

FACULTY OF ENGINEERING AND TECHNOLOGY

**A REPORT ABOUT THE APPLICATION OF NUMERICAL APPROXIMATION METHODS AND DIFFERENTIAL EQUATIONS SOLVERS IN REAL-WORLD PROBLEM SOLVIG USING MATLAB**

COURSE UNIT: COMPUTER PROGRAMMING

LECTURER: Mr. MASERUKA BENEDICTO

SUBMITTED BY: GROUP 16

E-MAIL: [matlabgroup16@gmail.com](mailto:matlabgroup16@gmail.com)

*Submitted in partial fulfillment of the requirements of* COMPUTER PROGRAMMING

*DATE OF SUBMISSION.............../............../..............*

*SUBMITTED TO:**.......................................................*

**DECLARATION**

We, the undersigned members of group 16, do hereby declare that this report is the result of our own work carried out in partial fulfillment of the requirements of this course.

**NAME SIGNATURE**



KABWERU ANDREW .............................



CHEMONGES MIKIRAR ........ .....................



NAGASHA RITTA ............ ..................



DIKITAL JOHN ............. .................

SANYU JOY ...............................

OULE SADOCK ............. ..................



WANGUSI DAVID .............. ..................

SEBATIKA COLLINE ................................



ATYANG MILDRED ................................

KITUTU LEONARD .................................

# APPROVAL

This report has been submitted and prepared by group 16 as part of the requirements for the completion of module 1to 4 under the guidance of our lecture Mr Maseruka Bendicto

Computer programming lecturer

Name: ......................................

Signature: ..................................

# ACKNOWLEDGEMENT

We extend our sincere gratitude to our lecturer and the department of water resource engineering for the guidance, support and provision of resources that enabled us to complete this assignment successfully. We also thank our group members for their cooperation, dedication and collective efforts towards achieving this work

# DEDICATION

This report is dedicated to our families, friends and lecturers whose support, encouragement, and guidance have continually inspired us to work hard and pursue excellence in our studies.

We also dedicate it to our fellow students in group 16, whose team work, commitment, and collaboration made this report possible. Lastly, we dedicate this report to the pursuit of knowledge and academic growth, which remains the foundation of our future professional success.

# ABSTRACT

This report presents the application of numerical approximation methods for solving non linear equations and numerical techniques for differential equations using MATLAB. The newton-Raphson and secant methods were implemented to determine roots of non-linear functions, while Euler and Runge Kutta methods were employed in solving ordinary differential equations. The methods were tested on both theoretical and practical problems, and the results were compared with analytical solutions. The study highlights the accuracy, efficiency, and computational performance of the methods, demonstrating their importance in addressing real- world engineering and scientific problems

# DEDICATION

This work is dedicated to our families, colleagues, and mentors who continuously support and inspire us in our academic journey

# LIST OF ACRONYMS/ ABBREVIATIONS

1. MATLAB – Matrix Laboratory
2. abs - absolute

Contents

[DECLARATION 2](#_Toc210144211)

[APPROVAL 3](#_Toc210144212)

[ACKNOWLEDGEMENT 4](#_Toc210144213)

[DEDICATION 5](#_Toc210144214)

[ABSTRACT 6](#_Toc210144215)

[DEDICATION 7](#_Toc210144216)

[LIST OF ACRONYMS/ ABBREVIATIONS 8](#_Toc210144217)

[CHAPTER ONE: INTRODUCTION 10](#_Toc210144218)

[1.1 Background. 10](#_Toc210144219)

[**1.2 Historical Development** 10](#_Toc210144220)

[CHAPTER TWO: STUDY METHODOLOGY 11](#_Toc210144221)

[2.1 Introduction 11](#_Toc210144222)

[2.2 Steps followed 11](#_Toc210144223)

[CHAPTER THREE: QUESTION ONE 12](#_Toc210144224)

[3.1 introduction 12](#_Toc210144225)

[3.2 steps 12](#_Toc210144226)

[3.3 RESULTS: 15](#_Toc210144227)

