eliminate the numerical butterfly effect in the Lorenz [23,31], Rossler [24,31], Chen [25,31], Chua [26,31], Hadley [27,31], ACT [28,31], and Diffusionless-Lorenz chaotic systems of equations[29,31]. Solutions to these equations were computed with and without the Micro-Integrator using four numerical integration methods, the Euler, Modified Euler, Simpson, and Fourth Order Runge-Kutta methods [30,32]. It is a continuation of previous work done in the referenced paper [7] where the Micro-Integrator was first applied to the Lorenz Equations alone and computed using only Euler's method. Here the study has been extended to include the six other sets of chaotic partial differential equation sets and three additional numerical integration methods.

2.2 The Sets of Chaotic Partial Differential Equations used in this Study

The different sets of chaotic partial differential equations used in this study are listed here, and their Poincaré maps as generated by the Modified Euler's method using MATLAB are shown. The Poincare map of a chaotic partial differential equation set is a plot of its x variable versus its y variable over a length of time. It serves as a standard means of identification of different chaotic systems.

2.2.1 The Lorenz System

The Lorenz System of equations is given below:

$$\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} (10y - 10x)dt \\ (28x - y - xz)dt \\ (xy - (8z \div 3))dt \end{pmatrix} \quad (2.1)$$

The initial conditions are $x_0 = -11.2, y_0 = -8.4, z_0 = 33.4$ from references [23,31]. The Poincaré chaotic attractor map for the Lorenz system of equations is shown in Figure. 2.1.

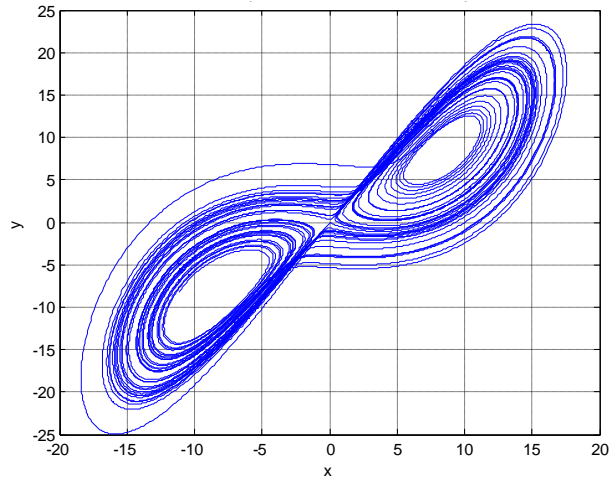


Figure 2.1 Poincaré Chaotic Attractor Map of the Lorenz System.

2.2.2 The Rossler System

The Rossler System of equations is given below:

$$\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} (-y - z)dt \\ (x + 0.2y)dt \\ (0.2 + z(x - 5.7))dt \end{pmatrix} \quad (2.2)$$

The initial conditions are $x_0 = -11.2, y_0 = -2.4, z_0 = 0$ from references [24,31]. The Poincaré chaotic attractor map of the Rossler system of equations is shown in Figure. 2.2.

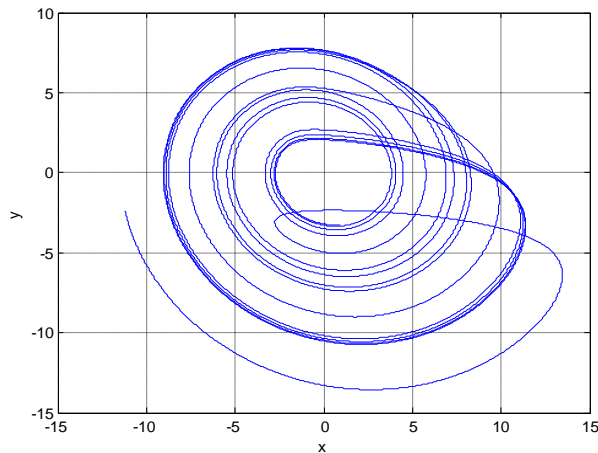


Figure 2.2 Poincaré Chaotic Attractor Map of the Rossler System.

2.2.3 The Chen System

The Chen System of equations is given below

$$\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} (35(y-x))dt \\ (-8xz+28y)dt \\ (xy-3z)dt \end{pmatrix} \quad (2.3)$$

The initial conditions are $x_0 = -10, y_0 = 0, z_0 = 37$ from references [25,31]. The Poincaré chaotic attractor map of the Chen system of equations is shown in Figure. 2.3.

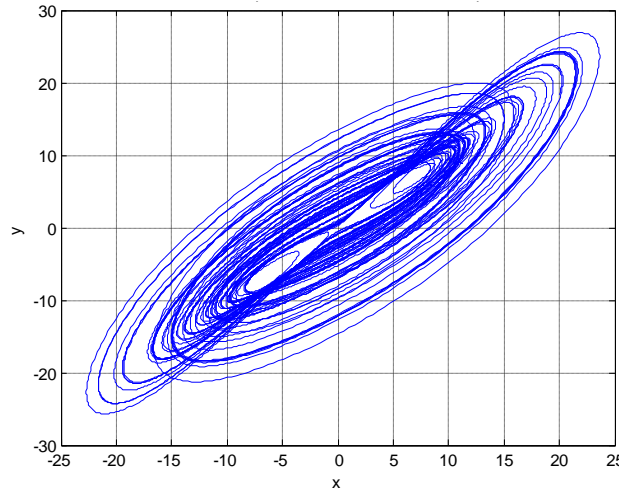


Figure 2.3 Poincare Chaotic Attractor Map of the Chen System.

2.2.4 The Chua System

The Chua System of equations is given below:

$$\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} 9(y-x+(5x/7)+((1.5/7)(|x+1|-|x-1|)))dt \\ (x-y+z)dt \\ (-100y/7)dt \end{pmatrix} \quad (2.4)$$

The initial conditions are $x_0 = 0, y_0 = 0, z_0 = 0.6$ from references [26,31]. The Poincaré chaotic attractor map of the Chua system of equations is shown in Figure. 2.4.

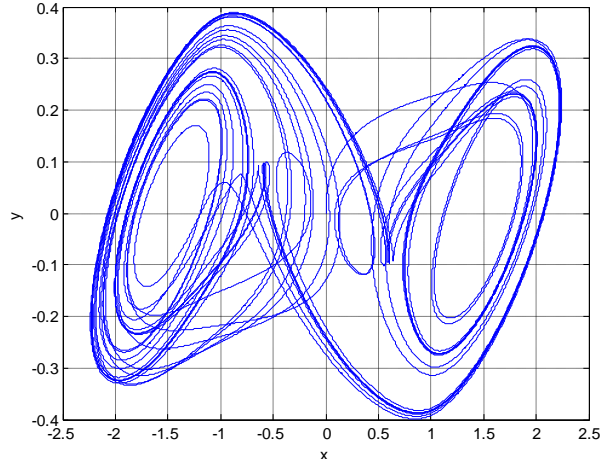


Figure 2.4 Poincaré Chaotic Attractor Map of the Chua System.

2.2.5 The Hadley System

The Hadley System of equations is given below:

$$\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} (-y^2 - z^2 - 0.25x + 2)dt \\ (xy - 4xz - y + 1)dt \\ (4xy + xz - z)dt \end{pmatrix} \quad (2.5)$$

The initial conditions are $x_0 = 0, y_0 = 0, z_0 = 1.3$ from references [27,31]. The Poincaré chaotic attractor map of the Hadley system of equations is shown in Figure. 2.5.

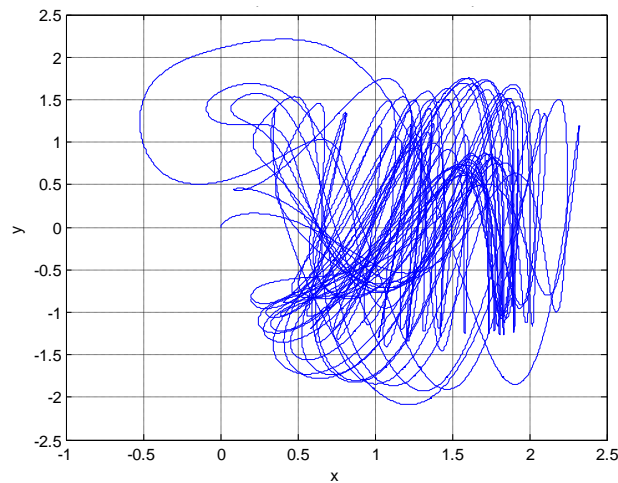


Figure 2.5 Poincaré Chaotic Attractor Map of the Hadley System.

2.2.6 The ACT System

The ACT System of equations is given below.

$$\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} (1.8(x-y))dt \\ ((-4)(1.8y) + xz + 0.02x^3)dt \\ ((-1.5)(1.8)z + xy + (-0.07z^2))dt \end{pmatrix} \quad (2.6)$$

The initial conditions are $x_0 = 0.5, y_0 = 0, z_0 = 0$ from references [28,31]. The Poincaré chaotic attractor map for the ACT system of equations is shown in Figure. 2.6.

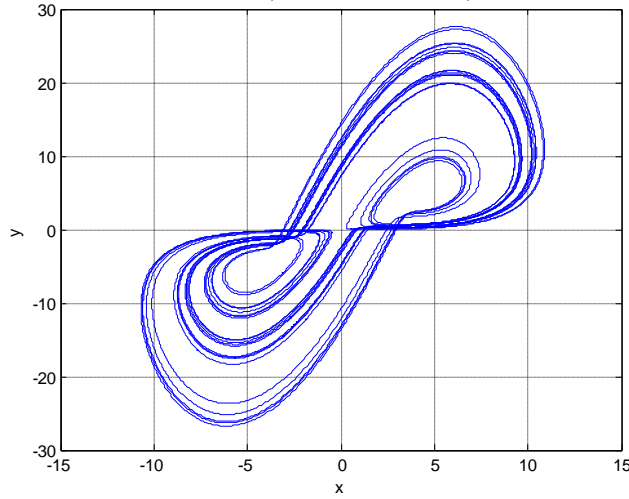


Figure 2.6 Poincaré Chaotic Attractor Map of the ACT System.

2.2.7 The Diffusionless Lorenz System

The Diffusionless Lorenz System of equations is given below:

$$\begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} (-y-x)dt \\ -(xz)dt \\ (xy+1)dt \end{pmatrix} \quad (2.7)$$

The initial conditions are $x_0 = 1, y_0 = -1, z_0 = 0.01$ from references [29,31]. The Poincaré chaotic attractor map for the Diffusionless Lorenz equations is shown in Figure. 2.7.

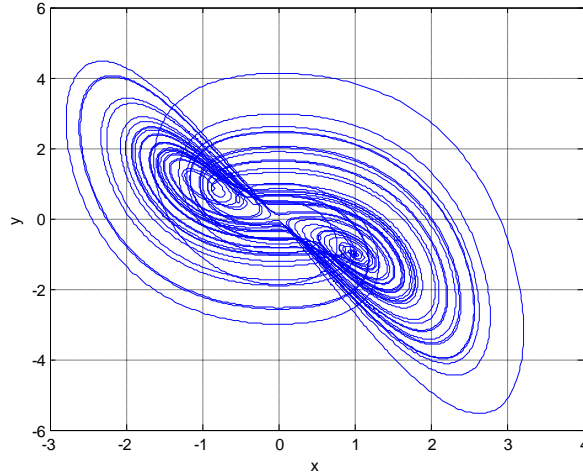


Figure 2.7 Poincare Chaotic Attractor Map of the Diffusionless Lorenz System.

2.3 Numerical Integration Algorithms.

The numerical integration methods used in this study were: the Euler's Method [23], Modified Euler's Method [23], Simpson's Method [21], and the fourth order Runge-Kutta Method [21].

2.3.1 Euler's Method

Euler's method operates according to Figure 2.8 and Equations (2.8a-b).

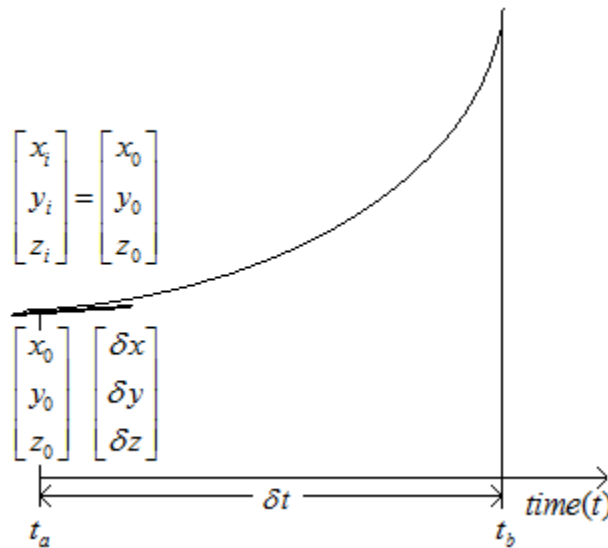


Figure 2.8 Visualization Sketch for Euler's Method.

$$\begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} = \begin{bmatrix} f_1(x_0, y_0, z_0)\delta t \\ f_2(x_0, y_0, z_0)\delta t \\ f_3(x_0, y_0, z_0)\delta t \end{bmatrix} \quad (2.8a)$$

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} \quad (2.8b)$$

For each iteration, the incremental set was obtained according to the specific equation or set of equations being integrated. The incremental set was then added to the initial values to create the next set of values in the solution array.

2.3.2 The Modified Euler's Method

The Modified Euler's method comes next in order of complexity, it works as described by Figure 2.9 and Equations (2.9a-e).

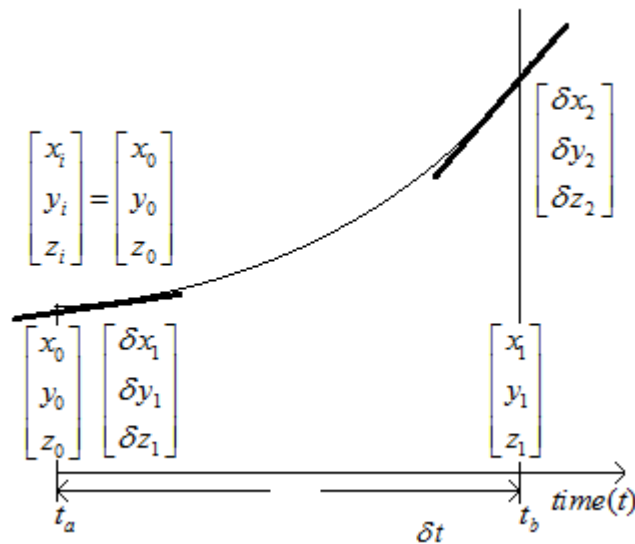


Figure 2.9 Visualization Sketch for the Modified Euler's Method.

$$\begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} = \begin{bmatrix} f_1(x_0, y_0, z_0)\delta t \\ f_2(x_0, y_0, z_0)\delta t \\ f_3(x_0, y_0, z_0)\delta t \end{bmatrix} \quad (2.9a)$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} \quad (2.9b)$$

$$\begin{bmatrix} \delta x_2 \\ \delta y_2 \\ \delta z_2 \end{bmatrix} = \begin{bmatrix} f_1(x_1, y_1, z_1)\delta t \\ f_2(x_1, y_1, z_1)\delta t \\ f_3(x_1, y_1, z_1)\delta t \end{bmatrix} \quad (2.9c)$$

$$\begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} = \frac{1}{2} \left[\begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} + \begin{bmatrix} \delta x_2 \\ \delta y_2 \\ \delta z_2 \end{bmatrix} \right] \quad (2.9d)$$

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} \quad (2.9e)$$

For each iteration, an incremental set was obtained according to the set of equations being integrated and was added to the initial values. The values obtained were then substituted back into the equations to create a second incremental set which was then averaged with the first to get a final incremental set. This set was used to obtain the next set of values in the solution array.

2.3.3 Simpson's Rule

Simpson's Rule is a Newton Cotes formula that approximates the solution to an integral using quadratic polynomials. It is a third order numerical integration method that is more complex than the Euler and Modified Euler methods, but not as complex as the Fourth Order Runge-Kutta Method. It operates as shown in Figure 2.10 and Equations (2.10a-g). For each iteration, three sets of increments are obtained successively, the first by substituting the initial values into the sets of equations, the second by adding the first increment to the initial values and substituting back into the sets of equations; and the third by adding the second to the initial values and substituting back into the sets of equations. The final sets of increments which are used to define the next values of the arrays, are obtained by adding the first increment to four times the second, and then to the last increment and dividing the sum by six.

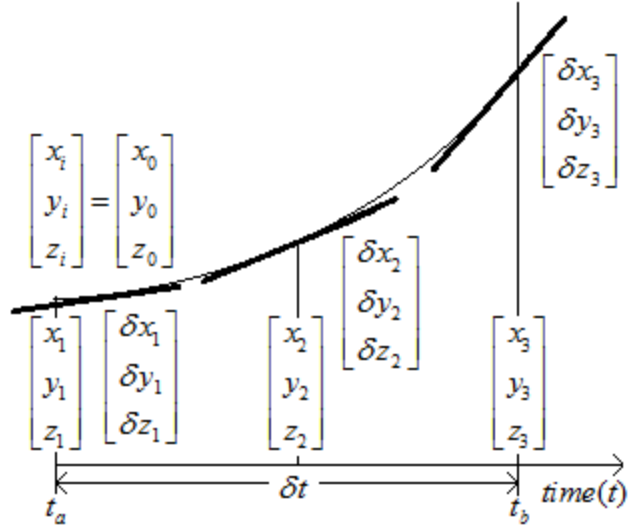


Figure 2.10 Visualization Sketch for Simpson's Rule.

$$\begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} = \begin{bmatrix} f_1(x_0, y_0, z_0) \delta t \\ f_2(x_0, y_0, z_0) \delta t \\ f_3(x_0, y_0, z_0) \delta t \end{bmatrix} \quad (2.10a)$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} \quad (2.10b)$$

$$\begin{bmatrix} \delta x_2 \\ \delta y_2 \\ \delta z_2 \end{bmatrix} = \begin{bmatrix} f_1(x_1, y_1, z_1) \delta t \\ f_2(x_1, y_1, z_1) \delta t \\ f_3(x_1, y_1, z_1) \delta t \end{bmatrix} \quad (2.10c)$$

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} \delta x_2 \\ \delta y_2 \\ \delta z_2 \end{bmatrix} \quad (2.10d)$$

$$\begin{bmatrix} \delta x_3 \\ \delta y_3 \\ \delta z_3 \end{bmatrix} = \begin{bmatrix} f_1(x_2, y_2, z_2) \delta t \\ f_2(x_2, y_2, z_2) \delta t \\ f_3(x_2, y_2, z_2) \delta t \end{bmatrix} \quad (2.10e)$$

$$\begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} = \frac{1}{6} \left(\begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} + 4 \times \begin{bmatrix} \delta x_2 \\ \delta y_2 \\ \delta z_2 \end{bmatrix} + \begin{bmatrix} \delta x_3 \\ \delta y_3 \\ \delta z_3 \end{bmatrix} \right) \quad (2.10f)$$

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} \quad (2.10 g)$$

2.3.4 The Fourth Order Runge-Kutta Method

The Fourth Order Runge-Kutta Method was the most complex of all the numerical methods used in this study. As shown in Equation (2.11a-i), four sets of increments were obtained during each iteration, the first by substituting the initial values into the sets of equations, the second by adding the initial values to one half the first set of increments, substituting back into the sets of equations; the third by adding the initial values to one half the second set of increments, then substituting back into the sets of equations and the fourth incremental by adding the initial values to the fourth set of increments, and then substituting back into the set of equations. The final sets of increments are obtained by adding the first increment to two times the sum of the second and third increments, and then to the last increment and dividing the sum by six.

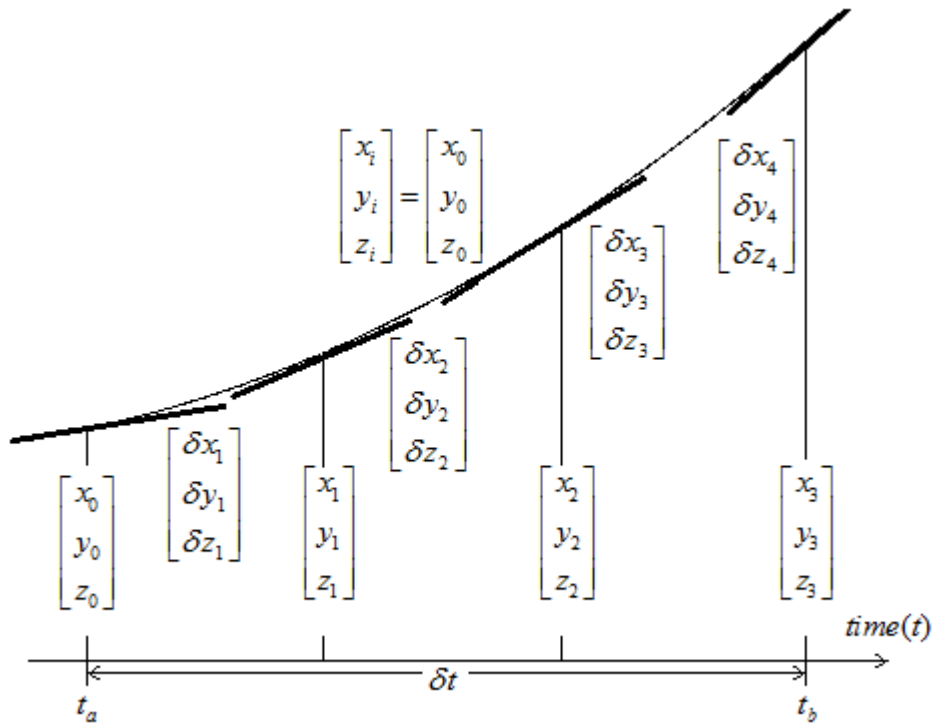


Figure 2.11 Visualization Sketch for Runge-Kutta's Method.

$$\begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} = \begin{bmatrix} f_1(x_0, y_0, z_0) \delta t \\ f_2(x_0, y_0, z_0) \delta t \\ f_3(x_0, y_0, z_0) \delta t \end{bmatrix} \quad (2.11a)$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + 0.5 \begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} \quad (2.11b)$$

$$\begin{bmatrix} \delta x_2 \\ \delta y_2 \\ \delta z_2 \end{bmatrix} = \begin{bmatrix} f_1(x_1, y_1, z_1) \delta t \\ f_2(x_1, y_1, z_1) \delta t \\ f_3(x_1, y_1, z_1) \delta t \end{bmatrix} \quad (2.11c)$$

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + 0.5 \begin{bmatrix} \delta x_2 \\ \delta y_2 \\ \delta z_2 \end{bmatrix} \quad (2.11d)$$

$$\begin{bmatrix} \delta x_3 \\ \delta y_3 \\ \delta z_3 \end{bmatrix} = \begin{bmatrix} f_1(x_2, y_2, z_2) \delta t \\ f_2(x_2, y_2, z_2) \delta t \\ f_3(x_2, y_2, z_2) \delta t \end{bmatrix} \quad (2.11e)$$

$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + 0.5 \times \begin{bmatrix} \delta x_3 \\ \delta y_3 \\ \delta z_3 \end{bmatrix} \quad (2.11 f)$$

$$\begin{bmatrix} \delta x_4 \\ \delta y_4 \\ \delta z_4 \end{bmatrix} = \begin{bmatrix} f_1(x_3, y_3, z_3) \delta t \\ f_2(x_3, y_3, z_3) \delta t \\ f_3(x_3, y_3, z_3) \delta t \end{bmatrix} \quad (2.11 g)$$

$$\begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} = \frac{1}{6} \left(\begin{bmatrix} \delta x_1 \\ \delta y_1 \\ \delta z_1 \end{bmatrix} + 2 \times \left[\begin{bmatrix} \delta x_2 \\ \delta y_2 \\ \delta z_2 \end{bmatrix} + \begin{bmatrix} \delta x_3 \\ \delta y_3 \\ \delta z_3 \end{bmatrix} \right] + \begin{bmatrix} \delta x_4 \\ \delta y_4 \\ \delta z_4 \end{bmatrix} \right) \quad (2.11 h)$$

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} \quad (2.11 i)$$

2.4 The Maximum Error Obtainable in Each Numerical Integration Method

A general equation for the integration estimate which covers the Simpson and Runge-Kutta methods is presented in Equation (2.12) [30]. Q_{mp} is the integration estimate for the Simpson and Runge-Kutta methods. A and B are constants of the method. f_i represents samples of the function to be integrated, and w_i are weight factors. The second term on the right hand side of the equation is related to the error bound.

$$Q_{mp} = A\delta t \sum_{-m/2}^{m/2} w_i f_i + B(\delta t)^{m+2} \quad (2.12)$$

For the Simpson's method, m is 1; and for the Fourth Order Runge-Kutta method, m is 2. The equation does not apply to the Euler and Modified Euler methods. The information contained in Equation (2.12) is displayed in Table 2.1. along with the data for the Euler and Modified Euler methods which was obtained directly from reference [30]. In the Table, ' \propto ' is a sign of proportionality.

Table 2.1 The error bound in each numerical method by Step Size order proportionality.

| Numerical Integration Method | Maximum Error |
|---------------------------------|------------------------|
| Euler's method | $\propto (\delta t)$ |
| Modified Euler's method | $\propto (\delta t)^2$ |
| Simpson's method | $\propto (\delta t)^3$ |
| Fourth Order Runge-Kutta method | $\propto (\delta t)^4$ |

2.5 Lyapunov Coefficients for the Chaotic Systems in this study

The value of the maximum positive Lyapunov coefficient is thought to impact the sensitivity of chaotic equations to initial conditions [31]. It is expected that the higher the coefficient the greater the sensitivity to initial conditions. This sensitivity is expected to impact the resolution parameter in the computation with more resolution being required for higher initial condition sensitivity. The observations derived from this work regarding these points will be commented on in the future work section of the conclusion.

Table 2.2 Lyapunov Coefficients for the chaotic systems tested.

| | Chaotic System | Lyapunov Coefficients [31] |
|---|-----------------------------|-----------------------------------|
| 1 | Lorenz System | 0.90, -14.57 |
| 2 | Rosler System | 0.07, -5.39 |
| 3 | Chen System | 2.03, -12.03 |
| 4 | Chua System | 0.33, -2.52 |
| 5 | Hadley System | 0.17, -4.45 |
| 6 | ACT System | 0.16, -9.21 |
| 7 | Diffusionless Lorenz System | 0.21, -1.21 |

Chapter 3

Introduction to the Micro-Integrator scheme

3.1 Overview

This chapter is a brief self-contained introduction to the Micro-Integrator scheme. To a significant degree, the introduction is based on recent work contained in references [7,33]. To illustrate the main ideas, the Lorenz chaotic partial differential equation set was numerically integrated using the Modified Euler's method.

It was observed while integrating the Lorenz and Rossler equations [33] using the Euler's method that adjusting the parameter 'N', used to set the number of iterations caused noticeable changes in the approximate solutions to the differential equations associated with these systems. Increasing it to the point of eliminating the numerical butterfly effect proved to be impractical, as the computer displayed an out of memory error at every attempt to go past a certain error level. This memory problem inspired the need for the design of the Micro-Integrator algorithm.

3.2 Integration Without the Micro-Integrator

Integration without the Micro-Integrator was done using standard numerical integration routines that can be shown with the following Figure;

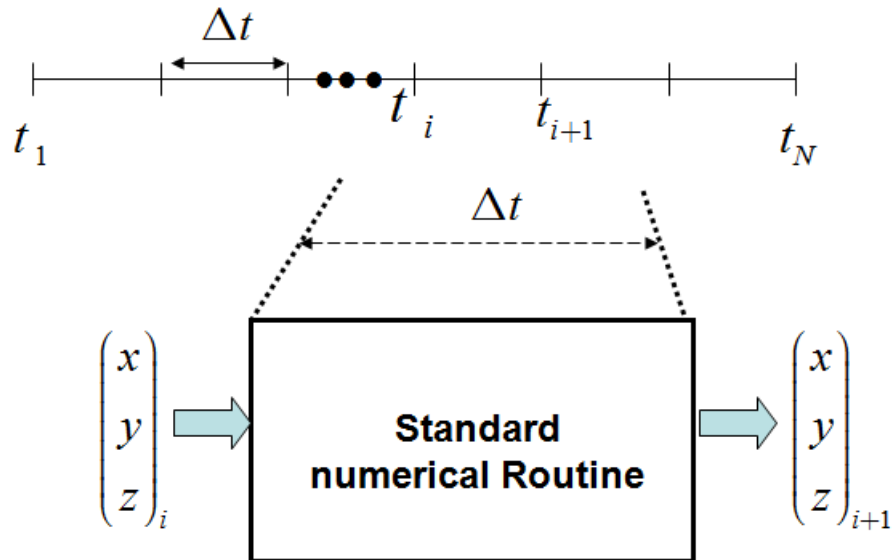


Figure 3.1 Numerical Integration Without the Micro-Integrator.

The time step Δt , is the amount of time between successive elements in the solution vector. The integration time frame T is the amount of time over which the solution array is spread. The resolution parameter N is the number of elements in the solution array. These parameters are related by the formula;

$$\Delta t = T / (N - 1) \quad (3.1)$$

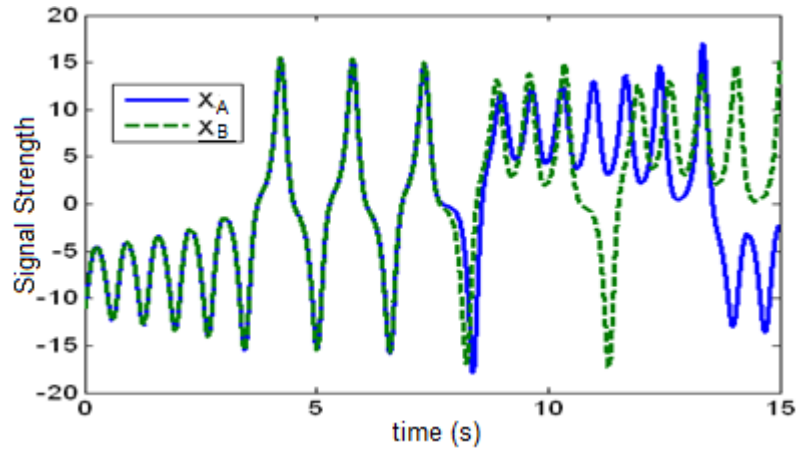


Figure 3.2 Plot of the x variable of the Lorenz equations against time for $N_A = 5,000$ and for $T = 15$ s.

Figure 3.2 was obtained by integrating the Lorenz Equations using the Modified Euler's method and plotting its x values against time. The problem of the numerical butterfly effect is visible here. It can be seen the solution does not yet exhibit the N -stable condition. The x_A, x_B solutions, corresponding to $N_A = 5,000$ and $N_B = 10,000$ respectively, start to diverge about 6 seconds into the $T=15$ second simulation. Although not shown here the N -stable convergence property is eventually achieved with the Lorenz problem for $N_A = 50,000$.

3.3 The Proposal of the Micro-Integrator Solution

The Micro-Integrator was proposed [7] as a solution to the memory problems with selecting a large resolution parameter. The tests in this section support the observations

that there are an unlimited number of discrete time solutions to differential equations with specified initial conditions. If the resolution is high enough (extremely short time step Δt), the discrete time solution becomes insensitive to the time step size and equally important indistinguishable from the unique continuous time solution.

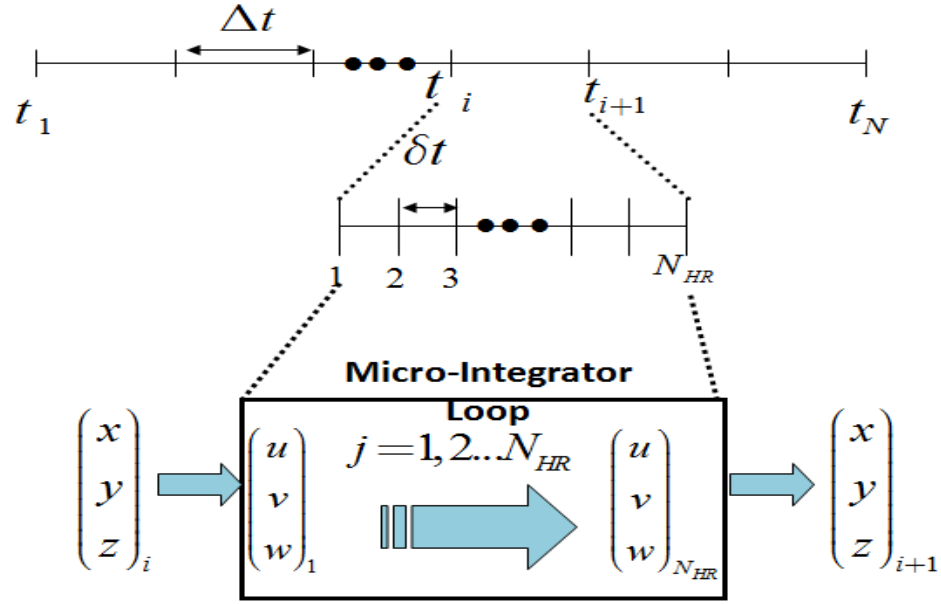


Figure 3.3 Schematic for the Micro-Integrator Concept.

