

FACULTY OF ENGINEERING AND TECHNOLOGY

**A REPORT ABOUT THE APPLICATION OF NUMERICAL APPROXIMATION METHODS AND DIFFERENTIAL EQUATIONS USING MATLAB BASED ON RECURSIVE PROGRAMMING AND** **COMPARATIVE ANALYSIS OF RECURSIVE AND DYNAMIC PROGRAMMING APPROACHES FOR THE KNAPSACK AND FIBONACCI PROBLEMS**

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 REG NO SIGNATURE**



KABWERU ANDREW BU/UG/2024/5057 .............................



CHEMONGES MIKIRA BU/UP/2024/1005 ........ .....................



NAGASHA RITTA BU/UG/2024/5055 ............ ..................



DIKITAL JOHN BU/UP/2024/1125 ............. .................

SANYU JOY BU/UP/2024/5348 ...............................



OULE SADOCK BU/UP/2024/0844 ............. ..................



WANGUSI DAVID BU/UP/2024/5350 .............. ..................

SEBATIKA COLLINE BU/UP/2024/0845 ................................



ATYANG MILDRED BU/UP/2024/1019 ................................

KITUTU LEONARD BU/UP/2024/1053 .................................

# 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

Of course. Here are the standard academic sections for your report, tailored to the topic of recursive and dynamic programming for the Knapsack and Fibonacci problems.

We would like to express our profound gratitude and sincere appreciation to all the group 16 members who supported and guided us throughout the completion of this report.

We are also immensely grateful to the academic community and the developers of the Matlab programming language and its powerful libraries which were instrumental in conducting the experiments and generating the comparative graphs for this study

Finally, we would wish to thank our families and friends for their unwavering support, constant encouragement, and patience during this endeavour. This accomplishment would not have been possible without them.

# DEDICATION

This work is dedicated to our parents, for their endless love and sacrifices, and to all curious minds who find elegance in turning complex problems into simple, recursive steps.

# ABSTRACT

This report investigates the implementation and performance analysis of recursive programming versus dynamic programming for solving classic computational problems. The study begins by converting an iterative algorithm from a Numerical Methods assignment into an equivalent recursive function, demonstrating the paradigm shift in problem-solving approach.

The core of the report focuses on a comparative analysis of two fundamental problems: the Fibonacci sequence generation and the 0/1 Knapsack problem. For each problem, a naive recursive solution and an optimized dynamic programming solution are implemented. The primary objective is to empirically compare the computational efficiency of these approaches.

# LIST OF ACROYNMS/ ABBREVIATIONS

1. MATLAB – Matrix Laboratory
2. abs - absolute

Table of Contents

[**DECLARATION** ii](#_Toc211951528)

[**APPROVAL** iii](#_Toc211951529)

[**ACKNOWLEDGEMENT** iv](#_Toc211951530)

[**DEDICATION** v](#_Toc211951531)

[**ABSTRACT** vi](#_Toc211951532)

[**LIST OF ACROYNMS/ ABBREVIATIONS** vii](#_Toc211951533)

[**CHAPTER ONE: INTRODUCTION** 1](#_Toc211951534)

[1.1 Background. 1](#_Toc211951535)

[1.2 Historical Development 1](#_Toc211951536)

[**CHAPTER TWO: QUESTION ONE** 2](#_Toc211951537)

[2.1 introduction 2](#_Toc211951538)

[**SECTION 1: NEWTON RAPHSON AND SECANT** 2](#_Toc211951539)

[2.1.1 steps 2](#_Toc211951540)

[2.1.2 outputs 4](#_Toc211951541)

[**SECTION 2: EULER AND RANGE KUTTA FOUR** 5](#_Toc211951542)

[2.2.1 Steps 5](#_Toc211951543)

[**CHAPTER THREE: QUESTION TWO** 8](#_Toc211951544)

[3.1 introduction 8](#_Toc211951545)

[3.2 steps taken 8](#_Toc211951546)

[Knapsack problem 8](#_Toc211951547)

[Fibonacci problem 9](#_Toc211951548)

[**CHAPTER FOUR: CONCLUSION AND RECOMMENDATION 11**](#_Toc211951549)

[**4.1 CONCLUSIONS 11**](#_Toc211951550)

[**4.2 RECOMMENDATIONS 11**](#_Toc211951551)

[**CHAPTER SIX: REFERENCES 12**](#_Toc211951552)

[**APPENDICES 13**](#_Toc211951553)