[CHAPTER FOUR: QUESTION TWO 18](#_Toc210144228)

[4.1 Introduction 18](#_Toc210144229)

[4.2 Steps 18](#_Toc210144230)

[CHAPTER FIVE: CONCLUSION AND RECOMMENDATION 24](#_Toc210144231)

[5.1 CONCLUSION 24](#_Toc210144232)

[5.2 RECOMMENDATIONS 24](#_Toc210144233)

[CHAPTER SIX: REFERENCES 25](#_Toc210144234)

[APPENDICES 26](#_Toc210144235)

# CHAPTER ONE: INTRODUCTION

## 1.1 Background.

MATLAB, which stands for matrix laboratory, is a high-performance programming language and environment designed primarily for technical computing. Its origins trace back to the late 1970s when Cleve Moler, a professor of computer science, developed it to provide his students with easy access to mathematical software libraries without requiring them to learn Fortran.

1.2 Historical Development

* + Initial Development: The first version of MATLAB was created in Fortran in the late 1970s as a simple interactive matrix calculator. This early iteration included basic matrix operations and was built on top of two significant mathematical libraries: LINPACK and EISPACK, which were developed for numerical linear algebra and eigenvalue problems, respectively.
  + Commercial Launch: MATLAB was officially launched as a commercial product in 1984 by MathWorks, a company founded by Moler along with Jack Little and Steve Bangert. This marked the transition from a simple calculator to a comprehensive programming environment. The software was reimplemented in C, enhancing its capabilities with the addition of user-defined functions, toolboxes, and graphical interfaces.
  + Expansion and Toolboxes: Over the years, MATLAB has expanded significantly. By the late 1980s, it had introduced several specialized toolboxes for various applications, including control systems and signal processing. The introduction of the Simulink environment further allowed users to model and simulate dynamic systems graphically.
  + Modern Enhancements: Recent versions of MATLAB have introduced features like the Live Editor, which allows users to create interactive documents that combine code, output, and formatted text. This evolution reflects MATLAB's ongoing adaptation to meet the needs of its diverse user base across academia and industry.

# CHAPTER TWO: STUDY METHODOLOGY

## 2.1 Introduction

the study coverage adopted for this repot involved a systematic approach to numerical analysis methods such as Secant and newton Raphson method and analytical methods for solving ODEs such as Runge Kutta and Euler data for question 1 & 2 following the steps below;

## 2.2 Steps followed

1. **Problem identification**

The assignment required the application of numerical methods for root finding and solving ordinary differential equations (ODEs)

Two root finding techniques (Newton Raphson and Secant) and two ODE solving techniques (Euler and Runge – Kutta) were selected

1. **Mathematical formulation**

Definition of non-linear equation and ode to be solved

Derivation of analytical solutions for validation

1. **Algorithm development**

Flowcharts were drawn for each method to illustrate the algorithm steps:

* **Newton Raphson flowchart:** starts with initial guess, applies iteration formula until convergence
* **Secant flowchart:** uses two initial guesses and iterates until the root is approximated.
* **Euler flowchart:** updates solutions step by step using slope at the beginning of interval
* **Runge Kutta flowchart:** uses weighted slopes (k1, k2, k3, k4) to improve accuracy

1. **Implementation in MATLAB**

Codes were written foe each method to compute approximate solutions.

Loops and stopping criteria (tolerance levels) were incorporated to improve accuracy

Graphical plots were generated to compare numerical and analytical results

1. **Validation and comparison**

Results from numerical methods were compared with exact analytical solutions.

Errors and convergence rates were analysed

Computational performance (iterations and time taken) was recorded.

1. **Application to real world problems**

The methods were applied to practical scenarios such as cooling laws and pressure balance problems to demonstrate real world problems

# CHAPTER THREE: QUESTION ONE

## 3.1 introduction

Question one required us to utilise the knowledge of algorithm and control structures and modules 1 to 4 to do numerical approximation for finding the solutions to functions by newtons Raphson and secant methods

## 3.2 steps

**Step 1: choosing type of functions**

We the group 16 members decided to choose implementations on non linear functions

**Step 2: drawing flow charts**

READ: i, x0, X1



i = 0

Numerical method formular i.e.

