VERIFICATION OF VARIOUS NUMERICAL METHODS USING HARDWARE IMPLEMENTATION

Major Project Report

Submitted in Partial Fulfillment of the Requirements
for the degree of

MASTER OF TECHNOLOGY
IN
INSTRUMENTATION & CONTROL
(CONTROL & AUTOMATION)

By

Trivedi Tej T.
13MICC24

Instrumentation & Control Engineering Section
Department of Electrical Engineering
Institute of Technology
Nirma University
Ahmedabad-382481

May-2015

VERIFICATION OF VARIOUS NUMERICAL METHODS USING HARDWARE IMPLEMENTATION

Major Project Report

Submitted in Partial Fulfillment of the Requirements
for the degree of

Master of Technology
in
Instrumentation and Control
(Control & Automation)

By

Trivedi Tej T.
13MICC24

Under the Guidance of

Prof. Sandip A. Mehta

Instrumentation & Control Engineering Section
Department of Electrical Engineering
Institute of Technology
Nirma University
Ahmedabad-382481

May-2015

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Instrumentation and Control Engineering (Control and Automation) at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Trivedi Tej T.
13MICC24

Undertaking for Originality of the Work

I, **Trivedi Tej T.**, Roll No. **13MICC24**, give undertaking that the Major Project entitled **"VERIFICATION OF VARIOUS NUMERICAL METHODS USING HARDWARE IMPLEMENTATION"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Instrumentation & Control (Control & Automation)** of Nirma University, Ahmedabad, is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Mr. Tej T. Trivedi

Date:

Place: Ahmedabad

Endorsed by:

_____

Signature of Guide

Prof. S. A. Mehta

Certificate

This is to certify that the Major Project entitled **VERIFICATION OF VARIOUS NUMERICAL METHODS USING HARDWARE IMPLEMENTATION** submitted by **Mr. TRIVEDI TEJ T.** Roll No. **13MICC24**, towards the partial fulfillment of the requirements for the award of degree in Master of Technology (**Instrumentation & Control Engineering**) in the field of **Control & Automation** of Nirma University is the record of work carried out by him under our supervision and guidance. The work submitted has in our opinion reached a level required for been accepted for examination. The results embodied in this major project work to the best of our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Date: **Place:** Ahmedabad

Guide: **Program Coordinator:**

Prof. Sandip A. Mehta
Assistant Professor, IC
Institute of Technology
Nirma University

Prof. J. B. Patel
Sr. Associate Professor, IC
Institute of Technology
Nirma University

HOD: **Director:**

Dr. P. N. Tekwani
Professor, EE
Institute of Technology
Nirma University

Dr. K. Kotecha
Director
Institute of Technology
Nirma University

Acknowledgement

It is indeed a pleasure for me to express my sincere gratitude to those who have always helped me in my project work.

First of all, I would like to thank my Project Guide **Prof. Sandip A. Mehta**, who helped me selecting the project topic, understanding of the subject, stimulating suggestions, encouragement, constant motivation and also for writing of this report, I am sincerely thankful for his valuable guidance and help to enhance my practical skills.

I would also like to thank our Section Head, IC Department **Dr. D. M. Adhyaru** and Program Coordinator **Prof. J. B. Patel** for providing valuable guidance and also to the Nirma University for providing excellent infrastructure and facilities whenever and wherever required.

Finally, I would also like to thank all other respected faculty members as well as Lab Assistants for their support during the project work. Last but not the least; I would like to thank God almighty, my parents and my friends for being a constant source of support, whose constant motivation helped me to increase my enthusiasm during our whole journey.

Trivedi Tej T.
13MICC24

Abstract

Numerical analysis is the study of algorithms that use numerical approximations for the problems of mathematical analysis and control system. These Numerical Algorithms are dedicated to applying its unique expertise in numerical software engineering to delivering high quality computer software and high performance computing services. In this project different Numerical Methods will be simulated for different system models. The system will be modeled using Differential Equations. The differential equations will be solved using available numerical methods. The comparative analysis will be carried out for different numerical methods using advanced software tools. The hardware implementation of the numerical methods will be done using Real-Time (RT) embedded system/hardware. A Graphical User Interface (GUI) will be developed for selection and comparison among the different numerical methods for the simulation as well as hardware implementation.

List of Abbreviations

ABM	Adams-Bashforth-Moulton
CRIO	Compact Reconfigurable Input/Output
DSO	Digital Storage Oscilloscope
EPC	Euler's Predictor-Corrector
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface
I/O	Input-Output
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
MATLAB	Matrix Laboratory
NI	National Instruments
ODEs	Ordinary Differential Equations
PC	Predictor-Corrector
RK	Runge-Kutta
RT	Real-Time

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation & Objective

In Engineering, there are some real systems which can be modeled mathematically with differential equations that can be solved using various numerical techniques. Moreover, there are many control algorithms required for controlling such systems; like optimal control which requires Search techniques to find local or global minima/maxima with the help of Gradients & Hessians. These Gradients & Hessians are calculated from numerical methods and basis on that decision can be made to select the suitable solver.

The comparative analysis will be carried out for different numerical methods using advanced software tools. This analysis will be done based upon the type & order of the system. The differential equations can be obtained from transfer function or state-space model; that will be solved using available numerical methods. A Graphical User Interface (GUI) will be developed for selection of different numerical methods. The user will be prompted to enter the differential equations in the GUI and can also select the method to solve. The stand-alone system will be prepared for the specific system model that uses any one of the selected numerical methods using RT system/hardware.

1.2 Numerical Analysis

Linear Equations are required to solve for various systems continuously in different engineering problems. Large number of scientific & technical problems are described by means of single equation with one variable or system of **n** equations with **n** variables. These equations can be divided

into ordinary differential equations and partial differential equations that are dependent upon number of independent variables and its corresponding derivatives. Obtaining solution/s for such equations constitutes one of the most common and important problems in numerical mathematics.

Ordinary differential equations frequently occur in mathematical models that arise in many branches of science and engineering. These ordinary differential equations defined for single value of the independent variable are called initial value problems satisfying the given differential equation to solve with an initial condition $y(t_0) = y_0$. These concept was proposed by Cauchy in Cauchy's Theorem.
The various numerical methods which are used are discussed in the next chapter.

These equations can be defined as linear or nonlinear according to the character of functions appearing in these equations. The corresponding classification of such algebraic equations is given in figure (1.1).
The class of single nonlinear equations is divided into polynomial and transcendent equations.

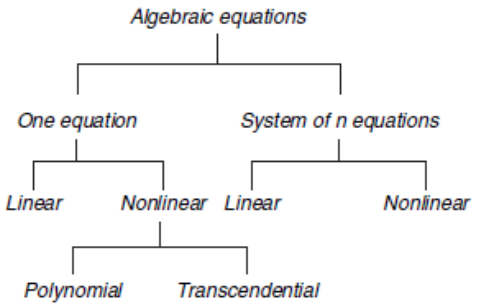

Figure 1.1: Non-Linear Equations as well as Equations with Complex-Roots

Combining math knowledge with some knowledge of programming and developing the ability of selecting appropriate numerical algorithms help to solve a large range of day-to-day problems. However, closed form analytical solutions for majority real world problems is not possible. Instead computer algorithms are used to calculate numerical solutions for such problems. Increase in complexity of the systems increases the number of differential equations; that can be solved with simple/complex programs written to run

on a programmable computers with high speed and low cost. The techniques for solving these differential equations were developed before existence of programmable computers. Currently, the computer can solve these problems that were though for the fastest supercomputers just a decade ago.

Computing numerical solutions to those problems for which there is no closed form algebraic solution (e.g. root finding, many integrals, approximation problems, etc); the focus is to obtain practical numerical solutions and applying them on large variety of real world problems. Unfortunately it is very rare that these equations have solutions which can be expressed in closed form, so it is common to seek approximate solutions by means of numerical methods. For solving such problems one should be able to program numerical algorithms in high-level programming softwares such as MATLAB/LabVIEW. MATLAB stands for Matrix Laboratory & LabVIEW stands for Laboratory Virtual Instrument Engineering Workbench. Also, the various numerical methods are programmed and implemented on the real-time target. The RT target support is provided by C-RIO stands for Compact Reconfigurable Input/Output.
More details for C-RIO are provided in later chapter/s.

1.3 Thesis Organization

In first chapter, motivation & objective of the project is discussed followed by introduction of the numerical methods and how it can be used to solve various systems using advance softwares. The second chapter covers the theoretical explanation of various numerical methods that will be studied and implemented. This chapter also covers the algorithm steps which will be useful to prepare the code of various numerical methods. In the third chapter, the simulated results are discussed. The results are available for the different systems that are solved using numerical methods prepared in MATLAB as well as LabVIEW. In the fourth chapter, the RT hardware (CRIO) is explored and the simulated results are verified on the real hardware. Also, real system is prepared to compare the results of CRIO. All these results are compared against each other to carry out comparative analysis and finally the conclusion is drawn for the same. The comparative analysis is shown in the tabular form in Appendix A while Appendix B covers some information about advance softwares and hardware specifications required for the project work.

Chapter 2

Various Numerical Methods

2.1 Methods for Ordinary Differential Equations

This chapter explores ordinary differential equations of the form

$$\frac{dy}{dt} = f(t, y) \tag{2.1}$$

which uses various numerical methods to obtain the solution for such equations. The method in general form is as follows:

$$\textbf{Next value} = \textbf{Previous value} + \textbf{slope} \times \textbf{step-size}$$

or, in mathematical terms,

$$y_{i+1} = y_i + \phi h$$

This formula is applicable for step by step computation and can trace the trajectory of the solution. It is also helpful to plot the polygon path joining the approximation points (t_k, y_k) and plot the trajectory as shown in figure (2.1).

Initial Value Problems can be solved using one-step numerical methods such as Euler's Method, Heun's Method, RK Method etc. All these self-starting & non self-starting methods are discussed in the subsections of this chapter. Runge-Kutta (RK) method is single-step self-starting method which is used to find the initial values for non self-starting methods such as Predictor-Corrector Methods (ABM, Milne's, Hamming etc.); this method is described first with its algorithm.

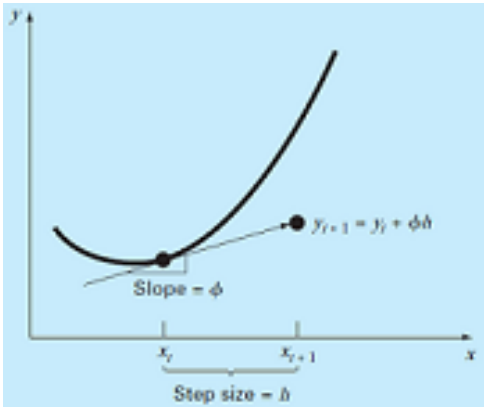

Figure 2.1: Graphical depiction of single step

Runge-Kutta (RK) Method:

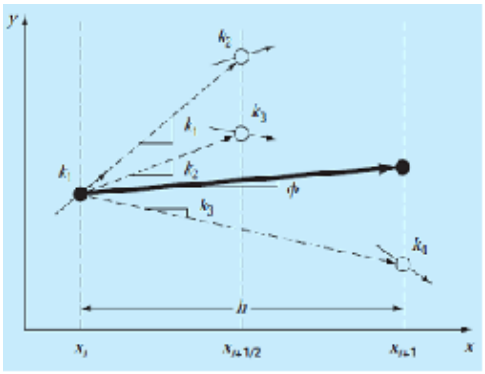

Figure 2.2: Graphical depiction for fourth order RK method

The Runge-Kutta (RK) method of order N = 4 is most popular. The fourth-order RK method gives better accuracy, stability and is also easy to program. This method is based on computing y_{k+1} and starts with the initial point $(t_0,\ y_0)$ and generates the sequence of approximation using following formula:

$$y_{k+1} = y_k + \frac{h(k_1 + 2k_2 + 2k_3 + k_4)}{6} \tag{2.2}$$

where,

$k_1 = f(t_k, y_k)$
$k_2 = f(t_k + \frac{h}{2}, y_k + \frac{h}{2} f_1)$
$k_3 = f(t_k + \frac{h}{2}, y_k + \frac{h}{2} f_2)$
$k_4 = f(t_k + h, y_k + h f_3)$

Algorithm for (RK) Method:

Algorithm steps for Runge-Kutta fourth order method are as follows:
1. Define f(t,y) [function of the differential equation y' = f(t, y)]
2. Enter t, y, h [initial values]
3. For i = 1(1) n, do till (12)
4. $k_1 \leftarrow$ hf(t, y)
5. $k_2 \leftarrow$ hf(t + h/2, y + k_1/2)
6. $k_3 \leftarrow$ hf(t + h/2, y + k_2/2)
7. t $\leftarrow$ t + h
8. $k_4 \leftarrow$ hf(t, y + k_3)
9. y $\leftarrow$ y + h(f_1 + 2f_2 + 2f_3 + f_4)/6
10. $y_m \leftarrow$ "the mathematical solution of the equation to be supplied"
11. Write t, y, y_m
12. Next i
13. End

Remarks: Initial values are assumed as t, y. Compute the value of y at the points t_0 + h, t_0 + 2h, ...

2.1.1 Single Step (Self Starting) Methods

Euler's Method

Euler Method is one of the simplest representative of the single-step methods, discussed below with the help of figure (2.3). Computation of consecutive values $y_n = y(t_n) = y(x_0 + n \cdot h)$ for n = 1, 2, 3, . . . , begins from the initial point $P_0 = (t_0, y_0)$, at which

$$\frac{dy(t)}{dt} = f[t_0, y(t_0)] = f[t_0, y_0] \tag{2.3}$$

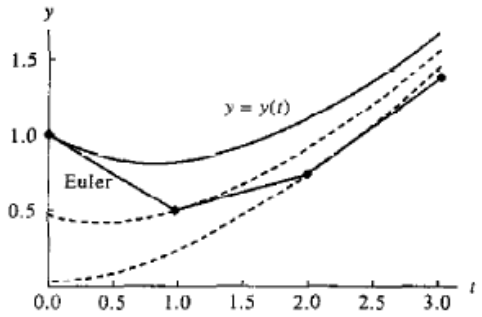

Figure 2.3: Euler's Approximation

The value of the function f(t_0, y_0) is equal to the tangent of the angle, at which the tangent to the curve y(t) satisfies the equation (2.3). Therefore, first computed value of this function is:

$$y_1 = y(t_0 + h) = y_0 + hf[t_0, y(t_0)] \tag{2.4}$$

Hence, point $P_1 = (t_1, y_1)$ is obtained, which is considered as the initial point in computing y_2, related to point $P_2 = (t_2, y_2)$. Iterating the process several times, calculates the discrete values y_n of the function approximating desired solution y(t).

Algorithm for Euler's Method:

Algorithm steps for One-Step Euler's Method are as follows:
1. Define f(t, y) [function of the differential equation y' = f(t, y)]
2. Enter t, y, h[initial values]
3. $y_i \leftarrow y_m \leftarrow y_s[y_i = y_m = y_s]$
4. For i = 1(1) n, do till (10)
5. $y_s \leftarrow y_s$ + h.f(t, y_s)
6. $t_n \leftarrow$ t + h
7. $y_e \leftarrow$ "the exact solution of the equation to be supplied, with t replaced by t_n"
8. Write t_n, y_s
9. t $\leftarrow t_n$
10. Next i
11. End

Remarks: Initial values are assumed as y_s in step 3 & in step 4 it is assumed as y_i and y_m. Compute the value of y at the points t_0 + h, t_0 + 2h, ...