Figure 3.3 briefly shows how the Micro-Integrator works. The large scale time intervals Δt and micro subintervals δt are illustrated in the top half. Given N_{HR} , a high resolution or micro-integrator parameter, which produces a partition of the interval Δt into $N_{HR} - 1$ equal segments. Along with the sub-interval spacing, it is given by:

$$\delta t = (t_{i+1} - t_i) / (N_{HR}) \quad (3.2)$$

As shown on Figure 3.3, δt is a subinterval on time intervals such as $[t_i, t_{i+1}]$. The program does not record the dynamic high resolution variable array; because of this, the memory storage requirements are not expanded. The primary objective is to more accurately propagate the time series of the dynamic variables between large-scale time

intervals $t_i \rightarrow t_{i+1}$ without high memory requirements. The Micro-Integrator scheme initially creates temporary dynamic variables (u, v, w) at the onset of the process:

$$(x, y, z)_i \rightarrow (u, v, w)_{j=1} \quad (3.3)$$

After the last j -index internal loop for the Micro-Integrator scheme as shown in Figure 3.3, the temporary dynamic variable sets are transferred back to a recorded dynamic variable set as per:

$$(u, v, w)_{j=N_{HR}} \rightarrow (x, y, z)_{i+1} \quad (3.4)$$

The Micro-integrator process is repeated for every large-scale time interval. For testing purposes the Modified Euler Method was also employed as the specific algorithm within the Micro-integrator. The Micro-Integrator temporary dynamic variable updates are distinguished with subscripts “old” and “new”. Since memory storage is not required formal indexing is obviated.

How high should the resolution parameter for the Micro-parameter, N_{HR} , be set in order to avoid the numerical butterfly effect? It will not only depend on the particular chaos system simulated but also the particular numerical integration technique selected. Therefore a straightforward numerical answer or formula is not readily available. On the other hand, there is a simple test for which the discrete time solution can be taken as a good approximation to the continuous time solution. In particular, the extended Micro-Integrator program is run initially with a resolution parameter seed value N_{HRA} and the $(x_A, y_A, z_A)_i$ $i = 1, 2, \dots, N$ solutions are recorded. Then the Micro-Integrator program is run with a new and higher resolution parameter N_{HRB} . It relates to N_{HRA} by the following formula:

$$N_{HRB} = m \times N_{HRA} \quad (3.5)$$

Where $m=2$ is a typical value. A solution $(x_B, y_B, z_B)_i, i=1, 2, \dots, N$ is then recorded. In step two a comparison is made to check if the initial low-resolution solution $(x_A, y_A, z_A)_i$ and the higher resolution solution $(x_B, y_B, z_B)_i$ have become indistinguishable. If indistinguishable then either the higher resolution B -solution $(x_B, y_B, z_B)_i$ or lower resolution A -solution would be considered a good approximation to the continuous time solution, insensitive to increases in N_{HR} , and free of the numerical butterfly effect. If A and B solutions are not close enough by some criterion then the process is repeated. In this situation the previously used higher resolution parameter N_{HRB} supplants N_{HRA} in the next test cycle. As described, this will require multiple pairs of program runs (N_{HRA}, N_{HRB}) for repeated test until the criterion that the two solutions are indistinguishable is met.

3.4 Calculation of the Error

The error is calculated as the difference between the computed chaotic signal x_A, y_A, z_A using a resolution of N_A and x_B, y_B, z_B using a resolution of $N_B = 2 \times N_A$.

$$d\{x_A, x_B\} = \frac{\frac{1}{N} \sum_{i=1}^N |x_A - x_B|}{\max\{|x_{A_1}|, \dots, |x_{A_N}|, |x_{B_1}|, \dots, |x_{B_N}|\}} \quad (3.6)$$

$$\begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{pmatrix} \equiv \begin{pmatrix} d\{x_A, x_B\} \\ d\{y_A, y_B\} \\ d\{z_A, z_B\} \end{pmatrix} \quad (3.7)$$

The terms $d\{x_A, x_B\}$, $d\{y_A, y_B\}$, $d\{z_A, z_B\}$ each represents a distance metric for the vectors x_A and x_B , y_A and y_B , z_A and z_B , respectively. The maximum error is obtained by taking the largest value from among the distance them as follows.

$$\varepsilon_{\max} \equiv \max \{ \varepsilon_x, \varepsilon_y, \varepsilon_z \} \quad (3.8)$$

ε_{\max} is then compared to an upper-bound error E which is one of the program inputs.

$$\text{If } \varepsilon_{\max} \leq E \text{ solution approximately free of ButterFly effect} \quad (3.9a)$$

$$\text{If } \varepsilon_{\max} \geq E \text{ Continue increasing resolution} \quad (3.9b)$$

3.4.1 Sample Error Calculation, In the case of the Lorenz Without the Micro-Integrator computed using the Fourth Order Runge-Kutta Method.

$$d \{x_A, x_B\} = \frac{0.0855}{17.2281} = 4.96 \times 10^{-3}$$

$$d \{y_A, y_B\} = \frac{0.1230}{22.7455} = 5.41 \times 10^{-3}$$

$$d \{z_A, z_B\} = \frac{0.1502}{43.1996} = 3.48 \times 10^{-3}$$

$$\begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{pmatrix} \equiv \begin{pmatrix} d \{x_A, x_B\} \\ d \{y_A, y_B\} \\ d \{z_A, z_B\} \end{pmatrix} \equiv \begin{pmatrix} 4.96 \times 10^{-3} \\ 5.41 \times 10^{-3} \\ 3.48 \times 10^{-3} \end{pmatrix}$$

$$\varepsilon_{\max} \equiv \max \{ \varepsilon_x, \varepsilon_y, \varepsilon_z \} \equiv \max \{ 4.96 \times 10^{-3}, 5.41 \times 10^{-3}, 3.48 \times 10^{-3} \}$$

$$\varepsilon_{\max} = 5.41 \times 10^{-3}$$

This answer can be verified from Table 5.1.

3.5 A Detailed Description of the Micro-Integrator

The flow chart process for the automated adaptive Micro-Integrator algorithm is shown in Figure 3.4 to clarify how these concepts are brought together. Besides initialization of the dynamic variables, the inputs to the algorithm include the large scale resolution parameter N , the error bound E , and a seed value for the Micro-Integrator resolution parameter N_{HRO} . The system description and initial conditions were not adjusted in the tests discussed and therefore not explicitly mentioned as inputs for the

algorithm. If the error test does not meet the required condition, the Micro-Integrator resolution parameter is increased by a factor ‘ m ’ then repeated. Once the error is satisfied, the automated process is stopped and the current values of all the resolution parameters are fixed.

N_{HRO} and N_{res_I} are used interchangeably here, they represent the initial values of the Micro-Integrator's resolution parameter N_{HRA} . Also, N_{HRA} and N_{res_A} are used interchangeably, they represent the real-time values of the Micro-Integrator's resolution parameter.

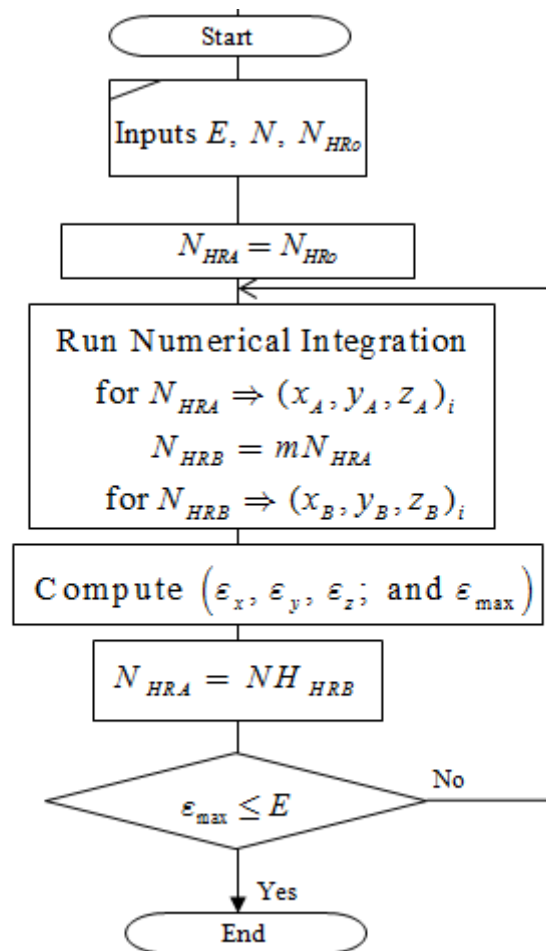


Figure 3.4, Automated Adaptive Extended Micro-Integrator Algorithm, typical $N_{HRO} = 2$,

$m = 2, E = 0.01$.

3.6 Testing the Micro-Integrator Algorithm

The test performed demonstrated that the Micro-Integrator process allowed a N -stable numerical solution. The Lorenz Equations were integrated using the Modified Euler's Method and a resolution parameter of $N_A = 5,000$. The algorithm described proceeds until $\varepsilon_{\max} \leq E = 0.01$. Figure 3.5 is a plot of the vector x obtained from the integration process. Convergence was obtained with only memory storage requirement associated with a (x, y, z) array size of 5000, not 50,000. This is a very good approximation for the exact continuous time solution vector to the Lorenz Equations. The error criterion was reached using a Micro-Integrator resolution parameter of $N_{res_A} = 16$.

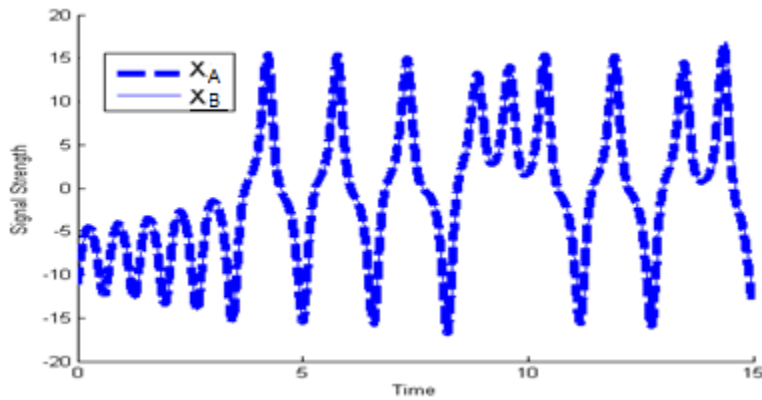


Figure 3.5 Convergence demonstration with Micro-Integrator algorithm, $N_A = 5000$

$$N_{HRO} = 2, m = 2, E = 0.01, T = 15s.$$

It was found that the modified-Euler's Method of numerical integration worked very well with the proposed Micro-Integrator algorithm. Hence it was possible to achieve convergence without excessive use of computer memory.

Chapter Four

Methodology for the Evaluation of the Performance factor of the Micro-Integrator Scheme

4.1 The Micro-Integrator Performance Factor

In order to gauge the utility of the Micro-Integrator scheme, a performance factor was introduced. This performance factor was defined as the ratio of memory requirements as defined by:

$$\eta = \frac{N_2}{N_1} \quad (4.1)$$

Where η represents the Micro-Integrator performance factor, N_1 and N_2 are the number of sampling points (N_A values) used when computing the chaotic signal with and without the Micro-Integrator, respectively. The values N_1 and N_2 are used as measures of the amount of memory required in each case.

4.1.1 The Memory Estimation Disclaimer

It is very important to state here that estimating the amount of memory required by the computer by measuring the size of its output array as was done in this study is accurate only to a certain extent. The actual memory used would be slightly different from this value. But for the purpose of this research, it was used as a good estimate.

4.2 Limiting the Error

In order to compare performance factors with various chaotic systems and numerical methods, it is required that the error ε_{\max} be reached within a band centered on the selected \mathcal{E} .

$$E\left(1 - \frac{\delta\varepsilon\%}{100}\right) \leq \varepsilon_{\max} \leq E\left(1 + \frac{\delta\varepsilon\%}{100}\right) \quad (4.2)$$

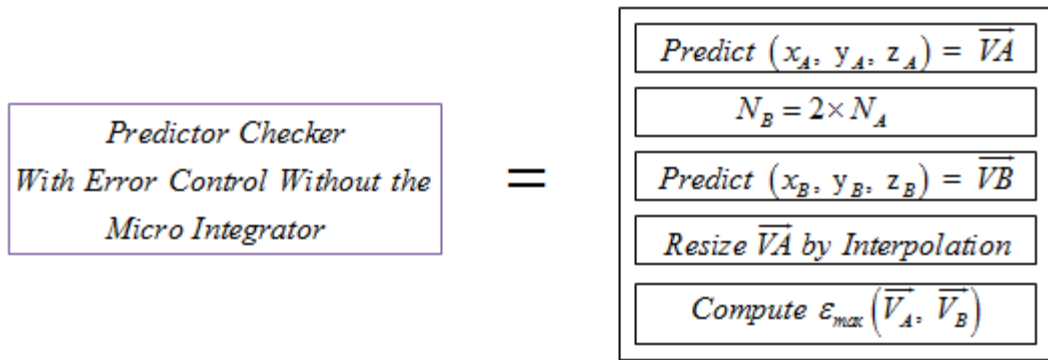
Where the percentage deviation allowed $\delta\varepsilon$ is specified as an error control computer program input parameter. A typical value for this in the numerical tests is 10%. Error control algorithms were designed to limit the computational error to within a fraction of a specified error limit. There were two versions, one for use with and the other for use without the Micro-Integrator. They were designed to ensure a more accurate comparison between computations with and without the Micro-Integrator.

4.3 Error Control Without the Micro-Integrator

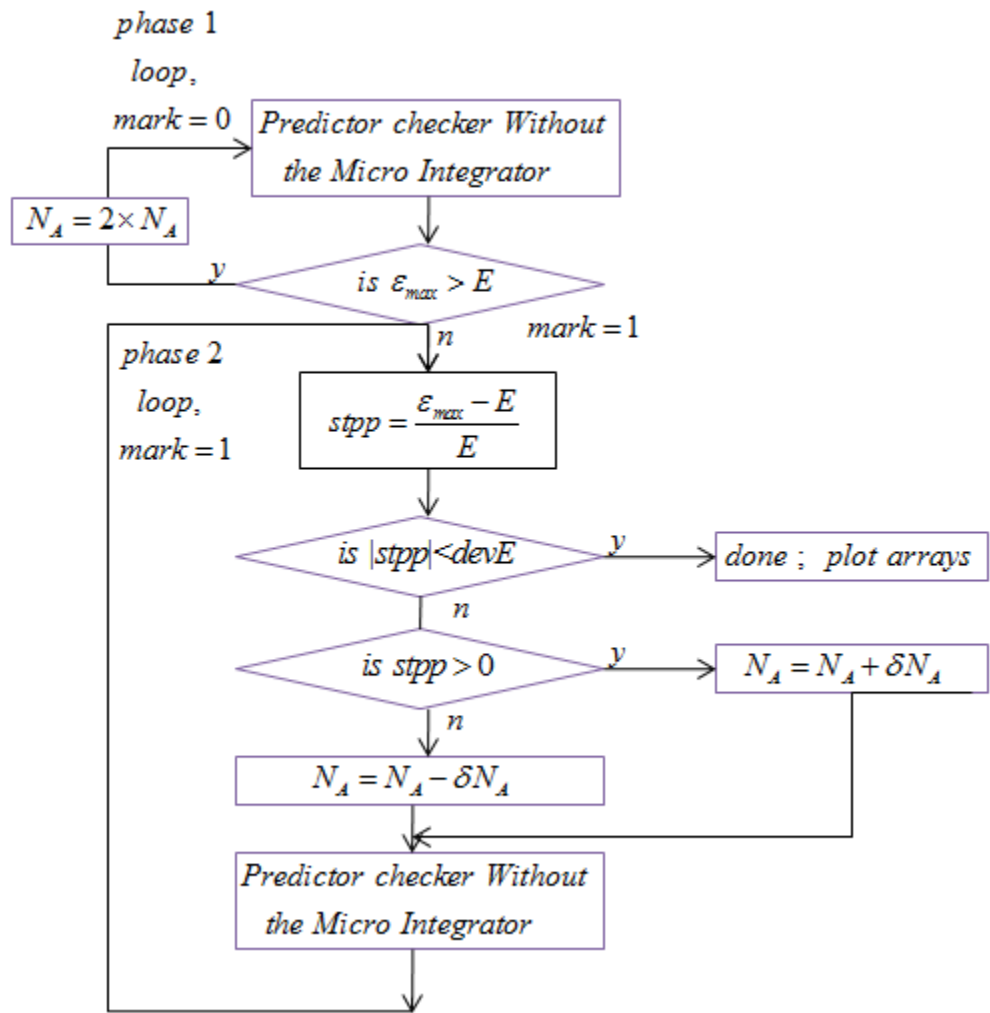
4.3.1 Algorithm for the Error Control Without the Micro-Integrator

Figure 4.1(a) is a predictor checker algorithm used to simplify the representation of the error control algorithm without the Micro-Integrator. It operates in five stages; the first stage integrates the set of equations using the input resolution parameter N_A and produces a vector \overline{VA} which is a set made up of x_A, y_A, z_A arrays. The second stage calculates the resolution parameter N_B by qdoubling N_A . The third stage integrates the system of equations using the resolution parameter N_B and produces a vector \overline{VB} which is a set made up of x_B, y_B, z_B arrays. At this stage, \overline{VB} is twice the size of \overline{VA} , the fourth stage uses an interpolation algorithm to double the size of \overline{VA} so that it can be accurately compared to \overline{VB} . In the fifth stage, the error ε_{max} between \overline{VA} and \overline{VB} is calculated using the error calculation process explained in chapter 3.

In Figure 4.1(b), the predictor checker algorithm is used twice in a flow chart representation of the error control algorithm. First, the program inputs are fed into the predictor checker algorithm. After the error ε_{max} between \overline{VA} and \overline{VB} has been calculated by the predictor checker algorithm, it is then compared to the set error limit 'E'. If ε_{max} is greater than E, then N_A is doubled, and the value obtained is used to replace its current value, the predictor checker stage is then repeated until ε_{max} becomes less than E. *stpp'* which is the deviation of ε_{max} from E is then calculated by dividing the absolute value of the difference between ε_{max} and E by E. The absolute value of '*stpp'*' is then compared to the maximum allowable deviation *devE*.



(a)



(b)

Figure 4.1: (a) Predictor Checker Algorithm Without the Micro-Integrator, (b) Error Control Algorithm Without the Micro-Integrator using the Predictor Checker Algorithm.

If '*stpp*' is greater than *devE*, a check is then performed to see if it is greater than zero. If it is greater than zero, N_A is then incremented by a fraction of its current value defined by a parameter *flucN_A* represented by the symbol δ in the flow chart. The value of N_A obtained is then fed back into the predictor checker, after which the program returns to the stage where '*stpp*' is being calculated. If '*stpp*' is less than zero, N_A is then decreased by *flucN_A*, the value of N_A obtained is then fed back into the predictor checker algorithm and the program returns to the stage where '*stpp*' is being calculated. This goes on until '*stpp*' becomes less than *devE*, when this happens, then the program is terminated and the current values of x_A and x_B are plotted against time.

4.3.2 Alternate Depiction of the Error Control Algorithm Without the Micro-Integrator

As shown in Figure 4.2, the error control algorithm for the case without the Micro-Integrator was designed to be fed the following input parameters: '*E*' the desired error limit, ' N_A ' the initial value of the resolution parameter, '*flucN_A*' the percentage fluctuation of N_A for each iteration, '*devE*' the maximum allowable percentage error deviation of the results from the desired error limit; and *kloopMax* the maximum number of iterations. For each iteration, the values x_A , y_A ; and z_A , are computed using the input resolution parameter N_A as the size of the solution array and to compute Δt . The values x_B , y_B ; and z_B are computed using $2 \times N_A$ in place of N_A . Due to the size differences in the *A* and *B* arrays, an interpolation algorithm had to be introduced to double the size of the array *A* to make it comparable to *B*. The error calculation process as explained in chapter 3 is then applied to calculate the error ' E_1 ' after which its deviation from the desired error limit '*stpp*' is calculated by subtracting the error E_1 from the error limit *E* and dividing the difference by E_1 . The deviation *stpp* is then compared to the maximum allowable error deviation *devE*; if *stpp* is greater than *devE*, then the deviation in the error is too large, the algorithm then performs a check to see if *stpp* is greater than zero, if *stpp* is greater than zero then the value of N_A used in the present iteration is added to its product with *flucN_A* rounded to the nearest significant Figure. The result obtained is

used as the value of N_A for the next iteration. If $stpp$ is less than zero, then the present N_A 's product with $flucN_A$ rounded to the nearest significant figure is subtracted from it and the result obtained is used as the value of N_A for the next iteration.

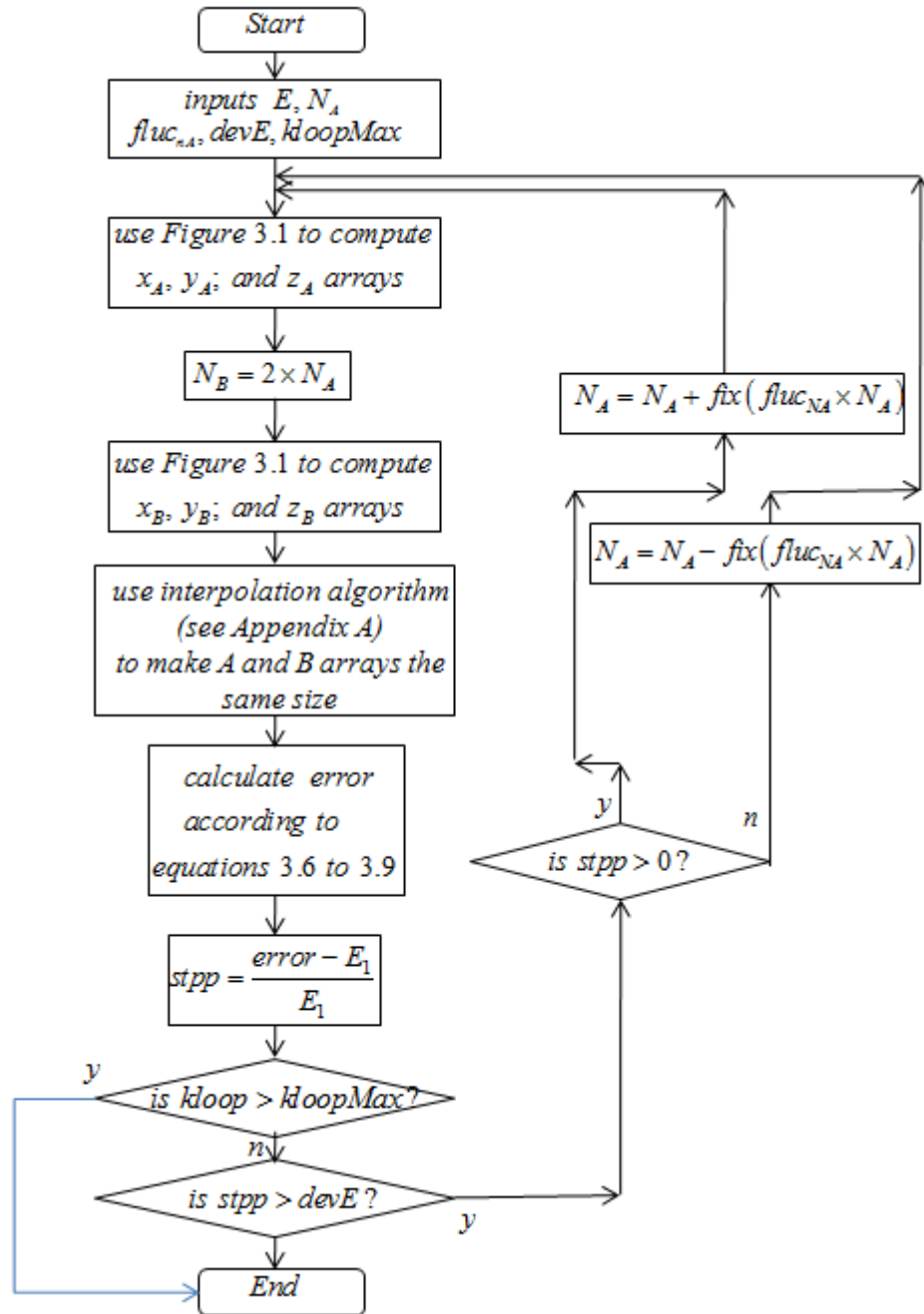


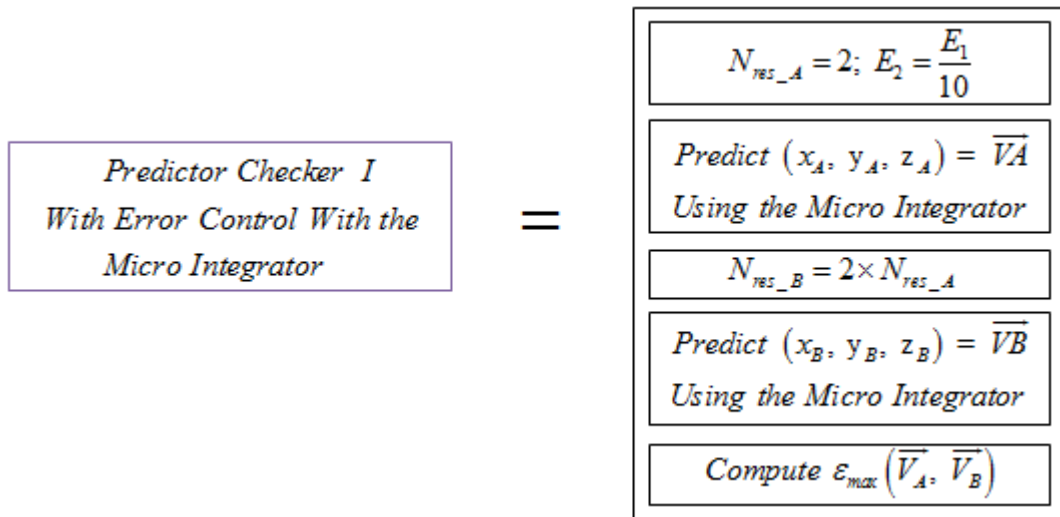
Figure 4.2 Alternate Depiction of the Error Control Algorithm without the Micro-Integrator.

This goes on until $stpp$ is less than $devE$, or when the failsafe is triggered. The failsafe is triggered when $kloop$ becomes greater than $kloopMax$. The values of the output vectors are then plotted against time which in this case is a discrete set of elements increasing by unity from 1 to $N_A - 1$ for the A arrays and from 1 to $N_B - 1$ for the B arrays.

4.4 Error Control With the Micro-Integrator

4.4.1 Algorithm for the error control with the Micro-Integrator

Figure 4.3(a) is the first predictor checker algorithm used in the error control algorithm with the Micro-Integrator. It works in five stages, in the first stage, two input parameters N_{res_A} (the initial value of the Micro-Integrator's resolution parameter) and E_1 (the error limit) are fed into the program, and E_2 is defined by dividing E_1 by ten. In the second stage, the set of equations is integrated using a fixed conventional resolution parameter N_A along with the input value of the Micro-integrator's resolution parameter N_{res_A} . In the third stage, a second Micro-Integrator resolution parameter N_{res_B} is calculated by doubling N_{res_A} .



(a)

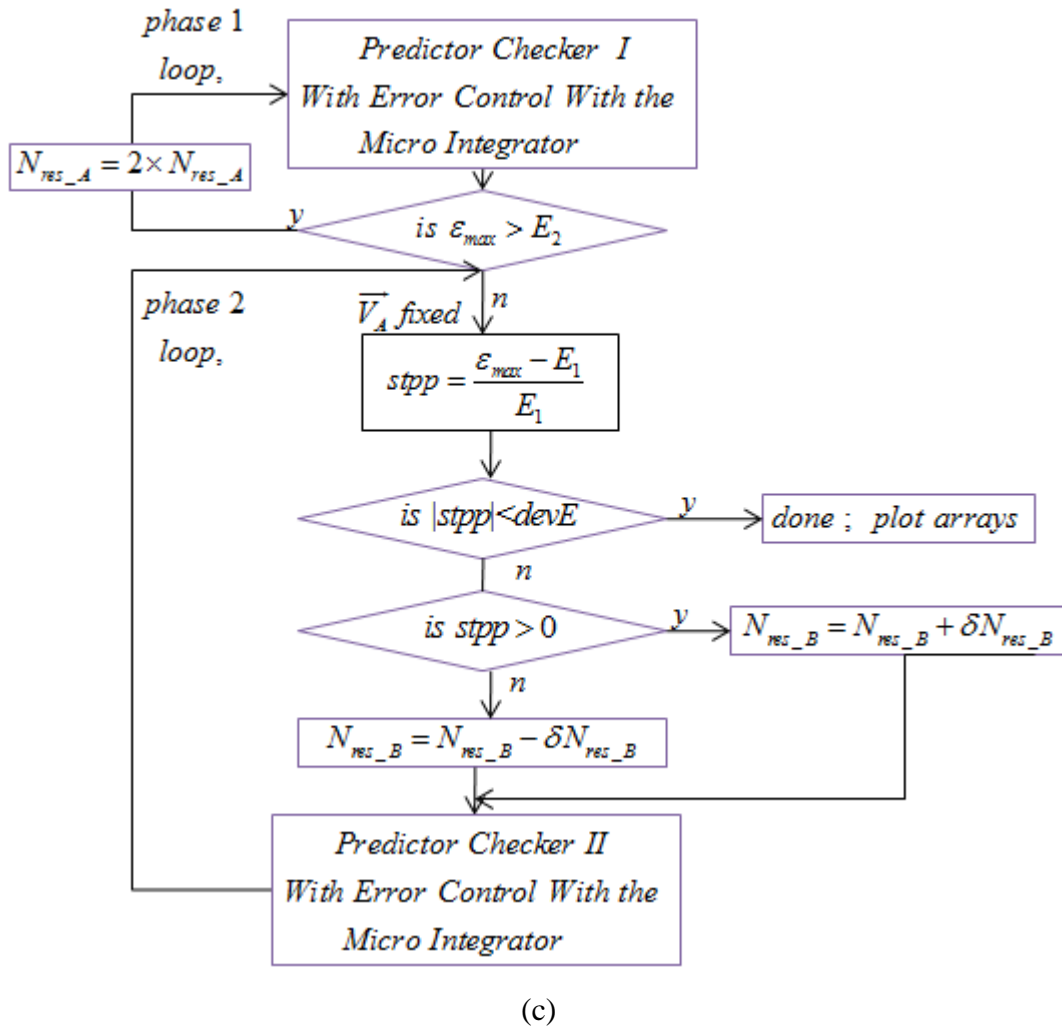
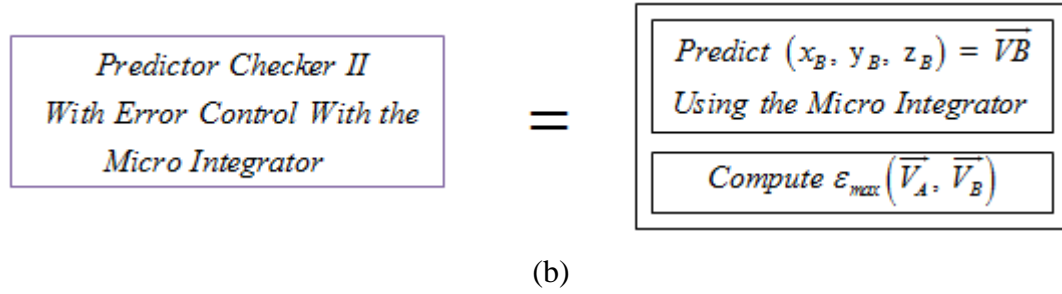


Figure 4.3: (a) First Predictor Checker Algorithm With the Micro-Integrator, (b) Second Predictor Checker Algorithm With the Micro-Integrator, (c) Error Control Algorithm With the Micro-Integrator using the Predictor Checker Algorithm.

In the fourth stage, the set of equations is integrated using the same fixed conventional resolution parameter N_A along with the newly calculated value of the Micro-integrator's resolution parameter N_{res_B} . In the fifth stage, the error ε_{max} between \overline{VA} and \overline{VB} is calculated using the error calculation process explained in chapter 3. There is no need for interpolation here because \overline{VA} and \overline{VB} are the same size.

Figure 4.3(b) is the second predictor checker algorithm used in the error control algorithm with the Micro-Integrator. It is used at a point where the \overline{VA} array is fixed. It simply integrates the equation set to obtain the \overline{VB} array and calculates the error between it and the fixed \overline{VA} array.

The complete error control algorithm is shown in Figure 4.3(c). At the start of the program, its inputs are fed into the first predictor checker algorithm. After the error ε_{max} between \overline{VA} and \overline{VB} has been calculated by the predictor checker algorithm, it is then compared to E_2 . If ε_{max} is greater than E_2 , then N_{res_A} is doubled, and the value obtained is used to replace its current value, the predictor checker stage is then repeated until ε_{max} becomes less than E_2 . '*stpp*' which is the deviation of ε_{max} from E_2 is then calculated by dividing the absolute value of its difference from ε_{max} by itself. The absolute value of '*stpp*' is then compared to the maximum allowable deviation $devE$. If '*stpp*' is greater than $devE$, a check is then performed to see if it is greater than zero. If it is greater than zero, N_{res_A} is then incremented by a fraction of its current value defined by a parameter $flucN_{res_A}$ represented by the symbol δ in the flow chart. The value of N_{res_A} obtained is then fed to the second predictor checker, after which the program returns to the stage where '*stpp*' is being calculated. If '*stpp*' is less than zero, N_{res_A} is then decreased by $flucN_{res_A}$, the value of N_{res_A} obtained is then fed to the second predictor checker algorithm and the program returns to the stage where '*stpp*' is being calculated. This goes on until '*stpp*' becomes less than $devE$, when this happens, then the program is terminated and the current values of x_A and x_B are plotted against a vector of elements increasing by 1 from 0 to $N_A - 1$.

