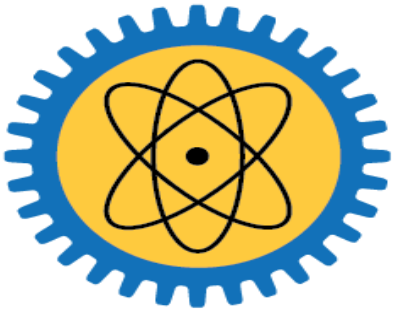


BUSITEMA UNIVERSITY

Pursuing excellence

FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER PROGRAMMING

REPORT OF ASSIGNMENT

BY

GROUP 14

COURSES: AMI, APE, MEB, WAR

ACKNOWLEDGEMENT

First and foremost, we would like to thank the Almighty God for giving us the knowledge and guidance while doing our assignment as group 14.

We extend our gratitude to all the people with whose help we managed to make it this far

The love of every group member invests time and provides all they can to see the assignment as a success.

Finally, we would like to express our gratitude to all the sources and references that have been cited in this report

ABSTRACT

We started our first meeting for the assignment, in the university library out of which we were exposed to various concepts on how to accomplish the task in MATLAB, we managed to achieve this through group work and division of tasks whereby everyone brought ideas.

DEDICATION.

We dedicate this report to all Group 14 members, who have been there with us in the process of researching and doing and compiling this report. To our lecturer Mr. Maseruka Benedicto, whose guidance and expertise have been so needful, your mentorship and lecturing have built our understanding.

DECLARATION

We declare that this information in this report is our own, to the best of our knowledge.

APPROVAL

We are presenting this report which has been written and produced under our efforts.

MR MASERUKA BENEDICTO

SIGNATURE…….

The following are the group members,

NAME	REG NO	COURSE	SIGNATURE
KATUMBA JEREMIAH	BU/UP/2024/5252	AMI	
TOSKIN CHRIS	BU/UP/2023/0843	WAR 3	
ARACH GLADYS	BU/UP/2024/1016	WAR	
MUTEBE JOEL	BU/UP/2024/1043	WAR	
TWALE JOSHUA	BU/UP/2024/5259	APE	
BASALIRWA ROBERT	BU/UP/2024/1004	WAR	
AWENE SOLOMON	BU/UP/2024/1021	WAR	
OMONA PATRICK	BU/UP/2024/0986	MEB	
ACHENG AUDREY LORNA	BU/UP/2024/0822	AMI	
ACIPA BRIDGET	BU/UP/2024/1078	WAR	

Contents

INTRODUCTION

HISTORICAL 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.

MATLAB is built around the concept of matrices, making it particularly effective for linear algebra and matrix manipulation. It provides a vast library of built-in functions for mathematical operations, statistics, optimization, and other specialized tasks.

MATLAB offers powerful tools for creating 2D and 3D plots, enabling users to visualize data effectively. Specialized toolboxes extend MATLAB's capabilities, providing functions tailored for specific applications like signal processing, image processing, control systems, and machine learning.

MATLAB can interface with other programming languages (like C, C++, and Python) and software tools, allowing for flexible integration into larger systems. Its interactive environment features a command window, workspace, and editor, making it accessible for both beginners and advanced users.

Historical Background

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.

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.

STUDY METHODOLOGY

Introduction

At the start, each member was given the task of doing research about the assignment before our first meeting. The research concepts were obtained through watching tutorials on YouTube and consultations from other continuing students especially those in year three and four.

QUESTION ONE

From the assignment of numerical methods, make equivalent code based on recursive programming.

SOLUTION

STEPS

Create a working folder

- ✓ Open MATLAB.
- ✓ On the top toolbar click Home, New Folder and name it
- ✓ In MATLAB's Current Folder browser, change to that folder
- ✓ Everything we create will be saved there.

Create the Live Script

- ✓ In MATLAB toolbar: Home, New Live Script.
- ✓ Save it as

Write the code in the live script
Add recursive function definitions
Scroll to the bottom of the. mlx and add another code containing the functions.

Recursive Root-Finding Methods
Root-finding algorithms determine the value of x where a nonlinear function f(x)=0.
Three recursive techniques were implemented:
1. Newton-Raphson Method
2.Secant Method
3.Bisection Method

CODE

```
%RECURSIVE ROOT FINDING
clear; clc;

f = @(x) x.^3 - x - 2;
df = @(x) 3*x.^2 - 1;
tol = 1e-6;
max_iter = 100;
```

NEWTON RAPHSON METHOD

```
% Newton Raphson Method
tic;
[x_nr, iter_nr] = newton_recursive(f, df, 1.5, tol, max_iter, 1);
```

```
time_nr = toc;
fprintf('Newton root: %.6f, iter: %d, time: %.6f s\n', x_nr, iter_nr, time_nr);
```

```
Newton root: 1.521380, iter: 3, time: 0.019795 s
```