Secant or newton Raphson

Is |Xi+1 -Xi| < error?

PRINT: Xi+1

i = i +1

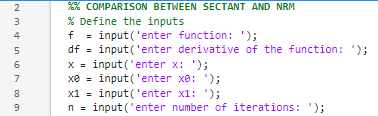
Xi = Xi+1



**Actual writing of codes**

**Step 3: reading the initial approximations and other inputs**

Using the input function, we allowed the input of initial approximation variables i.e. x0, x1, number of iterations N, function f(x) and it its derivative df(x)



**Step 4: finding the exact or true root**

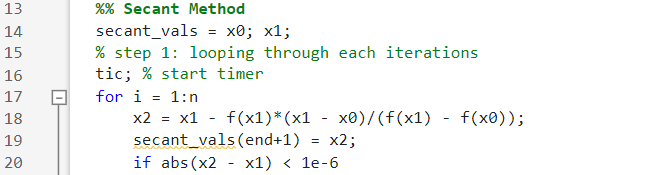
Using the code below, found the exact value of the root of the function f(x)



**For secant method**

**Step 5: looping through each iteration**

using the for function, we looped through each iterate and also using the if function, we commanded it to stop when |x2-x1| is less than error and we also started a stop clock at each time using the tic function



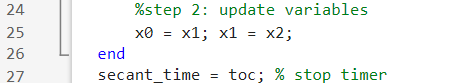
**Step 6: print the root xi**

Using the fprintf function we commanded Matlab to print the root xiwhen |x2-x1| is less than error and stop

****

**Step 7: update the valuables**

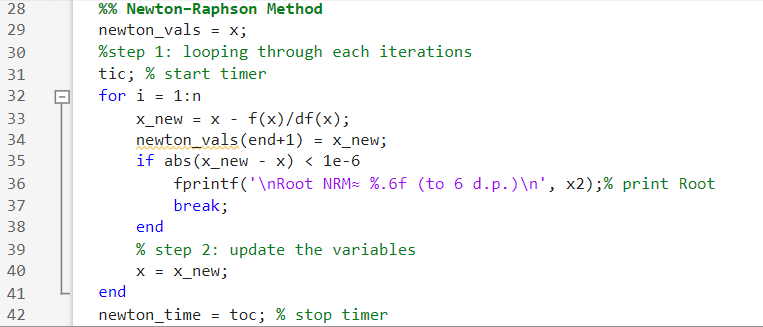
For |x2-x1| is grater than error, we commanded it to update variables such that x0 = x1 and x1=x2 and repeat the cycle. We also stopped our stop clock using toc function

****

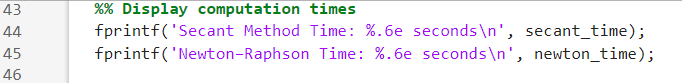
**For newtons Raphson method**

**Step 8:**

Following the same procedures as for secant, we looped through the iterates, commanded it to print the root when |x2-x1| is less than error and update values if its greater than error. And also measured the computational time using the tic and toc functions.



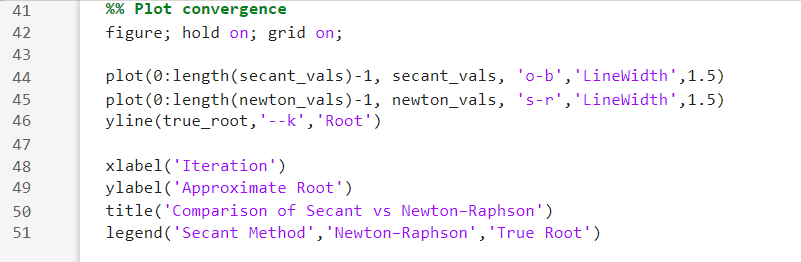
**Step 9: print the time of computation**