4.4.2 Alternate Depiction of the Algorithm for the error control with the Micro-Integrator

As shown in Figure 4.4, the error control algorithm for the case with the Micro-Integrator was designed to be fed the following input parameters: ' E_1 ' the error limit, ' N_A ' the standard resolution parameter, ' N_{resI} ' the initial value of the Micro-Integrator resolution parameter ' $fluc_{N_{resA}}$ ' which is the fluctuation per iteration of N_{resA} (the Micro-Integrator's resolution parameter), ' $devE$ ' the maximum allowable deviation from the error limit; and $kloopMax$, the maximum number of iterations. First, ' E_2 ' is defined as one tenth the value of E_1 . Then x_A , y_A ; and z_A are computed using Figure 3.3 (the basic Micro-Integrator algorithm). The Micro-Integrator resolution parameter is then doubled and the obtained value is used to compute x_B , y_B , and z_B arrays. The error is then computed as described in Chapter 3, and compared to E_2 . If it is greater than E_2 , then the Micro-Integrator's resolution parameter is doubled and the computation is re-done (using the doubled Micro-Integrator resolution parameter). If it is less than E_2 , then its B arrays are stored as a reference to be used in the second phase of the algorithm. The second phase of the algorithm uses in the stored B array and the Micro-Integrator's resolution parameter. It computes a new set of A arrays which are then compared to the stored arrays from the previous stage of computation and calculates the error. The deviation $stpp$ is then calculated by subtracting the error limit from the error just calculated and dividing the difference by the error limit. The result is then compared to $devE$. If $stpp$ is greater than $devE$, then a second check is performed to see if $stpp$ is greater than zero. If $stpp$ is greater than zero, if $stpp$ is greater than zero then the value of N_{res_A} used in the present iteration is added to its product with $fluc_{N_{res_A}}$ rounded to the nearest significant figure, and the result obtained is used as the new value of N_{res_A} while the A arrays are re-computed. If $stpp$ is less than zero, then the present N_{res_A} 's product with $fluc_{N_{res_A}}$ rounded to the nearest significant Figure is subtracted from it and the result obtained is used as the new value of N_{res_A} for the next iteration. This goes on until $stpp$ is less than $devE$, or when the failsafe is triggered. The failsafe is triggered when $kloop$ becomes greater than $kloopMax$. The output is then plotted.

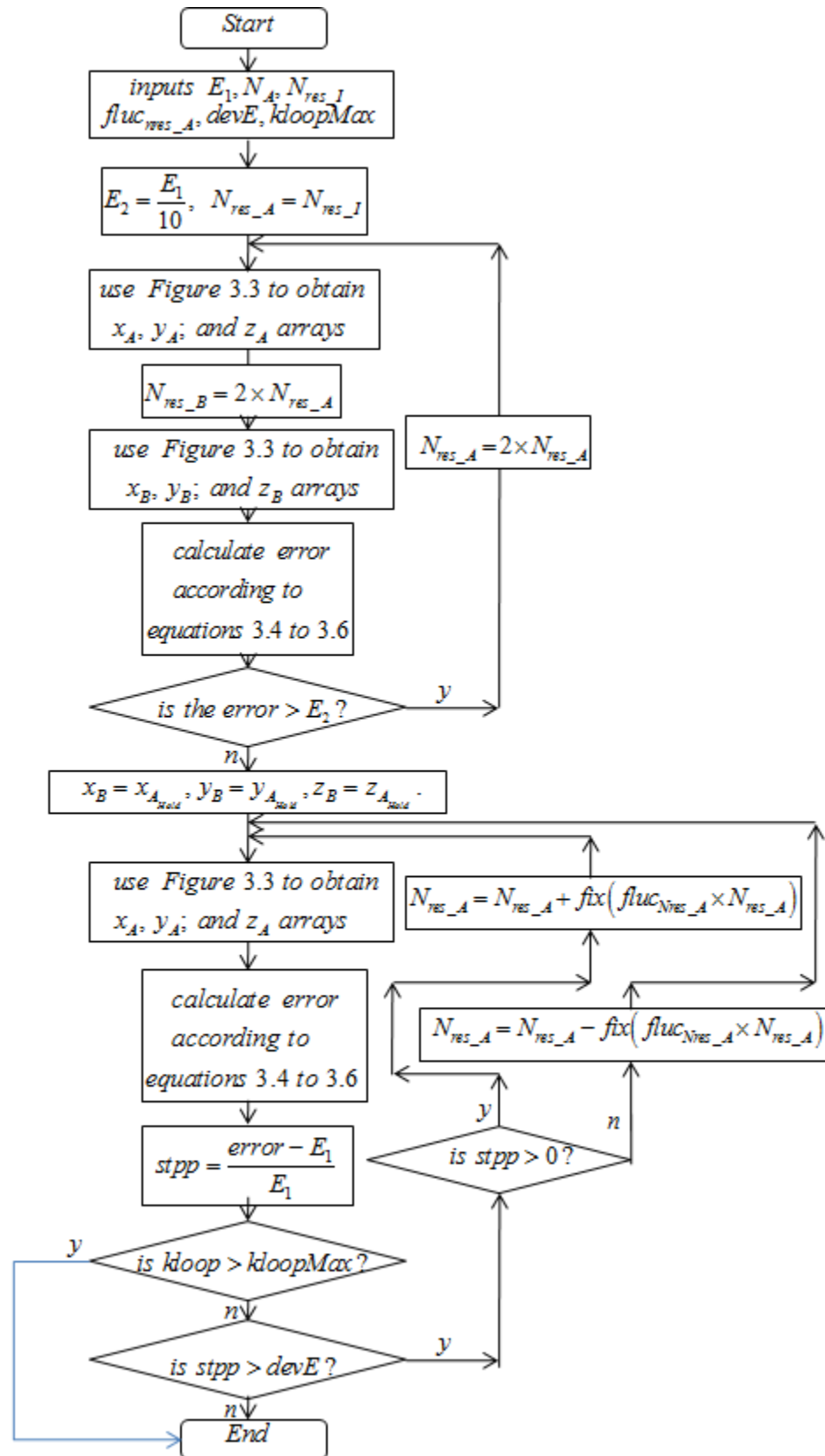


Figure 4.4 The error control algorithm with the Micro-Integrator.

The interpolation algorithm isn't needed in the computations with the Micro-Integrator because, the A and B arrays are always the same size.

4.5 Other Parameters Introduced

4.3.1 The Performance Measures

The Performance Measures are the logarithm to base ten of the Micro-Integrator performance factors of each of the chaotic systems of equations being integrated.

4.3.2 Percentage Reduction in CPU time

The percentage reduction in CPU time is defined by this formula:

$$\rho = \frac{t_1 - t_2}{t_1} \times 100\% \quad (4.3)$$

Where ρ is the percentage reduction in CPU time, t_1 is the CPU time required for integrating without using the Micro-Integrator; and t_2 is the CPU time required for integrating with the Micro-Integrator.

4.3.3 The Performance Ratio

The Performance ratio was introduced while studying a single Chaotic System of Equations being integrated twice for two different time frames; first with a low time frame; and then with a higher time frame. It could be represented by this formula:

$$r = \frac{\eta_{Hi}}{\eta_{Lo}} \quad (4.4)$$

Where r is the Performance Ratio, η_{Hi} is the Micro-Integrator performance factor obtained from integrating with the higher time frame; and η_{Lo} is the Micro-Integrator performance factor obtained from integrating with the lower time frame.

4.6 Problems with the Modular Programming approach

A complete modular approach was attempted for designing program codes for these algorithms for ease of implementation and debugging. Each chaotic differential stage was manually put into a custom function, so that in the integration stage; the differentials could be done by recalling the assigned functions. There were certain advantages gained by using this approach, but there were more disadvantages. The advantages were: relative ease of debugging as opposed to the non modular approach, little chances of introducing error, ease of understanding (for someone new to the code), ease of implementation; less bulky codes (than their non-modular versions). There was one big disadvantage with the modular programming approach in this experiment, the modular programs took too much more time to run than their non modular equivalents. This overhead in time came from the fact that the modular programs had to recall specific assigned functions (subroutines) stored separately within the computer. This disadvantage was enough to discourage further use of the modular approach in this study, as this research is directed towards computational efficiency in terms of memory and CPU time.

Chapter 5

Test Cases and Results

The purpose of this Chapter is to quantify the reduction in memory requirements from using the Micro-Integrator in terms of the performance factor introduced in Chapter 4. The result tables showing the integration input parameters, results, and output graphs from solving the Lorenz, Rossler, Chen, Chua, Hadley, ACT, and Diffusionless Lorenz chaotic partial differential equations, with and without the Micro-Integrator, are presented here.

5.1 A Brief Description of the Parameters used in the Tables

(1) Integration Time Frame

This is the difference between the initial and final time of the integration operation.

(2) Error Limit

This is the desired error limit. The true error varies within a set percentage of this.

(3) True Error

This is the actual error measured during the computational process.

(4) CPU Time

This is the time taken for the program to execute the integration process.

(5) Loop Count

This is the number of iterations the program takes to satisfy the error criteria.

All the parameters not described in this section were described in chapter 4.

5.2 Result Tables

This section contains all the result tables. For Tables 5.1 to 5.7, each Table represents a specific Chaotic System being integrated with and without the Micro-Integrator respectively using the four different numerical methods. The central bold columns containing N_2 and N_1 highlight the advantage in using the Micro-Integrator. The N_2 column displays the amount of memory required for integrating without the Micro-Integrator for each numerical method, while the N_1 column displays those required for integrating with the Micro-Integrator.

Table 5.1 Results from the Lorenz Equations with and without the Micro-Integrator.

| Lorenz System | Without the Micro-Integrator | | | | With the Micro-Integrator | | | | | |
|---|-------------------------------------|--------------------|-----------------------------|---------|--|-----------------------------|--------------------|------------|--------------|--------------------|
| | Seed $N_A=10^4$, fluc $N_A=30\%$, | | | | Seed $N_{res_A}=2$, fluc $N_{res_A}=30\%$ | | | | | |
| Integration time frame = 0-22 seconds, Set error limit = $0.005 \pm 10\%$ | | | | | | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Modified Euler | 20 | 87s | 4.59 | 3626132 | 1024 | 4.60 | 17s | 14+4 | 3935 | 3.54×10^3 |
| Simpson | 14 | 19s | 4.80 | 1946160 | 1024 | 4.75 | 13s | 13+7 | 2328 | 1.90×10^3 |
| Runge-Kutta | 13 | 18s | 5.41 | 10541 | 1024 | 5.49 | 1.2s | 5+3 | 12 | 10.29 |

Table 5.2 Results from the Rossler equations with and without the Micro-Integrator.

| Rossler System | Without the Micro-Integrator | | | | With the Micro-Integrator | | | | | |
|--|-------------------------------------|--------------------|-----------------------------|--------|--|-----------------------------|--------------------|------------|--------------|--------------------|
| | Seed $N_A=4000$, fluc $N_A=30\%$, | | | | Seed $N_{res_A}=2$, fluc $N_{res_A}=30\%$ | | | | | |
| Integration time frame = 0-200 seconds, Set error limit = $0.005 \pm 10\%$ | | | | | | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Modified Euler | 12 | 3.9s | 5.24 | 345984 | 1024 | 4.80 | 3.3s | 11+4 | 493 | 3.38×10^2 |
| Simpson's | 12 | 3.8s | 4.86 | 345984 | 1024 | 4.63 | 3.2s | 11+4 | 493 | 3.38×10^2 |
| Runge-Kutta | 8 | 1.3s | 5.33 | 68343 | 1024 | 4.98 | 2.2s | 8+5 | 81 | 66.74 |

Table 5.3 Results from the Chen equations with and without the Micro-Integrator.

| Chen System | Without the Micro-Integrator | | | | With the Micro-Integrator | | | | | |
|--|-------------------------------------|--------------------|-----------------------------|---------|--|-----------------------------|--------------------|------------|--------------|--------------------|
| | Seed $N_A=5000$, fluc $N_A=40\%$, | | | | Seed $N_{res_A}=2$, fluc $N_{res_A}=30\%$ | | | | | |
| Integration time frame = 0-8 seconds, Set error limit = $0.005 \pm 10\%$ | | | | | | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Modified Euler | 18 | 36s | 4.72 | 1839110 | 1024 | 4.61 | 10s | 13+16 | 2031 | 1.80×10^3 |
| Simpson | 13 | 7.5s | 5.12 | 648717 | 1024 | 4.58 | 3s | 11+2 | 738 | 6.34×10^2 |
| Runge-Kutta | 2 | 0.4s | 5.00 | 7500 | 1024 | 4.97 | 1.3s | 4+5 | 7 | 7.32 |

Table 5.4 Results from the Chua equations with and without the Micro-Integrator.

| Chua System | Without the Micro-Integrator | | | | With the Micro-Integrator | | | | | |
|---|-------------------------------------|--------------------|-----------------------------|-------|--|-----------------------------|--------------------|------------|--------------|--------|
| | Seed $N_A=5000$, fluc $N_A=40\%$, | | | | Seed $N_{res_A}=2$, fluc $N_{res_A}=30\%$ | | | | | |
| Integration time frame = 0-50 seconds, Set error limit = $0.005 \pm 10\%$ | | | | | | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Modified Euler | 20 | 13s | 4.92 | 80551 | 1024 | 5.19 | 1.5s | 6+9 | 49 | 78.66 |
| Simpson | 13 | 3.8s | 4.97 | 78210 | 1024 | 5.41 | 1.8s | 7+5 | 58 | 76.38 |
| Runge-Kutta | 4 | 1.3s | 5.21 | 46875 | 1024 | 4.64 | 1.9s | 8+13 | 59 | 45.78 |

Table 5.5 Results from the Hadley equations with and without the Micro-Integrator.

| Hadley System | Without the Micro-Integrator | | | | With the Micro-Integrator | | | | | |
|---|-------------------------------------|--------------------|-----------------------------|---------|--|-----------------------------|--------------------|------------|--------------|--------------------|
| | Seed $N_A=10^4$, fluc $N_A=30\%$, | | | | Seed $N_{res_A}=2$, fluc $N_{res_A}=30\%$ | | | | | |
| Integration time frame = 0-80 seconds, Set error limit = $0.005 \pm 10\%$ | | | | | | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Modified Euler | 14 | 24s | 5.31 | 1946160 | 1024 | 5.25 | 15s | 14+5 | 2755 | 1.90×10^3 |
| Simpson | 14 | 22s | 4.89 | 1946160 | 1024 | 4.57 | 9s | 13+5 | 2558 | 1.90×10^3 |
| Runge-Kutta | 10 | 4s | 5.45 | 155478 | 1024 | 4.73 | 1.2s | 6+13 | 26 | 1.52×10^2 |

Table 5.6 Results from the ACT equations with and without the Micro-Integrator.

| ACT System | Without the Micro-Integrator | | | | With the Micro-Integrator | | | | | |
|---|---|--------------------|-----------------------------|---------|--|-----------------------------|--------------------|------------|--------------|--------------------|
| | Seed $N_A=10^3$ to 10^4 , fluc $N_A=40\%$, | | | | Seed $N_{res_A}=2$, fluc $N_{res_A}=30\%$ | | | | | |
| Integration time frame = 0-80 seconds, Set error limit = $0.005 \pm 10\%$ | | | | | | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Modified Euler | 19 | 106s | 5.22 | 2059814 | 1024 | 5.30 | 67s | 13+7 | 2328 | 2.01×10^3 |
| Simpson | 19 | 93s | 4.55 | 1922492 | 1024 | 4.84 | 95s | 13+9 | 2119 | 1.88×10^3 |
| Runge-Kutta | 17 | 23s | 4.69 | 11110 | 1024 | 5.41 | 1.6s | 5+7 | 10 | 10.85 |

Table 5.7 Results from the Diffusionless Lorenz with and without the Micro-Integrator.

| Diffusionless Lorenz System | Without the Micro-Integrator | | | | With the Micro-Integrator | | | | | |
|--|-------------------------------------|--------------------|-----------------------------|--------|--|-----------------------------|--------------------|------------|--------------|--------------------|
| | Seed $N_A=5000$, fluc $N_A=30\%$, | | | | Seed $N_{res_A}=2$, fluc $N_{res_A}=30\%$ | | | | | |
| Integration time frame = 0-100 seconds, Set error limit = $0.005 \pm 10\%$ | | | | | | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Modified Euler | 23 | 23s | 4.60 | 633032 | 1024 | 4.61 | 2.5s | 11+3 | 703 | 6.18×10^2 |
| Simpson | 20 | 15s | 4.52 | 538295 | 1024 | 4.83 | 3.6s | 11+7 | 582 | 5.26×10^2 |
| Runge-Kutta | 6 | 0.5s | 5.04 | 5292 | 1024 | 5.07 | 1.9s | 4+4 | 5 | 5.17 |

Tables 5.8 and 5.9 show the results from integrating the seven chaotic systems using Euler's method. These results were kept separate from the other results because Euler's method was far less efficient than the other methods and they couldn't be compared on the same integration time frame. The difference between Tables 5.8 and 5.9 is that Table 5.9 contains values recorded for integrations using a larger time frame than Table 5.8. It is observable that the Micro-Integrator performance factors are higher on Table 5.9, this is evidence that the Micro-Integrator performance factor increases with an increase in the integration time frame.

Table 5.8 Results from computing all the listed chaotic sets of partial differential equations with Euler's method with and without the Micro-Integrator.

| Euler's Table of Computations with and without the Micro-Integrator | | | | | | | | | | |
|---|------------|--------------------|-----------------------------|---------------------------|-------|-----------------------------|--------------------|------------|--------------|--------------------|
| Without the Micro-Integrator | | | | With the Micro-Integrator | | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Lorenz 0-8s | 17 | 47s | 5.23 | 3722031 | 256 | 4.90 | 24s | 18+6 | 30842 | 1.45×10^4 |
| Rossler 0-80s | 16 | 10s | 4.74 | 708401 | 256 | 4.81 | 5.2s | 14+18 | 4923 | 2.77×10^3 |
| Chen 0-3s | 17 | 29s | 4.70 | 3284127 | 256 | 5.27 | 11s | 17+5 | 22031 | 1.28×10^4 |
| Chua 0-25s | 10 | 13s | 5.29 | 1537732 | 256 | 4.75 | 9.6s | 16+11 | 10970 | 6.00×10^3 |
| Hadley 0-20s | 18 | 60s | 4.81 | 3984762 | 256 | 4.59 | 25s | 18+6 | 30842 | 1.56×10^4 |
| ACT 0-20s | 18 | 11s | 5.09 | 367756 | 256 | 4.86 | 9s | 13+5 | 1378 | 1.44×10^3 |
| Diff Lorenz 0-50s | 12 | 4.1s | 4.70 | 432478 | 256 | 4.58 | 3.3s | 15+8 | 3510 | 1.69×10^3 |

Table 5.9 Results from integrating all the listed chaotic partial differential equations by Euler's method with and without the Micro-Integrator on an increased time window.

| Euler's Table of Computations with and without the Micro-Integrator | | | | | | | | | | |
|---|------------|--------------------|-----------------------------|---------|---------------------------|-----------------------------|--------------------|------------|--------------|--------------------|
| Without the Micro-Integrator | | | | | With the Micro-Integrator | | | | | |
| | Loop Count | CPU Time $\pm 5\%$ | True Error $\times 10^{-3}$ | N_2 | N_1 | True Error $\times 10^{-3}$ | CPU Time $\pm 5\%$ | Loop Count | N_{res_A} | η |
| Lorenz 0-8.1s | 17 | 58s | 5.01 | 6568290 | 256 | 4.57 | 29s | 18+8 | 52123 | 2.57×10^4 |
| Rosler 0-120s | 16 | 15s | 4.74 | 1751544 | 256 | 4.80 | 60s | 14+18 | 78696 | 6.84×10^3 |
| Chen 0-3.5s | 21 | 91s | 4.72 | 6206999 | 256 | 4.72 | 22s | 18+5 | 44060 | 2.43×10^4 |
| Chua 0-35s | 14 | 60s | 4.54 | 7784767 | 256 | 5.28 | 60s | 19+15 | 45803 | 3.04×10^4 |
| Hadley 0-35s | 17 | 53s | 5.43 | 6568290 | 256 | 4.84 | 48s | 19+6 | 56312 | 2.57×10^4 |
| ACT 0-30s | 23 | 121s | 5.23 | 7480318 | 256 | 4.66 | 252s | 18+6 | 30842 | 2.92×10^4 |
| Diff Lorenz 0-60s | 18 | 40s | 4.92 | 4926190 | 256 | 4.71 | 20s | 17+7 | 40094 | 1.92×10^4 |

Table 5.10 shows the importance of the error control algorithm, it has two major columns containing the error limits and actual errors obtained from integrating the Lorenz Equations with the Micro-Integrator, both with and without the error control algorithm. It can be confirmed from the Table that the error control algorithm brought the error to within an accuracy limit of $\pm 10\%$.

Table 5.10 Demonstrating the accuracy of the error control algorithm by computations with and without it for the Lorenz and Rosler systems with the Micro-Integrator.

| | Without Error Control Algorithm | | With Error Control Algorithm | | |
|--------------------|---------------------------------|-----------------------|------------------------------|-----------------------|------------|
| | Set error limit | Actual error | Set error limit | Actual error | Accuracy |
| Lorenz t=0-20s | 0.01 | 7.55×10^{-3} | 0.01 | 1.05×10^{-2} | $\pm 10\%$ |
| | 0.02 | 7.55×10^{-3} | 0.02 | 2.13×10^{-2} | $\pm 10\%$ |
| | 0.03 | 7.55×10^{-3} | 0.03 | 2.78×10^{-2} | $\pm 10\%$ |
| | 0.04 | 7.55×10^{-3} | 0.04 | 3.61×10^{-2} | $\pm 10\%$ |
| | 0.05 | 7.55×10^{-3} | 0.05 | 4.57×10^{-2} | $\pm 10\%$ |
| Rosler t=0-100s | 0.01 | 3.43×10^{-4} | 0.01 | 9.87×10^{-3} | $\pm 10\%$ |
| | 0.02 | 3.43×10^{-4} | 0.02 | 1.97×10^{-2} | $\pm 10\%$ |
| | 0.03 | 3.43×10^{-4} | 0.03 | 2.77×10^{-2} | $\pm 10\%$ |
| | 0.04 | 3.43×10^{-4} | 0.04 | 3.92×10^{-2} | $\pm 10\%$ |
| | 0.05 | 3.43×10^{-4} | 0.05 | 4.65×10^{-2} | $\pm 10\%$ |

5.3 Graphical Presentation of Results

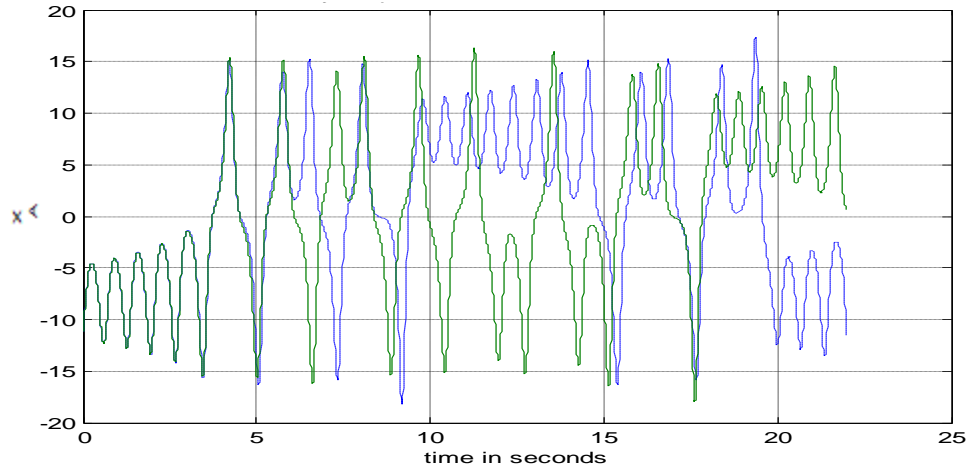
This section contains the output graphs for all the chaotic systems in this study. Figures 5.1a to 5.7c contain plots of the output x arrays for each of the chaotic systems. Figures 5.1a, 5.2a, 5.3a, 5.4a, 5.5a, 5.6a; and 5.7a were the outputs from integrating the chaotic systems without the Micro-Integrator using the Modified Euler's method with the same resolutions of $N_A = 2000$ and $N_B = 4000$ for the A and B arrays plotted using the blue dotted line and the green solid lines respectively. The errors varied by Chaotic System. Figures 5.1b, 5.2b, 5.3b, 5.4b, 5.5b, 5.6b; and 5.7b were integrated using the same numerical method without the Micro-Integrator to an error limit of 0.005 bound to $\pm 10\%$. The resolution parameter used for the green solid line was twice that used for the blue dotted line. Figures 5.1c, 5.2c, 5.3c, 5.4c, 5.5c, 5.6c; and 5.7c were integrated with the Micro-Integrator using the same numerical method to an error limit of 0.005 bound to $\pm 10\%$. The Micro-Integrator resolution parameter used for the green solid line was twice that for the blue dotted line.

Figures 5.8a-d were plotted from solving the Chen equations without the Micro-Integrator using the Euler, modified Euler, Simpson; and fourth order Runge-Kutta methods respectively for the same resolution parameters of $N_A = 2000$ and $N_B = 4000$. This was done to test the relative efficiency of these numerical methods. Observations were recorded in section 5.4 and in the results.

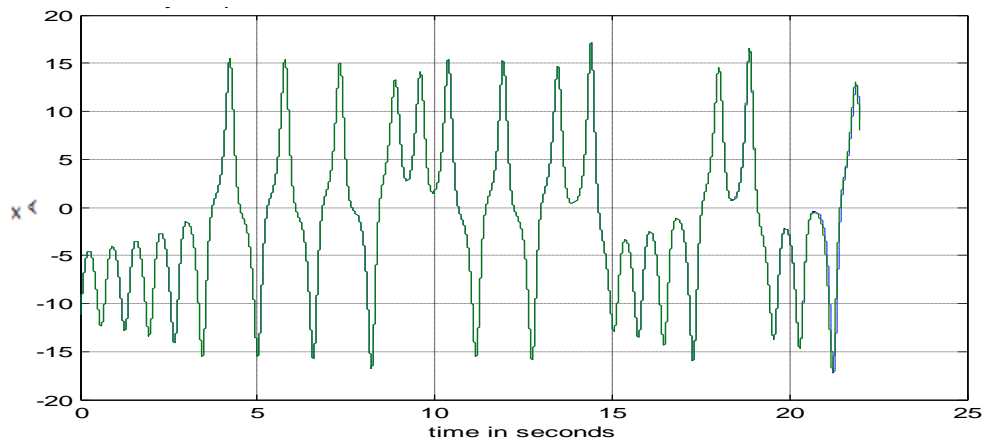
The integrations done with the Micro-Integrator to meet an error limit of 0.005 bound to $\pm 10\%$ produced output graphs that were visually indistinguishable for the different numerical methods. For the purpose of documentation, the results from using the modified Euler were included in this section.

5.4 Observations

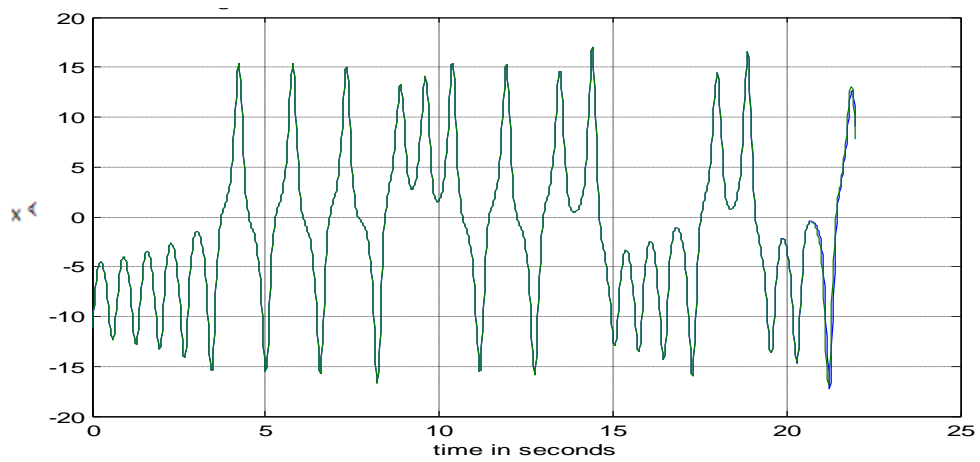
From the result tables, it could be observed that the integrations done using the Micro-Integrator required significantly less memory than those done without it. Also, the amount of CPU time required was much less for the integrations done with the Micro-Integrator, even though it first integrates to an accuracy ten times the desired limit, before gradually reducing the resolution parameter to meet the exact accuracy limit. Also, the higher order numerical methods performed much better than the lower ones.



(a) Without the Micro-Integrator $N_A = 2000$

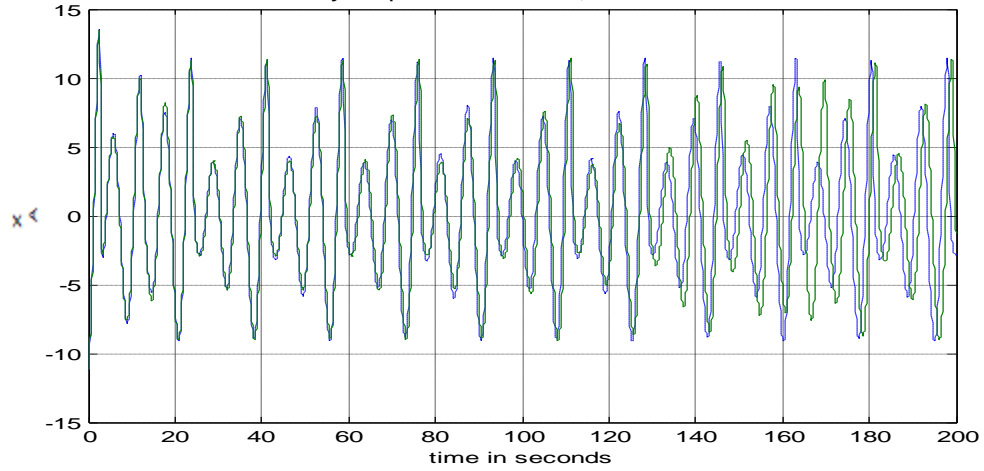


(b) Without the Micro-Integrator $N_A = 3626132$

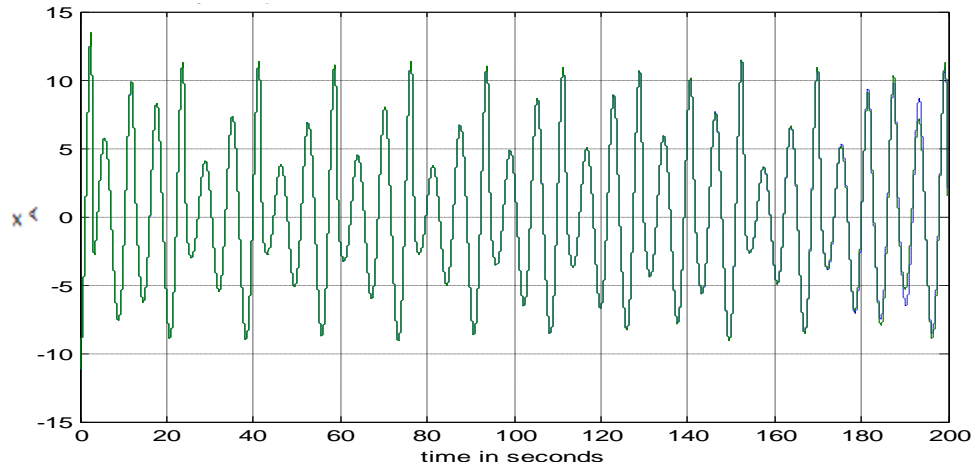


(c) With the Micro-Integrator $N_A = 1024$

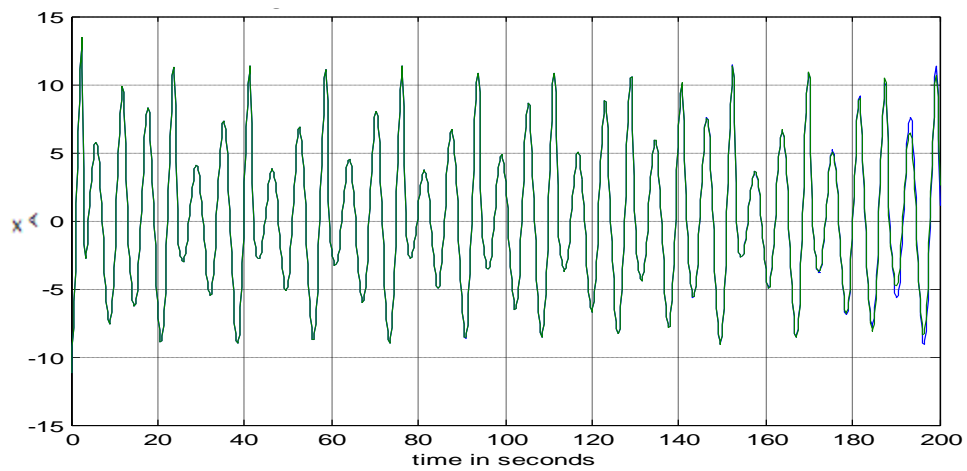
Figure 5.1 x vs t for the Lorenz System.