SECANT METHOD

```
% Secant
tic;
[x_sec, iter_sec] = secant_recursive(f, 1, 2, tol, max_iter, 1);
time_sec = toc;
fprintf('Secant root: %.6f, iter: %d, time: %.6f s\n', x_sec, iter_sec,
time_sec);
```

```
Secant root: 1.521380, iter: 7, time: 0.019741 s
```

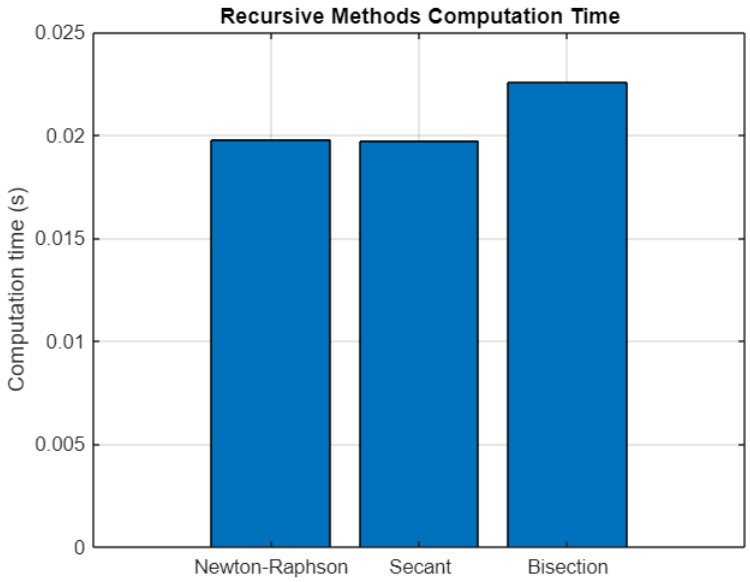
BISECTION METHOD

```
% Bisection
tic;
[x_bis, iter_bis] = bisection_recursive(f, 1, 2, tol, max_iter, 1);
time_bis = toc;
fprintf('Bisection root: %.6f, iter: %d, time: %.6f s\n', x_bis, iter_bis,
time_bis);
```

```
Bisection root: 1.521380, iter: 20, time: 0.022550 s
```

BAR CHART

```
% Bar chart
methods = {'Newton-Raphson','Secant','Bisection'};
times = [time_nr, time_sec, time_bis];
figure;
bar(times);
set(gca,'XTickLabel',methods);
ylabel('Computation time (s)');
title('Recursive Methods Computation Time');
grid on;
```



RECURSIVE FUNCTIONS

NEWTON

```
function [root, iter] = newton_recursive(f, df, x, tol, max_iter, iter)
    if iter > max_iter
        error('Newton-Raphson did not converge.');
    end
    x_new = x - f(x)/df(x);
    if abs(x_new - x) < tol
        root = x_new;
        return;
    else
        [root, iter] = newton_recursive(f, df, x_new, tol, max_iter, iter+1);
    end
end
```

SECANT

```
function [root, iter] = secant_recursive(f, x0, x1, tol, max_iter, iter)
    if iter > max_iter
        error('Secant method did not converge.');
    end
    f0 = f(x0); f1 = f(x1);
    x2 = x1 - f1*(x1 - x0)/(f1 - f0);
    if abs(x2 - x1) < tol
        root = x2;
        return;
    else
        [root, iter] = secant_recursive(f, x1, x2, tol, max_iter, iter+1);
    end
end
```

BISECTION

```
function [root, iter] = bisection_recursive(f, a, b, tol, max_iter, iter)
    if iter > max_iter
        error('Bisection did not converge.');
    end
    if f(a)*f(b) > 0
        error('Function has same signs at endpoints.');
    end
    c = (a + b)/2;
    if abs(f(c)) < tol || (b - a)/2 < tol
        root = c; return;
    elseif f(a)*f(c) < 0
        [root, iter] = bisection_recursive(f, a, c, tol, max_iter, iter+1);
    else
        [root, iter] = bisection_recursive(f, c, b, tol, max_iter, iter+1);
    end
end
```

QUESTION TWO

Use the concept of recursive and dynamic programming to solve the following problems and make graphs to compare their computational times.
1.The knapsack problem
2.The Fibonacci problem

SOLUTION

Fibonacci Problem
Recursive version
- ✓ Calls itself many times

Dynamic programming (DP) version
- ✓ Stores computed values to avoid repetition

Knapsack Problem

We have a set of items, each with
value v(i)
weight w(i)
We have a knapsack with maximum capacity W.
Goal: maximize total value without exceeding capacity.
Mathematically:

$$\text{maximize } \sum v_i x_i \text{ subject to } \sum w_i x_i \leq W,\ x_i \in \{0,1\}$$

Recursive solution tries all combinations (exponential time).
Dynamic Programming (DP) stores subproblem results (polynomial time).