# 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: QUESTION ONE

## 2.1 introduction

Question one required us to do numerical approximation for finding the solutions to functions by newtons Raphson ad secant methods based on recursive programming

# SECTION 1: NEWTON RAPHSON AND SECANT

## 2.1.1 steps

**Step 1: 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



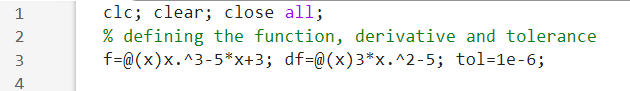
**Actual writing of codes**

**Step 2: suggestion on the equation to implement.**

As group 16, we decide to implement the secant and newtons Raphson’s method on the above equation together with initial approximations as shown below.

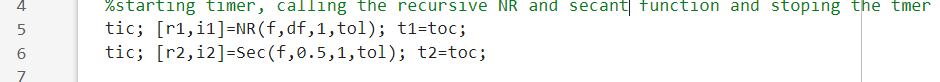
**Step 3: defining the function, derivative and tolerance**

we defined the function, derivative and tol as shown below

****

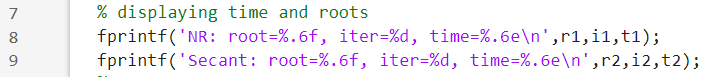
**Step 4: starting timer, calling the recursive NR function and stopping the timer**

We used inbuilt functions such as tic and toc to start and stop the timer. We then called the NR function with x0 =1 and secant with x0 =0.5, x1 = 1.



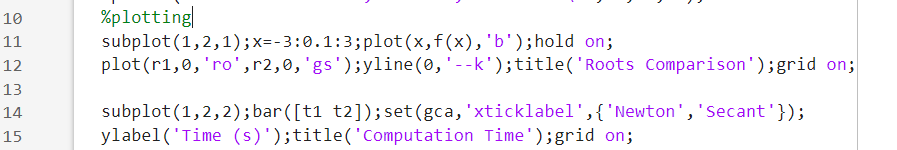
**Step 5: displaying the results**

Using the fprintf function, we displayed the time and roots to six decimal places

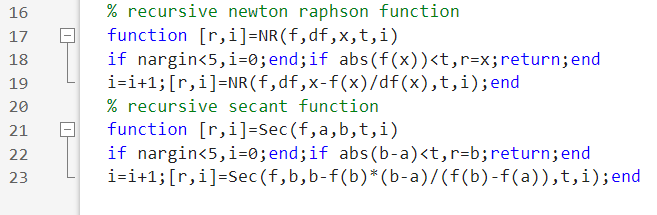


**Step 6: plotting function and roots**

Using the subplot function, we succeeded in plotting two plots on the same figure to compare the roots of the function by the two methods and their computation time



**Step 7: writing recursive formulars for both functions**

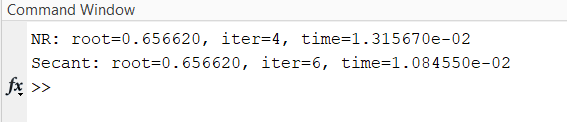
****

**Step 8: saving and running the code**

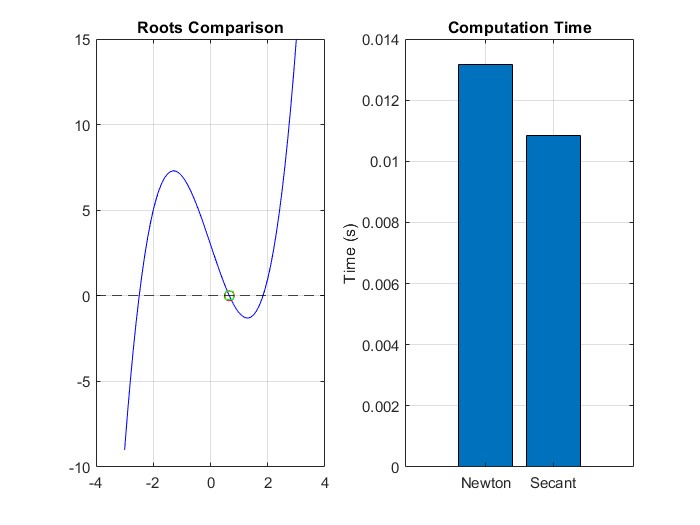
## 2.1.2 outputs

After saving and running the code above, we obtained the following results

**Roots and computation time**



**Visual comparison between newton Raphson and secant methods**

****

From the above observation, NR has a higher computational time than secant method