Heun's Method

In Heun's method, an auxiliary coordinate is computed first $y^*_{n+1} = y_n$ + hf (t_n, y_n) and used next to find the function value with quantity

$$f(t_{n+1}, y^*_{n+1}) \tag{2.5}$$

expressing the slope coefficient described by equation (2.6) and passing through the point $P^*_{n+1} = (t_{n+1}, y^*_{n+1})$ being first approximation of the required solution. The point, giving more better approximation, $P_{n+1} = (t_{n+1}, y_{n+1})$, has the coordinate y_{n+1} calculated from following formula:

$$y_{n+1} = y_n + \frac{1}{2}h[f(t_n, y_n) + f(t_{n+1}, y^*_{n+1}] \tag{2.6}$$

2.1.2 Predictor-Corrector Methods

In single-step methods discussed above, the value y_{n+1} = y(t_{n+1}) of the given function is calculated on the basis of only one value y_n = y(t_n), computed during the previous iteration. Whereas the multi-step methods not only calculates the value y_n = y(t_n) but also y_{n-k+1} = y(t_{n-k+1}), y_{n-k+2} = y(t_{n-k+2}), y_{n-k+3} = y(t_{n-k+3}), . . . , y_n = y(t_n), where the number of steps k = 1, 2, 3, . . . determines steps of the method.

ABM's Method

The Adams-Bashforth-Moulton (ABM) Method is a multi-step Predictor-Corrector Method derived from the fundamental theorem of calculus:

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(t, y(t))dt \tag{2.7}$$

The predictor of ABM requires the Lagrange Polynomial approximation for f(t, y(t)) based on the points (t_{k-3}, f_{k-3}), (t_{k-2}, f_{k-2}), (t_{k-1}, f_{k-1}) and

(t_k, f_k). The Adam-Bashforth Predictor is obtained by integrating over the interval [t_k, t_{k+1}] in equation (2.7).[Eq. (2.8)]

$$p_{k+1} = y_k + \frac{h}{24}(-9f_{k-3} + 37f_{k-2} - 59f_{k-1} + 55f_k) \tag{2.8}$$

The corrector is obtained similarly by using the value p_{k+1} that is computed by equation (2.8). A second Lagrange polynomial for f(t,y(t)) is formed, on the basis of points (t_{k-2}, f_{k-2}), (t_{k-1}, f_{k-1}), (t_k, f_k) and the new point (t_{k+1}, f_{k+1}) = (t_{k+1}, f(t_{k+1}, p_{k+1})). This polynomial is then integrated over [t_k, t_{k+1}], to get the Adams-Moulton corrector. [Eq. (2.9)]

$$y_{k+1} = y_k + \frac{h}{24}(f_{k-2} - 5f_{k-1} + 19f_k + 9f_{k+1}) \tag{2.9}$$

Algorithm for ABM Method:

Algorithm steps for Multi-Step ABM Method are as follows:
1. f(t, y) ← 'Function to be supplied'
2. Enter t_0, t_1, t_2, t_3
3. Enter y_0, y_1, y_2, y_3
4. h ← t_1 - t_0
5. For i = 1(1) n, do till (26)
6. f_0 ← f(t_0, y_0)
7. f_1 ← f(t_1, y_1)
8. f_2 ← f(t_2, y_2)
9. f_3 ← f(t_3, y_3)
10. y_p ← y_3 + h(-9f_0 + 37f_1 - 59f_2 + 55f_3)/24
11. t_4 ← t_3 + h
12. c ← y_3 + h(f_1 - 5f_2 + 19f_3)/24
13. y_c ← c + 9h f(t_4, y_p)/24
14. if | y_p - y_c | < 10^{-6}
15. Write t_4, y_c
16. Else y_p ← y_c
17. Goto (13)
18. t_0 ← t_1
19. t_1 ← t_2
20. t_2 ← t_3
21. t_3 ← t_4
22. y_0 ← y_1
23. y_1 ← y_2

24. $y_2 \leftarrow y_3$
25. $y_3 \leftarrow y_c$
26. Next i
27. End

Milne's Method

Another well-known Predictor-Corrector method is Milne-Simpson Method.

$$y(t_{k+1}) = y(t_{k-3}) + \int_{t_{k-3}}^{t_{k+1}} f(t, y(t))dt \tag{2.10}$$

The predictor requires the Lagrange Polynomial approximation for f(t, y(t)) based on the points $(t_{k-3},\ f_{k-3})$, $(t_{k-2},\ f_{k-2})$, $(t_{k-1},\ f_{k-1})$ and $(t_k,\ f_k)$. It is integrated over the interval $[t_{k-3},\ t_{k+1}]$ in equation (2.10) to obtained Milne Predictor. [Eq. (2.11)]

$$p_{k+1} = y_{k-3} + \frac{4h}{3}(2f_{k-2} - f_{k-1} + 2f_k) \tag{2.11}$$

The corrector is obtained similarly using the value p_{k+1} that is computed by equation (2.11). A second Lagrange polynomial for f(t,y(t)) is formed, basis on the points $(t_{k-2},\ f_{k-2})$, $(t_{k-1},\ f_{k-1})$, $(t_k,\ f_k)$ and the new point $(t_{k+1},\ f_{k+1}) = (t_{k+1},\ \mathrm{f}(t_{k+1},\ p_{k+1}))$. The polynomial is then integrated over $[t_{k-1},\ t_{k+1}]$, and the result is familiar Simpson's rule:

$$y_{k+1} = y_{k-1} + \frac{4h}{3}(f_{k-1} + 4f_k + f_{k+1}) \tag{2.12}$$

Algorithm for Milne's Method:

Algorithm steps for Multi-Step Milne's Method are as follows:
1. f(t, y) $\leftarrow$ 'Function to be supplied'
2. Enter t_0, t_1, t_2, t_3
3. Enter y_0, y_1, y_2, y_3
4. h $\leftarrow$ t_1 - t_0
5. For i = 1(1) n, do till (24)
6. $f_1 \leftarrow$ f(t_1, y_1)
7. $f_2 \leftarrow$ f(t_2, y_2)

8. $f_3 \leftarrow$ f(t_3, y_3)
9. $y_p \leftarrow y_0$ + 4h(2f_1 - f_2 + 2f_3)/3
10. $t_4 \leftarrow t_3$ + h
11. c $\leftarrow y_2$ + h(f_2 + 4f_3)/3
12. $y_c \leftarrow$ c + h f(t_4, y_p)/3
13. if | y_p - y_c | $< 10^{-6}$
14. Write t_4, y_c
15. Else $y_p \leftarrow y_c$
16. Goto (12)
17. $t_1 \leftarrow t_2$
18. $t_2 \leftarrow t_3$
19. $t_3 \leftarrow t_4$
20. $y_0 \leftarrow y_1$
21. $y_1 \leftarrow y_2$
22. $y_2 \leftarrow y_3$
23. $y_3 \leftarrow y_c$
24. Next i
25. End

Hamming's Method

Another Predictor-Corrector method is Hamming Method. Its predictor is same as Milne's Predictor but there is a modification in corrector compare to Milne-Simpson. The Corrector is given by:

$$y_{k+1} = \frac{-y_{k-2} + 9y_k}{8} + \frac{3h}{8}(-f_{k-1} + 2f_k + f_{k+1}) \tag{2.13}$$

Euler's Predictor Method

Euler's Predictor-Corrector Method uses the simple Euler's Formula as the predictor and the improved Euler's Formula as corrector. The simple Euler's formula is given by:

$$y(t+h) = y(t) + hf(t,y) \tag{2.14}$$

and the improved Euler's Formula is given by:

$$y(t+h) = y(t) + \frac{h}{2}[f(t,y) + f(t+h, y+hf(t,y))] \tag{2.15}$$

Algorithm for Euler's Predictor-Corrector:

Algorithm steps for Euler's Predictor-Corrector are as follows:
1. f(t, y) ← 'Function to be supplied'
2. Enter t_0, y_0, h
3. $y_p \leftarrow y_0$ + h f(t_0, y_0)
4. $t_1 \leftarrow t_0$ + h
5. c ← y_0 + h/2 f(t_0, y_0)
6. y_c ← c + h/2 f(t_1, y_p)
7. if | y_p - y_c | < 10^{-6}
8. $y_1 \leftarrow y_c$
9. Write t_1, y_1
10. Else $y_p \leftarrow y_c$
11. Goto (6)
12. For i = 1(1)n, do till (25)
13. $y_p \leftarrow y_0$ + 2h f(t_0, y_0)
14. $t_2 \leftarrow t_1$ + h
15. c ← y_1 + h/2 f(t_1, y_1)
16. y_c ← c + h/2 f(t_2, y_p)
17. if | y_p - y_c | < 10^{-6}
18. Goto (21)
19. Else $y_p \leftarrow y_c$
20. Goto (16)
21. Write t_2, y_c
22. $t_1 \leftarrow t_2$
23. $y_0 \leftarrow y_1$
24. $y_1 \leftarrow y_c$
25. Next i
26. End

Chapter 3

Simulation Results

The Simulated Results of different Numerical Methods is obtained using High-Level Programming Languages; mentioned earlier in the Introduction. Initially the MATLAB codes for different methods are written using m-file and results are tested. These same methods are also designed in LabVIEW on the basis of algorithms which are discussed in the previous chapter-Numerical Methods. To verify the simulation results, the simple first order ordinary differential equation is approximated and passed as a function inside the MATLAB as well as LabVIEW. Then after obtaining the same results in both softwares at different iterations using different numerical methods; these methods were used to solve different type of system/models.

The concept of Mathematical modeling is used to define the different type of system models. The mathematical modeling of electrical & electronics circuit is done and the equations are design and approximated in the first order differential equations. These set of ODEs are passed as a function inside both softwares and the comparative study of the results is carried out for different numerical methods. The analysis & verification of these models is done by real-time implementation on the hardware; (discussed in later chapter/s-RT Implementation).

3.1 MATLAB Simulation

In this section, the details of designing various numerical methods in MATLAB and the code for the same are discussed. All initial value problems can be solved and the approximate solution can be obtained. If solution with higher precision is required then more computing efforts and sophisticated algorithms are used. One of the computing software, MATLAB is used for

verification of results using *ode* function of MATLAB. (Designing of various methods is done in LabVIEW).

Besides *ode* function, some other important MATLAB functions are also used for the coding of numerical methods. These functions solve the purpose of lengthy scripts which were to be written in **Turbo C** and simplify the codes.

feval is MATLAB command to solve the value of function. Syntax is feval(*function name*, *function values*). Example- feval(*f*, *value1*, *value2*, . . .).
This function is solved by writing the m-file in MATLAB where the function values are passed as arguments and the function call is made in this file which is considered as the main program file. Also, *plot* command is used in MATLAB script to plot the graph which is the part of our solution shown in different figures below. This *plot* command is enrich with various features where multiple plots with legends, different colors & pattern are made possible. Moreover, *title* is also used to give the title to each plot.

Using various numerical methods, single first order differential equation-

$$\mathbf{y' = 30 - 5y} \tag{3.1}$$

is computed and the results are obtained at different iterations for various step size.
The initial conditions defined for computing this equation is y(0) = 1 and the size of step is a very small value.

3.1.1 Euler's Method

Euler's approximation is obtained by updating the resulting values obtained from the given initial conditions. These values are a sequence of points computed successively from the previous point and the chosen step size. The function for the approximation is as follows:
Euler's Function

```
function [T,Y] = eulers(f,a,b,y0,m)
h = % step size;
T = zeros(1,m+1);
Y = zeros(1,m+1);
T(1) = % starting point;
Y(1) = % initial condition;
```

```
for j=1:m,
    Y(j+1) = Y(j) + h*feval(f,T(j),Y(j));
    T(j+1) = a + h*j;
end
```

Results of Euler's Method

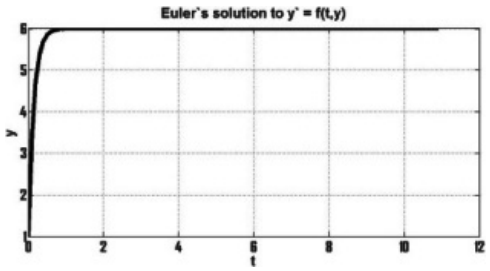

Figure 3.1: Euler's Approximation 109 iterations

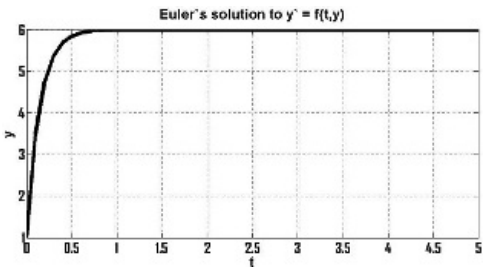

Figure 3.2: Euler's Approximation 50 iterations

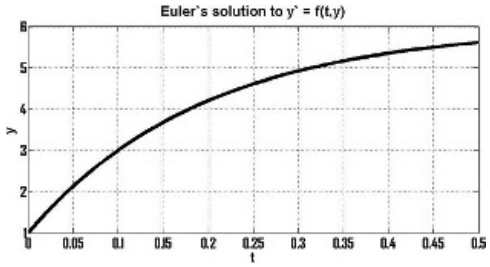

Figure 3.3: Euler's Approximation less iterations small step size

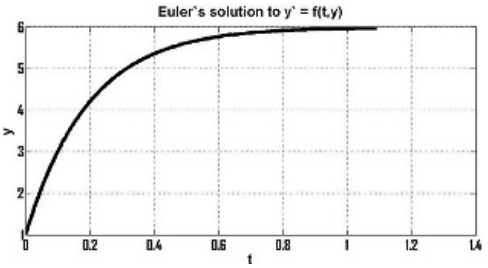

Figure 3.4: Euler's Approximation more iterations small step size

Different figures show the solution for different iterations using Euler's Method. The step-size is kept fix; h = 0.1 and the difference in graph is observed on the basis of variation in number of iterations. The figure (3.1) shows graph for more than 100 iterations and figure (3.2) shows graph for very less iterations.
If the step size is changed to h = 0.01 then the nature in the curve is changed drastically. The response is seen much slower if the iterations are very less; figure (3.3). But if the iterations are more, then the final value is achieved but time taken is much more compare to step size of h = 0.1.