****

**Plotting graph**

**Step 10:**

Using inbuilt functions such as plot, yline, xlabel, ylabel, title, and legend we managed to plot a well labelled graph to compare the solutions of both secant and newton Raphson method



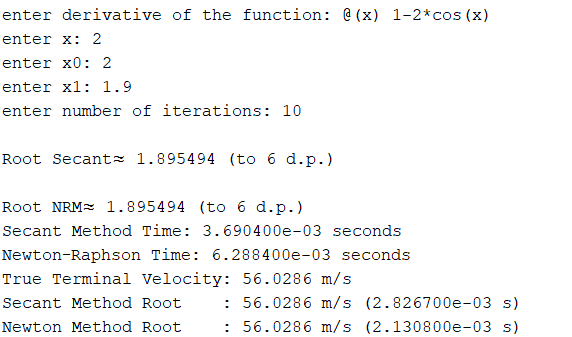
**Step 11: we then implemented the codes on two functions i.e.**



## 3.3 RESULTS:

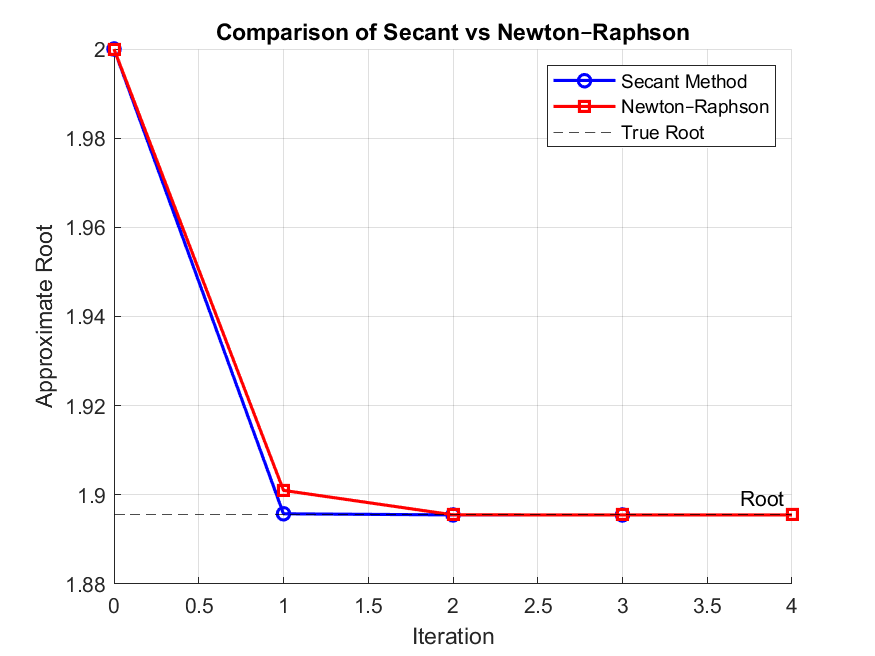
Implementing the code to function;

Using x = 2, x0=2, x1=1.9, and 10 iterations we obtained the following results;

****

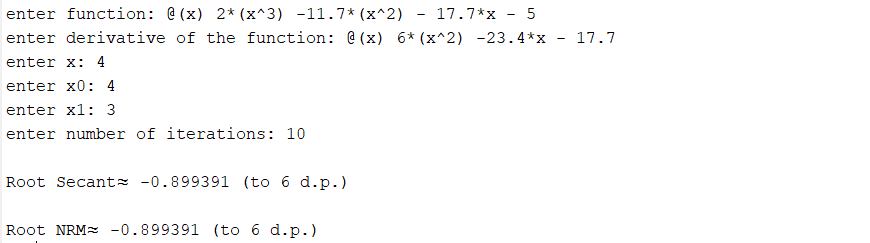
**Plot:**

From the plot, newton Raphson method reached the root faster meaning that its more efficient, faster and user friendly though its tedious since it requires finding the derivative of the function.