# SECTION 2: EULER AND RANGE KUTTA FOUR

## 2.2.1 Steps

**Step 1: drawing flow charts**

**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

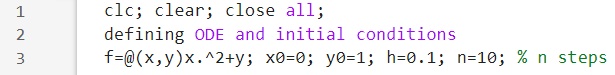
**Actual writing of codes**

**Step 2: choosing a problem to implement**

We as group 16, decide to implement our code to the differential equation and initial approximations below

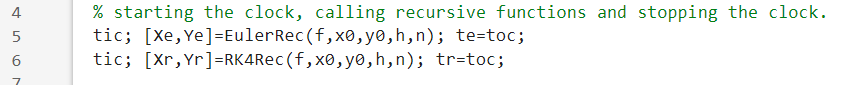
**Step 3: defining ODE and initial conditions**

We defined the ODE and its initial conditions as below



**Step 4: starting the clock, calling recursive functions and stopping the clock.**

We used inbuilt functions such as tic and toc to start and stop the timer.

****

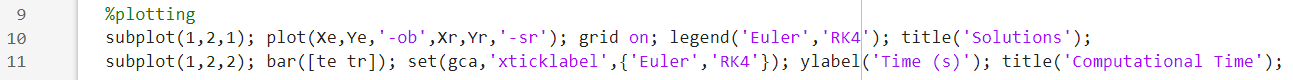
**Step 5: displaying computation time**

Using the fprintf function, we displayed the computational times of both methos to 6 d.p.s



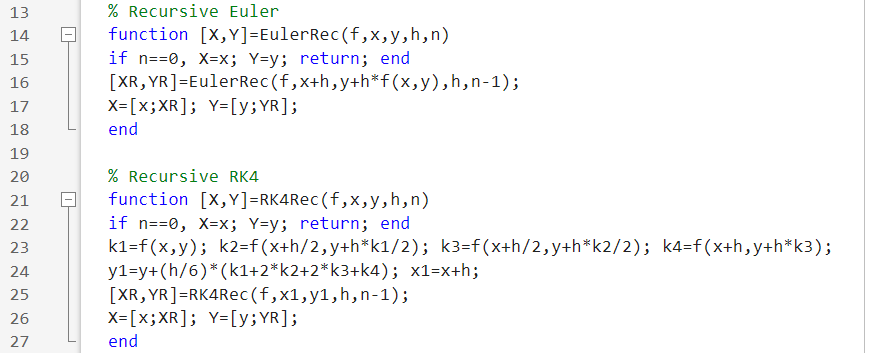
**Step 6: visualisation (plotting)**

We plotted two graphs on the same figure by the help of the subplot function to compare the roots and computation times of both methods



**Step 7: writing recursive formulars for both functions**

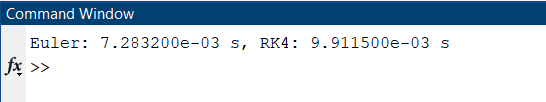
The recursive formulars were written as follows



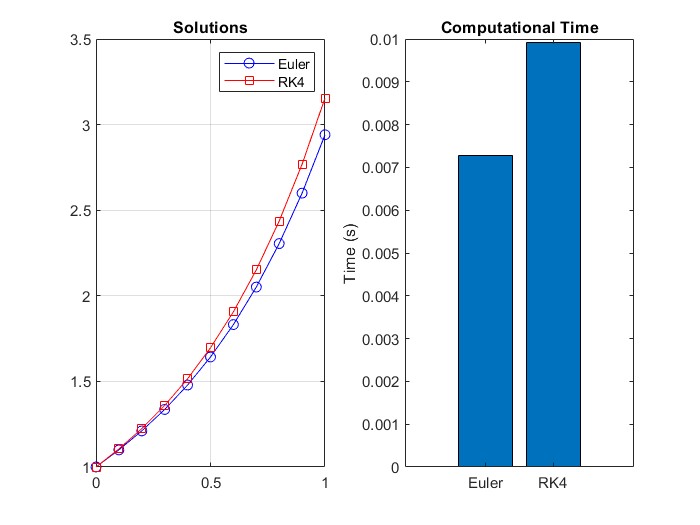
**Step 8: saving and running the code**

**2.2.2 outputs**

Running the codes produced the following outputs



**Visualisation**

****

From the above graphs we saw that Euler has a lower computational time than Runge Kutta which makes it more efficient though Runge Kutta is more accurate.