3.1.2 Heun's Method

Similar to Euler's Method, this method also depends on initial conditions and appropriate step size; but the successive sequence of points is approximated after updating the results twice with the same values. This can be made clear from the function written below for the Heun's Method:

Heun's Function

```
function [T,Y] = heun(f,a,b,y0,m)
h = % step size;
T = zeros(1,m+1);
Y = zeros(1,m+1);
T(1) = % starting point;
Y(1) = % initial condition;
for j=1:m,
    s1 = feval(f,T(j),Y(j));
    k = Y(j) + h*s1;
    T(j+1) = a + h*j;
    s2 = feval(f,T(j+1),k);
    Y(j+1) = Y(j) + h*(s1 + s2)/2;
end
```

Results of Heun's Method

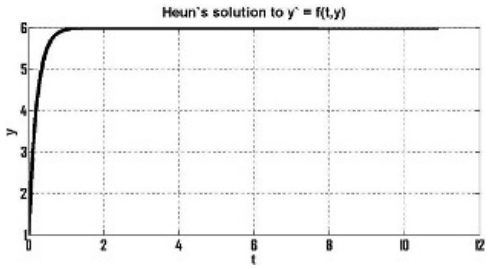

Figure 3.5: Heun's Method for 109 iterations

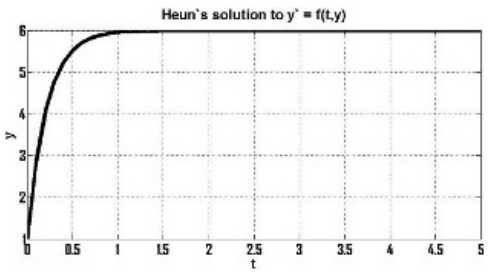

Figure 3.6: Heun's Method for 50 iterations

Similar type of results which were shown in Euler's Method are discussed in different figures for Heun's Method. Figures (3.5) & (3.6) are for step-size h = 0.1. Figures (3.7) & (3.8) are for step size h = 0.01.
Here, again the results for decreasing the step size and variation of iterations are same as that of Euler's Approximation.

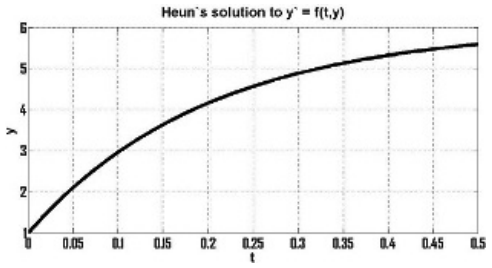

Figure 3.7: Heun's Method few iterations small step size

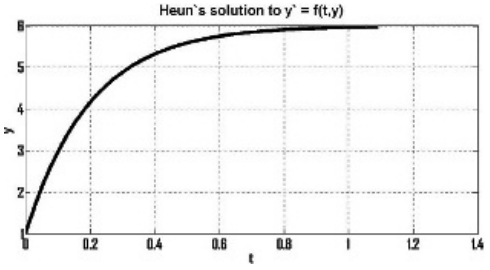

Figure 3.8: Heun's Method more iterations small step size

3.1.3 RK Method

Runge-Kutta (RK) Method is derived from Taylor Method to perform several function evaluations at each step and eliminates the necessity to compute higher derivatives. This method can be constructed for any order *N*. The function for fourth order (N=4) RK Method is as follows:

RK Function

```
function [T,Y] = rk_m(f,a,b,y0,m)
h = % step size;
T = zeros(1,m+1);
Y = zeros(1,m+1);
T(1) = % starting point;
Y(1) = % initial condition;
for j=1:m,
    t0 = T(j);
    y0 = Y(j);
    s1 = h*feval(f,t0,y0);
    s2 = h*feval(f,t0+h/2,y0+s1/2);
    s3 = h*feval(f,t0+h/2,y0+s2/2);
    s4 = h*feval(f,t0+h,y0+s3);
    Y(j+1) = yj + (s1 + 2*s2 + 2*s3 + s4)/6;
    T(j+1) = a + h*j;
end
```

This method is used to compute initial values of Predictor-Corrector (PC) Methods.

3.1.4 ABM Method

The multi-step ABM method requires four starting points which are computed by RK method. These points are passed in the predictor formula

and the next value is obtained. This value is passed in the corrector formula where is error is reduced and the minimal solution is obtained. Set of points are successively predicted, approximated and corrected based on the step size chosen for several iterations. The function for ABM is as follows:

ABM Function

```
function [T,Y] = abm(f,T,Y)
n = length(T);
if n<5, return, end;
F0 = feval(f,T(1),Y(1));
F1 = feval(f,T(2),Y(2));
F2 = feval(f,T(3),Y(3));
F3 = feval(f,T(4),Y(4));
h = T(2)-T(1); % step size
h2 = h/24;
a = T(1);
for k = 4:n-1,
    p = Y(k) + h2*(-9*F0 + 37*F1 - 59*F2 + 55*F3);
    T(k+1) = a + h*k;
    F4 = feval(f,T(k+1),p);
    Y(k+1) = Y(k) + h2*(F1 - 5*F2 + 19*F3 + 9*F4);
    F0 = F1;
    F1 = F2;
    F2 = F3;
    F3 = feval(f,T(k+1),Y(k+1));
end
```

Results of ABM Method

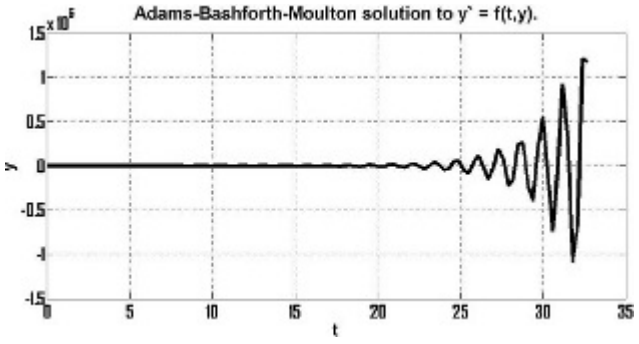

Figure 3.9: ABM Result for h = 0.15

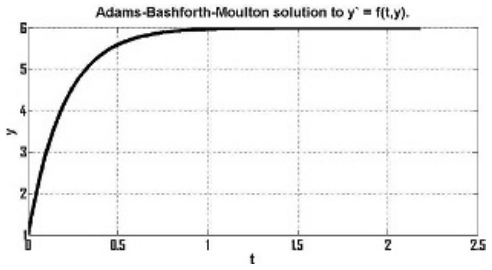

Figure 3.10: ABM Method for 109 iterations

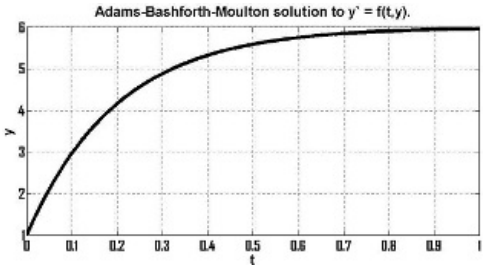

Figure 3.11: ABM Method for less iterations

Different figures show the solution for different iterations and the effect of change in graph with the change in step size. In this method, the accuracy of solution is more dependent on step size irrespective of the number of iterations. Higher number of iterations give faster response but slightly higher step value modifies the nature of curve; figure (3.9).
Figures (3.10) & (3.11) show the solution for very small step size h = 0.01 at different iterations.

3.1.5 Milne's Method

The multi-step Milne-Simpson method requires three starting points which are computed by RK method. These points are passed in the predictor formula and the next value is obtained. This value is passed in the corrector formula where is error is reduced and the minimal solution is obtained. Set of points are successively predicted, approximated and corrected based on the step size chosen for several iterations. The function for this method is as follows:

Milne Function

```
function [T,Y] = milne(f,T,Y)
n = length(T);
if n<5, return, end;
F0 = feval(f,T(1),Y(1));
F1 = feval(f,T(2),Y(2));
F2 = feval(f,T(3),Y(3));
F3 = feval(f,T(4),Y(4));
h = T(2)-T(1); % step size
a = T(1);
pold = 0;
yold = 0;
```

```
for k = 4:n-1,
    pnew = Y(k-3) + 4*h*(2*F1 - F2 + 2*F3)/3;
    pmod = pnew + 28*(yold-pold)/29;
    T(k+1) = a + h*k;
    F4 = feval(f,T(k+1),pmod);
    Y(k+1) = Y(k-1) + h*(F2 + 4*F3 + F4)/3;
    pold = pnew;
    yold = Y(k+1);
    F1 = F2;
    F2 = F3;
    F3 = feval(f,T(k+1),Y(k+1));
end
```

Results of Milne's Method

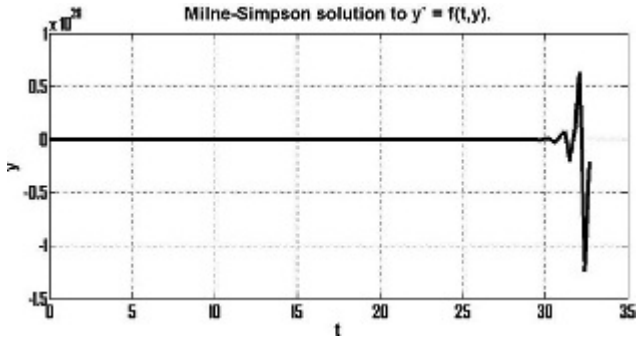

Figure 3.12: Milne Result for h = 0.15

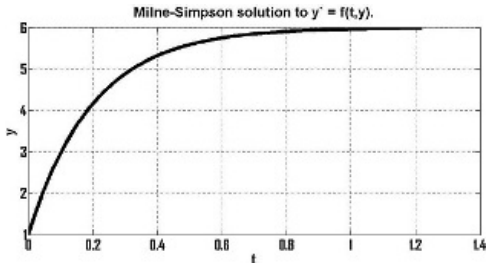

Figure 3.13: Milne's Method 61 iterations

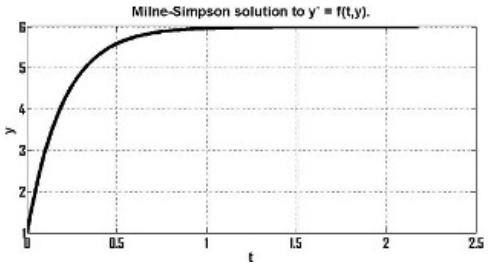

Figure 3.14: Milne's Method 109 iterations

Different figures show the solution for different iterations and the effect of change in graph with the change in step size. In this method, the accuracy of solution is more dependent on step size irrespective of the number of iterations. Higher number of iterations give faster response but slightly higher step value modifies the nature of curve; figure (3.12).
Figures (3.13) & (3.14) show the solution for very small step size h = 0.01 at different iterations.

3.1.6 Hamming's Method

The multi-step Hamming method is similar to Milne-Simpson Method except the correction formula as described earlier. The function for this method is as follows:

Hamming Function

```
function [T,Y] = hamming(f,T,Y)
n = length(T);
if n<5, return, end;
F0 = feval(f,T(1),Y(1));
F1 = feval(f,T(2),Y(2));
F2 = feval(f,T(3),Y(3));
F3 = feval(f,T(4),Y(4));
h = T(2)-T(1); %step size
a = T(1);
pold = 0;
cold = 0;
for k = 4:n-1,
    pnew = Y(k-3) + 4*h*(2*F1 - F2 + 2*F3)/3;
    pmod = pnew + 112*(cold-pold)/121;
    T(k+1) = a + h*k;
    F4 = feval(f,T(k+1),pmod);
    cnew = (9*Y(k) - Y(k-2) + 3*h*(-F2+2*F3+F4))/8;
    Y(k+1) = cnew + 9*(pnew-cnew)/121;
    pold = pnew;
    cold = cnew;
    F1 = F2;
    F2 = F3;
    F3 = feval(f,T(k+1),Y(k+1));
end
```

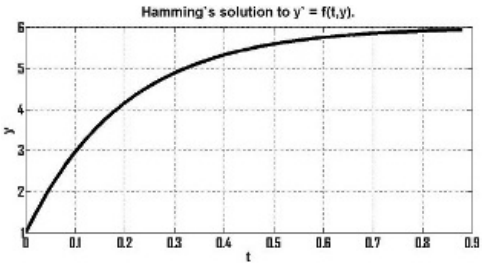

Figure 3.15: Hamming's Method 44 iterations

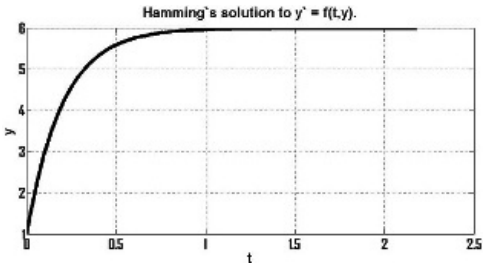

Figure 3.16: Hamming's Method 109 iterations

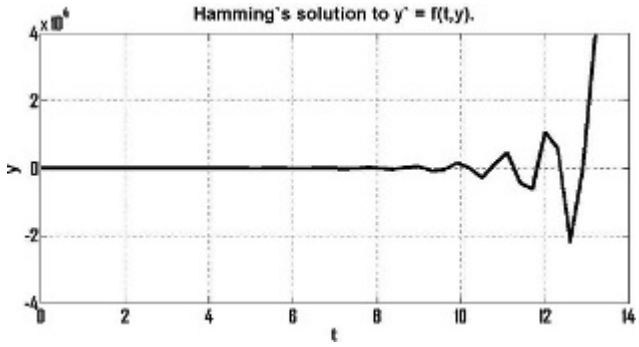

Figure 3.17: Hamming Result for h = 0.15

Results of Hamming's Method

Hamming's Method also have the results similar to above two predictive-corrective methods. Figures (3.15) & (3.16) show the solution for very small step size h = 0.01 at different iterations.
Figure (3.17) for slightly higher step size, h = 0.15

3.1.7 Simulation of Systems

The above methods can be used to solve the practical systems in the real-world. Any practical system can be modeled and simulated in the MATLAB using **ode solvers** or designing the model in **MATLAB Simulink** using various blocks. Few different real-world systems are taken into consideration for the simulation and verification on real-time as well. Here the simulation of these systems is discussed and the real-time implementation of all the system is discussed in the next chapter/s. One of the systems is **Full-wave Rectifier with Three-Element Filter**, other is **Unsymmetrical**

Voltage Doubler & another is a Chaotic System, **Lorenz Attractor**.

Full-wave Rectifier With Three-Element Filter

Functions of current and voltages are considered as state variables in this circuit to carry out time-domain analysis. The values of currents and voltages corresponds to state variables shown in the set of differential equations below:

$$\frac{dx(t)}{dt} = \frac{1}{C_1} * y(t) - \frac{1}{C_1 R} * x(t) \tag{3.2}$$

$$\frac{dy(t)}{dt} = \frac{1}{L} * z(t) - \frac{1}{L} * x(t) \tag{3.3}$$

$$\frac{dz(t)}{dt} = \frac{1}{C_2} * I_s[exp[\frac{|u_s(t)| - z(t)}{2 * V_T}] - 1] - \frac{1}{C_2} * y(t) \tag{3.4}$$

where, u_s(t) = 10 sin(2π*50*t) Volts.