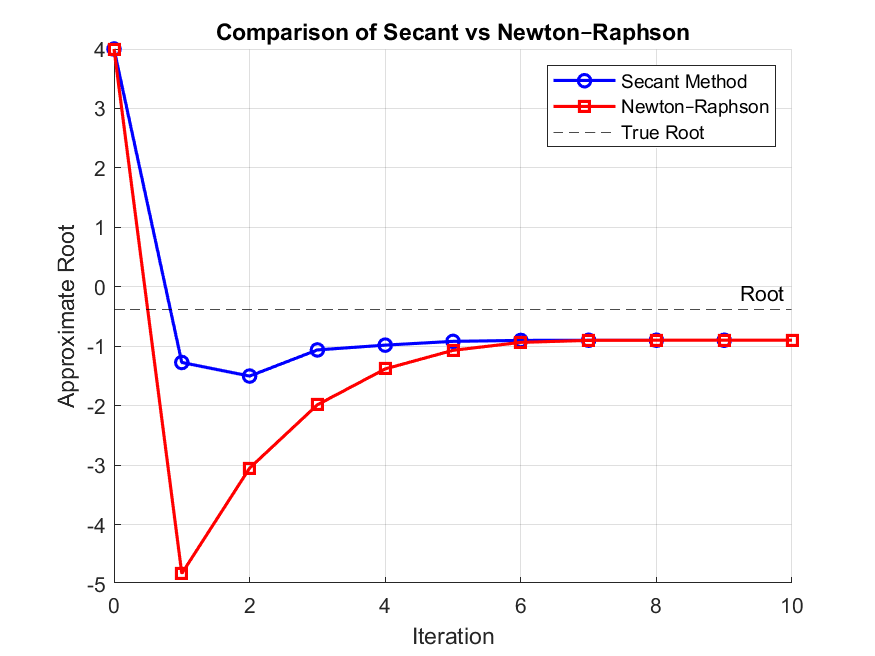


For the above expression, we used x0 = 4, x1 = 3 and 10 iterations, we obtained the following results.



**Plot:**

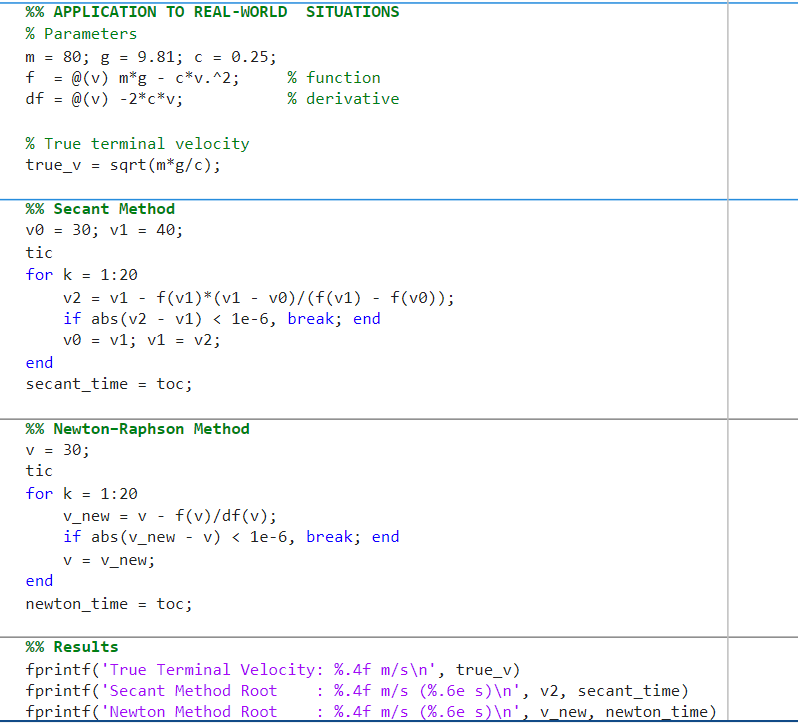
From the plot, newton Raphson method reached the root faster meaning that its more efficient and reliable than secant