(a) Without the Micro-Integrator $N_A = 2000$

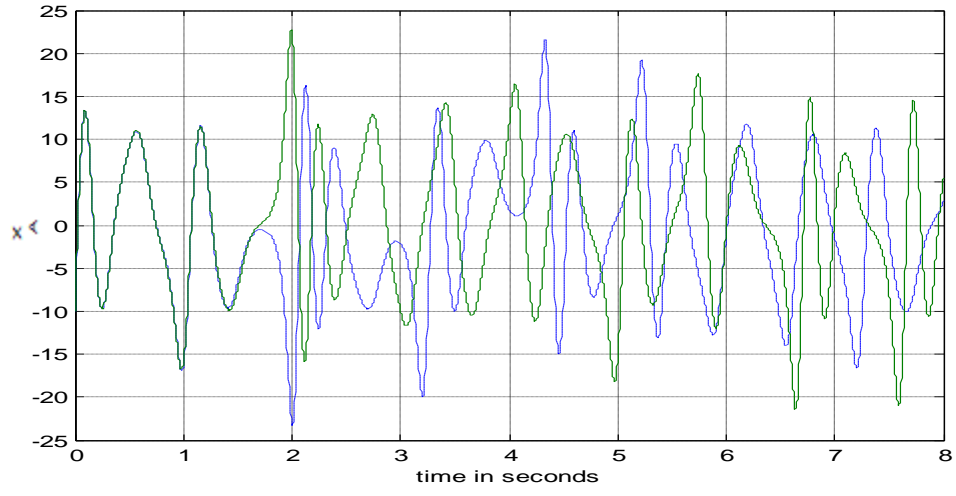


(b) Without the Micro-Integrator $N_A = 345984$

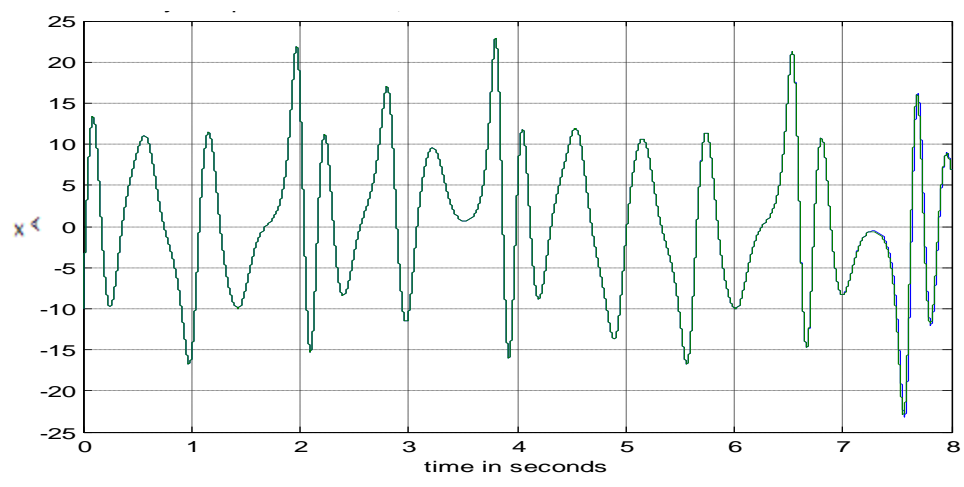


(c) With the Micro-Integrator $N_A = 1024$

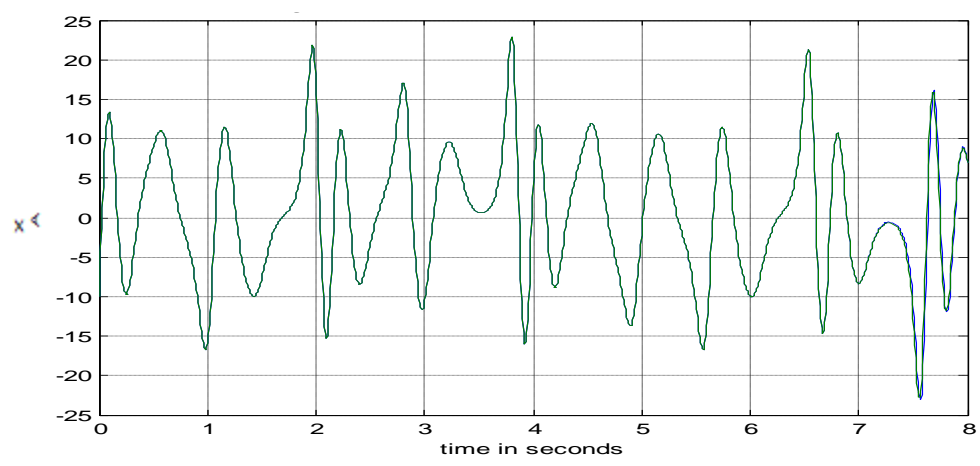
Figure 5.2 x vs t for the Rossler System.



(a) Without the Micro-Integrator $N_A = 2000$

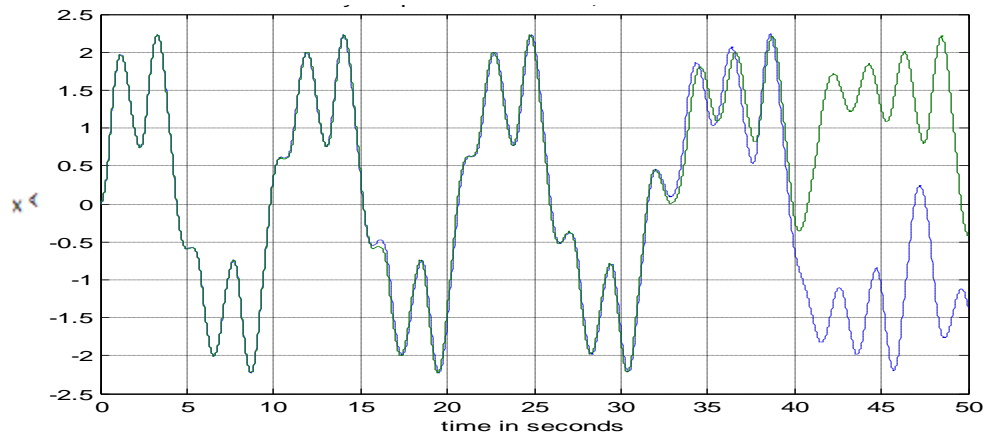


(b) Without the Micro-Integrator $N_A = 1839110$

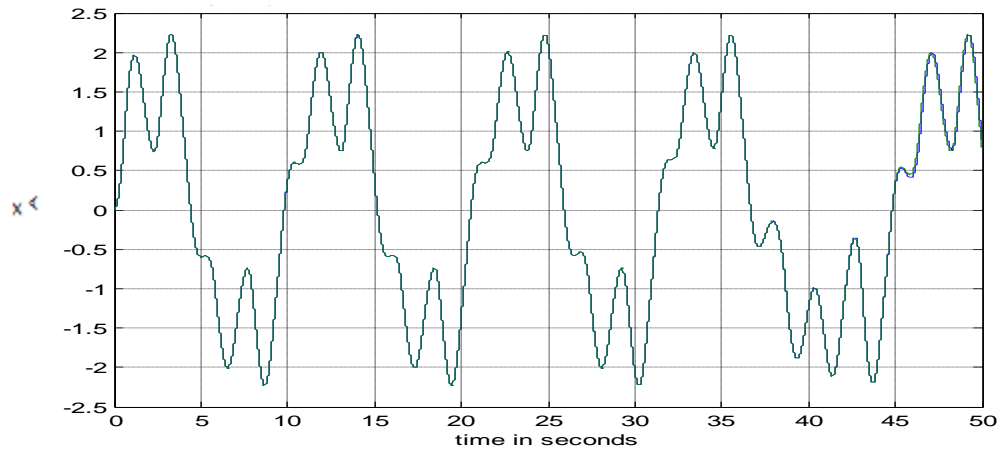


(c) With the Micro-Integrator $N_A = 1024$

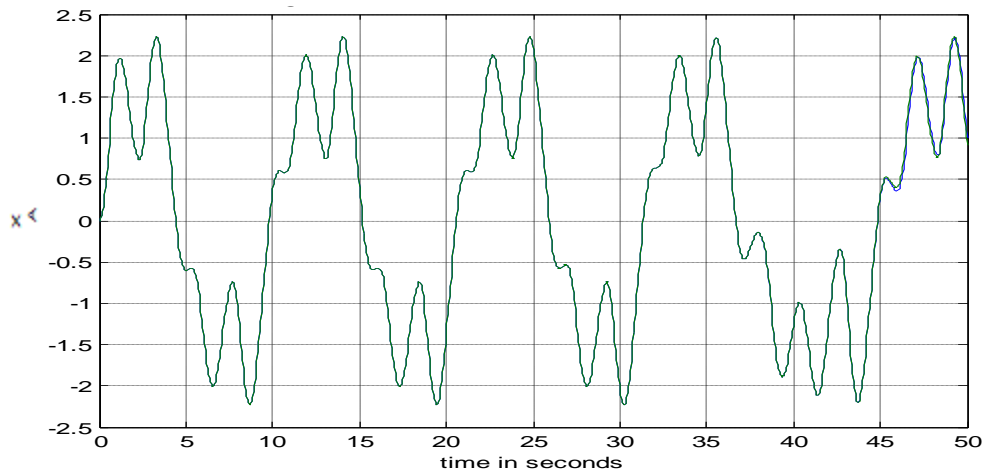
Figure 5.3 x vs t for the Chen System.



(a) Without the Micro-Integrator $N_A = 2000$

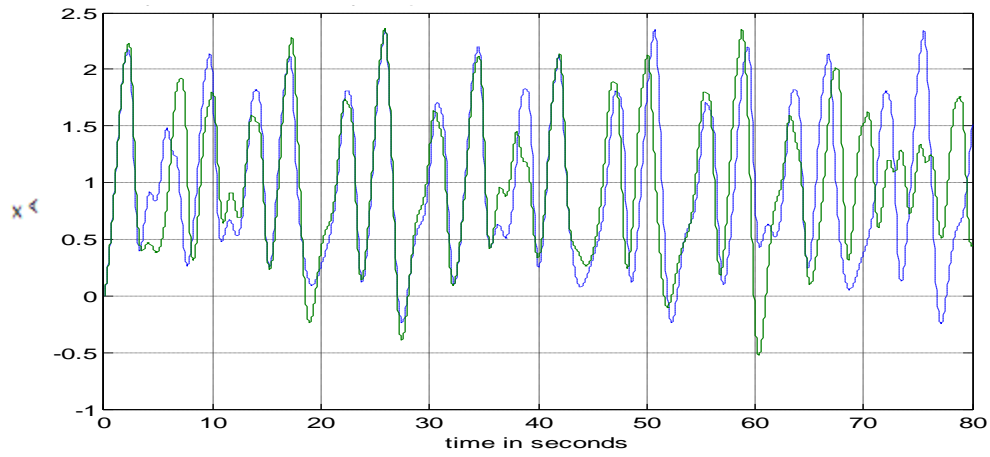


(b) Without the Micro-Integrator $N_A = 80551$

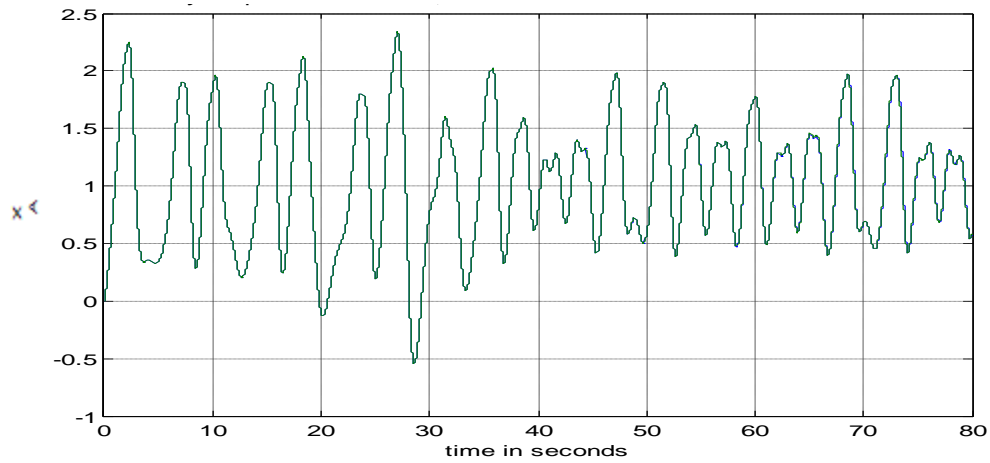


(c) With the Micro-Integrator $N_A = 1024$

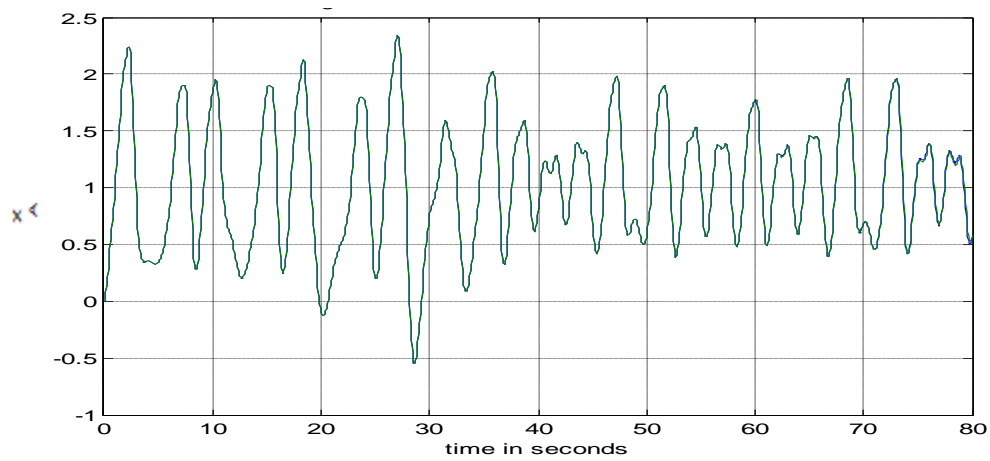
Figure 5.4 x vs t for the Chua System.



(a) Without the Micro-Integrator $N_A = 2000$

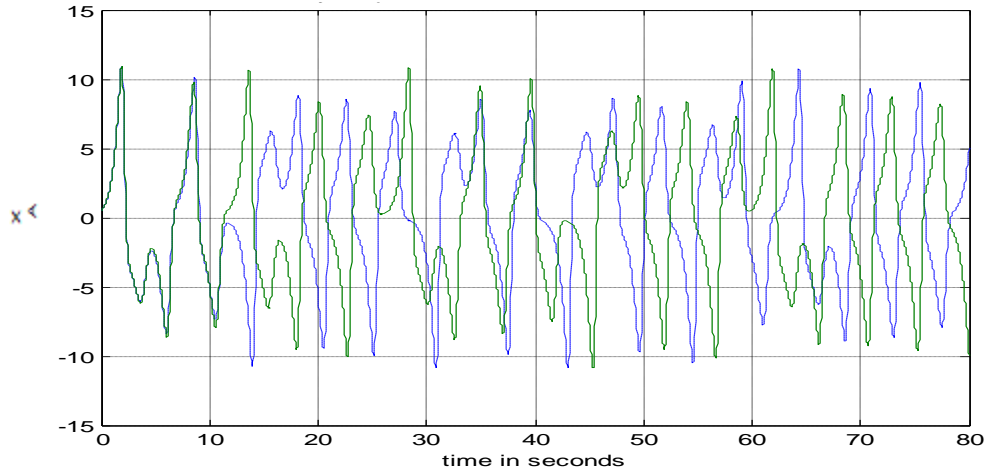


(b) Without the Micro-Integrator $N_A = 1946160$

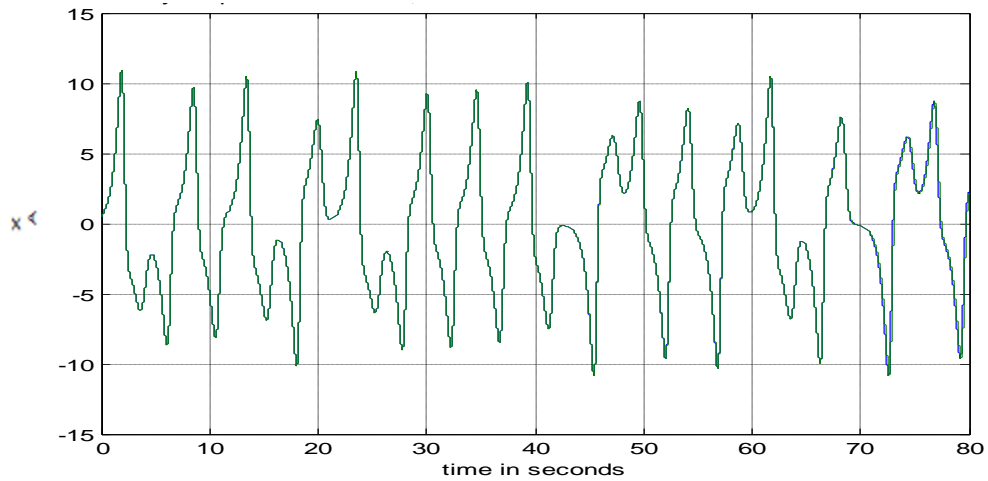


(c) With the Micro-Integrator $N_A = 1024$

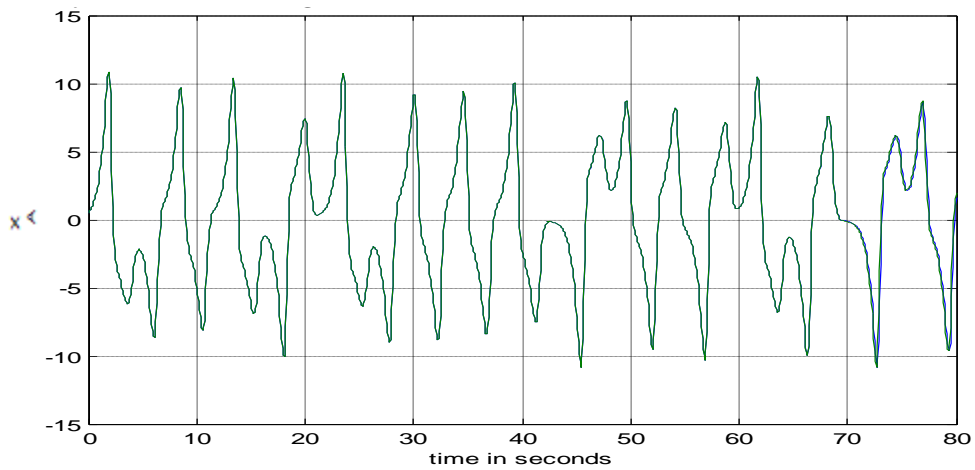
Figure 5.5 x vs t for the Hadley System.



(a) Without the Micro-Integrator $N_A = 2000$

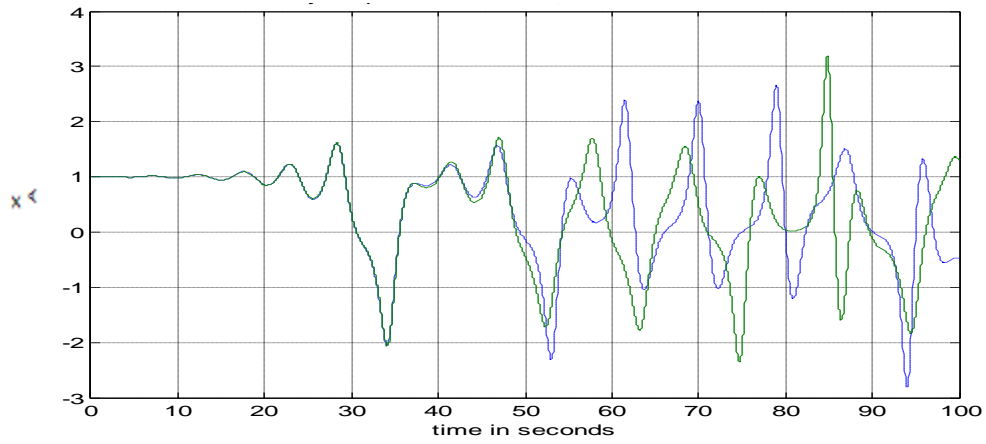


(b) Without the Micro-Integrator $N_A = 2059814$

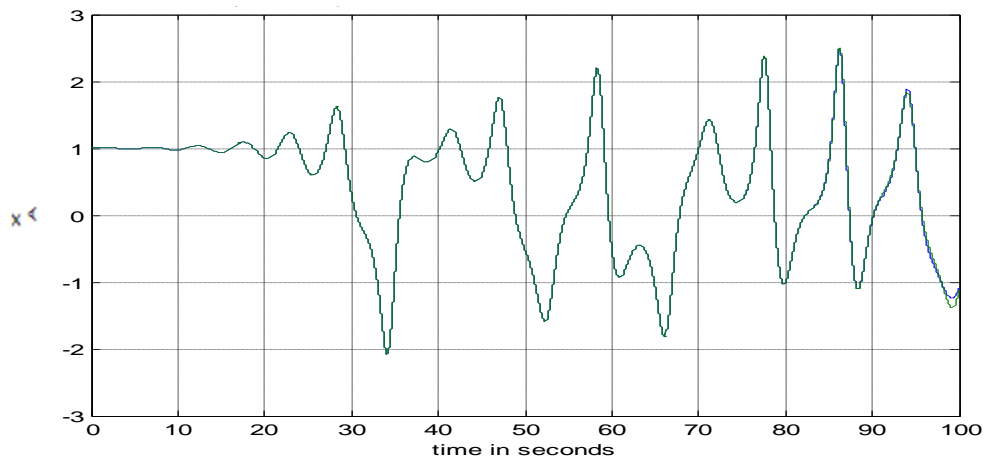


(c) With the Micro-Integrator $N_A = 1024$

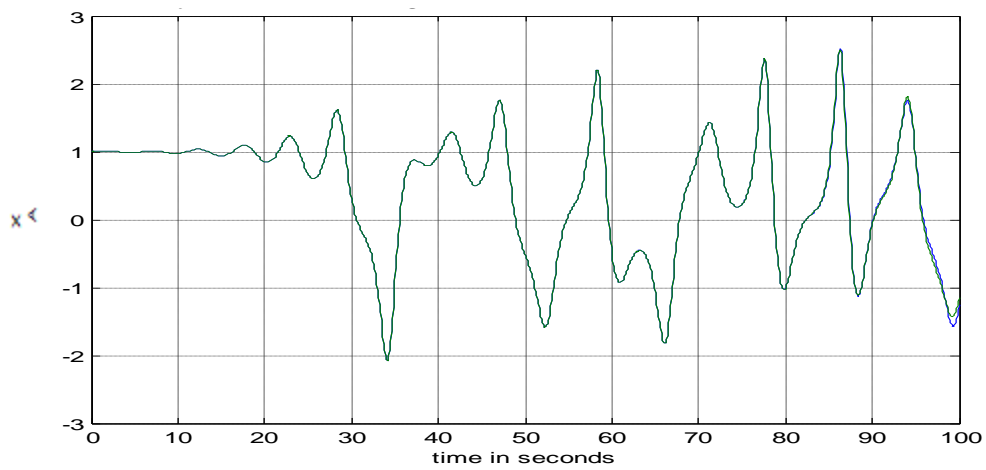
Figure 5.6 x vs t for the ACT System.



(a) Without the Micro-Integrator $N_A = 2000$

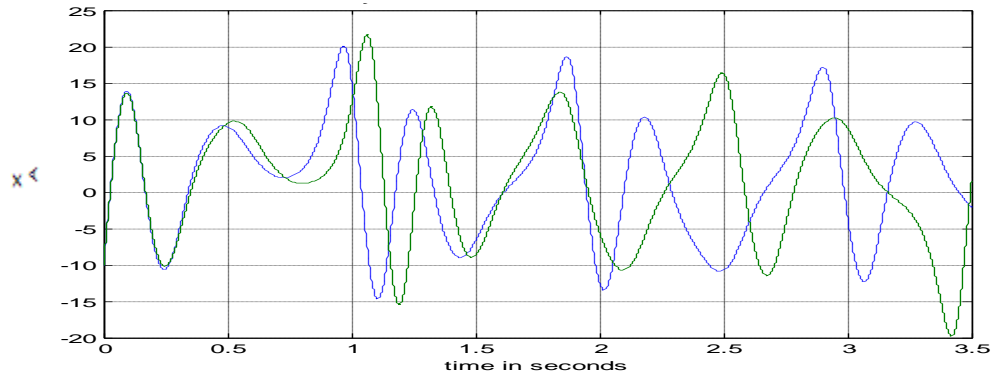


(b) Without the Micro-Integrator $N_A = 633032$

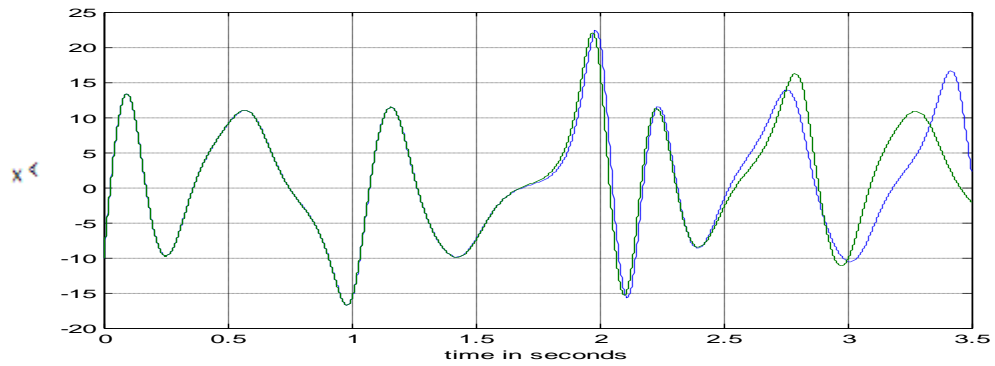


(c) With the Micro-Integrator $N_A = 1024$

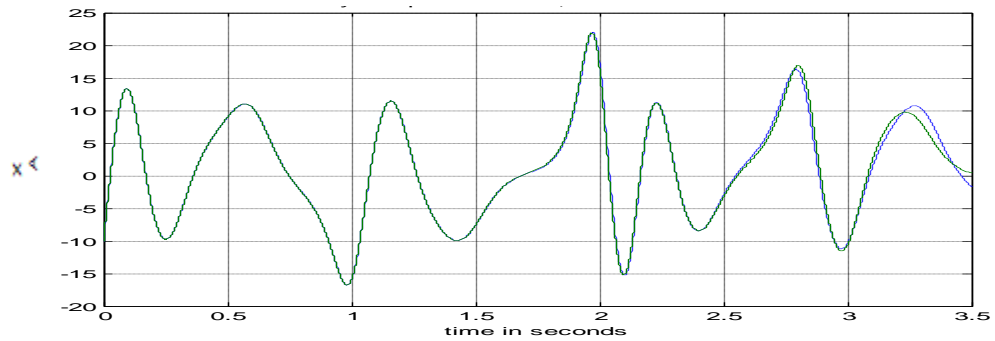
Figure 5.7 x vs t for the Diffusionless Lorenz System.



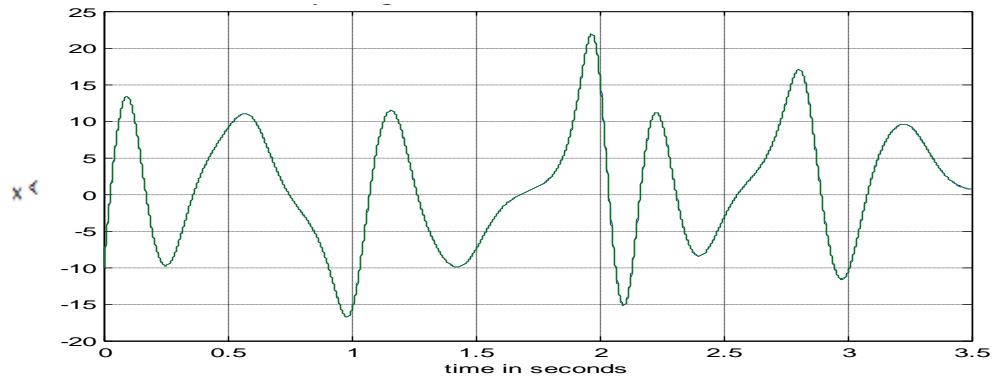
(a) Using Euler's Method, $N_A = 2000$



(b) Using the modified Euler's Method, $N_A = 2000$



(c) Using Simpson's Method, $N_A = 2000$



(d) Using Runge-Kutta's Method, $N_A = 2000$

Figure 5.8 x vs. t for the Chen equations Without the Micro Integrator.

Based on the results and expectations, A hierarchy could be drawn as follows; Euler, modified Euler, Simpson, Runge-Kutta. This represents an order of increasing complexity, and computational efficiency. A quick glance at this is presented in Figures 5.8a-d, from the graphs, the lag of the blue dotted line behind the green solid line shows how computationally efficient that algorithm was in carrying out that integration. The four integrations there were done for a specific value of the resolution parameter, yet some of the results were much more accurate than the others, this is evidence that certain numerical methods are more efficient than the others, and the hierarchy is correct.

5.5 Table Data Presented in Plots

5.5.1 Illustrating the Effectiveness of the Error Control Algorithms

The effectiveness of the error control algorithm as shown in Table 5.10 are represented graphically in Figures 5.9 and 5.10.

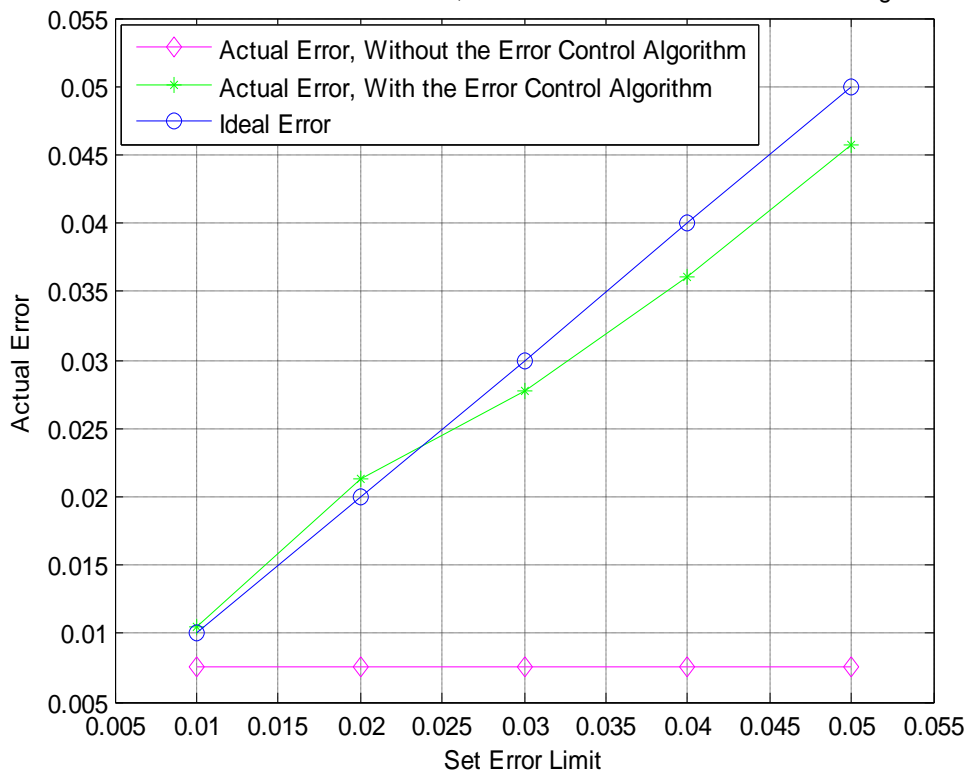


Figure 5.9 Illustrating the Effectiveness of the Error Control Algorithm using the Lorenz Equations.

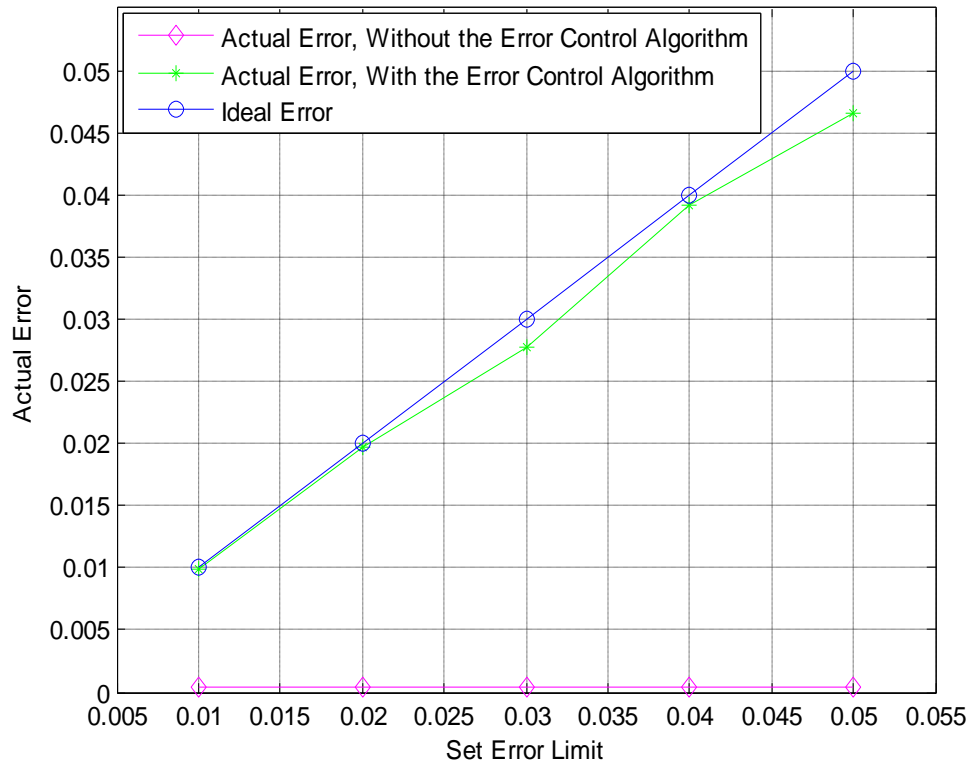


Figure 5.10 Illustrating the Effectiveness of the Error Control Algorithm using the Rossler Equations.