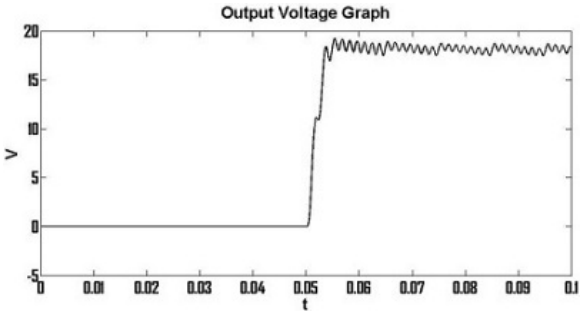

Figure 3.18: Result of solver ode113

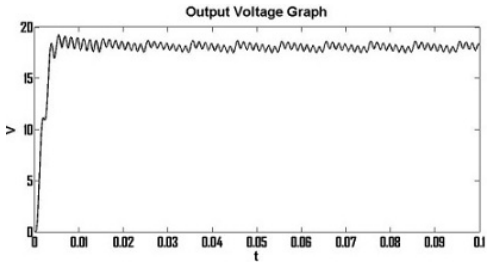

Figure 3.19: Result of solver ode23

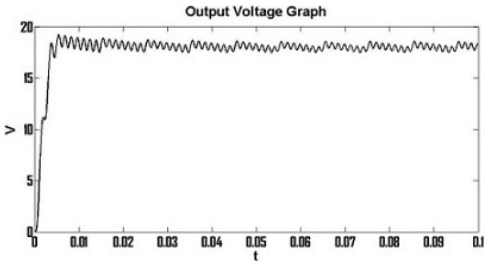

Figure 3.20: Result of solver ode23s

These set of equations (3.2), (3.3) & (3.4) are solved in MATLAB using *odeset*. The solutions obtained with zero initial conditions are shown in figures which are the simulation results of output voltage of the system. Graph shown in figure (3.18) is the result of *ode113*. ***ode113*** is a multi-step solver which is equivalent to ABM PC method. The other solver is *ode23* which is an implementation of an explicit Runge-Kutta (pair of Bogacki and Shampine) method. The result of which is shown in figure (3.19). Figure (3.20) shows the graph obtained from *ode23s* solver; based on a modified Rosenbrock formula of order 2.

Voltage Doubler

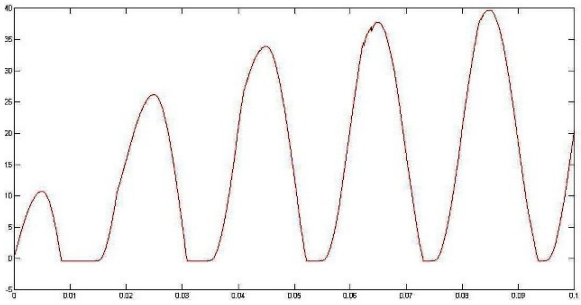

Figure 3.21: Result of solver ode113

Similar to previous system, this circuit is also analyzed using node voltages. These nodes are used as state variables. Here also, applying the concept of mathematical modeling for different state variables the first order differential equations are obtained as follows:

$$\frac{dx(t)}{dt} = \frac{du_s(t)}{dt} + \frac{I_s}{C_1}[exp(\frac{-x(t)}{V_T}) - 1] - \frac{I_s}{C_1}[exp(\frac{x(t) - y(t)}{V_T}) - 1] \quad (3.5)$$

$$\frac{dy(t)}{dt} = \frac{I_s}{C_2}[exp(\frac{x(t) - y(t)}{V_T}) - 1] - \frac{1}{RC_2}y(t) \quad (3.6)$$

where, $u_s(t) = V_m$ sin(2π*50*t) Volts.

These set of equations (3.5) & (3.6) are also solved in MATLAB using *odeset*. Similar to previous system the solutions obtained with zero initial conditions are shown in figures which are the simulation results of output

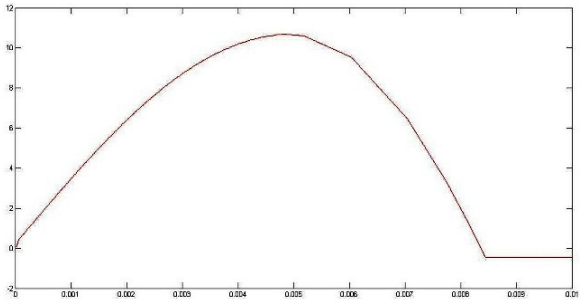

Figure 3.22: Result of solver ode23

voltage of the system. Graph shown in figure (3.21) is the result of *ode113* a multi-step solver equivalent to ABM PC method. The other solver is *ode23* which is an implementation of an explicit Runge-Kutta (pair of Bogacki and Shampine) method. The result of which is shown in figure (3.22). Figure (3.23) shows the graph obtained from *ode23s* solver; based on a modified Rosenbrock formula of order 2.

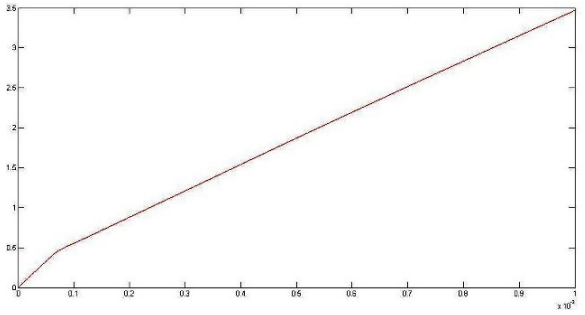

Figure 3.23: Result of solver ode23s

Lorenz Attractor

The Lorenz equations, given by Edward Lorenz are well-known for having chaotic solutions for several parameter values and specific initial conditions. The model consists of three ordinary differential equations known as the Lorenz equations:

$$\frac{dx}{dt} = 10(y - x) \tag{3.7}$$

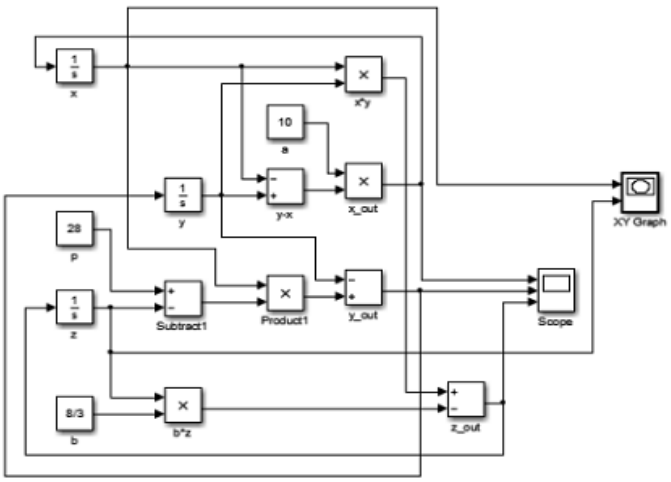

Figure 3.24: Lorenz Attractor Design in Simulink

$$\frac{dy}{dt} = x(28 - z) - y \tag{3.8}$$

$$\frac{dz}{dt} = xy - \frac{8}{3}z \tag{3.9}$$

The above set of equations (3.7), (3.8) & (3.9) are designed in MATLAB Simulink, figure (3.24). The initial conditions are set to [0.6 0.6 0.6] and solver ode23s is used to solve the system and accordingly the chaotic response is generated as shown in figure (3.25).

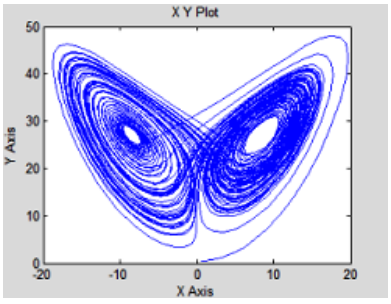

Figure 3.25: Lorenz Attractor Chaotic Response

3.2 LabVIEW Simulation

In this section, the details of designing various numerical methods in LabVIEW using the various algorithms are discussed. The design of various algorithms is done in the simplest way in LabVIEW with the graphical programming facility of LabVIEW. The LabVIEW is a high level programming language with the ready-made blocks available and the designer/ programmer can get rid of the program codes. The description for various blocks used in the design of methods is done under the different methods describing each blocks used for designing the algorithm.

The various blocks used for the design in the **Block Diagram** of LabVIEW are from the ***Express VI & Programming Section*** in LabVIEW. Also, ***Mathematics Section*** is used more for designing various equations & formulas of the numerical methods. The complex design is converted into subVI using **Edit** → **SubVI** and the solution obtained from these subVIs are processed further. Arrays, Numerics, Comparison, Graphs and other controls/indicators are used from the **Front Panel** of LabVIEW. The same equation (3.1) is solved using different numerical methods for comparison and verification of the results. In LabVIEW this equation is passed through formula node were the equation is solved with the given initial conditions and chosen step size. The equation is solved for fix number of iterations in all the methods so that the comparative analysis can be done easily.

3.2.1 Euler's Method

In figure (3.26), the outer most loop is the **For Loop** located on ***Functions*** → ***Programming*** → ***Structures*** Palette. The blocks under the area of that loop are executed as per For Loop logic of C program. The two terminals in blue **N** & ***i*** are count terminal and iteration terminal respectively. The number of iterations for which the loop is executed can be set by the count value. Until the count value is connected the loop is broken and the code execution is not possible showing the error in For Loop. The number of iterations executed can be monitored from the terminal **i**.

The loop inside for loop where the equation is written, is **Formula node** located on ***Functions*** → ***Mathematics*** → ***Scripts & Formulas***. The formula node supports the different mathematical equations which can be written in the form of script under the area of node. The border of the node is used to connect the inputs and outputs of the formula. These input-outputs are added by right clicking on the border of the node. This node

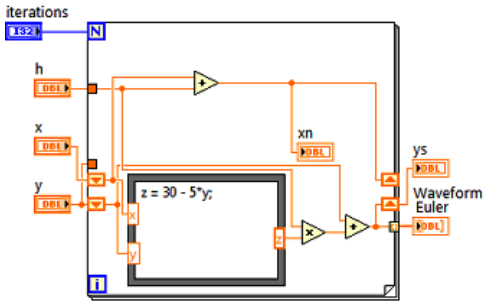

Figure 3.26: Euler Method LabVIEW code

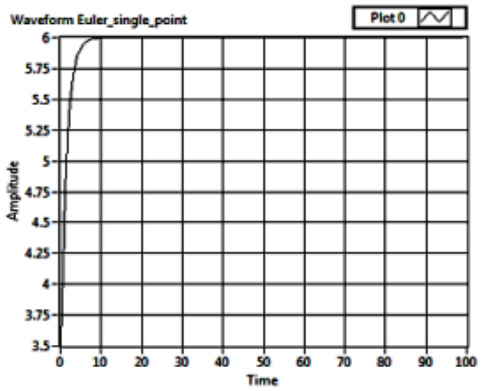

Figure 3.27: Euler Method 100 iterations

evaluates the formula as per the input values and the solution can be seen by connecting the indicator with desired output connector.

The rest of the blocks are used to design the Euler's formula using **add**, **multiply** etc. blocks located at ***Functions*** → ***Express Numeric***. The initial conditions and step size required for the calculation are passed through connection of controls; where a user can set the desired values. These controls are placed in the Front Panel which is also called a GUI of LabVIEW.

The figure (3.27), shows the graph of equation (3.1) solved using Euler's Method. This graph is displayed using the **Controls** → **Graph Indicator** in Front Panel.

3.2.2 Heun's Method

The figure (3.28) shows the Heun's Code designed in LabVIEW. The blocks used here are mostly similar to that of Euler's Method. But there are two formula node used in the Heun's code. As discussed in the function of Heun's Method in section (3.1.2) the result of the equation is updated twice in the same iteration and hence two nodes are required which passes different input values updated from the equation in that same iteration. In addition to this the rest of the blocks are **add**, **multiply**, **divide** etc. from the Numeric Palette.

The figure (3.29), shows the graph of equation (3.1) solved using Heun's Method. This graph is displayed using the **Waveform Graph** in Front

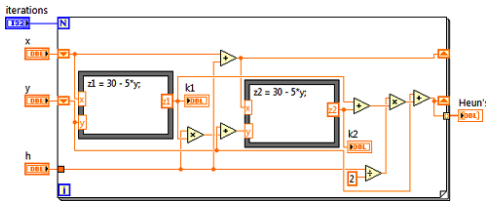

Figure 3.28: Heun's Method LabVIEW code

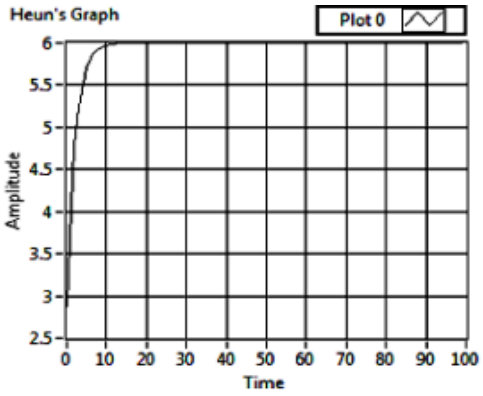

Figure 3.29: Heun's Method 100 iterations

Panel. The number of iterations in Heun's as well as Euler's Method was kept same for the comparison of results for single point methods.

3.2.3 RK Method

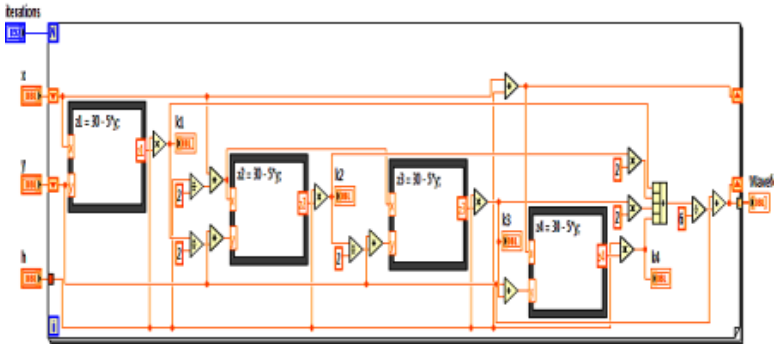

Figure 3.30: Fourth Order RK Method LabVIEW code

The fourth order Runge-Kutta (RK) method is considered the standard single point numerical method. This method is design in LabVIEW (figure (3.30)) for calculating the initial points of Predictive-Corrective (PC) methods. The fourth order calculations require **four formula nodes** in which the weights are updated four times in order to obtained more accurate solution directly after one iteration.

3.2.4 ABM Method

One of the Predictive-Corrective (PC) methods is ABM method, shown in figure (3.31). The Predictive & Corrective formulas are very complex and their design in LabVIEW is very lengthy. So this method has different SubVIs designed in the LabVIEW which have had reduced the complexity of the code. Different SubVIs are made for step-size calculation, passing the equation in formula node, Predictive formula, Corrective formula etc.