# CHAPTER THREE: QUESTION TWO

## 3.1 introduction

Question two required us to use the concept of recursive and dynamic programming to solve Knapsack and Fibonacci problems, making graphs to compare their computational times

## 3.2 steps taken

## Knapsack problem

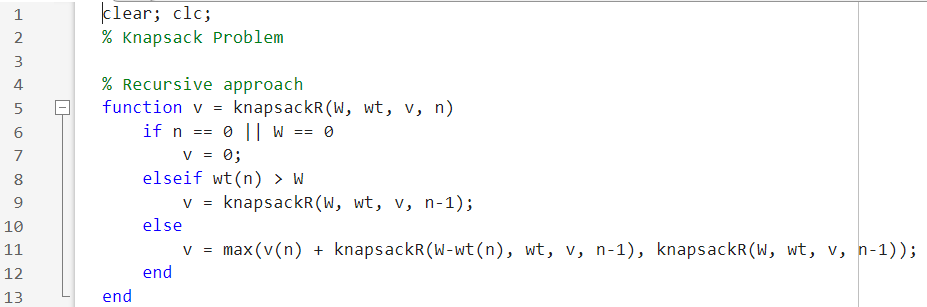
**Step 1; analysing the problem to be worked on**

The codes were to compute the maximum value with capacity W, item weights wt, item values v, considering first n terms using both dynamic and recursive programming techniques

**Step 2:** **writing functions for Knapsack in both recursive and dynamic programming**

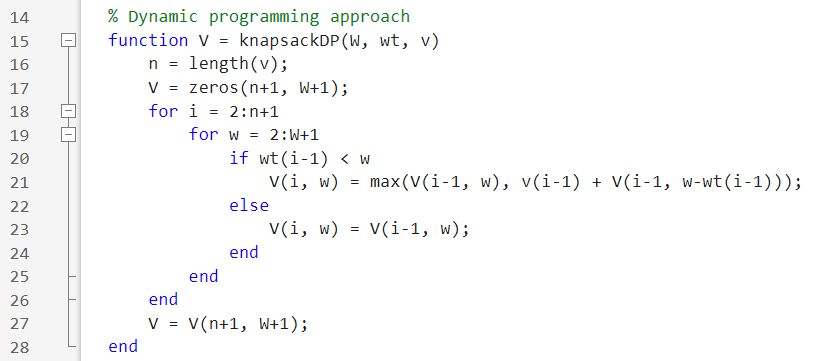
**(i). Recursive programming**

We wrote codes basing on the above parameters compute the maximum value with capacity W, item weights wt, item values v, considering first n terms as follows



**(ii). Dynamic programming**

For the above problem in dynamic programming, we wrote the code below to compute the maximum value for capacity. We used loops, if condition and return function to simulate the code as below

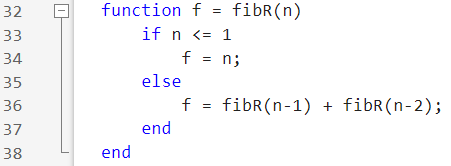


## Fibonacci problem

**Step 3: writing functions for Fibonacci in both recursive and dynamic programming**

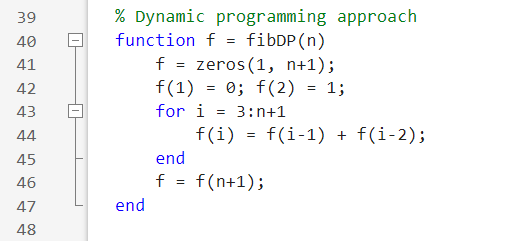
**(i). Recursive programming**

We wrote the Fibonacci function in recursive programming as follows with base cases as n=0 or n=1



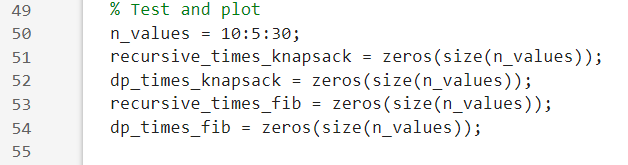
**(ii). Dynamic programming**

we wrote a function that builds a Fibonacci sequence in an array and returns F(n), runs in linear time 0(n) and uses 0(n) memory.