The pink line with diamond shaped points was obtained from plotting the errors achieved without the error control algorithm versus the set error limit. The green line with asterix points was obtained from plotting the errors achieved with the error control algorithm versus the set error limit. The blue line with circle shaped points is an ideal reference obtained by plotting the set error limits against each other.

5.5.2 The Performance Measures of the Chaotic Systems

The Performance Measures of each of the chaotic systems as defined in Chapter 4 are represented graphically in this subsection. Figure 5.11 shows those for the Lorenz Equations, Figure 5.12 shows those for the Rossler Equations, Figure 5.13 shows those for the Chen Equations, Figure 5.14 shows those for the Chua Equations, Figure 5.15 shows those for the Hadley Equations, Figure 5.16 shows those for the ACT Equations; and Figure 5.17 shows those for the Diffusionless Lorenz Equations.

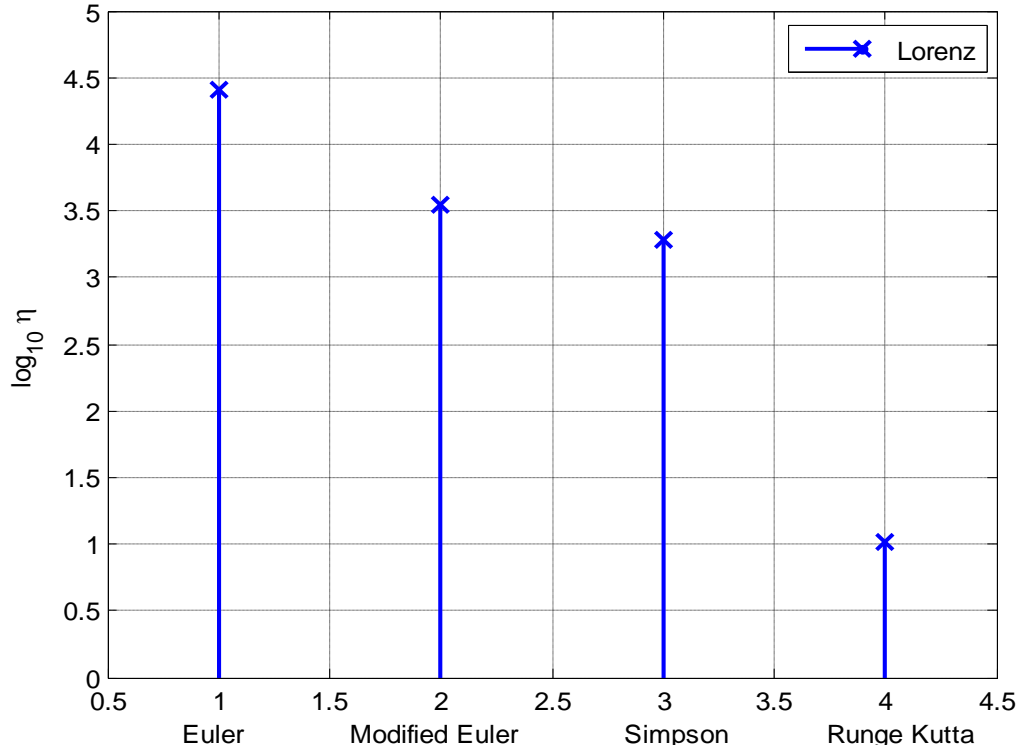


Figure 5.11 Performance Measures of the Lorenz System.

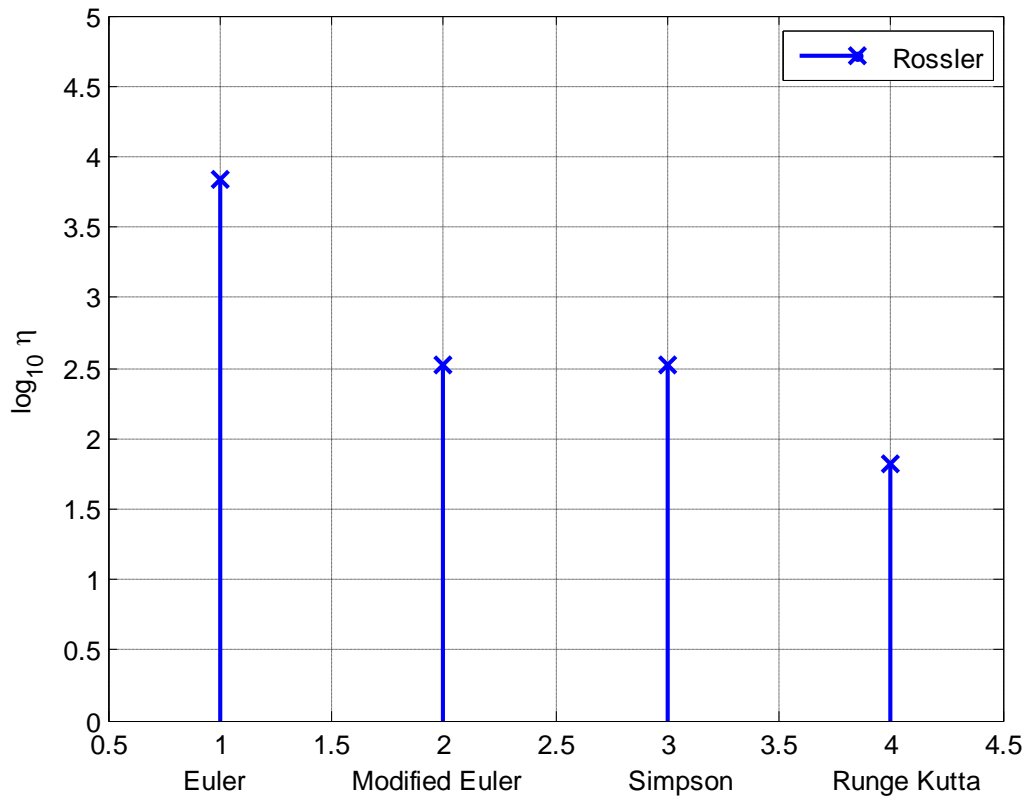


Figure 5.12 Performance Measures of the Rossler System.

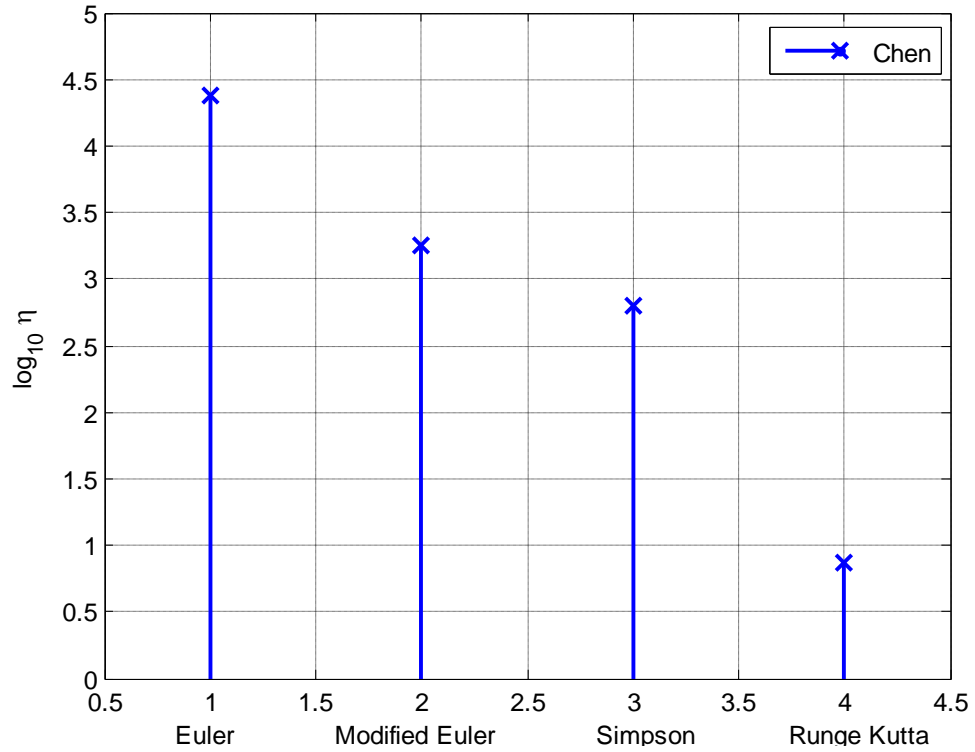


Figure 5.13 Performance Measures of the Chen System.

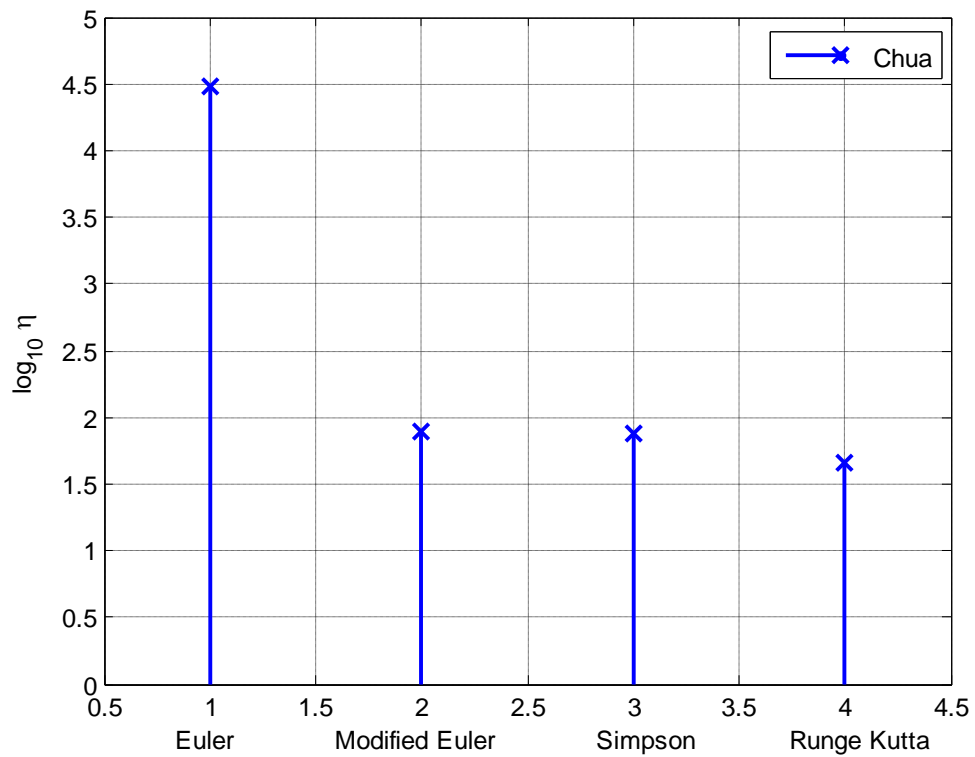


Figure 5.14 Performance Measures of the Chua System.

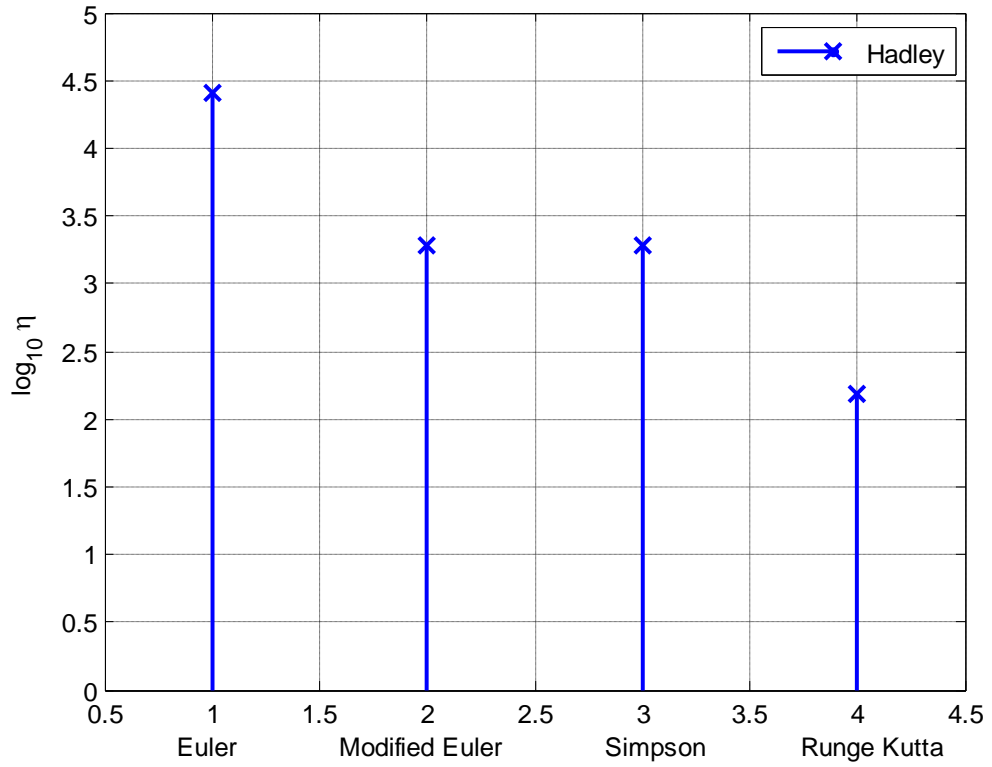


Figure 5.15 Performance Measures of the Hadley System.

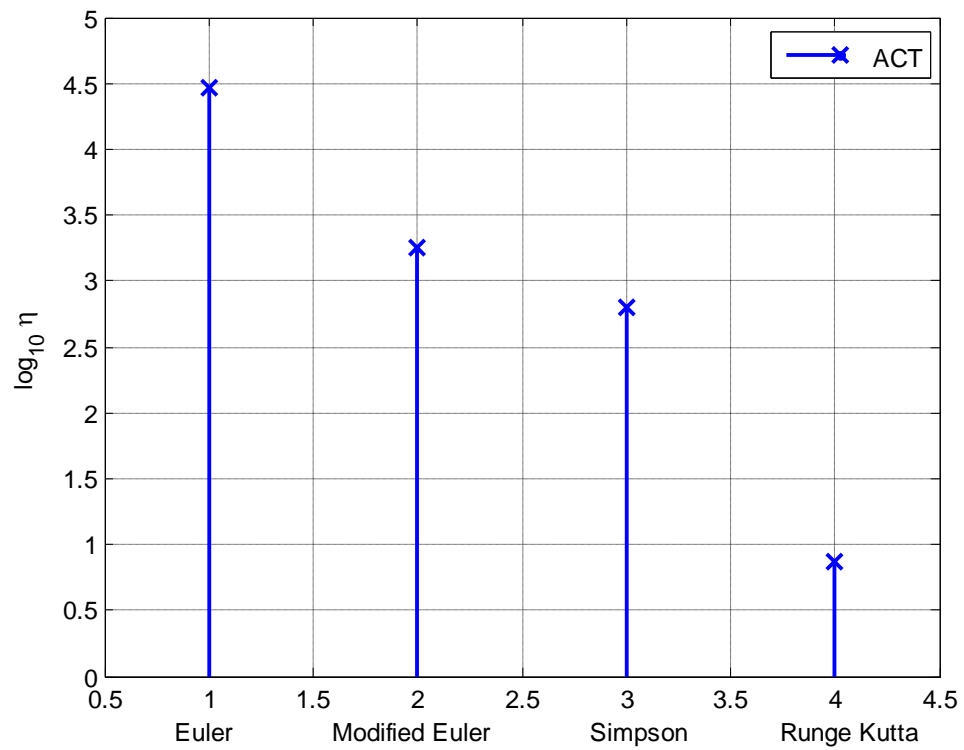


Figure 5.16 Performance Measures of the ACT System.

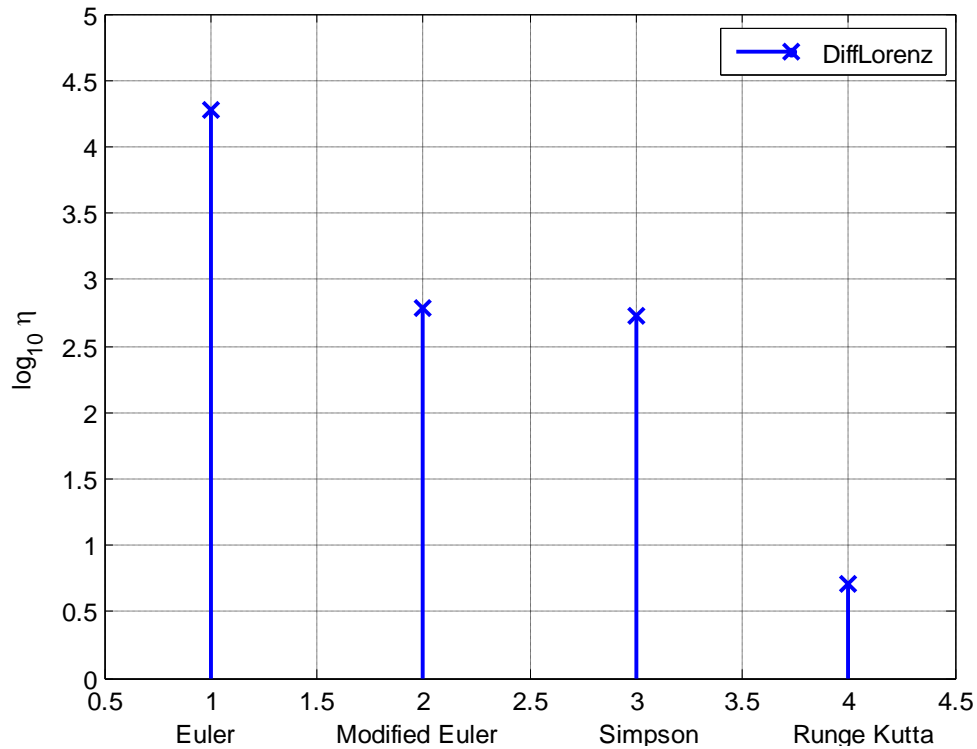


Figure 5.17 Performance Measures of the Diffusionless Lorenz System.

5.5.3 Percentage Reduction in CPU time

The percentage reduction in CPU time recorded for integrating all the chaotic systems of Equations with and without the Micro-Integrator as represented by the data in Tables 5.1 to 5.9 are shown graphically in Figure 5.18. The y axis is the percentage reduction in CPU time, ordered by the numerical methods used. All the cases apart from the Runge-Kutta showed reduction in CPU time with using the Micro-Integrator, while the Runge-Kutta showed an increase.

5.5.4 The Performance Ratios

The performance ratios obtained from integrating all the chaotic systems of Equations for two different integration time frames by the Euler's method as represented by Tables 5.8 and 5.9 are shown graphically in Figure 5.19. The y axis is the base ten logarithm of each of the performance factors, and each of the chaotic systems are placed on the x axis for comparison.

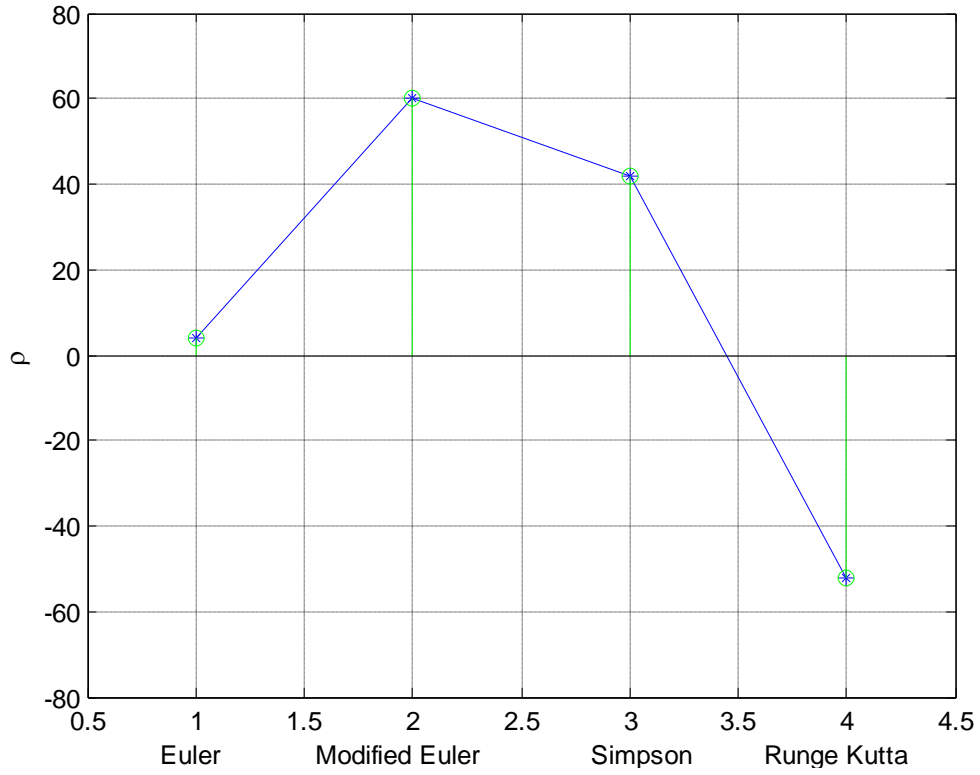


Figure 5.18 Percentage Reduction in CPU time.

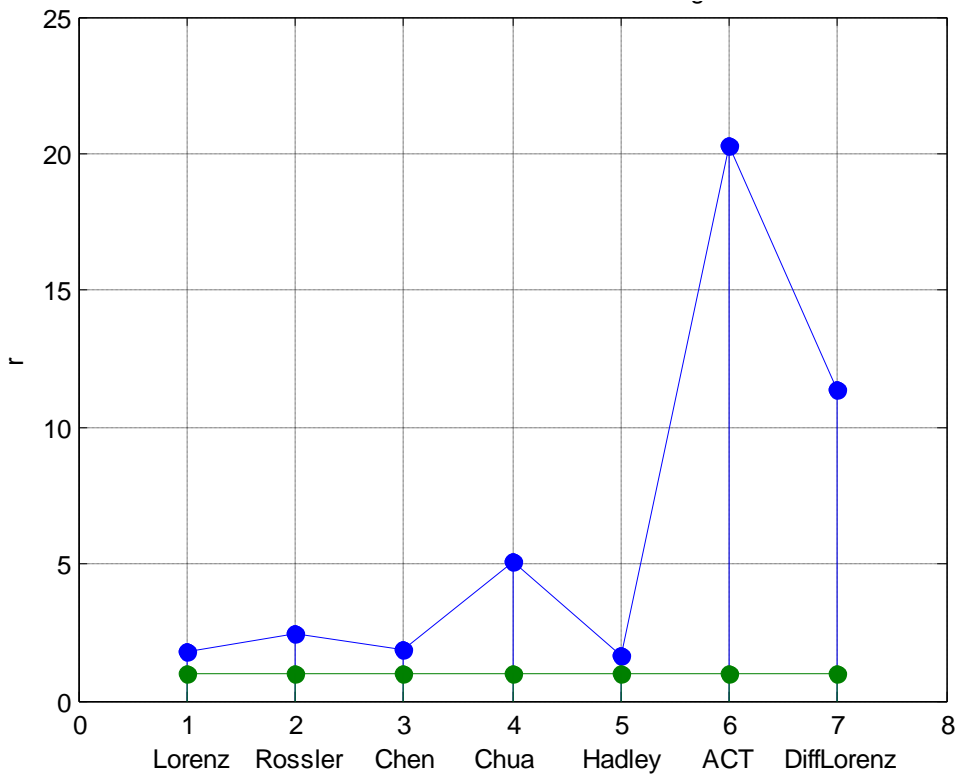


Figure 5.19 Performance Ratios for the chaotic systems.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In the results obtained, the advantages of the Micro-Integrator were visible and have been quantified. In order to accurately gauge the differences in performances, an error control algorithm was introduced during this research to limit the error to within a certain percentage of a specified error limit. It was developed and used in the tests.

A performance factor was also introduced to quantify the advantages gained from using the Micro-Integrator in terms of reduction in memory requirements. In at least 75% of the cases, the performance factors exceeded a memory ratio of 1000. It was observed that the numerical integration methods that were used in this research played a significant role in the performance of the Micro-Integrator algorithm. The less computationally efficient algorithms produced higher performance factors.

It was also noted that CPU times were affected by the selection of the numerical integration algorithm, and the Micro-Integrator was very efficient in lowering CPU times. The performance factors seemed to be influenced by the time windows for integration. Higher time windows produced higher performance factors.

6.2 Future Work

The Lyapunov coefficients for each of the chaotic systems showed some level of correlation with the time windows for integration, which in turn influenced the Micro-Integrator's performance factors. This would indirectly suggest that the Micro-Integrator's performance factor has some dependence on the Lyapunov coefficients of the chaotic systems. There are more investigations to be done here, possibly more advanced versions of the Micro-Integrator algorithm would make it easier to explore this correlation. The Micro-Integrator algorithm could be made more advanced than it currently is, for instance, automatic selection of the conventional resolution parameter, alternate structures to its current error control algorithm, tests with simpler error calculation algorithms. More importantly, it could be applied to more sets of equations and put to use in practical situations, to solve practical memory problems in real time simulations.

References

- [1] Benoit B. Mandelbrot, "Self-Similar Error Clusters in Communication Systems and the Concept of Conditional Stationarity," *IEEE Transactions on Communications Technology*, vol. 13, Issue 1, pp 71-90, March 1965.
- [2] Edward N. Lorenz, "Computational Chaos, a Prelude to Computational Instability," *Physica D: Non-linear Phenomena*, vol. 35, Issue 3, pp. 299-317, 1989.
- [3] Douglas R. Hofstadter, "Strange Attractors: Mathematical Patterns delicately poised between Order and Chaos," *Scientific American*, vol. 245, pp. 22-43, November 1981.
- [4] R. J. Pieper, "Observations on Convergence Problems of Pipeline Networks" *30th Southeastern Symposium on System Theory*, Morgantown WV, pp. 39-42, March 1998.
- [5] K. M. Cuomo, A. V. Oppenheim, "Synchronization of Lorenz-Based Chaotic Circuits with Applications to Communications," *IEEE Transactions on Circuits and Systems*, vol. 40, pp. 626 – 633, October 1993.
- [6] D. J. Blair, R. J. Pieper, "Observations on Message Transmission Using Rossler and Lorenz Chaos Systems with PSpice and Matlab Models," in *Proceedings of the Fortieth Southeastern Symposium on System Theory*, New Orleans LA, pp. 69 – 73, March 2008.
- [7] R. J. Pieper, D. J. Blair, "A Practical Solution to the Numerical Butterfly Effect in Chaotic Systems for Fast but Memory Limited Computers" *42nd Southeastern Symposium on System Theory*, Tyler TX, pp. 335-339, March 2010.
- [8] Joao Teixeira, Carolyn A. Reynolds, Kevin Judd, "Time Step Sensitivity of Non-linear Atmospheric Models: Numerical Convergence, Truncation Error Growth, and Ensemble Design," *Journal of the Atmospheric Sciences*, vol. 64, pp. 175 – 189, January 2007.
- [9] Lun-Shin Yao, "Computed Chaos or Numerical Errors", *Non-linear Analysis: Modelling and Control*, vol. 15, No.1, pp. 109-126, 2010.
- [10] Shouliang Bu, "Quantum trajectory of an individual particle in the presence of chaos", *Physica D: Non-linear Phenomena*, vol. 217, Issue 2, 15, pp. 103-106, May 2006.
- [11] T. M. Fromhold , P. B. Wilkinson, P. M. Martin, A. Thornton, L. Eaves, F. W. Sheard, P. C. Main, M. Henini "Chaos in quantum wells and analogous optical systems", *Physica E: Low-dimensional Systems and Nanostructures*, vol. 11, Issue 2-3, pp. 114-117, October 2001.

- [12] I.V. Gornya, A.D. Mirlina, "Wave function correlations on the ballistic scale: from quantum disorder to quantum chaos", *Physica E: Low-dimensional Systems and Nanostructures*, vol. 12, Issues 1-4, pp. 845-848, January 2002.
- [13] Jonathan N. Blakely, Ned J. Corron, Shawn D. Pethel, "Equivalence of the continuum limit of the generalized Rossler system and the chaotic transmission line oscillator", *Physica D: Non-linear Phenomena*, vol. 207, Issues 3-4, pp. 161-170, August 2005.
- [14] Stephen R. Addison, John E. Gray, "Chaos and Encryption, Problems and Potential", *38th Southeastern Symposium on System Theory*, Cookeville TN, pp. 275-279, March 2006.
- [15] Zhu Yu, Zhou Zhe, Yang Haibing, Pan Wenjie, Zhang Yunpeng, "A Chaos-Based Image Encryption Algorithm Using Wavelet Transform", *2nd International Congress on Image and Signal Processing*, Shenyang China, pp. 1-5, October 2009.
- [16] Shuo Zhang, Ruhua Cai, Yingchun Jiang, Shiping Guo, "An Image Encryption Algorithm Based on Multiple Chaos and Wavelet Transform", *International Congress on Image and Signal Processing*, Tianjin China, pp. 1-5, October 2009.
- [17] Shu-Mei Guo, Wen-Hsin Chang a, Jason S.-H. Tsai, Bo-Liang Zhuang, Li-Chun Chen, "JPEG 2000 wavelet filter design framework with chaos evolutionary programming", *Signal Processing Archive*, Amsterdam Netherlands, vol. 88 Issue 10, pp 2542-2553, October 2008.
- [18] Wang Juan, "Image Encryption Algorithm Based on 2-D Wavelet Transform and Chaos Sequences", *International Conference on Computational Intelligence and Software Engineering*, Wuhan China, pp. 1-3, December 2009.
- [19] John E. Gray and Stephen R. Addison, "Symbolic Noise, Signal Processing, and Signal Enhancement by the use of Chaos", *38th Southeastern Symposium on System Theory*, Cookeville TN, pp. 417-421, March 2006.
- [20] Leon O. Chua and Rabinder N. Madan, "Sights and sounds of chaos", *IEEE Circuits and Devices Magazine*, vol.4, pp. 3-13, January 1988.
- [21] J. C. Sprott, "Simple Chaotic Systems and circuits," *American Journal of Physics*, vol. 68, Issue 8, pp. 758-763, 2000.
- [22] R. Newcomb and S. Sathyan, "An RC op amp chaos generator," *IEEE Transactions on Circuits and Systems*, vol.30, no.1, pp. 54- 56, January 1983.
- [23] Edward N. Lorenz, "Deterministic Nonperiodic Flow," *Journal of Atmospheric Sciences*, vol.20, pp130-141, 1963.

- [24] O. E. Rossler, "An Equation for Continuous Chaos," *Physics Letters*, vol. 57A, pp. 397-398, 1976.
- [25] G. Chen and T. Ueta, "Yet Another Chaotic Attractor," *International Journal of Bifurcation and Chaos*, vol.9, pp.1465, 1999.
- [26] Leon O. Chua; T. Matsumoto; and M. Komuro, "The Double Scroll," *IEEE Transactions on Circuits and Systems*, vol. 33, pp. 798-818, August 1985.
- [27] Edward N. Lorenz, "Irregularity: A Fundamental Property of the Atmosphere," *Tellus* vol. 36A, pp. 98-110, 1984.
- [28] A. Arneodo; P. Coulet; and C. Tresser, "A Possible New Mechanism for the Onset of Turbulence," *Physics Letters A*, vol. 81, pp 197-201, 1981.
- [29] G. Van Der Schrier and L. R. M. Maas, "The Diffusionless Lorenz Equations; Shilnikov bifurcations and reduction to an explicit map" *Physica D*, vol.141, pp19-36, 2000.
- [30] Kaiser S. Kunz, "*Numerical Analysis*", McGraw-Hill, New York, 1957.
- [31] Julien Clinton Sprott, "*Chaos and Time-Series Analysis*," Oxford University Press, New York, 2003.
- [32] Steven C. Chapra; Raymond P. Canale; "*Numerical Methods for Engineers*" McGraw-Hill, New York, 2010.
- [33] Daniel J. Blair; "*Observations on Modeling Chaotic Circuits and Systems with Applications to Secure Communications*," Master of Engineering Project Report, University of Texas at Tyler, Department of Electrical Engineering, April 2008.
- [34] Warwick Tucker, "Computing Accurate Poincaré Maps," *Physica D*, vol.171, pp 127- 137, 2002.