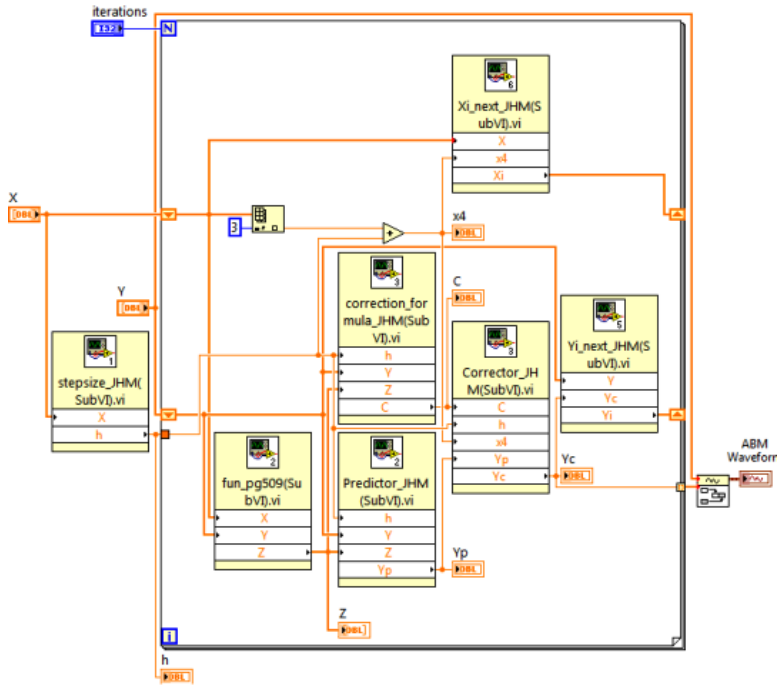

Figure 3.31: ABM code in LabVIEW

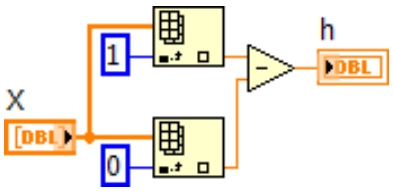

Figure 3.32: ABM step-size

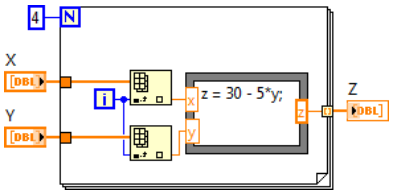

Figure 3.33: Equation in Formula Node

Figure (3.32) shows the SubVI of step-size. There is need of specific step-size in these PC methods. As we discussed earlier that four initial points are needed to start ABM method, 1-D array is used in which four values are stored. These values are required to pass one after the other and hence indexing is to be done. The two blocks connected with '0' & '1' before *subtract* block, are **Index Array** blocks located at ***Functions*** → ***Programming*** → ***Array*** Palette. Other figure (3.33) shows SubVI for formula-node. Here also the formula node is used to solve the equation (3.1) as in previous methods. The for loop is used here with fix number of iterations; that is '4'. This gives four values as output and is stored in 1-D array as mentioned above. These values are used in predictive and corrective approximation as discussed below.

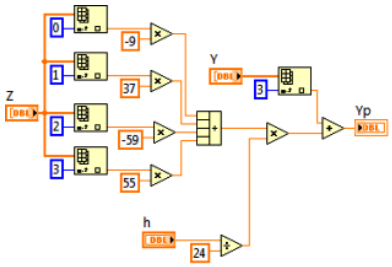

Figure 3.34: ABM Predictor

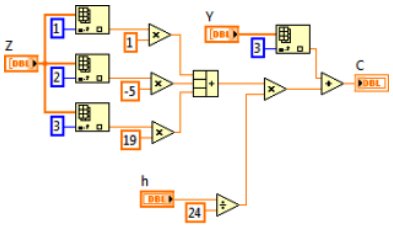

Figure 3.35: ABM Corrector

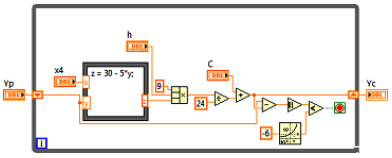

Figure 3.36: ABM Error $< 10^{-6}$

Figure (3.34) shows the predictive design of ABM method & figure (3.35) shows corrective design. These Predictor and Corrector are design with the simple **Mathematics** blocks. The predictive and corrective formulas are designed such that all four values stored in 1-D array are required in the calculation in same iteration. As these points are passed through array, indexing is required. *Index Array* blocks are used along with the *numeric constants* where the numeric constants are the fix (constant) values of the formula. These blocks are connected with different math blocks such that

the required formula is designed. This predictive corrective action is done until the error is not reduced to the minimum.

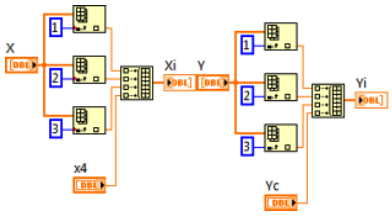

Figure 3.37: ABM Next Values

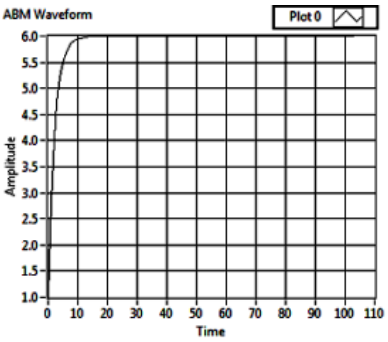

Figure 3.38: ABM Method for 100 iterations

As per the algorithm in section (2.2.2) the error must be less than 10^{-6}. Figure (3.36) shows the LabVIEW code for the error reduction and giving the corrected values as output. The outer loop in this figure (3.36) is ***while loop*** located in ***Functions*** $\rightarrow$ ***Programming*** $\rightarrow$ ***Structures*** Palette. The predicted points are passed in the formula node and the equation is evaluated till the condition is satisfied, that is error is less than 10^{-6}. The blocks under **while loop** are connected with **stop if true** logic, which is indicated by red button type icon at right corner of while loop. The comparison block is used to design the logic for comparing the error value. Comparison blocks are found in ***Functions*** $\rightarrow$ ***Programming*** $\rightarrow$ ***Comparison***. These corrected values are passed for the next iteration (figure (3.37)) where the another set of values will be approximated and at the end graph is obtained as shown in figure (3.38)

3.2.5 Milne's Method

Milne's Predictive-Corrective Method has three initial points. Figure (3.39) shows the Milne's LabVIEW code. Similar to ABM, this method also has some lengthy formulas due to which the coding becomes complex. The complexity can be reduced by converting several codes of LabVIEW into SubVIs.

Figure (3.40) shows the SubVI for step-size calculation which is similar to ABM method. Also from the figure (3.41) it is clear that the formula

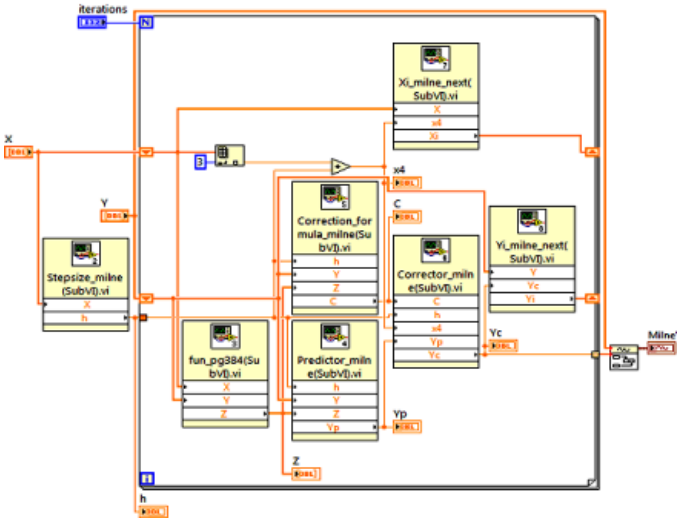

Figure 3.39: Milne's code in LabVIEW

node is used to solve the equation (3.1); but here 3 initial points are required and the zeroth point is not considered later on. Therefore, the for loop is fixed with 3 iterations and the first iteration is incremented by one so that the iterations start from '1' instead of '0'. These evaluated values are stored in 1-D array as it was in the case of ABM method and then the Predictive-Corrective SubVIs were designed to obtain the solution for next iterations [figure (3.42) & (3.43)].

The Predictive-Corrective design for this method is done as per same logic used in ABM method. Here also the math blocks are connected to design the required formula. Also, the algorithm is such that the absolute error must be less than 10^{-6}. Therefore, the logic for obtaining the corrected points for next iterations is similar to ABM method. Only difference we will see is that the initial point from which the step-size is calculated is never updated; (it can be made clear from the algorithm in section (2.2.2)). Figure (3.45) shows that how the corrected values are updated for next iterations and the graph is seen in figure (3.46).

***Note:** Some Research is required in this method to maintain stability & obtaining bounded solution.

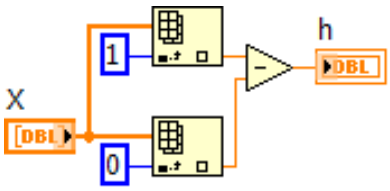

Figure 3.40: Milne step-size

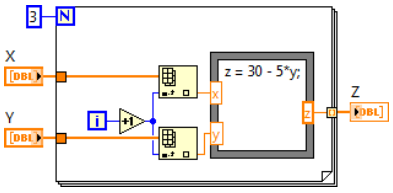

Figure 3.41: Equation in Formula Node

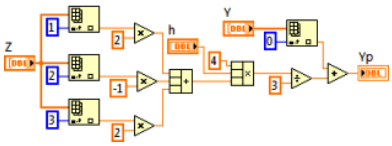

Figure 3.42: Milne Predictor

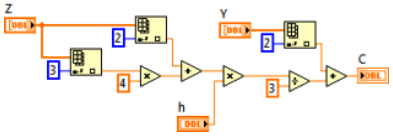

Figure 3.43: Milne Corrector

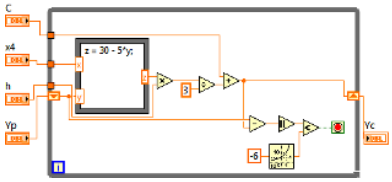

Figure 3.44: Milne's Error $< 10^{-6}$

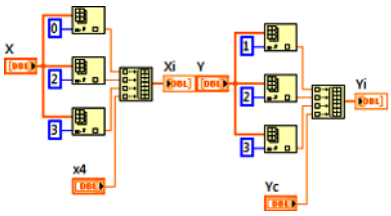

Figure 3.45: Milne Next Values

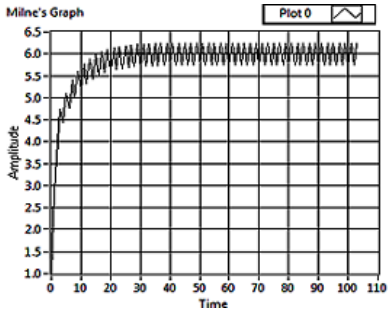

Figure 3.46: Milne's Method (unstable)* 100 iterations

3.2.6 Euler's Predictor-Corrector Method

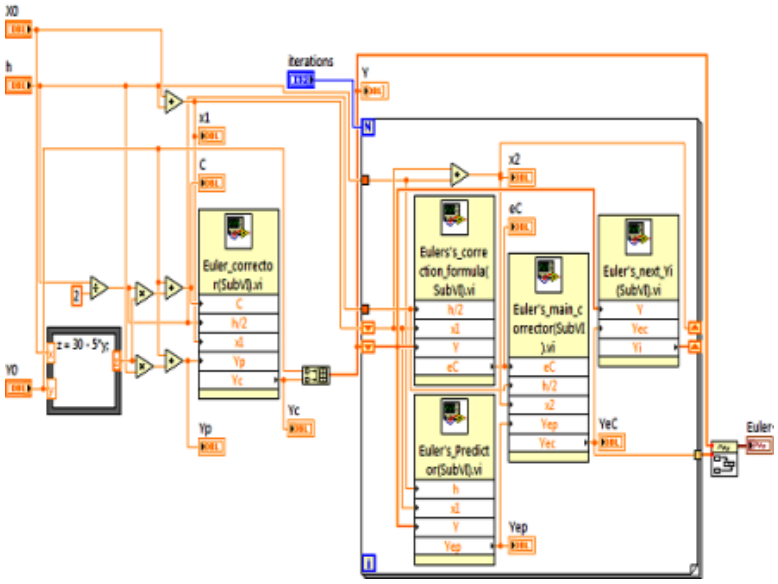

Figure 3.47: Euler's PC in LabVIEW

Figure (3.47) shows the LabVIEW code for Euler's Predictor-Corrector method. This method is single-point, self-starting, predictor-corrector method. The single point is approximated as per Euler's approximation and it is updated till the absolute error is less than 10^{-6}. This value is passed into the predictor-corrector algorithm and the predictive-corrective values are obtained. These values are again updated till the error is less than 10^{-6} (figure (3.48)).

Initially, the method is single point, but the predictive & corrective values are obtained on the basis of present and previous values. But the method uses its own updated value, it is still considered as self-starting method. The PC algorithms design logic is similar to that of previous two PC methods. Figures (3.49) & (3.50), respectively shows predictor & corrector code designed in LabVIEW for this method. The absolute error is minimize up to less than 10^{-6} and the updated values are further solved in the next iterations by this algorithm and the resulting graph is obtained as shown in figure (3.51).

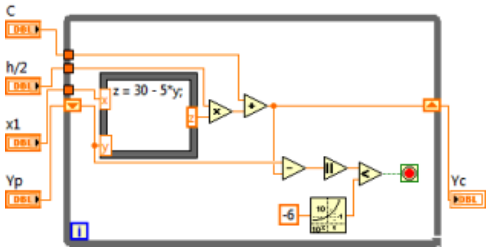

Figure 3.48: Euler's PC Error $< 10^{-6}$

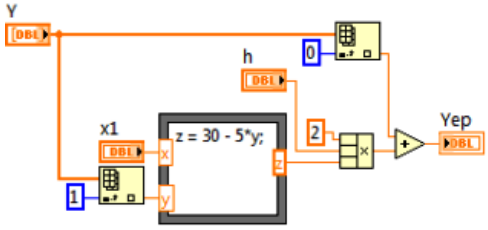

Figure 3.49: Euler's Predictor

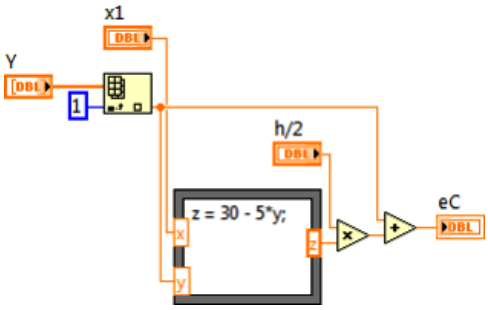

Figure 3.50: Euler's Corrector

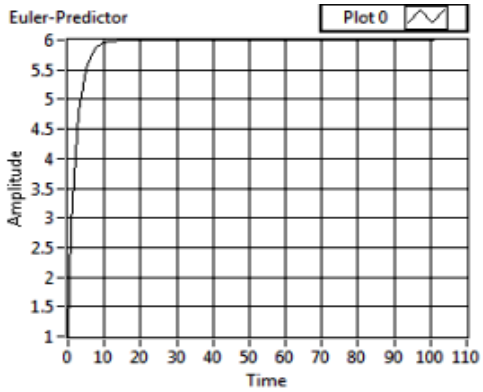

Figure 3.51: Euler's PC for 100 iterations

3.2.7 Simulation of Systems

The numerical methods designed in LabVIEW are to be used for solving the practical systems of real-world. The mathematical modeling of real systems is done and the ordinary differential equations are approximated in the form of simple equations. These set of equations are solved using numerical methods described above and simulated results can be obtained. The comparative analysis is discussed in Appendix A.