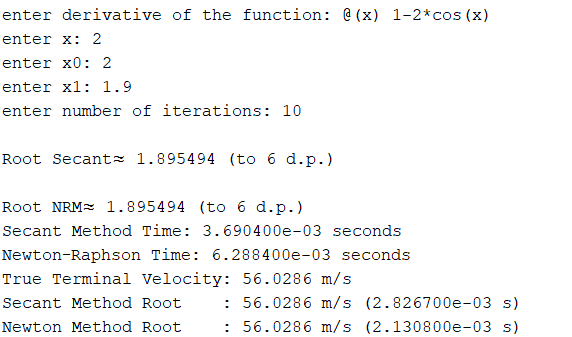
From the above observations, we as group 16 members recommend use of newtons Raphson other than secant because it is more efficient, runs faster and give a very accurate answer.

**Step 10: application to real world problems**

we then applied it to real world problem of finding terminal velocity (v) of particles with mass =80, g=9.81, c=0.25

****

After running the code, we then obtained the terminal velocity



# CHAPTER FOUR: QUESTION TWO

## 4.1 Introduction

Question two required us to utilise the knowledge of algorithm development, control structures, models one to four to solve the differential equations numerically while using methods such as Euler, Runge Kutta

## 4.2 Steps

**Step 1: choosing type of functions**

We the group 16 members decided to choose implementations on first order differential equations relating variable y with time t

**Step 2: flowcharts:**

**Euler: Runge Kutta:**



dy/dx=f(x,y), h, N, y(t)

Yn+1 = yn +h\*f(xn, yn)

Is xn+1 >= xend

Print yn

yn =yn+1, xn=xn+1

dy/dx=f(x,y), h, N, y(t)

Yn+1 =yn +6h(k1 +2k2+2k3+k4

Is xn+1 >= xend

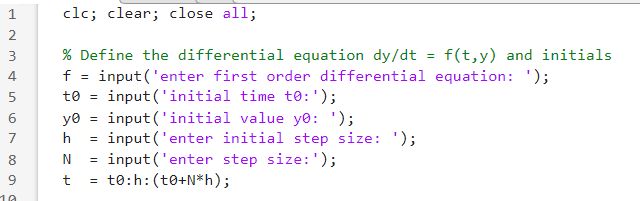
Print yn

Runge Kutta 4 formula;

**Actual writing of codes**

**Step 3: reading the initial approximations and other inputs**

Using the inbuilt function input, we allowed entry of the inputs such as step size, differential equation, initial value, initial time etc.

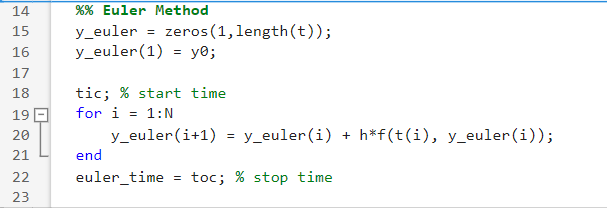


**Step 4: computing the exact value**

Using the code below we computed the exact value **but note** that the exact value of each function is found differently

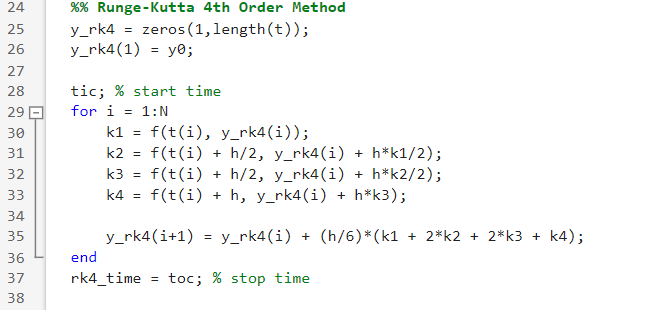
**Step 5: for Euler**

Using inbuilt functions such as tic, toc, for etc, we looped through each cycle, started a stop clock and then stopped it when the cycles are done



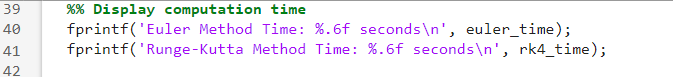
**Step 6: for Runge Kutta**

Also using inbuilt functions such as tic, toc, for etc, we looped through each cycle, started a stop clock and then stopped it when the cycles are done