Appendix A: Interpolation Algorithm

The interpolation algorithm as depicted in Figures A.1 to A.3, was executed in a Matlab code. Figures A.1 and A.2 are alternate graphical representations of the procedure, and Figure A.3 is its algorithm in a flow chart. It is designed to increment lower size arrays to match higher ones for better comparison. It uses the basic interpolation rule, it calculates the slope and multiplies it to the differential time interval under consideration and adds this product to the current value to find the next value. All the parameters here are the same as defined in the thesis, and x_{A_i} represents the interpolated array of x_A .

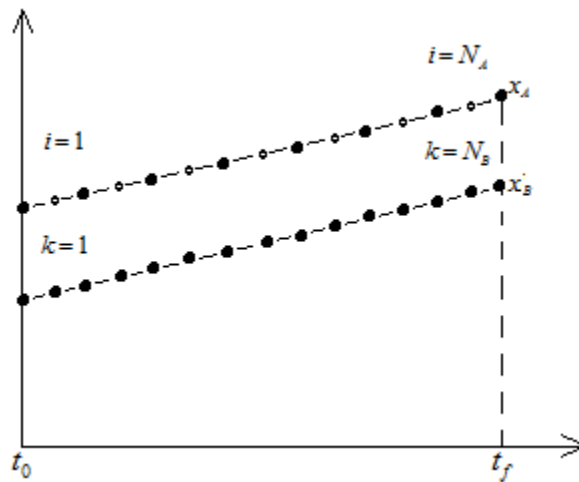


Figure A.1 Graphical representation of the Interpolation concept.

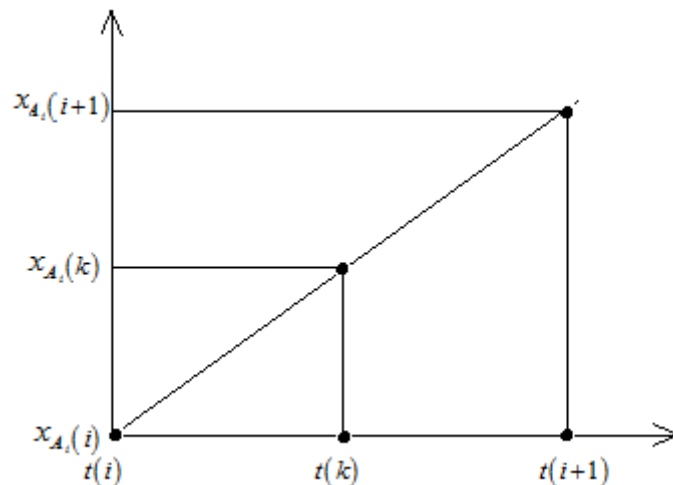


Figure A.2 Alternate Graphical representation of the Interpolation concept.

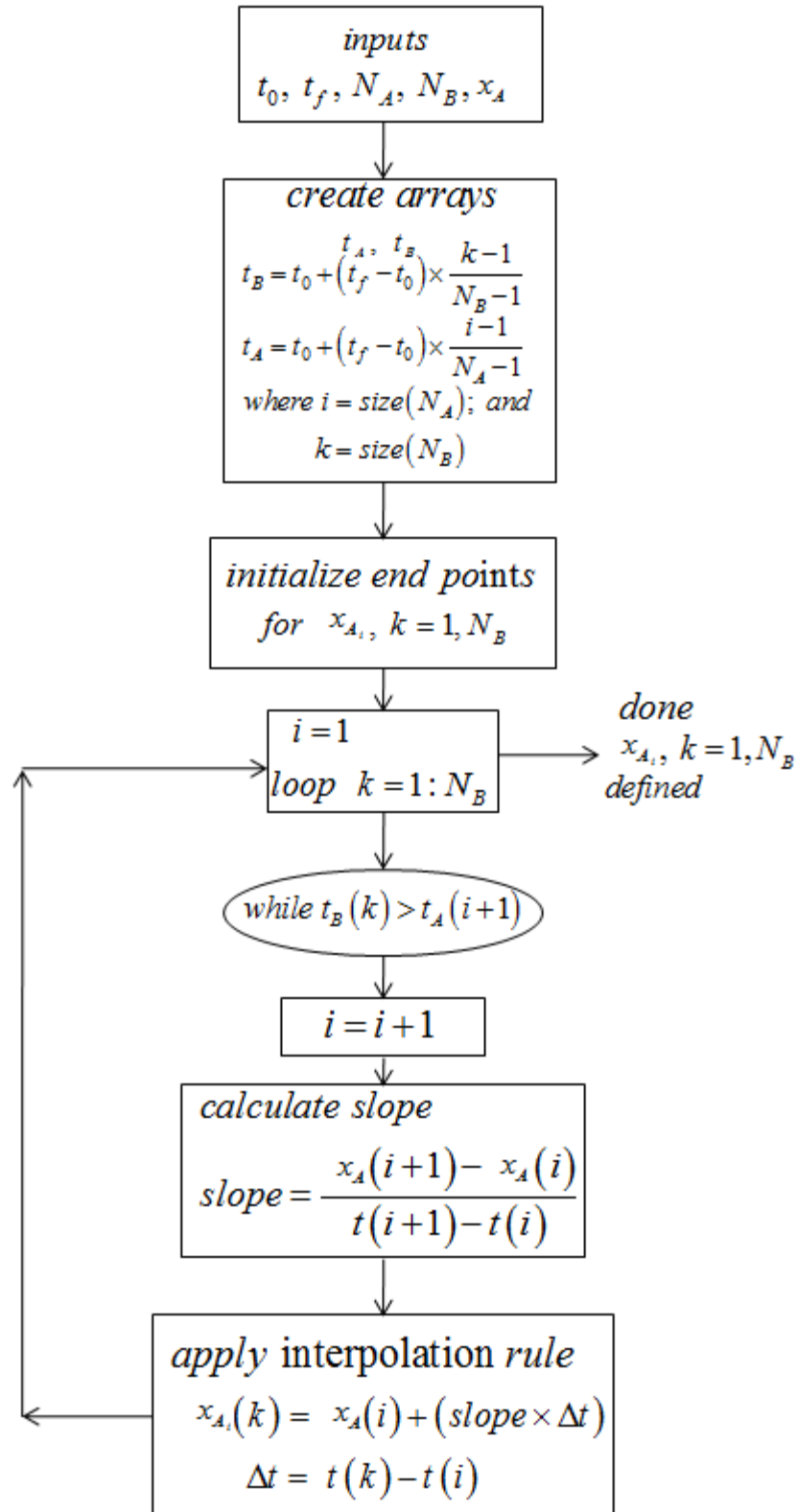


Figure A.3 The Interpolation algorithm.

Appendix B: The Matlab Codes

There were many Matlab codes used in this research, covering in excess of 300 pages. For ease of documentation, only those used to integrate the Lorenz Equations were included in this Appendix.

Appendix B-1: Lorenz Equations by Euler's Method without the Micro-Integrator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This Program uses the Eulers Method to solve the Lorenz Non-linear
Chaotic Partial Differential Equation Set%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;
close all;
%Pre-Integration Program Inputs%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=20; %upper time limit%
N=40000; %Resolution parameter%
dt=(tf-ti)/N;%integration step%
x=0:N; %primitive x array%
y=0:N; %primitive y array%
z=0:N; %primitive z array%
x(1)=xi; %allocating initial values%
y(1)=yi; %allocating initial values%
z(1)=zi; %allocating initial values%
for i=1:N;
    %The Integration%
    dx1=(10*y(i)-10*x(i))*dt;
    dy1=(28*x(i)-y(i)-x(i)*z(i))*dt;
    dz1=(x(i)*y(i)-8*z(i)/3)*dt;
    %    dx=(10*(y(i)+(0.5*dy1))-10*(x(i)+(0.5*dx1)))*dt;
    %    dy=(28*(x(i)+(0.5*dx1))-(y(i)+(0.5*dy1))-
    (x(i)+0.5*dx1)*(z(i)+(0.5*dz1)))*dt;
    %    dz=((x(i)+(0.5*dx1))*(y(i)+(0.5*dy1))-8*(z(i)+(0.5*dz1))/3)*dt;
    x(i+1)=x(i)+dx1;y(i+1)=y(i)+dy1;z(i+1)=z(i)+dz1;
end;
%plot of 2-D poincare map%
figure(1)
plot(x,y)
title('2D Poincare Map - Lorenz Without Micro - Eulers Method')
xlabel('x')
ylabel('y')
grid
%plot of 3-D poincare map%
figure(2)
plot3(x,y,z)
title('3D Poincare Map - Lorenz Without Micro - Eulers Method')
```

Appendix B (Continued)

```

xlabel('x')
ylabel('y')
zlabel('z')
grid
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=8.1; %upper time limit%
NVectA=[1000,2000,4000]; %Matrix of NA values%
NVectB=2*NVectA; %Matrix of NB values%
%Loop to use all values in the NA and NB matrices%
for k = 1:3;
    NA=NVectA(k);%Selecting NA from the matrix%
    NB=NVectB(k);%Selecting NB from the matrix%
    tA=(0:(NA-1))*(tf-ti)/(NA-1)+ti;%time vector%
    tB=(0:(NB-1))*(tf-ti)/(NB-1)+ti;%time vector%
    xA=1:NA; %Primitive xA array%
    yA=1:NA; %Primitive yA array%
    zA=1:NA; %Primitive zA array%
    xB=1:NB; %Primitive xB array%
    yB=1:NB; %Primitive yB array%
    zB=1:NB; %Primitive zB array%
    dtA=(tf-ti)/NA;%time step A%
    dtB=(tf-ti)/NB;%time step B%
    xA(1)=xi; %allocating initial values%
    yA(1)=yi; %allocating initial values%
    zA(1)=zi; %allocating initial values%
    starttime=cputime;
for i=1:(NA-1);
    %The Integration%
    dxA1=(10*yA(i)-10*xA(i))*dtA;
    dyA1=(28*xA(i)-yA(i)-xA(i)*zA(i))*dtA;
    dzA1=(xA(i)*yA(i)-8*zA(i)/3)*dtA;
%    dxA=(10*(yA(i)+(0.5*dyA1))-10*(xA(i)+(0.5*dxA1)))*dtA;
%    dyA=(28*(xA(i)+(0.5*dxA1))-(yA(i)+(0.5*dyA1))-
(xA(i)+0.5*dxA1)*(zA(i)+(0.5*dzA1)))*dtA;
%    dzA=((xA(i)+(0.5*dxA1))*(yA(i)+(0.5*dyA1))-
8*(zA(i)+(0.5*dzA1))/3)*dtA;
    xA(i+1)=xA(i)+dxA1;yA(i+1)=yA(i)+dyA1;zA(i+1)=zA(i)+dzA1;
end;
    finishtime=cputime;usedtime=cputime-starttime;
    xB(1)=xi; %allocating initial values%
    yB(1)=yi; %allocating initial values%
    zB(1)=zi; %allocating initial values%
for i=1:(NB-1);
    %The Integration%
    dxB1=(10*yB(i)-10*xB(i))*dtB;
    dyB1=(28*xB(i)-yB(i)-xB(i)*zB(i))*dtB;
    dzB1=(xB(i)*yB(i)-8*zB(i)/3)*dtB;
%    dxB=(10*(yB(i)+(0.5*dyB1))-10*(xB(i)+(0.5*dxB1)))*dtB;
%    dyB=(28*(xB(i)+(0.5*dxB1))-(yB(i)+(0.5*dyB1))-
(xB(i)+0.5*dxB1)*(zB(i)+(0.5*dzB1)))*dtB;

```

Appendix B (Continued)

```

%      dzB=((xB(i)+(0.5*dxB1))*(yB(i)+(0.5*dyB1))-
8*(zB(i)+(0.5*dzB1))/3)*dtB;
xB(i+1)=xB(i)+dxB1;yB(i+1)=yB(i)+dyB1;zB(i+1)=zB(i)+dzB1;
end;
XAi=funcinterp(ti,tf,NA,NB,xA);
YAi=funcinterp(ti,tf,NA,NB,yA);
ZAi=funcinterp(ti,tf,NA,NB,zA);
err=functerr(XAi,xB,YAi,yB,ZAi,zB);
%Plotting for the first set of NA and NB values from the input
Matrices%
if k==1 ;
    figure (3)
    plot(tA, xA, tB,xB)
    title(['Lorenz Without Micro by Eulers Method, NA=', num2str(NA), '
NB=',num2str( NB), ' err=', num2str(err) ] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end
%Plotting for the second set of NA and NB values from the input
Matrices%
if k==2 ;
    figure (4)
    plot(tA,xA,'g',tB,xB,'r')
    title(['Lorenz Without Micro by Eulers Method, NA=', num2str(NA), '
NB=',num2str( NB), ' err=', num2str(err)] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end
%Plotting for the third set of NA and NB values from the input
Matrices%
if k==3 ;
    figure (5)
    plot(tA,xA,tB,xB)
    title(['Lorenz Without Micro by Eulers Method, NA=', num2str(NA), '
NB=',num2str( NB), ' err=', num2str(err)] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end
end
start=cputime;
%Re-Computing with the error limit%
NA=10000;E=0.005;stpp=1;
devEper=10;flucperNA=15;
mark=0;devE=devEper*0.01;
flucNA=flucperNA*0.01;
KLoop=1;KLoopMax=40;
while abs(stpp)>devE ; NB=2*NA;
    tA=(0:(NA-1))*(tf-ti)/(NA-1)+ti;%time vector%
    tB=(0:(NB-1))*(tf-ti)/(NB-1)+ti;%time vector%
    xA=1:NA;%Primitive xA array%
    yA=1:NA;%Primitive yA array%

```

Appendix B (Continued)

```

zA=1:NA;%Primitive zA array%
xB=1:NB;%Primitive xB array%
yB=1:NB;%Primitive yB array%
zB=1:NB;%Primitive zB array%
dtA=(tf-ti)/NA;%increment A%
dtB=(tf-ti)/NB;%increment B%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
for i=1:(NA-1);
    %The Integration%
    dxA1=(10*yA(i)-10*xA(i))*dtA;
    dyA1=(28*xA(i)-yA(i)-xA(i)*zA(i))*dtA;
    dzA1=(xA(i)*yA(i)-8*zA(i)/3)*dtA;
    %    dxA=(10*(yA(i)+(0.5*dyA1))-10*(xA(i)+(0.5*dxA1)))*dtA;
    %    dyA=(28*(xA(i)+(0.5*dxA1))-(yA(i)+(0.5*dyA1))-
    (xA(i)+0.5*dxA1)*(zA(i)+(0.5*dzA1)))*dtA;
    %    dzA=((xA(i)+(0.5*dxA1))*(yA(i)+(0.5*dyA1))-
    8*(zA(i)+(0.5*dzA1))/3)*dtA;
    xA(i+1)=xA(i)+dxA1;yA(i+1)=yA(i)+dyA1;zA(i+1)=zA(i)+dzA1;
end;
xB(1)=xi; %allocating initial values%
yB(1)=yi; %allocating initial values%
zB(1)=zi; %allocating initial values%
for i=1:(NB-1);
    %The Integration%
    dxB1=(10*yB(i)-10*xB(i))*dtB;
    dyB1=(28*xB(i)-yB(i)-xB(i)*zB(i))*dtB;
    dzB1=(xB(i)*yB(i)-8*zB(i)/3)*dtB;
    %    dxB=(10*(yB(i)+(0.5*dyB1))-10*(xB(i)+(0.5*dxB1)))*dtB;
    %    dyB=(28*(xB(i)+(0.5*dxB1))-(yB(i)+(0.5*dyB1))-
    (xB(i)+0.5*dxB1)*(zB(i)+(0.5*dzB1)))*dtB;
    %    dzB=((xB(i)+(0.5*dxB1))*(yB(i)+(0.5*dyB1))-
    8*(zB(i)+(0.5*dzB1))/3)*dtB;
    xB(i+1)=xB(i)+dxB1;yB(i+1)=yB(i)+dyB1;zB(i+1)=zB(i)+dzB1;
end;
XAi=funcinterp(ti,tf,NA,NB,xA);
YAi=funcinterp(ti,tf,NA,NB,yA);
ZAi=funcinterp(ti,tf,NA,NB,zA);
err=functerr(XAi,xB,YAi,yB,ZAi,zB);
stpp=(err-E)/E;

if stpp<0; % this implies the error is less than specification E
    if mark==0 % preset value needs to change to serve as flag error
specification crossed
        mark=1; % one time here
    end
    if abs(stpp)> devE; % in which case the measured deviation is
greater than specified error deviation
        NA=fix( NA-flucNA*NA); % decreasing NA should lower measured
error getting it closer to E
    end;
end
if stpp>0;% this implies the error is lgreater than specification E

```

Appendix B (Continued)

```
    if mark==0; % Check if flag has not been set to use NA increase
rule on fast track
        if abs(stpp)> devE; % only want ot increase NA if measured
deviaiton is greater than specified
            %NA=NA+ fix(NA/log(NA));
            NA=fix(1.5*NA );
        end
    end

    end
    if mark ==1 ;%
    if abs(stpp)> devE; % error is greater than apesified and flag
has been set to stop rapid increase NA
        NA=fix( NA+flucNA*NA);
    end
    end
end
KLoop=KLoop+1;
Kloop=KLoop-1
if KLoop == KLoopMax
    stop
end

end; % end on while condition based on abs(stpp)> devE;
figure (6)
    plot(tA, xA, tB,xB)
    title(['Lorenz Without Micro by Eulers Method, With error control
NA=', num2str(NA), ' NB=', num2str( NB), ' err=', num2str(err)] );
    xlabel('time in seconds')
    ylabel('XA')
    grid

cputime=cputime-start
epsilon=err
```

Appendix: B-2: Lorenz Equations by the Modified Euler's Method without the

Micro-Integrator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%This Program uses the Modified Euler Method to solve the Lorenz Non-
linear Chaotic Partial Differential Equation Set without the Micro-
Integrator%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
clc;
clear all;
close all;
%Pre-Integration Program Inputs%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
```


Appendix B (Continued)

```

zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=22; %upper time limit%
N=40000; %Resolution parameter%
dt=(tf-ti)/N;%integration step%
x=0:N; %primitive x array%
y=0:N; %primitive y array%
z=0:N; %primitive z array%
x(1)=xi; %allocating initial values%
y(1)=yi; %allocating initial values%
z(1)=zi; %allocating initial values%
for i=1:N;
    %The Integration%
    dx1=(10*y(i)-10*x(i))*dt;
    dy1=(28*x(i)-y(i)-x(i)*z(i))*dt;
    dz1=(x(i)*y(i)-8*z(i)/3)*dt;
    dx2=(10*(y(i)+dy1)-10*(x(i)+dx1))*dt;
    dy2=(28*(x(i)+dx1)-(y(i)+dy1)-(x(i)+dx1)*(z(i)+dz1))*dt;
    dz2=((x(i)+dx1)*(y(i)+dy1)-8*(z(i)+dz1)/3)*dt;
    dx=0.5*(dx1+dx2);
    dy=0.5*(dy1+dy2);
    dz=0.5*(dz1+dz2);
    x(i+1)=x(i)+dx;
    y(i+1)=y(i)+dy;
    z(i+1)=z(i)+dz;
end;
%plot of 2-D poincare map%
figure(1)
plot(x,y)
title('2D Poincare Map - Lorenz Without Micro - Modified Euler')
xlabel('x')
ylabel('y')
grid
%plot of 3-D poincare map%
figure(2)
plot3(x,y,z)
title('3D Poincare Map - Lorenz Without Micro - Modified Euler')
xlabel('x')
ylabel('y')
zlabel('z')
grid
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=22; %upper time limit%
NVectA=[1000,2000,4000]; %Matrix of NA values%
NVectB=2*NVectA; %Matrix of NB values%
%Loop to use all values in the NA and NB matrices%
for k = 1:3;
    NA=NVectA(k);%Selecting NA from the matrix%
    NB=NVectB(k);%Selecting NB from the matrix%
    tA=(0:(NA-1))*(tf-ti)/(NA-1)+ti;%time vector%
    tB=(0:(NB-1))*(tf-ti)/(NB-1)+ti;%time vector%

```

Appendix B (Continued)

```

xA=1:NA; %Primitive xA array%
yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xB=1:NB; %Primitive xB array%
yB=1:NB; %Primitive yB array%
zB=1:NB; %Primitive zB array%
dtA=(tf-ti)/NA;%time step A%
dtB=(tf-ti)/NB;%time step B%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
starttime=cputime;
for i=1:(NA-1);
    %The Integration%
    dxA1=(10*yA(i)-10*xA(i))*dtA;
    dyA1=(28*xA(i)-yA(i)-xA(i)*zA(i))*dtA;
    dzA1=(xA(i)*yA(i)-8*zA(i)/3)*dtA;
    dxA2=(10*(yA(i)+dyA1)-10*(xA(i)+dxA1))*dtA;
    dyA2=(28*(xA(i)+dxA1)-(yA(i)+dyA1)-(xA(i)+dxA1)*(zA(i)+dzA1))*dtA;
    dzA2=((xA(i)+dxA1)*(yA(i)+dyA1)-8*(zA(i)+dzA1)/3)*dtA;
    dxA=0.5*(dxA1+dxA2);
    dyA=0.5*(dyA1+dyA2);
    dzA=0.5*(dzA1+dzA2);
    xA(i+1)=xA(i)+dxA;
    yA(i+1)=yA(i)+dyA;
    zA(i+1)=zA(i)+dzA;
end;
finishtime=cputime;usedtime=finishtime-starttime;
xB(1)=xi; %allocating initial values%
yB(1)=yi; %allocating initial values%
zB(1)=zi; %allocating initial values%
for i=1:(NB-1);
    %The Integration%
    dxB1=(10*yB(i)-10*xB(i))*dtB;
    dyB1=(28*xB(i)-yB(i)-xB(i)*zB(i))*dtB;
    dzB1=(xB(i)*yB(i)-8*zB(i)/3)*dtB;
    dxB2=(10*(yB(i)+dyB1)-10*(xB(i)+dxB1))*dtB;
    dyB2=(28*(xB(i)+dxB1)-(yB(i)+dyB1)-(xB(i)+dxB1)*(zB(i)+dzB1))*dtB;
    dzB2=((xB(i)+dxB1)*(yB(i)+dyB1)-8*(zB(i)+dzB1)/3)*dtB;
    dxB=0.5*(dxB1+dxB2);
    dyB=0.5*(dyB1+dyB2);
    dzB=0.5*(dzB1+dzB2);
    xB(i+1)=xB(i)+dxB;
    yB(i+1)=yB(i)+dyB;
    zB(i+1)=zB(i)+dzB;
end;
XAi=funcinterp(ti,tf,NA,NB,xA);
YAi=funcinterp(ti,tf,NA,NB,yA);
ZAi=funcinterp(ti,tf,NA,NB,zA);
err=functerr(XAi,xB,YAi,yB,ZAi,zB);
%Plotting for the first set of NA and NB values from the input
Matrices%
if k==1 ;
    figure (3)

```

Appendix B (Continued)

```
plot(tA, xA, tB, xB)
title(['Lorenz Without Micro by Modified Euler Method, NA=',
num2str(NA), ' NB=', num2str( NB), ' err=', num2str(err) ] );
xlabel('time in seconds')
ylabel('XA')
grid
end
%Plotting for the second set of NA and NB values from the input
Matrices%
if k==2 ;
figure (4)
plot(tA, xA, 'm--', tB, xB)
title(['Lorenz Without Micro by Modified Euler Method, NA=',
num2str(NA), ' NB=', num2str( NB), ' err=', num2str(err) ] );
xlabel('time in seconds')
legend('integration time step=0.011', 'integration time step=
0.0055')
ylabel('XA')
grid
end
%Plotting for the third set of NA and NB values from the input
Matrices%
if k==3 ;
figure (5)
plot(tA, xA, tB, xB)
title(['Lorenz Without Micro by Modified Euler Method, NA=',
num2str(NA), ' NB=', num2str( NB), ' err=', num2str(err), '
cputime=', num2str(usedtime), 's' ] );
xlabel('time in seconds')
ylabel('XA')
grid
end
end
start=cputime;
%Re-Computing with the error limit%
NA=10000;E=0.005;stpp=1;
devEper=10;flucperNA=30;
mark=0;devE=devEper*0.01;
flucNA=flucperNA*0.01;
KLoop=1;KLoopMax=40;
while abs(stpp) > devE ;
NB=2*NA;
tA=(0:(NA-1))*(tf-ti)/(NA-1)+ti;%time vector%
tB=(0:(NB-1))*(tf-ti)/(NB-1)+ti;%time vector%
xA=1:NA;%Primitive xA array%
yA=1:NA;%Primitive yA array%
zA=1:NA;%Primitive zA array%
xB=1:NB;%Primitive xB array%
yB=1:NB;%Primitive yB array%
zB=1:NB;%Primitive zB array%
dtA=(tf-ti)/NA;%increment A%
dtB=(tf-ti)/NB;%increment B%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
```

Appendix B (Continued)

```

    zA(1)=zi; %allocating initial values%
    starttime1=cputime;
for i=1:(NA-1);
    %The Integration%
    dxA1=(10*yA(i)-10*xA(i))*dtA;
    dyA1=(28*xA(i)-yA(i)-xA(i)*zA(i))*dtA;
    dzA1=(xA(i)*yA(i)-8*zA(i)/3)*dtA;
    dxA2=(10*(yA(i)+dyA1)-10*(xA(i)+dxA1))*dtA;
    dyA2=(28*(xA(i)+dxA1)-(yA(i)+dyA1)-(xA(i)+dxA1)*(zA(i)+dzA1))*dtA;
    dzA2=((xA(i)+dxA1)*(yA(i)+dyA1)-8*(zA(i)+dzA1)/3)*dtA;
    dxA=0.5*(dxA1+dxA2);
    dyA=0.5*(dyA1+dyA2);
    dzA=0.5*(dzA1+dzA2);
    xA(i+1)=xA(i)+dxA;
    yA(i+1)=yA(i)+dyA;
    zA(i+1)=zA(i)+dzA;
end;
    finishtime1=cputime;
    usedtime1=finishtime1-starttime1;
    xB(1)=xi; %allocating initial values%
    yB(1)=yi; %allocating initial values%
    zB(1)=zi; %allocating initial values%
for i=1:(NB-1);
    %The Integration%
    dxB1=(10*yB(i)-10*xB(i))*dtB;
    dyB1=(28*xB(i)-yB(i)-xB(i)*zB(i))*dtB;
    dzB1=(xB(i)*yB(i)-8*zB(i)/3)*dtB;
    dxB2=(10*(yB(i)+dyB1)-10*(xB(i)+dxB1))*dtB;
    dyB2=(28*(xB(i)+dxB1)-(yB(i)+dyB1)-(xB(i)+dxB1)*(zB(i)+dzB1))*dtB;
    dzB2=((xB(i)+dxB1)*(yB(i)+dyB1)-8*(zB(i)+dzB1)/3)*dtB;
    dxB=0.5*(dxB1+dxB2);
    dyB=0.5*(dyB1+dyB2);
    dzB=0.5*(dzB1+dzB2);
    xB(i+1)=xB(i)+dxB;
    yB(i+1)=yB(i)+dyB;
    zB(i+1)=zB(i)+dzB;
end;
XAi=funcinterp(ti,tf,NA,NB,xA);
YAi=funcinterp(ti,tf,NA,NB,yA);
ZAi=funcinterp(ti,tf,NA,NB,zA);
err=functerr(XAi,xB,YAi,yB,ZAi,zB);
stpp=(err-E)/E;

if stpp<0; % this implies the error is less than specification E
    if mark==0 % preset value needs to change to serve as flag error
        specification crossed
            mark=1; % one time here
    end
    if abs(stpp)> devE; % in which case the measured deviation is
        greater than specified error deviation
        NA=fix( NA-flucNA*NA); % decreasing NA should lower measured
        error getting it closer to E
    end;
end
end

```

Appendix B (Continued)

```
if stpp>0;% this implies the error is lgreater than specification E
    if mark==0; % Check if flag has not been set to use NA increase
rule on fast track
        if abs(stpp)> devE; % only want ot increase NA if measured
deviaiton is greater than specified
            %NA=NA+ fix(NA/log(NA));
            NA=fix(1.5*NA );
        end

    end

    if mark ==1 ;%
    if abs(stpp)> devE; % error is greater than apecified and flag
has been set to stop rapid increase NA
        NA=fix( NA+flucNA*NA);
    end
    end
end
KLoop=KLoop+1;
Kloop=KLoop-1
if KLoop == KLoopMax
    stop
end

end; % end on while condition based on abs(stpp)> devE;
figure (6)
plot(tA, xA, '--', tB,xB)
title(['Lorenz Without Micro by Modified Euler Method, With error
control NA=', num2str(NA), ' NB=',num2str( NB), ' err=',
num2str(err)] );
xlabel('time in seconds')
ylabel('XA')
grid

cputime=cputime-start
epsilon=err
```

Appendix B-3: Lorenz Equations by Simpson's Method without the Micro-

Integrator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This Program uses the Simpson's Method to solve the Lorenz Non-linear
Chaotic Partial Differential Equation Set With The Micro-Integrator%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all
close all
%Pre-Integration Program Inputs%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
```

Appendix B (Continued)

```
ti=0; %lower time limit%
tf=20; %upper time limit%
N=40000; %Resolution parameter%
dt=(tf-ti)/N;%integration step%
x=0:N; %primitive x array%
y=0:N; %primitive y array%
z=0:N; %primitive z array%
x(1)=xi; %allocating initial values%
y(1)=yi; %allocating initial values%
z(1)=zi; %allocating initial values%
for i=1:N;
    %The Integration%
    dx1=(10*y(i)-10*x(i))*0.5*dt;
    dy1=(28*x(i)-y(i)-x(i)*z(i))*0.5*dt;
    dz1=(x(i)*y(i)-8*z(i)/3)*0.5*dt;
    xhf=x(i)+dx1;
    yhf=y(i)+dy1;
    zhf=z(i)+dz1;
    dx2=(10*yhf-10*xhf)*0.5*dt;
    dy2=(28*xhf-yhf-xhf*zhf)*0.5*dt;
    dz2=(xhf*yhf-8*zhf/3)*0.5*dt;
    xfl=xhf+dx2;
    yfl=yhf+dy2;
    zfl=zhf+dz2;
    dx3=(10*yfl-10*xfl)*0.5*dt;
    dy3=(28*xfl-yfl-xfl*zfl)*0.5*dt;
    dz3=(xfl*yfl-8*zfl/3)*0.5*dt;
    dx=(1/3)*(dx1+(4*dx2)+dx3);
    dy=(1/3)*(dy1+(4*dy2)+dy3);
    dz=(1/3)*(dz1+(4*dz2)+dz3);
    x(i+1)=x(i)+dx;
    y(i+1)=y(i)+dy;
    z(i+1)=z(i)+dz;
end;
%plot of 2-D poincare map%
figure(1)
plot(x,y)
title('Poincare Map - Lorenz Without Micro - Simpson')
xlabel('x')
ylabel('y')
grid
%plot of 3-D poincare map%
figure(2)
plot3(x,y,z)
title('3D Poincare Map - Lorenz Without Micro - Simpson')
xlabel('x')
ylabel('y')
zlabel('z')
grid
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=22; %upper time limit%
```

Appendix B (Continued)

```

NVectA=[1000,2000,4000]; %Matrix of NA values%
NVectB=2*NVectA;          %Matrix of NB values%
%Loop to use all values in the NA and NB matrices%
for k = 1:3;
    NA=NVectA(k); %Selecting NA from the matrix%
    NB=NVectB(k); %Selecting NB from the matrix%
    tA=(0:(NA-1))*(tf-ti)/(NA-1)+ti; %time vector%
    tB=(0:(NB-1))*(tf-ti)/(NB-1)+ti; %time vector%
    xA=1:NA; %Primitive xA array%
    yA=1:NA; %Primitive yA array%
    zA=1:NA; %Primitive zA array%
    xB=1:NB; %Primitive xB array%
    yB=1:NB; %Primitive yB array%
    zB=1:NB; %Primitive zB array%
    dtA=(tf-ti)/NA; %time step A%
    dtB=(tf-ti)/NB; %time step B%
    xA(1)=xi; %allocating initial values%
    yA(1)=yi; %allocating initial values%
    zA(1)=zi; %allocating initial values%
for i=1:(NA-1);
    %The Integration%
    dxA1=(10*yA(i)-10*xA(i))*0.5*dtA;
    dyA1=(28*xA(i)-yA(i)-xA(i)*zA(i))*0.5*dtA;
    dzA1=(xA(i)*yA(i)-8*zA(i)/3)*0.5*dtA;
    xAhf=xA(i)+dxA1;
    yAhf=yA(i)+dyA1;
    zAhf=zA(i)+dzA1;
    dxA2=(10*yAhf-10*xAhf)*0.5*dtA;
    dyA2=(28*xAhf-yAhf-xAhf*zAhf)*0.5*dtA;
    dzA2=(xAhf*yAhf-8*zAhf/3)*0.5*dtA;
    xAfl=xAhf+dxA2;
    yAfl=yAhf+dyA2;
    zAfl=zAhf+dzA2;
    dxA3=(10*yAfl-10*xAfl)*0.5*dtA;
    dyA3=(28*xAfl-yAfl-xAfl*zAfl)*0.5*dtA;
    dzA3=(xAfl*yAfl-8*zAfl/3)*0.5*dtA;
    dxA=(1/3)*(dxA1+(4*dxA2)+dxA3);
    dyA=(1/3)*(dyA1+(4*dyA2)+dyA3);
    dzA=(1/3)*(dzA1+(4*dzA2)+dzA3);
    xA(i+1)=xA(i)+dxA;
    yA(i+1)=yA(i)+dyA;
    zA(i+1)=zA(i)+dzA;
end;
    xB(1)=xi; %allocating initial values%
    yB(1)=yi; %allocating initial values%
    zB(1)=zi; %allocating initial values%
for i=1:(NB-1);
    %The Integration%
    dxB1=(10*yB(i)-10*xB(i))*0.5*dtB;
    dyB1=(28*xB(i)-yB(i)-xB(i)*zB(i))*0.5*dtB;
    dzB1=(xB(i)*yB(i)-8*zB(i)/3)*0.5*dtB;
    xBhf=xB(i)+dxB1;
    yBhf=yB(i)+dyB1;
    zBhf=zB(i)+dzB1;

```

Appendix B (Continued)