The same systems which are simulated in MATLAB are used for the simulation in LabVIEW. The single point numerical methods fail to simulate such non-linear and chaotic systems. Predictor-Corrector methods are used to solve such systems. The set of equations (3.2), (3.3) & (3.4) resembles the model of full-wave rectifier, equations (3.5) & (3.6) resembles to voltage doubler model and equations (3.7), (3.8) & (3.9) are for chaotic system. These systems are solved using different PC methods and the results are obtained. The comparative analysis for these systems is also discussed in Appendix A.

Full-wave Rectifier With Three Element Filter

The Euler's and ABM PC methods produce similar response and it is shown in figure (3.52). Figure (3.53) shows the output response of Milne's PC method which produces slower response compare to previous two. Also this method gives unbounded output in some cases and hence, it is considered least effective method.

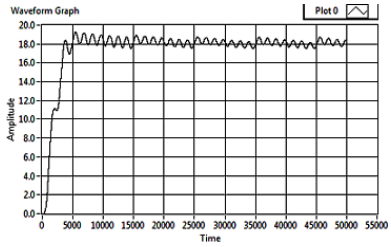

Figure 3.52: Output Response of EPC & ABM method

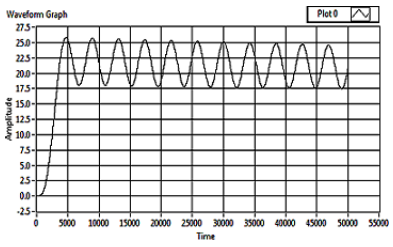

Figure 3.53: Output Response of Milne method

Unsymmetrical Voltage Doubler

The equations (3.5) & equation (3.6) are also solved in LabVIEW using various predictor-corrector numerical methods like it was done for previous

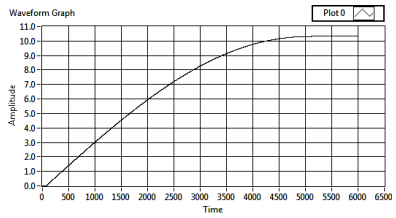

Figure 3.54: ABM Predictor Corrector Result

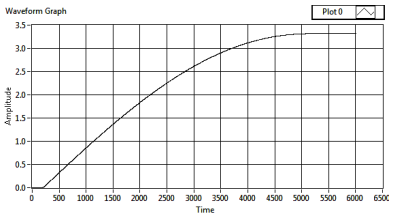

Figure 3.55: Milne Predictor Corrector Result

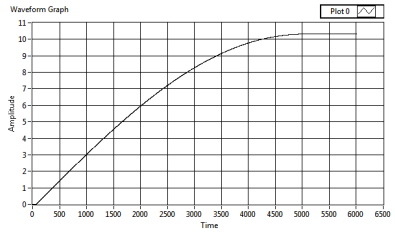

Figure 3.56: Euler Predictor Corrector Result

system. Simulated results of ABM and Milne methods are obtained as shown in figures (3.54) & (3.55) respectively. Also figure (3.56) shows the result of Euler's PC method which gives more accurate solution than above two methods.

Chaotic System

The set of equations (3.7), (3.8) & (3.9) are ordinary differential equations for Lorenz Attractor. Here also Euler's and ABM PC methods produce similar chaotic response as shown in figure (3.57). But Milne's PC method fails to generate the chaotic response (figure (3.58)) as stability is not guarantee.

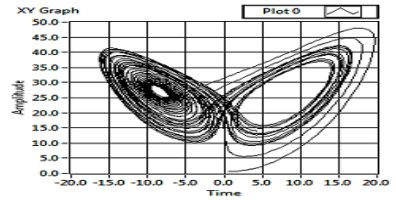

Figure 3.57: Chaotic Response of EPC & ABM method

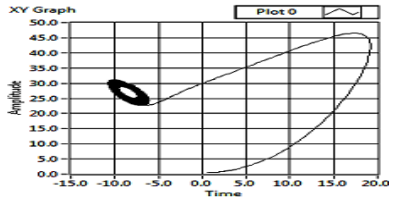

Figure 3.58: Response of Milne method

Chapter 4

Real-Time (RT) Implementation

A real-time implementation of any application/program or algorithm functions within a time frame that the user senses as immediate or current. In other words, RT implementation must be deterministic; the latency must be less than a defined value, usually measured in seconds. The defined task or set of tasks requiring the maximum length of time on a given hardware platform is known as its implementation time.

Real-Time implementation can be possible with NI C-RIO which is the hardware support from National Instruments the developers of LabVIEW. To develop a Compact-RIO application in LabVIEW using the RIO Scan Interface there is a requirement of Compact-RIO Reconfigurable Embedded system with LabVIEW to make a simple process-control VI. It is mandatory to learn concepts and techniques for developing Compact-RIO applications with Scan Interface. The Scan Interface enables the use of C Series modules directly from LabVIEW Real-Time. The complete description for this is discussed in section below.

4.1 C-RIO

The NI C-RIO-9073 is an integrated system with the combination of a real-time processor and a reconfigurable field-programmable gate array (FPGA) within the same chassis for embedded machine control and monitoring applications. The Specification Summary is given in Appendix B Table (B.1).

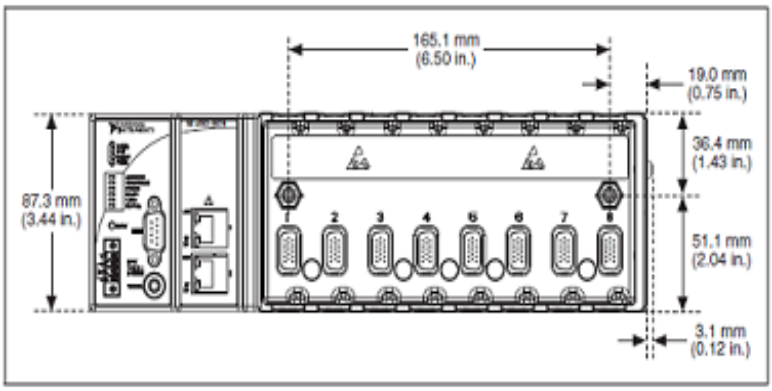

Figure 4.1: C-RIO Layout

To start with Compact RIO Reconfiguration Embedded System there is a need to install following supports:

- Compact-RIO reconfigurable embedded chassis with integrated intelligent real-time controller.
- C Series I/O modules.
- DIN rail mount kit (for DIN rail mounting only).
- Two M4 or number 10 pan-head screws (for panel mounting only).
- Power supply.

Required Components are listed in table (4.1) below:

Creating a Project in Scan Interface Mode; use a LabVIEW project to manage VIs, targets, and I/O modules on the development computer. To create a project follow the steps below:

a. Start LabVIEW.

b. Create Empty Project through Getting Started window to open the Project Explorer window. Or select File→New Project to open the Project Explorer window.

Required Software	
NI LabVIEW 2009 or later	
NI LabVIEW Real-Time Module 2009 or later	
NI LabVIEW FPGA Module 2009 or later	optional
NI-RIO 3.2 or later	
Required Hardware	
Power supply for the controller	
Ethernet connection and cable	
Analog Input (AI) module	NI 9201/NI 9205/NI 9206/ NI 9215/NI 9221
Analog Output (AO) module	NI 9263/NI 9264/NI 9269
Digital Input (DI) module	NI 9401/NI 9411/NI 9421/ NI 9423
Digital Output (DO) module	NI 9401/NI 9472/NI 9474
For Scan Interface	
cRIO-9073/cRIO-9074 Integrated Real-Time Controller and Chassis	
For FPGA Interface	
Compact-RIO controller and chassis	

Table 4.1: NI C-RIO Requirements

c. Right-click the top-level project item in the Project Explorer window and select New→Targets and Devices to display the Add Targets and Devices dialog box.

d. Select 'Existing target or device' radio button.

e. Expand Real-Time Compact-RIO.

f. Select the Compact-RIO controller to add to the project and click OK.

g. If LabVIEW FPGA is installed, then Select Programming Mode dialog box appears. Select Scan Interface to put the system into Scan Interface mode.

h. Click Continue. LabVIEW adds the controller, the chassis, and all the modules to the project.

i. Select 'Discover' in the Discover C Series Modules dialog box if it appears.

j. Select File→Save Project and save the project as x.lvproj.

Adding AI and AO to the VI; in real-world applications, the AI channel receives input from same device or another device and the AO channel sends a voltage to a device in a physical process. Devices can be connected to the analog modules and controlled or monitored, using the following steps by adding AI and AO to the VI:

a. Expand the AI/AO module item in the Project Explorer window and the I/O variable items for that module channels will be available.

b. Drag and drop the required I/O variable from the Project Explorer window on to the block diagram of the VI.

c. Right-click the I/O variable and select Create→Control/Indicator to create a control/indicator on the front panel. Or connect the wire of interested variable in the existing VI to get its physical value.

4.1.1 Analog Input (AI) & Analog Output (AO) Modules

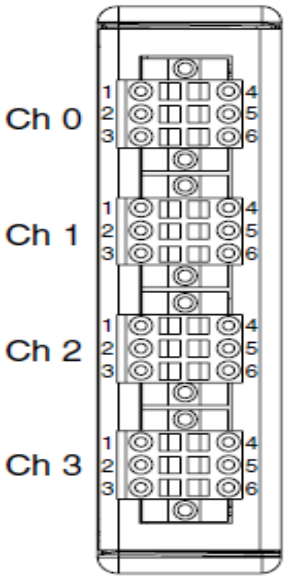

Figure 4.2: NI 9219 AI Module

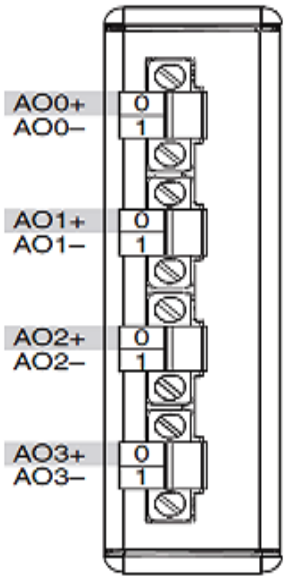

Figure 4.3: NI 9269 AO Module

The NI 9219 is a 4-channel universal C series AI module for multipurpose testing in NI Compact-RIO and NI Compact-DAQ C Series chassis. Individual channel selection is possible to measure various signals from different sensors. Measurement ranges and different sensor support is provided in the specifications summary given in Appendix B Table (B.2). To use I/O from this module in a VI, drag and drop I/O variables from the Project Explorer window to the block diagram of the VI (discussed in previous section). The I/O variables write floating-point values to the channels in volts.

The NI 9269 is also a 4-channel AO module for NI Compact-RIO and NI Compact-DAQ C Series chassis. The channel-to-channel isolation in this module increases safety, improved signal quality, and the ability to stack channels to output up to 40V. Channel-to-channel isolation is commonly needed for applications that have multiple electrical systems, such as automotive test, or industrial applications where the noise is more and hence it often requires multiple ground planes; which is supported in this module. The specification summary is given Appendix B Table (B.3). Similar to AI module one can use I/O from this module in a VI, drag and drop I/O variables from the Project Explorer window to the block diagram of the VI (discussed in previous section). The I/O variables write floating-point values to the channels in volts.

4.1.2 Results of Real-Time Implementation

With all these hardware setup and configurations of devices discussed above; the real-time implementation of various numerical methods is possible. The following figures show the result captured on Digital Storage Oscilloscope (DSO) from the AO module.

Figures (4.4) to (4.8) are the results of equation (3.1); whereas figures (4.9) and (4.10) show the result of full-wave rectifier, figures (4.11) and (4.12) show the result of voltage doubler and figure (4.13) shows the chaotic behavior of Lorenz Attractor (equations (3.7), (3.8) & (3.9)). Figures (4.9) to (4.12) are results comparing Euler's PC method and Milne PC method.

Figure 4.4: Euler single-point on RT target

Figure 4.5: Heun's on RT target

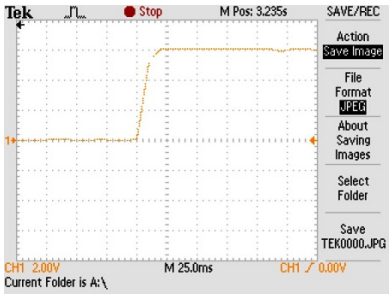

Figure 4.6: ABM on RT target

Figure 4.7: Euler's Predictor-Corrector on RT target

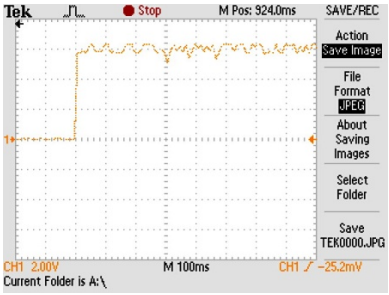

Figure 4.8: Milne's Predictor-Corrector on RT target

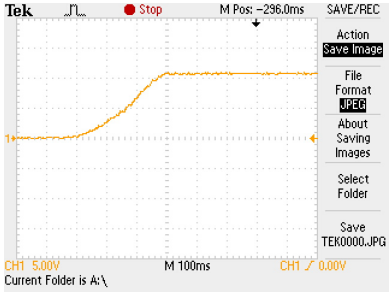

Figure 4.9: Euler PC Full-wave Rectifier Filter Response

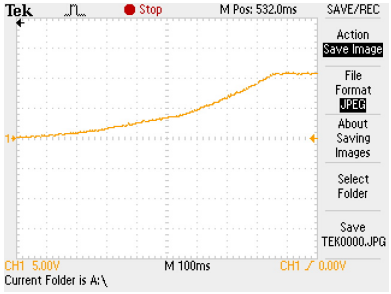

Figure 4.10: Milne PC Full-wave Rectifier Filter Response

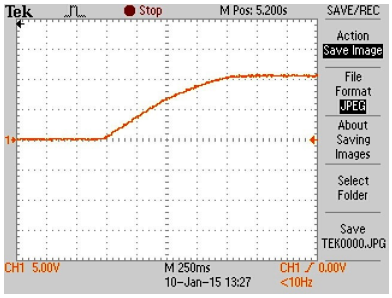

Figure 4.11: Euler PC Voltage Doubler Response

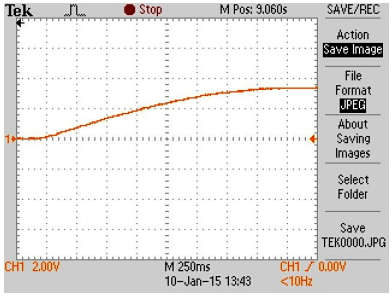

Figure 4.12: Milne PC Voltage Doubler Response

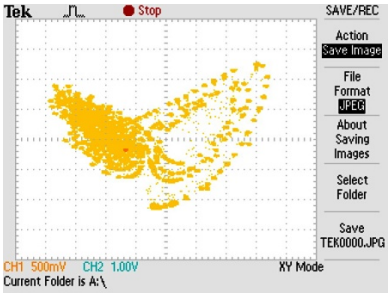

Figure 4.13: Chaotic Response on RT target

4.2 Circuit Design

The various numerical methods are simulated, tested and implemented on real-time hardware. Also simulation of some real system is also carried out and the results are discussed above. Thus, for the verification of all the above results, it is necessary to prepare the required real system and the results obtained can be compared and verified. Hence, there is a need to design specific circuits for the corresponding systems and practical observations are recorded. The details described below contain type of system design, its specifications, component values used, other requirements etc.