**Step 4: testing and timing setup**

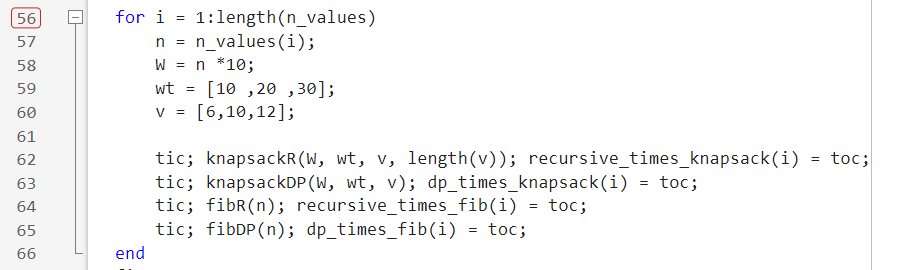
We allocated n values from 10-30, pre allocated arrays to store execution times for each method problem.



**Step 5: loops that run test and records times**

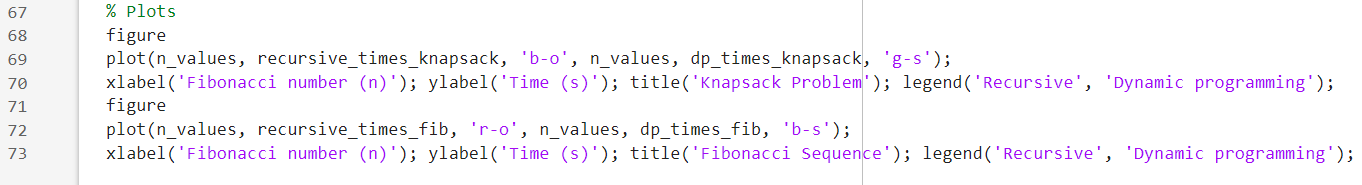
For each n,

W set to n\*10, wt and v fixed small item sets and also stored the elapsed times for both functions.



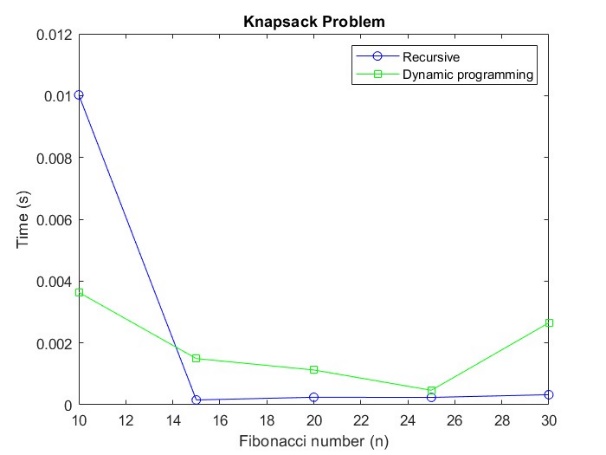
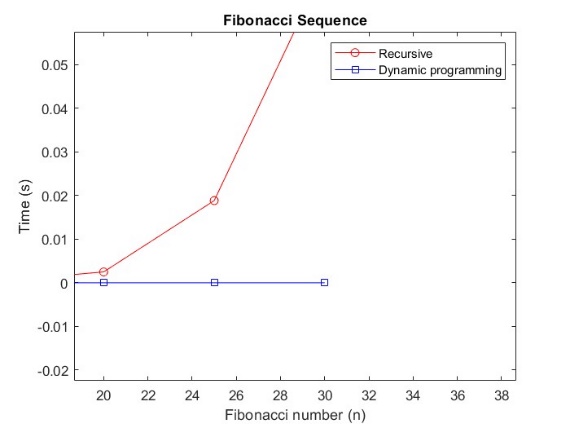
**Step 6: plotting both Knapsack and Fibonacci results**

We wrote plotting codes for both functions as follows;



**3.3 running the code.**

Running the produced the following results



# CHAPTER FOUR: CONCLUSION AND RECOMMENDATION

## 4.1 CONCLUSIONS

Dynamic programming (DP) is far superior to naive recursion for problems with overlapping subproblems, like Fibonacci and Knapsack. Graphs show DP's time grows slowly, while naive recursion becomes too slow for larger inputs. The trade-off of using extra memory for massive speed gains is justified

## 4.2 RECOMMENDATIONS

* Use DP over recursion for problems with overlapping subproblems.
* Apply memorization to recursive functions as a simple performance fix.
* Explore other DP problems like Longest Common Subsequence or Coin Change.
* Compare top-down and bottom-up DP methods in future work.

# CHAPTER SIX: REFERENCES

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

# APPENDICES