```
dxB2=(10*yBhf-10*xBhf)*0.5*dtB;
dyB2=(28*xBhf-yBhf-xBhf*zBhf)*0.5*dtB;
dzB2=(xBhf*yBhf-8*zBhf/3)*0.5*dtB;
xBf1=xBhf+dxB2;
yBf1=yBhf+dyB2;
zBf1=zBhf+dzB2;
dxB3=(10*yBf1-10*xBf1)*0.5*dtB;
dyB3=(28*xBf1-yBf1-xBf1*zBf1)*0.5*dtB;
dzB3=(xBf1*yBf1-8*zBf1/3)*0.5*dtB;
dxB=(1/3)*(dxB1+(4*dxB2)+dxB3);
dyB=(1/3)*(dyB1+(4*dyB2)+dyB3);
dzB=(1/3)*(dzB1+(4*dzB2)+dzB3);
xB(i+1)=xB(i)+dxB;
yB(i+1)=yB(i)+dyB;
zB(i+1)=zB(i)+dzB;
end;
XAi=funcinterp(ti,tf,NA,NB,xA);
YAi=funcinterp(ti,tf,NA,NB,yA);
ZAi=funcinterp(ti,tf,NA,NB,zA);
err=functerr(XAi,xB,YAi,yB,ZAi,zB);
%Plotting for the first set of NA and NB values from the input
Matrices%
if k==1 ;
    figure (3)
    plot(tA, xA, tB,xB)
    title(['Lorenz Without Micro by Simpsons Method, NA=', num2str(NA),
' NB=',num2str( NB), ' err=', num2str(err) ] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end
%Plotting for the second set of NA and NB values from the input
Matrices%
if k==2 ;
    figure (4)
    plot(tA,xA,'--',tB,xB)
    title(['Lorenz Without Micro by Simpsons Method, NA=', num2str(NA),
' NB=',num2str( NB), ' err=', num2str(err)] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end
%Plotting for the third set of NA and NB values from the input
Matrices%
if k==3 ;
    figure (5)
    plot(tA,xA,tB,xB)
    title(['Lorenz Without Micro by Simpsons Method, NA=', num2str(NA),
' NB=',num2str( NB), ' err=', num2str(err)] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end
end
```


Appendix B (Continued)

```

start=cputime;
%Re-Computing with the error limit%
NA=10000;E=0.005;stpp=1;
devEper=10;flucperNA=30;
mark=0;devE=devEper*0.01;
flucNA=flucperNA*0.01;
KLoop=1;KLoopMax=40;
while abs(stpp) > devE ;
    NB=2*NA;
    tA=(0:(NA-1))*(tf-ti)/(NA-1)+ti;%time vector%
    tB=(0:(NB-1))*(tf-ti)/(NB-1)+ti;%time vector%
    xA=1:NA;%Primitive xA array%
    yA=1:NA;%Primitive yA array%
    zA=1:NA;%Primitive zA array%
    xB=1:NB;%Primitive xB array%
    yB=1:NB;%Primitive yB array%
    zB=1:NB;%Primitive zB array%
    dtA=(tf-ti)/NA;%increment A%
    dtB=(tf-ti)/NB;%increment B%
    xA(1)=xi; %allocating initial values%
    yA(1)=yi; %allocating initial values%
    zA(1)=zi; %allocating initial values%
    starttimel=cputime;
for i=1:(NA-1);
    %The Integration%
    dxA1=(10*yA(i)-10*xA(i))*0.5*dtA;
    dyA1=(28*xA(i)-yA(i)-xA(i)*zA(i))*0.5*dtA;
    dzA1=(xA(i)*yA(i)-8*zA(i)/3)*0.5*dtA;
    xAhf=xA(i)+dxA1;
    yAhf=yA(i)+dyA1;
    zAhf=zA(i)+dzA1;
    dxA2=(10*yAhf-10*xAhf)*0.5*dtA;
    dyA2=(28*xAhf-yAhf-xAhf*zAhf)*0.5*dtA;
    dzA2=(xAhf*yAhf-8*zAhf/3)*0.5*dtA;
    xAfl=xAhf+dxA2;
    yAfl=yAhf+dyA2;
    zAfl=zAhf+dzA2;
    dxA3=(10*yAfl-10*xAfl)*0.5*dtA;
    dyA3=(28*xAfl-yAfl-xAfl*zAfl)*0.5*dtA;
    dzA3=(xAfl*yAfl-8*zAfl/3)*0.5*dtA;
    dxA=(1/3)*(dxA1+(4*dxA2)+dxA3);
    dyA=(1/3)*(dyA1+(4*dyA2)+dyA3);
    dzA=(1/3)*(dzA1+(4*dzA2)+dzA3);
    xA(i+1)=xA(i)+dxA;
    yA(i+1)=yA(i)+dyA;
    zA(i+1)=zA(i)+dzA;
end;
    finishtimel=cputime;
    usedtimel=finishtimel-starttimel;
    xB(1)=xi; %allocating initial values%
    yB(1)=yi; %allocating initial values%
    zB(1)=zi; %allocating initial values%
for i=1:(NB-1);
    %The Integration%

```

Appendix B (Continued)

```

dxB1=(10*yB(i)-10*xB(i))*0.5*dtB;
dyB1=(28*xB(i)-yB(i)-xB(i)*zB(i))*0.5*dtB;
dzB1=(xB(i)*yB(i)-8*zB(i)/3)*0.5*dtB;
xBhf=xB(i)+dxB1;
yBhf=yB(i)+dyB1;
zBhf=zB(i)+dzB1;
dxB2=(10*yBhf-10*xBhf)*0.5*dtB;
dyB2=(28*xBhf-yBhf-xBhf*zBhf)*0.5*dtB;
dzB2=(xBhf*yBhf-8*zBhf/3)*0.5*dtB;
xBf1=xBhf+dxB2;
yBf1=yBhf+dyB2;
zBf1=zBhf+dzB2;
dxB3=(10*yBf1-10*xBf1)*0.5*dtB;
dyB3=(28*xBf1-yBf1-xBf1*zBf1)*0.5*dtB;
dzB3=(xBf1*yBf1-8*zBf1/3)*0.5*dtB;
dxB=(1/3)*(dxB1+(4*dxB2)+dxB3);
dyB=(1/3)*(dyB1+(4*dyB2)+dyB3);
dzB=(1/3)*(dzB1+(4*dzB2)+dzB3);
xB(i+1)=xB(i)+dxB;
yB(i+1)=yB(i)+dyB;
zB(i+1)=zB(i)+dzB;
end;
XAi=funcinterp(ti,tf,NA,NB,xA);
YAi=funcinterp(ti,tf,NA,NB,yA);
ZAi=funcinterp(ti,tf,NA,NB,zA);
err=functerr(XAi,xB,YAi,yB,ZAi,zB);
stpp=(err-E)/E;
%Plot for the correct Error Limit%

if stpp<0; % this implies the error is less than specification E
    if mark==0 % preset value needs to change to serve as flag error
specification crossed
        mark=1; % one time here
    end
    if abs(stpp)> devE; % in which case the measured deviation is
greater than specified error deviation
        NA=fix( NA-flucNA*NA); % decreasing NA should lower measured
error getting it closer to E
    end;
end
if stpp>0;% this implies the error is lgreater than specification E
    if mark==0; % Check if flag has not been set to use NA increase
rule on fast track
        if abs(stpp)> devE; % only want ot increase NA if measured
deviaiton is greater than specified
            %NA=NA+ fix(NA/log(NA));
            NA=fix(1.5*NA );
        end
    end
    if mark ==1 ;%
    if abs(stpp)> devE; % error is greater than apesified and flag
has been set to stop rapid increase NA
        NA=fix( NA+flucNA*NA);
    end
end

```

Appendix B (Continued)

```
    end
end
KLoop=KLoop+1;
Kloop=KLoop-1
if KLoop == KLoopMax
    stop
end

end; % end on while condition based on abs(stpp)> devE;
figure (6)
    plot(tA, xA, tB,xB)
    title(['Lorenz Without Micro by Simpsons Method, With error control
NA=', num2str(NA), ' NB=',num2str( NB), ' err=', num2str(err)] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
cputime=cputime-start
epsilon=err
```

Appendix B-4: Lorenz Equations by Runge-Kutta's Method without the Micro-Integrator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This Program uses the 4th Order Runge-Kutta Method to solve the Lorenz
Non-linear Chaotic Partial Differential Equation Set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;
close all;
%Pre-Integration Program Inputs%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=20; %upper time limit%
N=40000; %Resolution parameter%
dt=(tf-ti)/N;%integration step%
x=0:N; %primitive x array%
y=0:N; %primitive y array%
z=0:N; %primitive z array%
x(1)=xi; %allocating initial values%
y(1)=yi; %allocating initial values%
z(1)=zi; %allocating initial values%
for i=1:N;
    %The Integration%
    dx1=(10*y(i)-10*x(i))*dt;
    dy1=(28*x(i)-y(i)-x(i)*z(i))*dt;
    dz1=(x(i)*y(i)-8*z(i)/3)*dt;
    dx2=(10*(y(i)+(0.5*dy1))-10*(x(i)+(0.5*dx1)))*dt;
```

Appendix B (Continued)

```

dy2=(28*(x(i)+(0.5*dx1))-(y(i)+(0.5*dy1))-
(x(i)+(0.5*dx1))*(z(i)+(0.5*dz1)))*dt;
dz2=((x(i)+(0.5*dx1))*(y(i)+(0.5*dy1))-8*(z(i)+(0.5*dz1))/3)*dt;
dx3=(10*(y(i)+(0.5*dy2))-10*(x(i)+(0.5*dx2)))*dt;
dy3=(28*(x(i)+(0.5*dx2))-(y(i)+(0.5*dy2))-
(x(i)+(0.5*dx2))*(z(i)+(0.5*dz2)))*dt;
dz3=((x(i)+(0.5*dx2))*(y(i)+(0.5*dy2))-8*(z(i)+(0.5*dz2))/3)*dt;
dx4=(10*(y(i)+(dy3))-10*(x(i)+(dx3)))*dt;
dy4=(28*(x(i)+(dx3))-(y(i)+(dy3))-(x(i)+(dx3))*(z(i)+(dz3)))*dt;
dz4=((x(i)+(dx3))*(y(i)+(dy3))-8*(z(i)+(dz3))/3)*dt;
dx=(1/6)*(dx1+2*(dx2+dx3)+dx4);
dy=(1/6)*(dy1+2*(dy2+dy3)+dy4);
dz=(1/6)*(dz1+2*(dz2+dz3)+dz4);
x(i+1)=x(i)+dx;
y(i+1)=y(i)+dy;
z(i+1)=z(i)+dz;

end;
%plot of 2-D poincare map%
figure(1)
plot(x,y)
title('1D Poincare Map - Lorenz Without Micro - Runge-Kutta')
xlabel('x')
ylabel('y')
grid
%plot of 3-D poincare map%
figure(2)
plot3(x,y,z)
title('3D Poincare Map - Lorenz Without Micro - Runge-Kutta')
xlabel('x')
ylabel('y')
zlabel('z')
grid
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=22; %upper time limit%
NVectA=[1000,2000,4000]; %Matrix of NA values%
NVectB=2*NVectA; %Matrix of NB values%
for k = 1:3;
NA=NVectA(k);%Selecting NA from the matrix%
NB=NVectB(k);%Selecting NB from the matrix%
tA=(0:(NA-1))*(tf-ti)/(NA-1)+ti;%time vector%
tB=(0:(NB-1))*(tf-ti)/(NB-1)+ti;%time vector%
xA=1:NA; %Primitive xA array%
yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xB=1:NB; %Primitive xB array%
yB=1:NB; %Primitive yB array%
zB=1:NB; %Primitive zB array%
dtA=(tf-ti)/NA;%time step A%
dtB=(tf-ti)/NB;%time step B%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%

```

Appendix B (Continued)

```

zA(1)=zi; %allocating initial values%
starttime=cputime;
for i=1:(NA-1);
%The Integration%
dxA1=(10*yA(i)-10*xA(i))*dtA;
dyA1=(28*xA(i)-yA(i)-xA(i)*zA(i))*dtA;
dzA1=(xA(i)*yA(i)-8*zA(i)/3)*dtA;
dxA2=(10*(yA(i)+(0.5*dyA1))-10*(xA(i)+(0.5*dxA1)))*dtA;
dyA2=(28*(xA(i)+(0.5*dxA1))-(yA(i)+(0.5*dyA1))-
(xA(i)+(0.5*dxA1))*(zA(i)+(0.5*dzA1)))*dtA;
dzA2=((xA(i)+(0.5*dxA1))*(yA(i)+(0.5*dyA1))-
8*(zA(i)+(0.5*dzA1))/3)*dtA;
dxA3=(10*(yA(i)+(0.5*dyA2))-10*(xA(i)+(0.5*dxA2)))*dtA;
dyA3=(28*(xA(i)+(0.5*dxA2))-(yA(i)+(0.5*dyA2))-
(xA(i)+(0.5*dxA2))*(zA(i)+(0.5*dzA2)))*dtA;
dzA3=((xA(i)+(0.5*dxA2))*(yA(i)+(0.5*dyA2))-
8*(zA(i)+(0.5*dzA2))/3)*dtA;
dxA4=(10*(yA(i)+(dyA3))-10*(xA(i)+(dxA3)))*dtA;
dyA4=(28*(xA(i)+(dxA3))-(yA(i)+(dyA3))-
(xA(i)+(dxA3))*(zA(i)+(dzA3)))*dtA;
dzA4=((xA(i)+(dxA3))*(yA(i)+(dyA3))-8*(zA(i)+(dzA3))/3)*dtA;
dxA=(1/6)*(dxA1+2*(dxA2+dxA3)+dxA4);
dyA=(1/6)*(dyA1+2*(dyA2+dyA3)+dyA4);
dzA=(1/6)*(dzA1+2*(dzA2+dzA3)+dzA4);
xA(i+1)=xA(i)+dxA;
yA(i+1)=yA(i)+dyA;
zA(i+1)=zA(i)+dzA;
end;
finishtime=cputime;usedtime=finishtime-starttime;
xB(1)=xi; %allocating initial values%
yB(1)=yi; %allocating initial values%
zB(1)=zi; %allocating initial values%
for i=1:(NB-1);
%The Integrator%
dxB1=(10*yB(i)-10*xB(i))*dtB;
dyB1=(28*xB(i)-yB(i)-xB(i)*zB(i))*dtB;
dzB1=(xB(i)*yB(i)-8*zB(i)/3)*dtB;
dxB2=(10*(yB(i)+(0.5*dyB1))-10*(xB(i)+(0.5*dxB1)))*dtB;
dyB2=(28*(xB(i)+(0.5*dxB1))-(yB(i)+(0.5*dyB1))-
(xB(i)+(0.5*dxB1))*(zB(i)+(0.5*dzB1)))*dtB;
dzB2=((xB(i)+(0.5*dxB1))*(yB(i)+(0.5*dyB1))-
8*(zB(i)+(0.5*dzB1))/3)*dtB;
dxB3=(10*(yB(i)+(0.5*dyB2))-10*(xB(i)+(0.5*dxB2)))*dtB;
dyB3=(28*(xB(i)+(0.5*dxB2))-(yB(i)+(0.5*dyB2))-
(xB(i)+(0.5*dxB2))*(zB(i)+(0.5*dzB2)))*dtB;
dzB3=((xB(i)+(0.5*dxB2))*(yB(i)+(0.5*dyB2))-
8*(zB(i)+(0.5*dzB2))/3)*dtB;
dxB4=(10*(yB(i)+(dyB3))-10*(xB(i)+(dxB3)))*dtB;
dyB4=(28*(xB(i)+(dxB3))-(yB(i)+(dyB3))-
(xB(i)+(dxB3))*(zB(i)+(dzB3)))*dtB;
dzB4=((xB(i)+(dxB3))*(yB(i)+(dyB3))-8*(zB(i)+(dzB3))/3)*dtB;
dxB=(1/6)*(dxB1+2*(dxB2+dxB3)+dxB4);
dyB=(1/6)*(dyB1+2*(dyB2+dyB3)+dyB4);
dzB=(1/6)*(dzB1+2*(dzB2+dzB3)+dzB4);

```

Appendix B (Continued)

```
    xB(i+1)=xB(i)+dxB;
    yB(i+1)=yB(i)+dyB;
    zB(i+1)=zB(i)+dzB;
end;
XAi=funcinterp(ti,tf,NA,NB,xA);
YAi=funcinterp(ti,tf,NA,NB,yA);
ZAi=funcinterp(ti,tf,NA,NB,zA);
err=functerr(XAi,xB,YAi,yB,ZAi,zB);
%Plotting for the first set of NA and NB values from the input
Matrices%
if k==1 ;
    figure (3)
    plot(tA, xA, tB,xB)
    title(['Lorenz Without Micro by Runge-Kutta Method, NA=',
num2str(NA), ' NB=',num2str( NB), ' err=', num2str(err) ] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end
%Plotting for the second set of NA and NB values from the input
Matrices%
if k==2 ;
    figure (4)
    plot(tA,xA,tB,xB)
    title(['Lorenz Without Micro by Runge-Kutta Method, NA=',
num2str(NA), ' NB=',num2str( NB), ' err=', num2str(err) ] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end
%Plotting for the third set of NA and NB values from the input
Matrices%
if k==3 ;
    figure (5)
    plot(tA,xA,tB,xB)
    title(['Lorenz Without Micro by Runge-Kutta Method, NA=',
num2str(NA), ' NB=',num2str( NB), ' err=', num2str(err) ] );
    xlabel('time in seconds')
    ylabel('XA')
    grid
end

end
start=cputime;
%Re-Computing with the error limit%
NA=10000;E=0.005;stpp=1;
devEper=10;flucperNA=30;
mark=0;devE=devEper*0.01;
flucNA=flucperNA*0.01;
KLoop=1;KLoopMax=40;
while abs(stpp)>devE;NB=2*NA;
    tA=(0:(NA-1))*(tf-ti)/(NA-1)+ti;%time vector%
    tB=(0:(NB-1))*(tf-ti)/(NB-1)+ti;%time vector%
    xA=1:NA;%Primitive xA array%
```

Appendix B (Continued)

```

yA=1:NA;%Primitive yA array%
zA=1:NA;%Primitive zA array%
xB=1:NA;%Primitive xB array%
yB=1:NA;%Primitive yB array%
zB=1:NA;%Primitive zB array%
dtA=(tf-ti)/NA;%increment A%
dtB=(tf-ti)/NB;%increment B%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
starttime1=cputime;
for i=1:(NA-1);
    %The Integration%
    dxA1=(10*yA(i)-10*xA(i))*dtA;
    dyA1=(28*xA(i)-yA(i)-xA(i)*zA(i))*dtA;
    dzA1=(xA(i)*yA(i)-8*zA(i)/3)*dtA;
    dxA2=(10*(yA(i)+(0.5*dyA1))-10*(xA(i)+(0.5*dxA1)))*dtA;
    dyA2=(28*(xA(i)+(0.5*dxA1))-(yA(i)+(0.5*dyA1))-
(xA(i)+(0.5*dxA1))*(zA(i)+(0.5*dzA1)))*dtA;
    dzA2=((xA(i)+(0.5*dxA1))*(yA(i)+(0.5*dyA1))-
8*(zA(i)+(0.5*dzA1))/3)*dtA;
    dxA3=(10*(yA(i)+(0.5*dyA2))-10*(xA(i)+(0.5*dxA2)))*dtA;
    dyA3=(28*(xA(i)+(0.5*dxA2))-(yA(i)+(0.5*dyA2))-
(xA(i)+(0.5*dxA2))*(zA(i)+(0.5*dzA2)))*dtA;
    dzA3=((xA(i)+(0.5*dxA2))*(yA(i)+(0.5*dyA2))-
8*(zA(i)+(0.5*dzA2))/3)*dtA;
    dxA4=(10*(yA(i)+(dyA3))-10*(xA(i)+(dxA3)))*dtA;
    dyA4=(28*(xA(i)+(dxA3))-(yA(i)+(dyA3))-
(xA(i)+(dxA3))*(zA(i)+(dzA3)))*dtA;
    dzA4=((xA(i)+(dxA3))*(yA(i)+(dyA3))-8*(zA(i)+(dzA3))/3)*dtA;
    dxA=(1/6)*(dxA1+2*(dxA2+dxA3)+dxA4);
    dyA=(1/6)*(dyA1+2*(dyA2+dyA3)+dyA4);
    dzA=(1/6)*(dzA1+2*(dzA2+dzA3)+dzA4);
    xA(i+1)=xA(i)+dxA;
    yA(i+1)=yA(i)+dyA;
    zA(i+1)=zA(i)+dzA;
end;
finishtime1=cputime;
usedtime1=finishtime1-starttime1;
xB(1)=xi; %allocating initial values%
yB(1)=yi; %allocating initial values%
zB(1)=zi; %allocating initial values%
for i=1:(NB-1);
    %The Integration%
    dxB1=(10*yB(i)-10*xB(i))*dtB;
    dyB1=(28*xB(i)-yB(i)-xB(i)*zB(i))*dtB;
    dzB1=(xB(i)*yB(i)-8*zB(i)/3)*dtB;
    dxB2=(10*(yB(i)+(0.5*dyB1))-10*(xB(i)+(0.5*dxB1)))*dtB;
    dyB2=(28*(xB(i)+(0.5*dxB1))-(yB(i)+(0.5*dyB1))-
(xB(i)+(0.5*dxB1))*(zB(i)+(0.5*dzB1)))*dtB;
    dzB2=((xB(i)+(0.5*dxB1))*(yB(i)+(0.5*dyB1))-
8*(zB(i)+(0.5*dzB1))/3)*dtB;
    dxB3=(10*(yB(i)+(0.5*dyB2))-10*(xB(i)+(0.5*dxB2)))*dtB;

```

Appendix B (Continued)

```

dyB3=(28*(xB(i)+(0.5*dxB2))-(yB(i)+(0.5*dyB2))-
(xB(i)+(0.5*dxB2))*(zB(i)+(0.5*dzB2)))*dtB;
dzB3=((xB(i)+(0.5*dxB2))*(yB(i)+(0.5*dyB2))-
8*(zB(i)+(0.5*dzB2))/3)*dtB;
dxB4=(10*(yB(i)+(dyB3))-10*(xB(i)+(dxB3)))*dtB;
dyB4=(28*(xB(i)+(dxB3))-(yB(i)+(dyB3))-
(xB(i)+(dxB3))*(zB(i)+(dzB3)))*dtB;
dzB4=((xB(i)+(dxB3))*(yB(i)+(dyB3))-8*(zB(i)+(dzB3))/3)*dtB;
dxB=(1/6)*(dxB1+2*(dxB2+dxB3)+dxB4);
dyB=(1/6)*(dyB1+2*(dyB2+dyB3)+dyB4);
dzB=(1/6)*(dzB1+2*(dzB2+dzB3)+dzB4);
xB(i+1)=xB(i)+dxB;
yB(i+1)=yB(i)+dyB;
zB(i+1)=zB(i)+dzB;
end;
XAi=funcinterp(ti,tf,NA,NB,xA);
YAi=funcinterp(ti,tf,NA,NB,yA);
ZAi=funcinterp(ti,tf,NA,NB,zA);
err=functerr(XAi,xB,YAi,yB,ZAi,zB);
stpp=(err-E)/E;
%Plot for the correct Error Limit%

if stpp<0; % this implies the error is less than specification E
    if mark==0 % preset value needs to change to serve as flag error
specification crossed
        mark=1; % one time here
    end
    if abs(stpp)> devE; % in which case the measured deviation is
greater than specified error deviation
        NA=fix( NA-flucNA*NA); % decreasing NA should lower measured
error getting it closer to E
    end;
end

if stpp>0;% this implies the error is lgreater than specification E
    if mark==0; % Check if flag has not been set to use NA increse
rule on fast track
        if abs(stpp)> devE; % only want ot increase NA if measured
deviaiton is greater than specified
            %NA=NA+ fix(NA/log(NA));
            NA=fix(1.5*NA );
        end
    end

    if mark ==1 ;%
        if abs(stpp)> devE; % error is greater than apecified and flag
has been set to stop rapid increase NA
            NA=fix( NA+flucNA*NA);
        end
    end
end
KLoop=KLoop+1;
Kloop=KLoop-1
if KLoop == KLoopMax

```


Appendix B (Continued)

```
        stop
    end

end; % end on while condition based on abs(stpp)> devE;
figure (6)
    plot(tA, xA, tB,xB)
    title(['Lorenz Without Micro by Runge-Kutta Method, With error
control NA=', num2str(NA), ' NB=', num2str( NB), ' err=',
num2str(err)] );
    xlabel('time in seconds')
    ylabel('XA')
    grid

cputime=cputime-start
epsilon=err
```

Appendix B-5: Lorenz Equations by Euler's Method with the Micro-Integrator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This Program uses the Eulers Method to solve the Lorenz Non-linear
Chaotic Partial Differential Equation Set With The Micro-Integrator%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;
close all;
%Pre-Integration Program Inputs%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=20; %upper time limit%
N=40000; %Resolution parameter%
dt=(tf-ti)/N;%integration step%
x=0:N; %primitive x array%
y=0:N; %primitive y array%
z=0:N; %primitive z array%
x(1)=xi; %allocating initial values%
y(1)=yi; %allocating initial values%
z(1)=zi; %allocating initial values%
for i=1:N;
    %The Integration%
    dx1=(10*y(i)-10*x(i))*dt;
    dy1=(28*x(i)-y(i)-x(i)*z(i))*dt;
    dz1=(x(i)*y(i)-8*z(i)/3)*dt;
    %
    dx=(10*(y(i)+(0.5*dy1))-10*(x(i)+(0.5*dx1)))*dt;
    %
    dy=(28*(x(i)+(0.5*dx1))-(y(i)+(0.5*dy1))-
(x(i)+0.5*dx1)*(z(i)+(0.5*dz1)))*dt;
    %
    dz=((x(i)+(0.5*dx1))*(y(i)+(0.5*dy1))-8*(z(i)+(0.5*dz1))/3)*dt;
    x(i+1)=x(i)+dx1;
    y(i+1)=y(i)+dy1;
    z(i+1)=z(i)+dz1;
end
```

Appendix B (Continued)

```
end;
%plot of 2-D poincare map%
figure(1)
plot(x,y)
title('2D Poincare Map - Lorenz With Micro - Eulers Method')
xlabel('x')
ylabel('y')
grid
%plot of 3-D poincare map%
figure(2)
plot3(x,y,z)
title('3D Poincare Map - Lorenz With Micro - Eulers Method')
xlabel('x')
ylabel('y')
zlabel('z')
grid
clear
start=cputime;
%Computing with error/10%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=8.1; %upper time limit%
E1=0.005; %Error Limit%
N_resI=2;%Initial micro resolution%
EHR=E1/10;%error/10%
NA=256; %Resolution Parameter%
clear xA yA zA xB yB zB
t=(0:(NA-1))*(tf-ti)/(NA-1)+ti;
xA=1:NA; %Primitive xA array%
yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xB=1:NA; %Primitive xB array%
yB=1:NA; %Primitive yB array%
zB=1:NA; %Primitive zB array%
dt=(tf-ti)/(NA-1);%increment%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
xB(1)=xi; %allocating initial values%
yB(1)=yi; %allocating initial values%
zB(1)=zi; %allocating initial values%
N_resA=N_resI;%Initial micro resolution%
epsilon=2*E1; %defining epsilon%
loop=1; %Initializing the first loop count parameter%
%The Micro-Integrator%
while epsilon > EHR %error constraint loop%
starttime=cputime;
    for i=1:(NA-1);
        u=xA(i);%creating the first dynamic variable %
        v=yA(i);%creating the second dynamic variable%
        w=zA(i);%creating the third dynamic variable %
        delta = (t(i+1)-t(i))/(N_resA);
```

Appendix B (Continued)

```

    for j=1:N_resA;
        ui=u;vi=v;wi=w;
        %The Integration%
        du1=(10*v-10*u)*delta;
        dv1=(28*u-v-u*w)*delta;
        dw1=(u*v-8*w/3)*delta;
    %
        du=(10*(v+(0.5*dv1))-10*(u+(0.5*du1)))*delta;
    %
        dv=(28*(u+(0.5*du1))-(v+(0.5*dv1))-
(u+(0.5*du1))*(w+(0.5*dw1)))*delta;
    %
        dw=((u+(0.5*du1))*(v+(0.5*dv1))-8*(w+(0.5*dw1))/3)*delta;
        u_new=ui+du1;v_new=vi+dv1;w_new=wi+dw1;
        u=u_new;v=v_new;w=w_new;
    end;
    xA(i+1)=u;yA(i+1)=v;zA(i+1)=w;
end;
finishtime=cputime;usedtime=finishtime-starttime;
N_resB=2*N_resA;
for i=1:(NA-1);
    u=xB(i);%creating the first dynamic variable %
    v=yB(i);%creating the second dynamic variable%
    w=zB(i);%creating the third dynamic variable %
    delta = (t(i+1)-t(i))/(N_resB);
    for j=1:N_resB;
        ui=u;vi=v;wi=w;
        %The Integration%
        du1=(10*v-10*u)*delta;
        dv1=(28*u-v-u*w)*delta;
        dw1=(u*v-8*w/3)*delta;
    %
        du=(10*(v+(0.5*dv1))-10*(u+(0.5*du1)))*delta;
    %
        dv=(28*(u+(0.5*du1))-(v+(0.5*dv1))-
(u+(0.5*du1))*(w+(0.5*dw1)))*delta;
    %
        dw=((u+(0.5*du1))*(v+(0.5*dv1))-8*(w+(0.5*dw1))/3)*delta;
        u_new=ui+du1;v_new=vi+dv1;w_new=wi+dw1;
        u=u_new;v=v_new;w=w_new;
    end;
    xB(i+1)=u;yB(i+1)=v;zB(i+1)=w;
end;
epsilon=functerr(xB,xA,yB,yA,zB,zA);
N_resA=2*N_resA;
loop=loop+1; %Incremental setup for the loop count%
loop_1=loop-1 %correcting the forward lag of +1 in the loop count%
end;
N_resA=0.5*N_resA;
figure(3)
plot( t, xA, t,xB)
title(['Lorenz With Micro Eulers Method with error/10 NA=',
num2str(NA), ' NresA=',num2str( N_resA),' error=', num2str(epsilon)]
);
xlabel('time in seconds')
ylabel('XA')
grid
%computing with actual error%
E=E1; %defining the error parameter%
flucperNAres=30;%fluctuation in N_resA per loop%

```

Appendix B (Continued)

```

devEper=10;           %percentage error deviation allowed%
Kloop=1;             %Initializing the second loop count parameter%
KloopMax=25;        %maximum number of loops before program termination%
devE=devEper*0.01;  flucNAres=flucperNAres*0.01;
t=(0:(NA-1))*(tf-ti)/(NA-1)+ti; stpp=(epsilon-E)/E;
xB1=xB; yB1=yB; zB1=zB; xAhold=xB; yAhold=yB; zAhold=zB;
NAresAhold=N_resA; mark=0; X=1;
while abs(stpp)>devE;
    if stpp>0;
        N_resA=N_resA+fix(flucNAres*N_resA);
    else
        N_resA=N_resA-fix(flucNAres*N_resA);
    end;
clear xA yA zA
xA=1:NA; %Primitive xA array% yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xA(1)=xi; %allocating initial values% yA(1)=yi; %allocating initial
values% zA(1)=zi; %allocating initial values%
    for i=1:(NA-1);
        u=xA(i); %creating the first dynamic variable %
        v=yA(i); %creating the second dynamic variable%
        w=zA(i); %creating the third dynamic variable %
        delta=(t(i+1)-t(i))/(N_resA);
        for j=1:N_resA;
            ui=u; vi=v; wi=w;
            %The Integration%
            du1=(10*v-10*u)*delta;
            dv1=(28*u-v-u*w)*delta;
            dw1=(u*v-8*w/3)*delta;
            %
            du=(10*(v+(0.5*dv1))-10*(u+(0.5*du1)))*delta;
            %
            dv=(28*(u+(0.5*du1))-(v+(0.5*dv1)))-
            (u+(0.5*du1))*(w+(0.5*dw1))*delta;
            %
            dw=((u+(0.5*du1))*(v+(0.5*dv1))-8*(w+(0.5*dw1))/3)*delta;
            u_new=ui+du1; v_new=vi+dv1; w_new=wi+dw1;
            u=u_new; v=v_new; w=w_new;
        end;
        xA(i+1)=u; yA(i+1)=v; zA(i+1)=w;
    end;
    epsilon=functerr(xA, xAhold, yA, yAhold, zA, zAhold); % here the A
array is smaller
    Kloop=Kloop+1; %Incremental setup for the loop count%
    loop_2=Kloop-1 %correcting the forward lag of +1 in the loop
count%
    if Kloop >KloopMax;
        stop
    end
stpp=(epsilon-E)/E;
end;
figure(4)
plot(t, xA, t, xB)
title(['Lorenz With Micro Eulers Method With error control NA=',
num2str(NA), ' NresA=', num2str(N_resA), ' error=', num2str(epsilon)]
);
xlabel('time in seconds')

```

Appendix B (Continued)