4.2.1 Full-wave Rectifier With Three Element Filter

One is Full-wave Rectifier Integrated with Three-Element Filter circuit, shown in Figure (4.14). The design (figure (4.14)) is clear that the three-element filter is a low-pass filter. The circuit is supplied by alternating voltage, which after transforming has practical value equal to $u_s = 19\sin(2\pi 50t)$ V. 12-0-12 transformer is used for transforming the voltage which gives practically, output rms voltage equal to 14V (measured across center-tapped). Hence, peak-to-peak $V_{p-p} = 19$V. The time-domain analysis of circuit is done by mathematical modeling of control systems which gives the system model. This model is prepared by set of first order ordinary differential equations as in equation (3.2),(3.3) & (3.4).

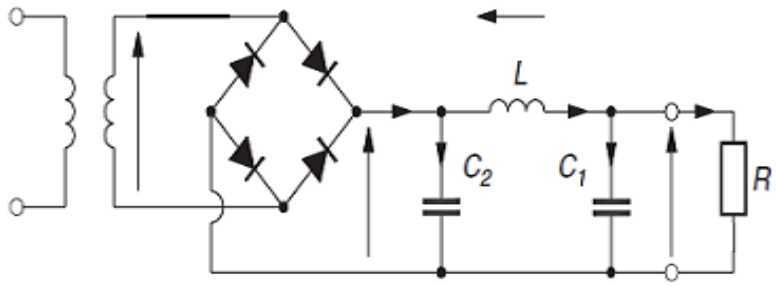

Figure 4.14: Full-wave Rectifier With Three Element Filter Circuit

The components required for designing this circuit are transformer (12-0-12; center-tapped) $\rightarrow$ 1, diode-bridge $\rightarrow$ 1, two capacitors, one inductor and a resistor. The combination of R-L-C forms the low-pass filter design. The value of capacitors is 1000 μF, 0.1 mH inductor and 150 Ω, 10 W resistor.

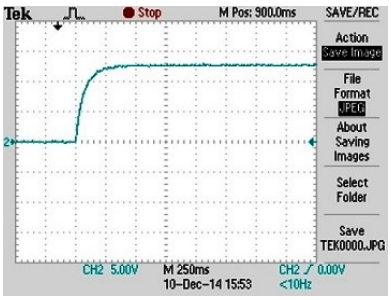

Figure 4.15: Output Voltage Graph of Circuit

DSO is used to observe the output voltage in the form of graph which is shown in figure (4.15).

4.2.2 Unsymmetrical Voltage Doubler

Another is Unsymmetrical Voltage Doubler circuit shown in Figure (4.16). The circuit is supplied by alternating voltage, which after transforming has practical value equal to $u_s = 21\sin(2\pi 50t)$ V. 6-0-6 transformer is used for transforming the voltage which gives practically, output rms voltage equal to 14V. Hence, peak-to-peak $V_{p-p} = 21$V. The time-domain analysis of circuit is done by mathematical modeling of control systems which gives the system model. This model is prepared by set of first order ordinary differential equations (3.5) & (3.6).

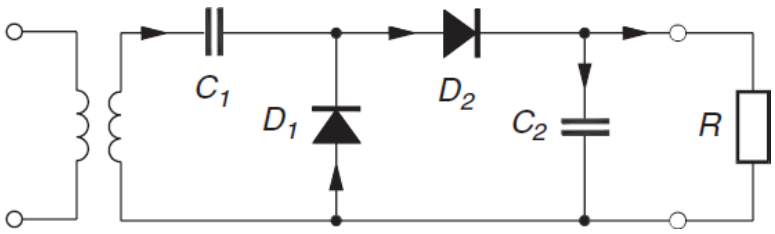

Figure 4.16: Unsymmetrical Voltage Doubler Circuit

The components required for designing this circuit are transformer (6-0-6; center-tapped) $\rightarrow$ 1, two silicon-diodes, two capacitors and a resistor. The value of capacitors is 1000 μF and 10 kΩ resistor. DSO is used to observe the output voltage in the form of graph which is shown in figure (4.17).

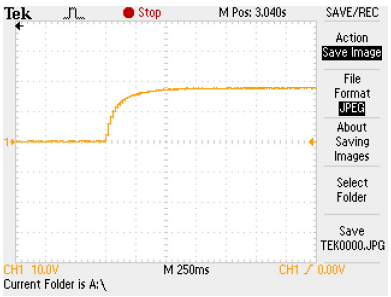

Figure 4.17: Output Voltage Graph of Circuit

4.3 Real-Time Data Acquisition in LabVIEW

There are different types of USB DAQs & C-RIOs available to acquire the data. For acquiring analog voltage USB 6008 (shown in figure (4.18)) is used. The specifications for the same are listed in Appendix B Table (B.4).

Figure 4.18: USB 6008 DAQ

The above systems Full-Wave Rectifier With Three Element Filter and Unsymmetrical Voltage Doubler are supplied with the step-down transformer input. The transformer was connected with USB DAQ but due to faster sampling; at higher frequency, numerical methods fail to give appropriate solution. Hence, the frequency analysis of both systems was carried out using low frequency signal. To solve the mathematical model of both the systems using real-time input the low frequency sine wave is given as input through USB DAQ in the LabVIEW VI using Function Generator.

Frequency analysis of Full-Wave Rectifier Filter was carried out for 1000 iterations. Following figure (4.19) shows the input waveform. Figures (4.20) & (4.21) shows the output response using ABM and Euler's PC methods respectively. Again the single point numerical method fails to solve third order systems whereas in Milne PC Method also the computation was done at very small step-size and fails to give appropriate output.

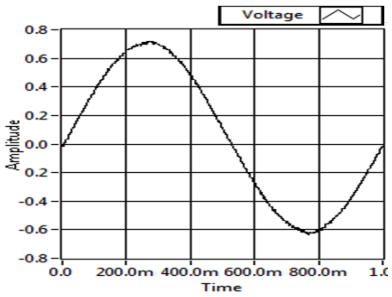

Figure 4.19: Low Frequency Sine-Wave

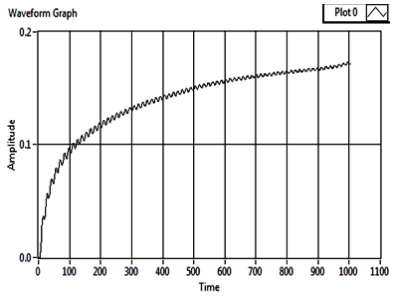

Figure 4.20: Full-wave Rectifier Response using ABM

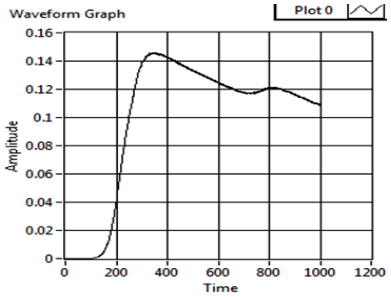

Figure 4.21: Full-wave Rectifier Response using EPC

Similarly frequency analysis of Voltage Doubler model was carried out for 500 iterations. Same input shown in figure (4.19) was given to USB DAQ. Figures (4.22) to (4.25) shows the output response obtained using various numerical methods. The single point numerical method fails to solve second order system as shown in figure (4.22).

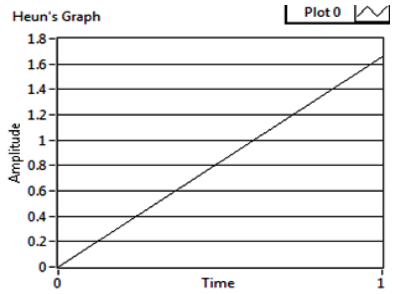

Figure 4.22: Voltage Doubler Response using Single Point Method

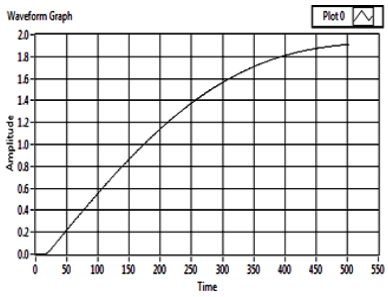

Figure 4.23: Voltage Doubler Response using ABM

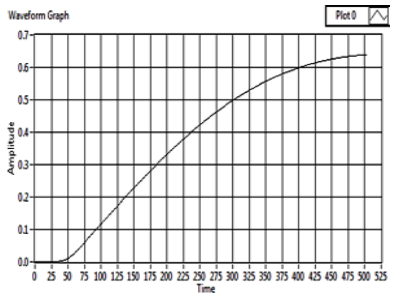

Figure 4.24: Voltage Doubler Response using Milne PC Method

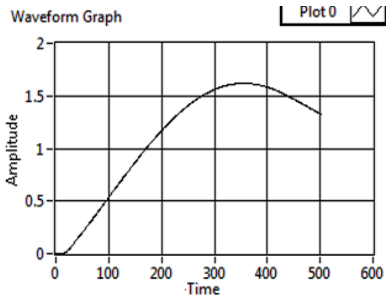

Figure 4.25: Voltage Doubler Response using EPC

4.4 Standalone System

The Standalone System was build on CRIO hardware setup using Application Builder Toolkit in LabVIEW. Verification of algorithm can be done by building a real-time standalone system where the code is dumped into CRIO along with the model of the system to be solved. After that the real-time input can be given to the standalone system through AI module to obtain the solution using inbuilt algorithm on RT target and the output can be observed through the physical output channel of AO module. The output of the AO module can be connected with the indicator or any fusible display device.

Verification of numerical methods can also done by building a real-time standalone system where the code for numerical method is dumped into CRIO along with the model of the system to be solved. After that the real-time input can be given through AI module to obtain the solution using

inbuilt algorithm of numerical method on RT target and the output can be observed through the physical output channel of AO module. The output of the AO module is connected with DSO to observe the output waveform of various systems.

The Real-Time Standalone System can be prepared by following few mandatory steps along with the appropriate hardware and software support. It is mandatory to create a RT target; steps for which are mentioned in the above section (4.1). It is must to have a LabVIEW software that supports CRIO specifications. Here NI CRIO-9073 is configured as standalone system which supports LabVIEW 2009 and higher versions. After preparing the algorithm in LabVIEW under the RT target of CRIO, following steps are needed to build the real-time standalone system:

a. Open the Project Explorer Window where the RT Target is created.

b. Right-click 'Build Specifications' under the RT target and select New→Real-Time Application to display the Real-Time Application Properties dialog box.

c. Give the name for standalone real-time system with .rtexe extension on Information page of the dialog box which was open after above step.

d. Explore the other options from the same dialog box.

e. Select the Startup VI from the number of VIs under RT target. It may contain other SubVIs (Dependencies); but select the main calling VI as Startup VI.

f. Configure destination settings from Destination page in the dialog box.

g. Browse the path in the Source File Settings page to change the destination of Builds. (Builds is the folder where .rtexe will be created and executed at later; once all steps are completed.)

h. Click Generate Preview button from Preview page if you want to see the preview of files that will be generated.

i. Click 'Ok' and save all the settings.

j. After saving all settings, right-click on 'Build Specification' and several options will be available. First Build the application, then from the same menu select deploy to dump the algorithm.

k. Select 'Set as Startup'.

l. Finally click on 'Run as Startup'. It will prompt to reboot. Let the system reboot process completed then after you can see the standalone real-time system started running.

m. If you want to update the VI, Rebuild whole application.

From the simulations and implementation of various numerical methods discussed in chapter 3 as well as in above sections of this chapter it is clear that Euler's PC method gives faster solution and also it is useful for solving higher order systems. Hence, the standalone system build using above steps uses Euler's Predictor-Corrector method to solve the above two systems; Full-wave Rectifier Filter and Voltage Doubler. As described above, the real-time input from transformer is given to AI module and output is observed on the DSO through AO module. Full-wave Rectifier Filter circuit model is dumped with EPC code. Here channel-3 of NI 9219 is configured as input and that is followed by channel-2 of output module NI 9269 to show the sinusoidal input in blue colour (Figure (4.26)). In figure (4.26), yellow colour shows the output response of Full-wave Rectifier. From the figure (4.26) it can be verified that the numerical method is a bit slower and hence it fails to hold the previous value due to faster sampling and fails to calculate next value. Thus, the difference in system output response can be seen by comparing with previous figures of the same system.

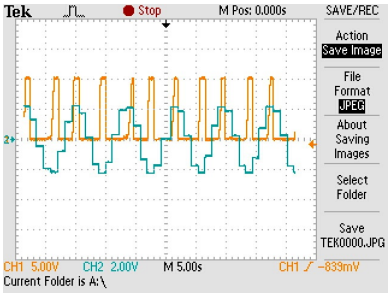

Figure 4.26: Full-wave Rectifier Response on Standalone System

Similarly the real-time standalone system is also prepared for Voltage Doubler and same code is used to solve this system. The figures (4.27) & (4.28) shows the output response of voltage doubler for different load values

10K and 100K respectively. Here also the method is bit slower to hold the previous value due to faster sampling. The output response is of yellow colour configured through AO module which can be compare with previous results of this same system. The blue waveform is of the input which is given through AI module. The results are taken on DSO.

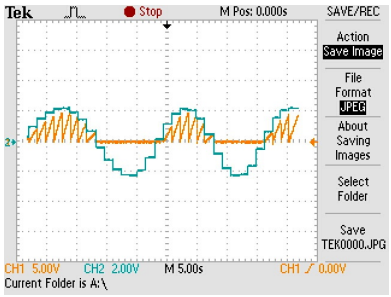

Figure 4.27: Voltage Doubler Response with 10K load on Standalone System

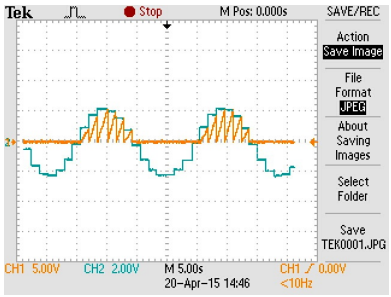

Figure 4.28: Voltage Doubler Response with 100K load on Standalone System

Chapter 5

Graphical User Interface (GUI)

Graphical User Interface (GUI) allows user to interact with the system code during run-time. The interaction in GUI can be made using drop down boxes or using arrows keys for selecting various parameters. The selection of the parameters can be done on the basis of the nature of system code. If the code has numerous iterations then change in value of the parameter may not be taken into consideration at that time instantly. The user has to wait till the completion of all iterations after that the next selected value/parameter will be applicable in the system. Moreover, there are different types of GUIs having least user interfacing controls to the most flexible GUIs. There are some GUIs in which the objects and blocks are available in which the user can have only "start-stop" button available through which one can only visualize the process and can't modify anything else whereas in certain GUIs there are some parameters available for user to control/change the value as per requirement. Also there are flexible GUIs available for user to interact with each and every control and can try all possible combination as per his/her desire to run the system. All this is depended upon the programmer or designer of the GUI.