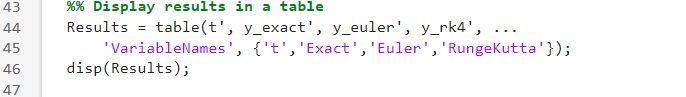
**Step 7: displaying computation time**

Using the inbuilt function fprintf, we displayed the computation time for both Euler and Runge Kutta

****

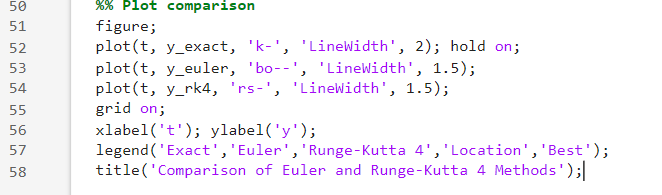
**Step 8: displaying results in a table**

Using functions such as table and disp, we displayed the results of both Euler and Runge Kutta



**Step 9: plotting**

We the created a figure and using plot function we managed to plot line graph with results of both Euler and Runge Kutta on the same axes



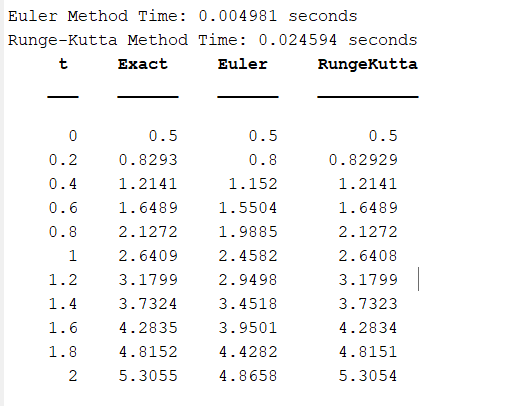
**5.3 RESULTS**

We implemented the code on two differential equations i.e.

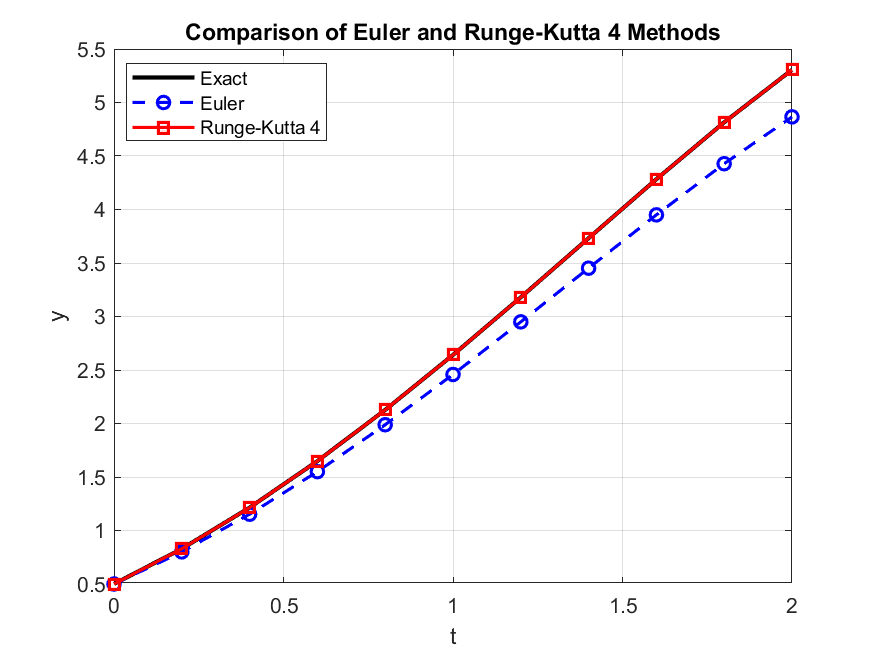
Using exact value as;

****

We obtained the following results;