```
ylabel('XA')
grid
cputime=cputime-start
loop_1
loop_2
epsilon
```

Appendix B-6: Lorenz Equations by the Modified Euler's Method with the Micro-Integrator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This Program uses the Modified Euler Method to solve the Lorenz Non-
linear Chaotic Partial Differential Equation Set With The Micro-
Integrator%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all
close all
%Pre-Integration Program Inputs%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=20; %upper time limit%
N=40000; %Resolution parameter%
dt=(tf-ti)/N;%integration step%
x=0:N; %primitive x array%
y=0:N; %primitive y array%
z=0:N; %primitive z array%
x(1)=xi; %allocating initial values%
y(1)=yi; %allocating initial values%
z(1)=zi; %allocating initial values%
for i=1:N;
    %The Integration%
    dx=(10*y(i)-10*x(i))*dt;
    dy=(28*x(i)-y(i)-x(i)*z(i))*dt;
    dz=(x(i)*y(i)-8*z(i)/3)*dt;
    x(i+1)=x(i)+dx;
    y(i+1)=y(i)+dy;
    z(i+1)=z(i)+dz;
    dx=0.5*((10*y(i)-10*x(i))+(10*y(i+1)-10*x(i+1)))*dt;
    dy=0.5*((28*x(i)-y(i)-x(i)*z(i))+(28*x(i+1)-y(i+1)-
x(i+1)*z(i+1)))*dt;
    dz=0.5*((x(i)*y(i)-8*z(i)/3)+(x(i+1)*y(i+1)-8*z(i+1)/3))*dt;
    x(i+1)=x(i)+dx;
    y(i+1)=y(i)+dy;
    z(i+1)=z(i)+dz;
end;
%plot of 2-D poincare map%
figure(1)
```

Appendix B (Continued)

```
plot(x,y)
title('Poincare Map - Lorenz With Micro - Modified Euler Method')
xlabel('x')
ylabel('y')
grid
%plot of 3-D poincare map%
figure(2)
plot3(x,y,z)
title('3D Poincare Map - Lorenz With Micro - Modified Euler Method')
xlabel('x')
ylabel('y')
zlabel('z')
grid
clear
start=cputime;
%Computing with error/10%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=22; %upper time limit%
E1=0.005; %Error Limit%
N_resI=2;%Initial micro resolution%
EHR=E1/10;%error/10%
NA=1024; %Resolution Parameter%
clear xA yA zA xB yB zB
t=(0:(NA-1))*(tf-ti)/(NA-1)+ti;
xA=1:NA; %Primitive xA array%
yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xB=1:NA; %Primitive xB array%
yB=1:NA; %Primitive yB array%
zB=1:NA; %Primitive zB array%
dt=(tf-ti)/(NA-1);%increment%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
xB(1)=xi; %allocating initial values%
yB(1)=yi; %allocating initial values%
zB(1)=zi; %allocating initial values%
N_resA=N_resI;%Initial micro resolution%
epsilon=2*E1; %defining epsilon%
loop=1; %Initializing the first loop count parameter%
%The Micro-Integrator%
while epsilon > EHR %error constraint loop%
    starttime=cputime;
    for i=1:(NA-1);
        u=xA(i);%creating the first dynamic variable %
        v=yA(i);%creating the second dynamic variable%
        w=zA(i);%creating the third dynamic variable %
        delta = (t(i+1)-t(i))/(N_resA);
        for j=1:N_resA;
            ui=u;vi=v;wi=w;
            %The Integration%
```

Appendix B (Continued)

```

        du_new=(10*v-10*u)*delta;
        dv_new=(28*u-v-u*w)*delta;
        dw_new=(u*v-8*w/3)*delta;
        u=ui+du_new;
        v=vi+dv_new;
        w=wi+dw_new;
        du=du_new/2+(10*v-10*u)*delta/2;
        dv=dv_new/2+(28*u-v-u*w)*delta/2;
        dw=dw_new/2+(u*v-8*w/3)*delta/2;
        u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
        u=u_new;v=v_new;w=w_new;
    end;
    xA(i+1)=u;yA(i+1)=v;zA(i+1)=w;
end;
finishtime=cputime;
usedtime=finishtime-starttime;
N_resB=2*N_resA;
for i=1:(NA-1);
    u=xB(i);%creating the first dynamic variable %
    v=yB(i);%creating the second dynamic variable%
    w=zB(i);%creating the third dynamic variable %
    delta = (t(i+1)-t(i))/(N_resB);
    for j=1:N_resB;
        ui=u;vi=v;wi=w;
        %The Integration%
        du_new=(10*v-10*u)*delta;
        dv_new=(28*u-v-u*w)*delta;
        dw_new=(u*v-8*w/3)*delta;
        u=ui+du_new;
        v=vi+dv_new;
        w=wi+dw_new;
        du=du_new/2+(10*v-10*u)*delta/2;
        dv=dv_new/2+(28*u-v-u*w)*delta/2;
        dw=dw_new/2+(u*v-8*w/3)*delta/2;
        u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
        u=u_new;v=v_new;w=w_new;
    end;
    xB(i+1)=u;yB(i+1)=v;zB(i+1)=w;
end;
epsilon=functerr(xB,xA,yB,yA,zB,zA);
N_resA=2*N_resA;
loop=loop+1;    %Incremental setup for the loop count%
loop_1=loop-1  %correcting the forward lag of +1 in the loop count%
end;
N_resA=0.5*N_resA;
figure(3)
plot( t, xA, t,xB)
title (['Lorenz With Micro Modified Euler Method with error/10 NA=',
num2str(NA), ' NresA=',num2str( N_resA),' error=', num2str(epsilon),'
cputime=', num2str(usedtime),'s' ] );
xlabel('time in seconds')
ylabel('XA')
grid
%computing with actual error%

```

Appendix B (Continued)

```

E=E1;                %defining the error parameter%
flucperNAres=30;%fluctuation in N_resA per loop%
devEper=10;         %percentage error deviation allowed%
Kloop=1;           %Initializing the second loop count parameter%
KloopMax=25;       %maximum number of loops before program termination%
devE=devEper*0.01;flucNAres=flucperNAres*0.01;
t=(0:(NA-1))*(tf-ti)/(NA-1)+ti; stpp=(epsilon-E)/E;
xB1=xB;yB1=yB;zB1=zB;xAhold=xB;yAhold=yB;zAhold=zB;
NAresAhold=N_resA;mark=0;X=1;
while abs(stpp)>devE;

    if stpp>0;
        N_resA= N_resA+fix(flucNAres*N_resA);
    else
        N_resA =N_resA-fix(flucNAres*N_resA);
    end;
clear xA yA zA
xA=1:NA; %Primitive xA array%yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
for i=1:(NA-1);
    u=xA(i);%creating the first dynamic variable %
    v=yA(i);%creating the second dynamic variable%
    w=zA(i);%creating the third dynamic variable %
    delta=(t(i+1)-t(i))/(N_resA);
    for j=1:N_resA;
        ui=u;vi=v;wi=w;
        %The Integration%
        du_new=(10*v-10*u)*delta;
        dv_new=(28*u-v-u*w)*delta;
        dw_new=(u*v-8*w/3)*delta;
        u=ui+du_new;
        v=vi+dv_new;
        w=wi+dw_new;
        du=du_new/2+(10*v-10*u)*delta/2;
        dv=dv_new/2+(28*u-v-u*w)*delta/2;
        dw=dw_new/2+(u*v-8*w/3)*delta/2;
        u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
        u=u_new;v=v_new;w=w_new;
    end;
    xA(i+1)=u;yA(i+1)=v;zA(i+1)=w;
end;
epsilon=functerr(xA, xAhold, yA,yAhold, zA,zAhold);% here the A
array is smaller
Kloop=Kloop+1;      %Incremental setup for the loop count%
loop_2=Kloop-1     %correcting the forward lag of +1 in the loop
count%
if Kloop >KloopMax;
    stop
end
stpp=(epsilon-E)/E;
end;

```


Appendix B (Continued)

```
figure(4)
plot( t, xA, t,xB)
title (['Lorenz With Micro Modified Euler Method With error control
NA=', num2str(NA), ' NresA=',num2str( N_resA), ' error=',
num2str(epsilon) ] );
xlabel('time in seconds')
ylabel('XA')
grid
cputime=cputime-start
loop_1
loop_2
epsilon
```

Appendix B-7: Lorenz Equations by Simpson's Method with the Micro-Integrator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This Program uses the Simpson's Method to solve the Lorenz Non-linear
Chaotic Partial Differential Equation Set With The Micro-Integrator%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all
close all
%Pre-Integration Program Inputs%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=20; %upper time limit%
N=40000; %Resolution parameter%
dt=(tf-ti)/N;%integration step%
x=0:N; %primitive x array%
y=0:N; %primitive y array%
z=0:N; %primitive z array%
x(1)=xi; %allocating initial values%
y(1)=yi; %allocating initial values%
z(1)=zi; %allocating initial values%
for i=1:N;
    %The Integration%
    dx1=(10*y(i)-10*x(i))*0.5*dt;
    dy1=(28*x(i)-y(i)-x(i)*z(i))*0.5*dt;
    dz1=(x(i)*y(i)-8*z(i)/3)*0.5*dt;
    xhf=x(i)+dx1;
    yhf=y(i)+dy1;
    zhf=z(i)+dz1;
    dx2=(10*yhf-10*xhf)*0.5*dt;
    dy2=(28*xhf-yhf-xhf*zhf)*0.5*dt;
    dz2=(xhf*yhf-8*zhf/3)*0.5*dt;
    xfl=xhf+dx2;
    yfl=yhf+dy2;
    zfl=zhf+dz2;
    dx=(1/6)*((10*y(i)-10*x(i)))+(4*(10*yhf-10*xhf))+(10*yfl-
10*xfl))*dt;
```

Appendix B (Continued)

```
dy=(1/6)*((28*x(i)-y(i)-x(i)*z(i))+4*(28*xhf-yhf-
xhf*zhf))+28*xfl-yfl-xfl*zfl)*dt;
dz=(1/6)*((x(i)*y(i)-8*z(i)/3)+(4*(xhf*yhf-8*zhf/3))+(xfl*yfl-
8*zfl/3))*dt;
x(i+1)=x(i)+dx;
y(i+1)=y(i)+dy;
z(i+1)=z(i)+dz;
end;
%plot of 2-D poincare map%
figure(1)
plot(x,y)
title('Poincare Map - Lorenz With Micro - Simpson')
xlabel('x')
ylabel('y')
grid
%plot of 3-D poincare map%
figure(2)
plot3(x,y,z)
title('3D Poincare Map - Lorenz With Micro - Simpson')
xlabel('x')
ylabel('y')
zlabel('z')
grid
clear
start=cputime;
%Computing with error/10%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=22; %upper time limit%
E1=0.005; %Error Limit%
N_resI=2;%Initial micro resolution%
EHR=E1/10;%error/10%
NA=1024; %Resolution Parameter%
clear xA yA zA xB yB zB
t=(0:(NA-1))*(tf-ti)/(NA-1)+ti;
xA=1:NA; %Primitive xA array%
yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xB=1:NA; %Primitive xB array%
yB=1:NA; %Primitive yB array%
zB=1:NA; %Primitive zB array%
dt=(tf-ti)/(NA-1);%increment%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
xB(1)=xi; %allocating initial values%
yB(1)=yi; %allocating initial values%
zB(1)=zi; %allocating initial values%
N_resA=N_resI;%Initial micro resolution%
epsilon=2*E1; %defining epsilon%
loop=1; %Initializing the first loop count parameter%
%The Micro-Integrator%
```

Appendix B (Continued)

```

while epsilon > EHR %error constraint loop%
    starttime=cputime;
    for i=1:(NA-1);
        u=xA(i);%creating the first dynamic variable %
        v=yA(i);%creating the second dynamic variable%
        w=zA(i);%creating the third dynamic variable %
        delta = (t(i+1)-t(i))/(N_resA);

        for j=1:N_resA;% microintegrator loop
            ui=u;vi=v;wi=w;
            %The Integration%
            du1=(10*v-10*u)*0.5*delta;
            dv1=(28*u-v-u*w)*0.5*delta;
            dw1=(u*v-8*w/3)*0.5*delta;
            uhf=u+du1;
            vhf=v+dv1;
            whf=w+dw1;
            du2=(10*vhf-10*uhf)*0.5*delta;
            dv2=(28*uhf-vhf-uhf*whf)*0.5*delta;
            dw2=(uhf*vhf-8*whf/3)*0.5*delta;
            ufl=uhf+du2;
            vfl=vhf+dv2;
            wfl=whf+dw2;
            du=(1/6)*((10*v-10*u)+(4*(10*vhf-10*uhf))+(10*vfl-
10*ufl))*delta;
            dv=(1/6)*((28*u-v-u*w)+(4*(28*uhf-vhf-uhf*whf))+(28*ufl-
vfl-ufl*wfl))*delta;
            dw=(1/6)*((u*v-8*w/3)+(4*(uhf*vhf-8*whf/3))+(ufl*vfl-
8*wfl/3))*delta;
            u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
            u=u_new;v=v_new;w=w_new;
        end;
        xA(i+1)=u;yA(i+1)=v;zA(i+1)=w;
    end;
    finishtime=cputime;
    usedtime=finishtime-starttime;
    N_resB=2*N_resA;
    for i=1:(NA-1);
        u=xB(i);%creating the first dynamic variable %
        v=yB(i);%creating the second dynamic variable%
        w=zB(i);%creating the third dynamic variable %
        delta=(t(i+1)-t(i))/(N_resB);
        for j=1:N_resB;% microintegrator loop
            ui=u;vi=v;wi=w;
            %The Integration%
            du1=(10*v-10*u)*0.5*delta;
            dv1=(28*u-v-u*w)*0.5*delta;
            dw1=(u*v-8*w/3)*0.5*delta;
            uhf=u+du1;
            vhf=v+dv1;
            whf=w+dw1;
            du2=(10*vhf-10*uhf)*0.5*delta;
            dv2=(28*uhf-vhf-uhf*whf)*0.5*delta;
            dw2=(uhf*vhf-8*whf/3)*0.5*delta;

```

Appendix B (Continued)

```

        ufl=uhf+du2;
        vfl=vhf+dv2;
        wfl=whf+dw2;
        du=(1/6)*((10*v-10*u)+(4*(10*vhf-10*uhf))+(10*vfl-
10*ufl))*delta;
        dv=(1/6)*((28*u-v-u*w)+(4*(28*uhf-vhf-uhf*whf))+(28*ufl-vfl-
ufl*wfl))*delta;
        dw=(1/6)*((u*v-8*w/3)+(4*(uhf*vhf-8*whf/3))+(ufl*vfl-
8*wfl/3))*delta;
        u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
        u=u_new;v=v_new;w=w_new;
    end;
    xB(i+1)=u;yB(i+1)=v;zB(i+1)=w;
end;
epsilon=functerr(xB,xA,yB,yA,zB,zA);
N_resA=2*N_resA;
loop=loop+1;    %Incremental setup for the loop count%
loop_1=loop-1    %correcting the forward lag of +1 in the loop count%
end;
N_resA=0.5*N_resA;
figure(3)
plot(t,xA,t,xB)
title(['Lorenz With Micro Simpsons Method with error/10 NA=',
num2str(NA), ' NresA=',num2str( N_resA),' error= ',
num2str(epsilon),' cputime=', num2str(usedtime),'s' ] );
xlabel('time in seconds')
ylabel('XA')
grid
%computing with actual error%
E=E1;    %defining the error parameter%
flucperNAres=30;%fluctuation in N_resA per loop%
devEper=10;    %percentage error deviation allowed%
Kloop=1;    %Initializing the second loop count parameter%
KloopMax=25;    %maximum number of loops before program termination%
devE=devEper*0.01;flucNAres=flucperNAres*0.01;
t=(0:(NA-1))*(tf-ti)/(NA-1)+ti; stpp=(epsilon-E)/E;
xB1=xB;yB1=yB;zB1=zB;xAhold=xB;yAhold=yB;zAhold=zB;
NAresAhold=N_resA;mark=0;X=1;
while abs(stpp)>devE;
    if stpp>0;
        N_resA= N_resA+fix(flucNAres*N_resA);
    else
        N_resA =N_resA-fix(flucNAres*N_resA);
    end;
clear xA yA zA
xA=1:NA; %Primitive xA array%
yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
for i=1:(NA-1);
    u = xA(i);%creating the first dynamic variable %
    v = yA(i);%creating the second dynamic variable %

```

Appendix B (Continued)

```

w = zA(i);%creating the third dynamic variable %
delta=(t(i+1)-t(i))/(N_resA);
for j=1:N_resA;
    ui=u;vi=v;wi=w;
    %The Integration%
    du1=(10*v-10*u)*0.5*delta;
    dv1=(28*u-v-u*w)*0.5*delta;
    dw1=(u*v-8*w/3)*0.5*delta;
    uhf=u+du1;
    vhf=v+dv1;
    whf=w+dw1;
    du2=(10*vhf-10*uhf)*0.5*delta;
    dv2=(28*uhf-vhf-uhf*whf)*0.5*delta;
    dw2=(uhf*vhf-8*whf/3)*0.5*delta;
    ufl=uhf+du2;
    vfl=vhf+dv2;
    wfl=whf+dw2;
    du=(1/6)*((10*v-10*u)+(4*(10*vhf-10*uhf))+(10*vfl-
10*ufl))*delta;
    dv=(1/6)*((28*u-v-u*w)+(4*(28*uhf-vhf-uhf*whf))+(28*ufl-vfl-
ufl*wfl))*delta;
    dw=(1/6)*((u*v-8*w/3)+(4*(uhf*vhf-8*whf/3))+(ufl*vfl-
8*wfl/3))*delta;
    u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
    u=u_new;v=v_new;w=w_new;
    end;
    xA(i+1)=u;yA(i+1)=v;zA(i+1)=w;
end;
epsilon=functerr(xA, xAhold, yA,yAhold, zA,zAhold);
Kloop=Kloop+1;    %Incremental setup for the loop count%
loop_2=Kloop-1    %correcting the forward lag of +1 in the loop count%
    if Kloop >KloopMax;
        stop
    end
stpp=(epsilon-E)/E ;
end;
figure(4)
plot(t, xA, t, xB)
title(['Lorenz With Micro Simpsons Method With error control  NA=',
num2str(NA), '  NresA=',num2str( N_resA), '  error=',
num2str(epsilon)]);
xlabel('time in seconds')
ylabel('XA')
grid
cputime=cputime-start
loop_1
loop_2
epsilon

```

Appendix B-8: Lorenz Equations by Runge-Kutta's Method with the Micro-Integrator

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This Program uses the 4th Order Runge-Kutta Method to solve the Lorenz
Non-linear Chaotic Partial Differential Equation Set With The Micro-
Integrator%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;
close all;
%Pre-Integration Program Inputs%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=40; %Upper time limit%
N=40000; %Resolution parameter%
dt=(tf-ti)/N;%integration step%
x=0:N; %primitive x array%
y=0:N; %primitive y array%
z=0:N; %primitive z array%
x(1)=xi; %allocating initial values%
y(1)=yi; %allocating initial values%
z(1)=zi; %allocating initial values%
for i=1:N;
    %The Integration%
    dx1=(10*y(i)-10*x(i))*dt;
    dy1=(28*x(i)-y(i)-x(i)*z(i))*dt;
    dz1=(x(i)*y(i)-8*z(i)/3)*dt;
    dx2=(10*(y(i)+(0.5*dy1))-10*(x(i)+(0.5*dx1)))*dt;
    dy2=(28*(x(i)+(0.5*dx1))-(y(i)+(0.5*dy1))-
(x(i)+(0.5*dx1))*(z(i)+(0.5*dz1)))*dt;
    dz2=((x(i)+(0.5*dx1))*(y(i)+(0.5*dy1))-8*(z(i)+(0.5*dz1))/3)*dt;
    dx3=(10*(y(i)+(0.5*dy2))-10*(x(i)+(0.5*dx2)))*dt;
    dy3=(28*(x(i)+(0.5*dx2))-(y(i)+(0.5*dy2))-
(x(i)+(0.5*dx2))*(z(i)+(0.5*dz2)))*dt;
    dz3=((x(i)+(0.5*dx2))*(y(i)+(0.5*dy2))-8*(z(i)+(0.5*dz2))/3)*dt;
    dx4=(10*(y(i)+(dy3))-10*(x(i)+(dx3)))*dt;
    dy4=(28*(x(i)+(dx3))-(y(i)+(dy3))-(x(i)+(dx3))*(z(i)+(dz3)))*dt;
    dz4=((x(i)+(dx3))*(y(i)+(dy3))-8*(z(i)+(dz3))/3)*dt;
    dx=(1/6)*(dx1+2*(dx2+dx3)+dx4);
    dy=(1/6)*(dy1+2*(dy2+dy3)+dy4);
    dz=(1/6)*(dz1+2*(dz2+dz3)+dz4);
    x(i+1)=x(i)+dx;
    y(i+1)=y(i)+dy;
    z(i+1)=z(i)+dz;
end;
%plot of 2-D poincare map%
figure(1)
plot(x,y)

```

Appendix B (Continued)

```
title('Poincare Map - Lorenz With Micro - Runge-Kutta -4th Order')
xlabel('x')
ylabel('y')
grid
%plot of 3-D poincare map%
figure(2)
plot3(x,y,z)
title('3D Poincare Map - Lorenz With Micro - Runge-Kutta -4th Order')
xlabel('x')
ylabel('y')
zlabel('z')
grid
clear
start=cputime;
%Computing with error/10%
xi=-11.2;%initial value of x%
yi=-8.4; %initial value of y%
zi=33.4; %initial value of z%
ti=0; %lower time limit%
tf=22; %upper time limit%
E1=0.005; %Error Limit%
N_resI=2; %Initial micro resolution%
EHR=E1/10;%error/10%
NA=1024; %Resolution Parameter%
clear xA yA zA xB yB zB
t=(0:(NA-1))*(tf-ti)/(NA-1)+ti;
xA=1:NA; %Primitive xA array%
yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xB=1:NA; %Primitive xB array%
yB=1:NA; %Primitive yB array%
zB=1:NA; %Primitive zB array%
dt=(tf-ti)/(NA-1);%increment%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
xB(1)=xi; %allocating initial values%
yB(1)=yi; %allocating initial values%
zB(1)=zi; %allocating initial values%
N_resA=N_resI;%Initial micro resolution%
epsilon=2*E1; %defining epsilon%
loop=1; %Initializing the first loop count parameter%
%The Micro-Integrator%
while epsilon > EHR %error constraint loop%
    starttime=cputime;
    for i=1:(NA-1);
        u = xA(i);%creating the first dynamic variable %
        v = yA(i);%creating the second dynamic variable%
        w = zA(i);%creating the third dynamic variable %
        delta = (t(i+1)-t(i))/(N_resA);
        for j=1:N_resA;% microintegrator loop
            ui=u;vi=v;wi=w;
            %The Integration%
            dul=(10*v-10*u)*delta;
```

Appendix B (Continued)

```

        dv1=(28*u-v-u*w)*delta;
        dw1=(u*v-8*w/3)*delta;
        du2=(10*(v+(0.5*dv1))-10*(u+(0.5*du1)))*delta;
        dv2=(28*(u+(0.5*du1))-(v+(0.5*dv1))-(
(u+(0.5*du1))*(w+(0.5*dw1))))*delta;
        dw2=((u+(0.5*du1))*(v+(0.5*dv1))-8*(w+(0.5*dw1)))/3)*delta;
        du3=(10*(v+(0.5*dv2))-10*(u+(0.5*du2)))*delta;
        dv3=(28*(u+(0.5*du2))-(v+(0.5*dv2))-(
(u+(0.5*du2))*(w+(0.5*dw2))))*delta;
        dw3=((u+(0.5*du2))*(v+(0.5*dv2))-8*(w+(0.5*dw2)))/3)*delta;
        du4=(10*(v+(dv3))-10*(u+(du3)))*delta;
        dv4=(28*(u+(du3))-(v+(dv3))-(u+(du3))*(w+(dw3)))*delta;
        dw4=((u+(du3))*(v+(dv3))-8*(w+(dw3)))/3)*delta;
        du=(1/6)*(du1+2*(du2+du3)+du4);
        dv=(1/6)*(dv1+2*(dv2+dv3)+dv4);
        dw=(1/6)*(dw1+2*(dw2+dw3)+dw4);
        u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
        u=u_new;v=v_new;w=w_new;
    end;
    xA(i+1)=u;yA(i+1)=v;zA(i+1)=w;
end;
    finishtime=cputime;
    usedtime=finishtime-starttime;
    N_resB=2*N_resA;
for i=1:(NA-1);
    u=xB(i);%creating the first dynamic variable %
    v=yB(i);%creating the second dynamic variable%
    w=zB(i);%creating the third dynamic variable %
    delta=(t(i+1)-t(i))/(N_resB);
    for j=1:N_resB;% microintegrator loop
        ui=u;vi=v;wi=w;
        %The Integration%
        du1=(10*v-10*u)*delta;
        dv1=(28*u-v-u*w)*delta;
        dw1=(u*v-8*w/3)*delta;
        du2=(10*(v+(0.5*dv1))-10*(u+(0.5*du1)))*delta;
        dv2=(28*(u+(0.5*du1))-(v+(0.5*dv1))-(
(u+(0.5*du1))*(w+(0.5*dw1))))*delta;
        dw2=((u+(0.5*du1))*(v+(0.5*dv1))-8*(w+(0.5*dw1)))/3)*delta;
        du3=(10*(v+(0.5*dv2))-10*(u+(0.5*du2)))*delta;
        dv3=(28*(u+(0.5*du2))-(v+(0.5*dv2))-(
(u+(0.5*du2))*(w+(0.5*dw2))))*delta;
        dw3=((u+(0.5*du2))*(v+(0.5*dv2))-8*(w+(0.5*dw2)))/3)*delta;
        du4=(10*(v+(dv3))-10*(u+(du3)))*delta;
        dv4=(28*(u+(du3))-(v+(dv3))-(u+(du3))*(w+(dw3)))*delta;
        dw4=((u+(du3))*(v+(dv3))-8*(w+(dw3)))/3)*delta;
        du=(1/6)*(du1+2*(du2+du3)+du4);
        dv=(1/6)*(dv1+2*(dv2+dv3)+dv4);
        dw=(1/6)*(dw1+2*(dw2+dw3)+dw4);
        u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
        u=u_new;v=v_new;w=w_new;
    end;
    xB(i+1)=u;yB(i+1)=v;zB(i+1)=w;
end;

```


Appendix B (Continued)

```

epsilon=functerr(xB,xA,yB,yA,zB,zA);
N_resA=2*N_resA;
loop=loop+1;    %Incremental setup for the loop count%
loop_1=loop-1  %correcting the forward lag of +1 in the loop count%
end;
N_resA=0.5*N_resA;
figure(3)
plot(t,xA,t,xB)
title(['Lorenz With Micro Runge-Kutta 4th Order with error/10 NA=',
num2str(NA), ' NresA=',num2str(N_resA),' error= ', num2str(epsilon)]
);
xlabel('time in seconds')
ylabel('XA')
grid
%computing with actual error%
E=E1;          %defining the error parameter%
flucperNAres=30;%fluctuation in N_resA per loop%
devEper=10;    %percentage error deviation allowed%
Kloop=1;       %Initializing the second loop count parameter%
KloopMax=25;   %maximum number of loops before program termination%
devE=devEper*0.01;flucNAres=flucperNAres*0.01;
t=(0:(NA-1))*(tf-ti)/(NA-1)+ti; stpp=(epsilon-E)/E;
xB1=xB;yB1=yB;zB1=zB;xAhold=xB;yAhold=yB;zAhold=zB;
NAresAhold=N_resA;mark=0;X=1;
while abs(stpp)>devE;

    if stpp>0;
        N_resA= N_resA+fix(flucNAres*N_resA);
    else
        N_resA =N_resA-fix(flucNAres*N_resA);
    end;
clear xA yA zA
xA=1:NA; %Primitive xA array%
yA=1:NA; %Primitive yA array%
zA=1:NA; %Primitive zA array%
xA(1)=xi; %allocating initial values%
yA(1)=yi; %allocating initial values%
zA(1)=zi; %allocating initial values%
for i=1:(NA-1);
    u = xA(i);%creating the first dynamic variable %
    v = yA(i);%creating the second dynamic variable%
    w = zA(i);%creating the third dynamic variable %
    delta=(t(i+1)-t(i))/(N_resA);
    for j=1:N_resA;
        ui=u;vi=v;wi=w;
        %The Integration%
        du1=(10*v-10*u)*delta;
        dv1=(28*u-v-u*w)*delta;
        dw1=(u*v-8*w/3)*delta;
        du2=(10*(v+(0.5*dv1))-10*(u+(0.5*du1)))*delta;
        dv2=(28*(u+(0.5*du1))-(v+(0.5*dv1))-(
(u+(0.5*du1))*(w+(0.5*dw1)))*delta;
        dw2=((u+(0.5*du1))*(v+(0.5*dv1))-8*(w+(0.5*dw1))/3)*delta;
        du3=(10*(v+(0.5*dv2))-10*(u+(0.5*du2)))*delta;

```

Appendix B (Continued)

```
dv3=(28*(u+(0.5*du2))-(v+(0.5*dv2))-(
(u+(0.5*du2))*(w+(0.5*dw2)))*delta;
dw3=((u+(0.5*du2))*(v+(0.5*dv2))-8*(w+(0.5*dw2))/3)*delta;
du4=(10*(v+(dv3))-10*(u+(du3)))*delta;
dv4=(28*(u+(du3))-(v+(dv3))-(u+(du3))*(w+(dw3)))*delta;
dw4=((u+(du3))*(v+(dv3))-8*(w+(dw3))/3)*delta;
du=(1/6)*(du1+2*(du2+du3)+du4);
dv=(1/6)*(dv1+2*(dv2+dv3)+dv4);
dw=(1/6)*(dw1+2*(dw2+dw3)+dw4);
u_new=ui+du;v_new=vi+dv;w_new=wi+dw;
u=u_new;v=v_new;w=w_new;
end;
xA(i+1)=u;yA(i+1)=v;zA(i+1)=w;
end;

epsilon=functerr(xA, xAhold, yA,yAhold, zA,zAhold);
Kloop=Kloop+1; %Incremental setup for the loop count%
loop_2=Kloop-1 %correcting the forward lag of +1 in the loop count%
if Kloop >KloopMax;
stop
end
stpp=(epsilon-E)/E ;
end;
figure(4)
plot(t,xA,t,xB)
title(['Lorenz With Micro Runge-Kutta 4th Order With error control
NA=', num2str(NA), ' NresA=',num2str( N_resA), ' error=',
num2str(epsilon)]);
xlabel('time in seconds')
ylabel('XA')
grid
cputime=cputime-start
loop_1
loop_2
epsilon
```

Appendix C: Principal Subroutines

Appendix C-1: The Interpolation Subroutine

```
function XB=funcinterp(to,tf,NA,NB,XA)

pB=1:NB; %array 1 to size B%
pA=1:NA; %array 1 to size A%
XB(1) = XA(1); %defining the first values%
XB(NB)=XA(NA); %defining the last values%
tB=ones(size(pB)); %array of ones of size B%
tA=ones(size(XA)); %array of ones of size A%
tB=to*tB+(tf-to)/(NB-1)*(pB-tB);
tA=to*tA+(tf-to)/(NA-1)*(pA-tA);
i=1; %Initial value before iterations%
for k=1:(NB-1);
    while tB(k)> tA(i+1);
        i=i+1; %iteration increment%
    end;
    slope=(XA(i+1)-XA(i))/(tA(i+1)-tA(i)); %slope calculation%
    XB(k)=XA(i)+slope*(tB(k)-tA(i)); %interpolation rule%
end
```

Appendix C-2: The Error Calculation Subroutine

```
function Err=functerr(xB,xA,yB,yA,zB,zA)

%%%%% Error estimation ***** two arrays*** no loop in this
version, no corrective step on micro-loop
gX=[xA,xB]; %creates a concatenated array of x values%
gY=[yA,yB]; %creates a concatenated array of y values%
gZ=[zA,zB]; %creates a concatenated array of z values%

gXmax=max(abs(gX)) %maximum x value%
gYmax=max(abs(gY)) %maximum y value%
gZmax=max(abs(gZ)) %maximum z value%

gXmean=mean(abs(xA-xB)) %mean absolute x difference%
gYmean=mean(abs(yA-yB)) %mean absolute y difference%
gZmean=mean(abs(zA-zB)) %mean absolute z difference%

eps_x = gXmean/gXmax; %values normalized by maximum in x set%
eps_y = gYmean/gYmax; %values normalized by maximum in y set%
eps_z = gZmean/gZmax; %values normalized by maximum in z set%

epsilonV=[eps_x,eps_y,eps_z]; %combined error set%

epsilon = max(epsilonV) %maximum error%
g=0.5*max(size(xA)) %size of 'A' arrays%
Err=epsilon; %error%
```