The various numerical methods are designed in LabVIEW for solving ordinary differential equations which requires initial conditions, step-size, iterations etc. Also this differential equation obtained by concept of mathematical modeling like state-space analysis is require to enter in the form of an approximated formula. Here the GUI is prepared for single step methods and predictor-corrector methods with the flexibility given to the user for selecting one method at a time using arrow keys available as control. Also the initial conditions, step-size and number of iterations required are kept available for user to define accordingly as per the requirement.

5.1 Designing a GUI in LabVIEW

LabVIEW itself is a graphical programming language having wide facility of developing a GUI. LabVIEW software has front panel well-known for using it as GUI application. Here the LabVIEW codes for numerical methods designed previously are combined into one single VI and "Case-Structure" is used for the design of GUI. To select these numerical methods string control with up-down arrows called "Enumerate String Control" is used. This will select the name of the numerical method which user wants to use as solver. Also the formula node which was previously placed in the block diagram was replaced by "formula node vi". This has the flexibility to enter the formula through front panel of LabVIEW. Hence it can be user defined. Moreover, all other controls are designed placing the controls in front panel and hence the code is prepared as the most flexible GUI as all variables can be set to desire value via user interaction at any instant of time.

Three different GUIs are prepared for solving third order system, second order and first order system respectively. It is plan to keep a single GUI for selection of order of the system and accordingly the code can be available for user to solve the system model. Figures (4.1), (4.2) & (4.3) shows the screen-shot of the various GUI prepared for selecting various numerical methods.

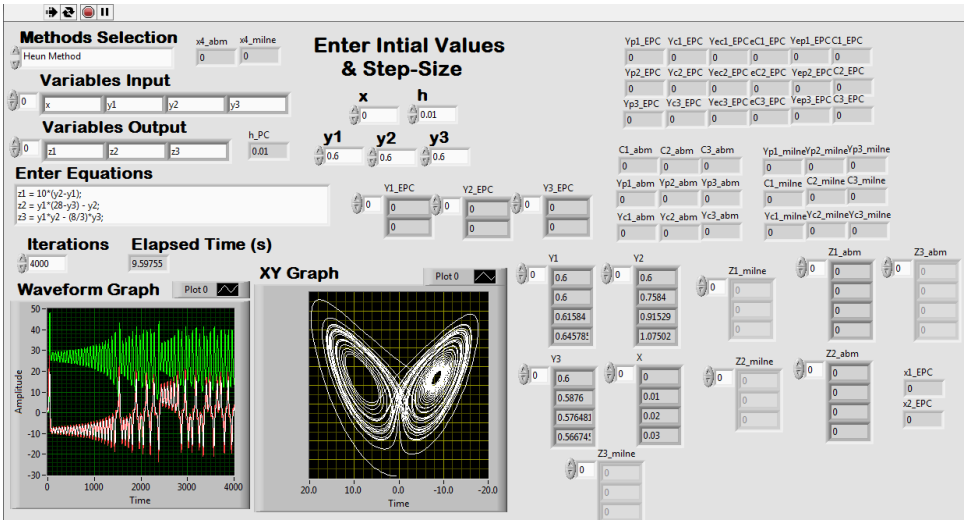

Figure 5.1: GUI for Third Order System

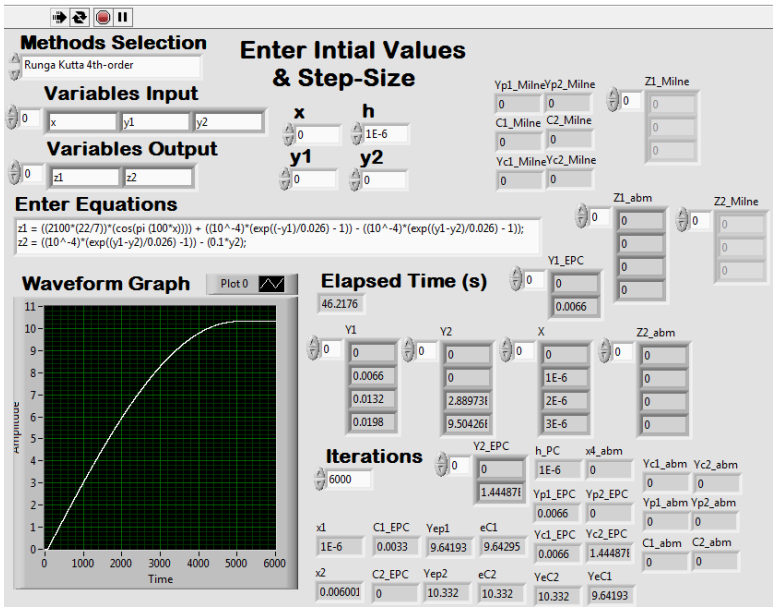

Figure 5.2: GUI for Second Order System

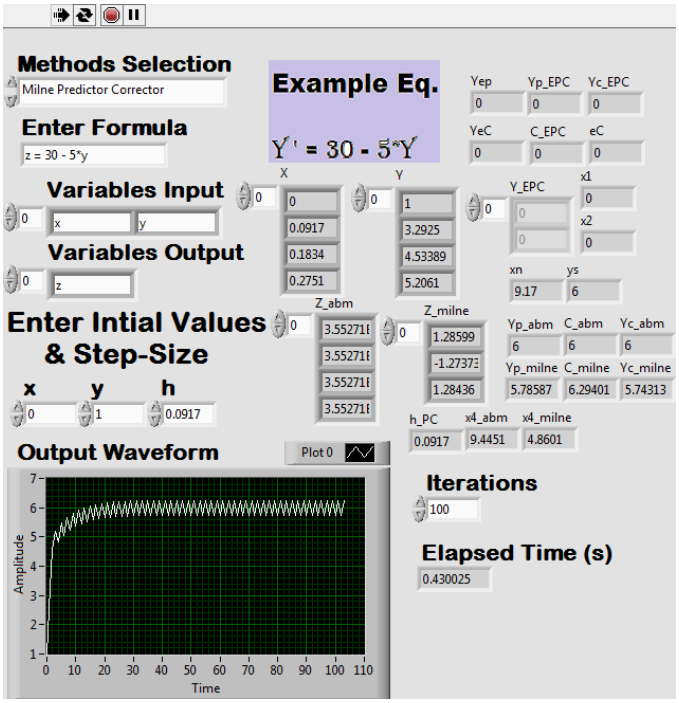

Figure 5.3: GUI for First Order System

Chapter 6

Conclusion

The Simulations of various Numerical Methods for different applications are carried out in MATLAB as well as LabVIEW to verify the results of each method. The solutions are obtained for the different applications which are compared with the real-time implementation on CRIO. Further the results of C-RIO and real system results are compared to justify that Euler Predictor-Corrector method is faster and its results are best fitted with results of real system whereas Milne's Predictor-Corrector method does not guarantee the stable results. Also Heun's Method gives more appropriate solution for 1^{st} order system. Moreover, GUI for various Numerical Methods is prepared in LabVIEW to simulate various systems according to the order of the system. Hence, the best method can be chose and the standalone system (supported by CRIO) can be prepared using Application Builder toolkit of LabVIEW.

By performing all type of simulations and preparing real-time applications it is also concluded that numerical methods are little bit slow for solving faster systems. Hence if the model is prepared and simulated, then it may give proper solution but the same cannot be possible if real-time solution is required using numerical methods. Moreover, mathematical model of such system is dumped into the C-RIO and it is also verified that due to faster sampling rate and slower response of numerical methods, the actual output value is dropped or it is unable to hold the previous value for calculating the next value. But the real-time implementation of various sluggish system can be advantageous as the controlling action through hardware in loop can be made possible when it is implemented on real-time embedded target.

Appendix A

Comparative Analysis

A.1 Results of 1^{st} Order Differential Equation

The Simulated Results of equation (3.1) using different numerical methods is compared against real-time implementation in C-RIO is shown in the following Table (A.1).

Methods	**Approx. Simulation Time* in LabVIEW**	**Real-Time Implementation (using C-RIO)**
Euler	4ms	50ms(approx.)
Heun	2ms	10ms
ABM	5s	20ms
Milne	Unstable	Unstable
Euler Predictor-Corrector	4ms	20ms

Table A.1: Comparison of Simulated Response Time against C-RIO Implementation

For Milne's Method the real-time solution is unstable due to 3 point method. Dahlquist Theorem proposed that the modulus of characteristic roots of the equation must be equal to 1 or the solution of the equation must be z-1 = 0. If the modulus of characteristic roots of the equation are greater than one then the equation does not obey the concept of zero-stability. Hence, it may happen that this method does not give proper solution in some cases.

A.2 Results of Full-wave Rectifier With Three Element Filter Circuit

The simulation results in LabVIEW, its Real-Time Implementation in C-RIO and real hardware response time is discussed in following Table (A.2).

Predictor-Corrector Methods	ABM	Milne	Euler
Approx. Simulation Time* in LabVIEW	130ms	150ms	120ms
Real-Time Implementation (using C-RIO)	625ms	800ms	300ms
Real Hardware Response Time	250ms		

Table A.2: Comparison of Simulated Response Time against C-RIO & Hardware Implementation

A.3 Results of Unsymmetrical Voltage Doubler Circuit

One more real system is simulated and tested on real hardware and its real-time implementation is also done using C-RIO. Table (A.3) below shows comparison values.

Predictor-Corrector Methods	ABM	Milne	Euler
Approx. Simulation Time* in LabVIEW	110ms	110ms	107ms
Real-Time Implementation (using C-RIO)	1.875s	2s	900ms
Real Hardware Response Time	800ms		

Table A.3: Comparison of Simulated Response Time against C-RIO & Hardware Implementation

***Note:** The numeric values showing simulation time in all tables is taken after performing few simulations in LabVIEW.

Appendix B

Specifications

B.1 CRIO Chassis

General	
Number of Slots	8
NI IO Modules	C Series
Operating System/Target	Real-Time
LabVIEW RT Support	Yes
Integrated Controller	Yes
Power Requirements	
Input Voltage Range	19V - 30V
Power Consumption	20W
Reconfigurable FPGA	
FPGA	Spartan-3
Physical Specifications	
Operating Temperature	-20°C - 55°C
Industrial RT Processor	266 MHz Integrated with 2M Gate FPGA
Dynamic RAM	64 MB For embedded operation
Nonvolatile Memory	128 MB For data logging
Ethernet port	10/100 Mb/s To conduct programmatic communication over the network
Built-in Servers	Web (HTTP) and file (FTP)

Table B.1: NI C-RIO Specifications

B.2 I/O Modules

General	
Product Family	Industrial I/O
Measurement Type	Voltage, Current, Resistance, Temperature, RTD, Strain/Bridged Sensor, Thermocouple
Form Factor	Compact-DAQ Compact-RIO
Isolation Type	Ch-Ch Isolation
Analog Input	
Number of Channels	4(0 to 3) Differential
Resolution	24 bits
Sample Rate	100S/s
Maximum Voltage Range	
Range	-60V - 60V
Accuracy	243 mV
Minimum Voltage Range	
Range	-0.125V - 0.125V
Accuracy	271μV
Maximum Current Range	
Range	-0.025A - 0.025A
Accuracy	152μA
Simultaneous Sampling	Yes
Bridge Configurations	Half, Full & Quarter Bridge
Physical Specifications	
I/O Connector	6-positions
Operating Temperature	-40°C - 70°C

Table B.2: NI 9219 Specifications

General	
Product Family	Industrial I/O
Measurement Type	Voltage
Form Factor	Compact-DAQ, Compact-RIO
Isolation Type	Ch-Ch Isolation
Analog Output	
Number of Channels	4(0 to 3)
Resolution	16 bits
Maximum Voltage Range	
Range	-10V - 10V
Accuracy	6.3mV
Update Rate	100kS/s
Current Drive Single	10mA
Current Drive All	20mA
Physical Specifications	
I/O Connector	2-positions
Operating Temperature	-40°C - 70°C

Table B.3: NI 9269 Specifications

Analog Inputs	
Number of Channels	8(ai0 - ai7) Single-ended 4 Differential
Sample Rate	10kS/s
Analog Output	
Number of Channels	2(ao0 - ao1)
Update Rate	150S/s
Resolution	12-bit
Digital I/O	12
Counter	1; 32-bit
OS Compatibility	Windows, LINUX, Mac OS, Pocket PC

Table B.4: NI USB 6008 Specifications

References

[1] John H. Mathews, Kurtis D. Fink, "NUMERICAL METHODS USING MATLAB", Fourth Edition, PHI.

[2] Stanislaw Rosloniec, "Fundamental Numerical Methods for Electrical Engineering", Springer.

[3] T Veerarajan, T Ramachandran, "Numerical Methods With Programs in C", Tata McGraw-Hill.

[4] Steven C. Chapra, Raymond P. Canale, "Numerical Methods for Engineers", Sixth Edition, McGraw-Hill (Higher Education).

[5] Dr. B. S. Grewal, "Numerical Methods in Engineering & Science with Programs in C & C++", Khanna Publishers.

[6] Jovitha Jerome, "Virtual Instrumentation Using LabVIEW", PHI.

[7] Jeffrey Travis, Jim Kring, "LabVIEW For Everyone Graphical Programming Made Easy and Fun", Third Edition, Pearson Education.

[8] S. Sumathi, P. Surekha, "LabVIEW based Advanced Instrumentation Systems", Springer.

[9] Ramakant A. Gayakwad, "Op-Amps and Linear Integrated Circuits", Fourth Edition, PHI.

[10] R. S. Sedha, "A TEXTBOOK OF APPLIED ELECTRONICS", Multicolour Edition, S. Chand.

[11] Robert Boylestad, Louis Nashelsky, "ELECTRONIC DEVICES AND CIRCUIT THEORY" Ninth Edition, PHI.

[12] Katsuhiko Ogata, "Modern Control Engineering", Fourth Edition, PHI.

[13] I. J. Nagrath, M. Gopal, "Control System Engineering", Fifth Edition, New Age International Publishers.

[14] Maciej Rosol, Adam Pilat, Andrzej Turnau, "Real-time controller design based on NI Compact-RIO", Proceedings of the International Multiconference on Computer Science and Information Technology pp. 825–830.

[15] Ned J. Corron, "A Simple Circuit Implementation of a Chaotic Lorenz System" [Online].

[16] National Instruments, "About LabVIEW & NI CRIO", [Online] Available: http://www.ni.com/getting-started/install-software/compactrio.

[17] LabVIEW, "Learning & Solving errors", [Online] Available: http://forums.ni.com/*topic name/error*.

[18] National Instruments, "Video Tutorials" & "White Papers", [Online] Available: http://ni.com/*topic name*.

[19] Downloading "Add-ons/Toolkits" for LabVIEW [Online] Available: http://sine.ni.com/*add-on/toolkit*.

[20] Chaotic System, [Online] http://en.wikipedia.org/wiki/Lorenz_system.

[21] Chaotic System, [Online] http://www.glensstuff.com/lorenzattractor/lorenz.htm.

List of Publications

[1] Prof. Sandip Mehta, Tej Trivedi, "Verification of Various Numerical Methods Using Hardware Implementation", International Conference on Futuristic Trends in Computational Analysis and Knowledge Management, INBUSH ERA 2015.