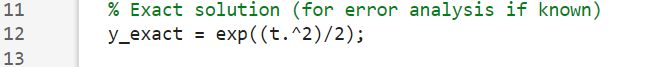


**Figure**

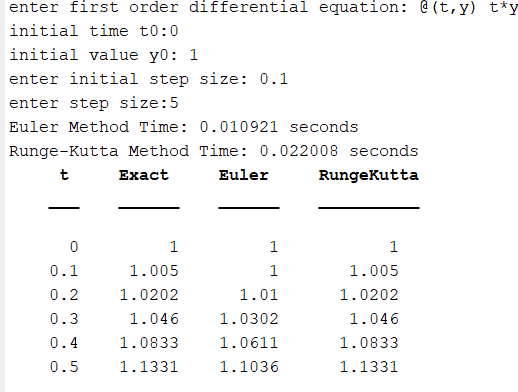
****

From the above observations, Runge Kutta provide more accurate answers compared to simpler methods like Euler, though it has a higher computation time than Euler. So we recommend Runge Kutta when accuracy is key and Euler for minimum computational time

Using the exact value as;



We obtained the following results



**Figure**

****

From the above observations, Runge Kutta provide more accurate answers compared to simpler methods like Euler, though it has a higher computation time than Euler. So we recommend Runge Kutta when accuracy is key and Euler for minimum computational time

**Step 10: application on real world problem**

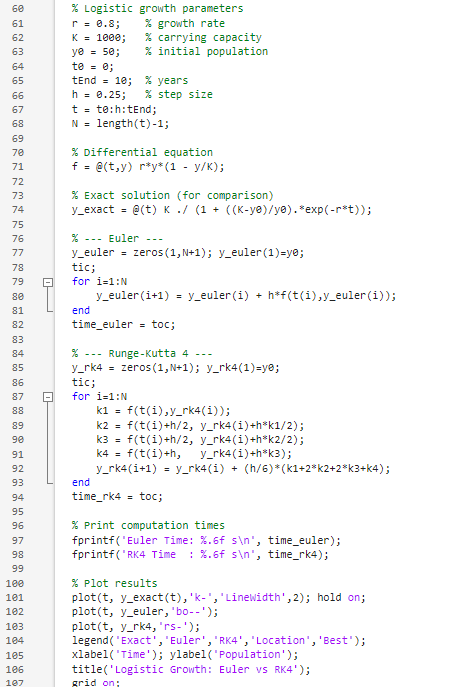
We used the code in solving the logistic population growth model, which describes how a population of bacteria grows over time when there are limited resources (space). using the equation below

y(0) = y0

Where y(t) – population at time t, r – growth rate, k – carrying capacity, y0 – initial population

Using the same functions and procedure as in above, we used the code to calculate the population of bacteria from 0 to 10 years with initial population of 50, 0.8 growth rate and carrying capacity of 1000 using step size of 0.25

**Code**

****

The produced the results computational times and figure below after running

**Computational time**

****

**Figure**



# CHAPTER FIVE: CONCLUSION AND RECOMMENDATION

## 5.1 CONCLUSION

In this study, numerical approximation methods were successfully implemented for finding roots and differential equation solving. The newton Raphson and secant methods provided accurate root estimation, while Euler and Runge Kutta methods gave approximate solutions to first order differential equations. The results show that higher order methods such as Runge Kutta provide more accurate answers compared to simpler methods like Euler, the overall assignment demonstrates the effectiveness of numerical techniques in solving in solving real world problems where analysis is difficult o impossible to obtain

## 5.2 RECOMMENDATIONS

* Higher order methods such as Runge Kutta should be prioritized when accuracy is required, while simpler methods such as Euler can be used for quick approximations.
* Computational time should always be considered in practical applications where efficiency is crucial
* Students and researchers should continue applying numerical methods to real world engineering, scientific, and economic problems for better decision making
* Future work may include extending these methods to systems of non linear equations and higher order differential equations

# CHAPTER SIX: REFERENCES

* course Lecture notes (module 1 – 4) and part of module 5
* Matlab documentation

# APPENDICES