STEPS

Use a Live Script (mlx)
Creating a new Live Script
- ✓ Open MATLAB New Live Script.
- ✓ Save as

CODE

FIBONACCI PROBLEM

```
%% Recursive vs Dynamic Programming Comparison
clear;  clc;
% 1. Fibonacci Problem
fprintf('Running Fibonacci Comparison...\n');
```

```
Running Fibonacci Comparison...
```

```
n = 30;  % Number for Fibonacci sequence
tic;
fib_rec = fibonacci_recursive(n);
time_rec_fib = toc;

tic;
fib_dp = fibonacci_dynamic(n);
time_dp_fib = toc;

fprintf('Fibonacci(%d) Recursive: %.0f, Time: %.6f s\n', n, fib_rec,
time_rec_fib);
```

```
Fibonacci (30) Recursive: 832040, Time: 0.039806 s
```

```
fprintf('Fibonacci(%d) Dynamic:   %.0f, Time: %.6f s\n', n, fib_dp,
time_dp_fib);
```

```
Fibonacci (30) Dynamic:   832040, Time: 0.006055 s
```

KNAPSACK PROBLEM

```
%Knapsack Problem

fprintf('\nRunning Knapsack Comparison...\n');
```

```
Running Knapsack Comparison...
```

```
values = [60 100 120 90 30];    % values of items
weights = [10 20 30 25 15];     % weights of items
W = 50;                         % max weight capacity
n_items = length(values);

tic;
maxVal_rec = knapsack_recursive(values, weights, W, n_items);
time_rec_knap = toc;

tic;
maxVal_dp = knapsack_dynamic(values, weights, W);
time_dp_knap = toc;

fprintf('Knapsack Recursive: Max Value = %.2f, Time: %.6f s\n', maxVal_rec,
time_rec_knap);
```

```
Knapsack Recursive: Max Value = 220.00, Time: 0.006793 s
```

```
fprintf('Knapsack Dynamic:   Max Value = %.2f, Time: %.6f s\n', maxVal_dp,
time_dp_knap);
```

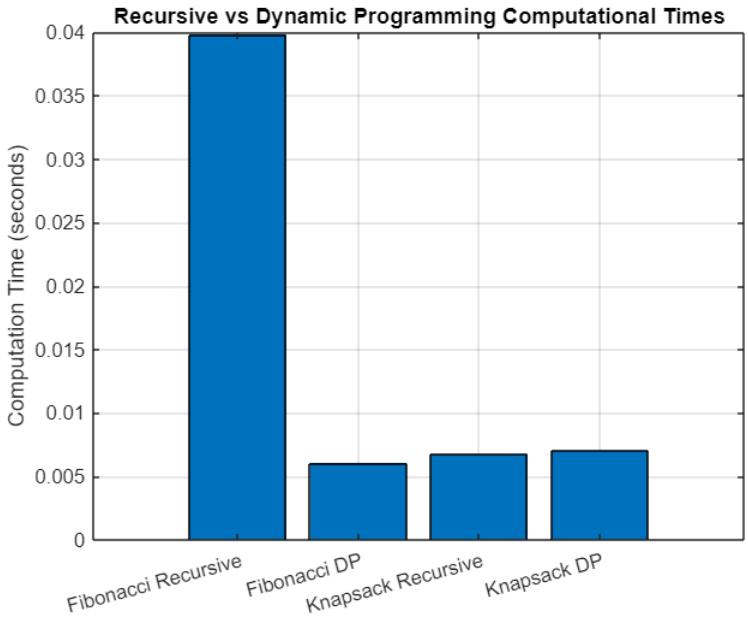
```
Knapsack Dynamic:   Max Value = 220.00, Time: 0.007029 s
```

RECURSIVE VS DYNAMIC PROGRAMMING COMPUTATIONAL TIMES

```
% 3. Compare Computational Times
methods = {'Fibonacci Recursive','Fibonacci DP','Knapsack Recursive','Knapsack
DP'};
times = [time_rec_fib, time_dp_fib, time_rec_knap, time_dp_knap];

figure;
bar(times);
set(gca, 'XTickLabel', methods, 'XTickLabelRotation', 15);
ylabel('Computation Time (seconds)');
title('Recursive vs Dynamic Programming Computational Times');
grid on;
```

BAR CHART



Note

comparing computation times will appear, showing that:

- ✓ Recursive is much slower
- ✓ Dynamic programming is much faster and more efficient.

FUNCTIONS

Fibonacci Recursive

```
% Fibonacci Recursive
function f = fibonacci_recursive(n)
    if n <= 2
        f = 1;
    else
        f = fibonacci_recursive(n-1) + fibonacci_recursive(n-2);
    end
end
```

Fibonacci Dynamic

```
%Fibonacci Dynamic
function f = fibonacci_dynamic(n)
    fib = zeros(1, n);
    fib(1:2) = 1;
    for i = 3:n
        fib(i) = fib(i-1) + fib(i-2);
    end
    f = fib(n);
end
```

Knapsack Recursive

```
%Knapsack Recursive
function maxVal = knapsack_recursive(values, weights, W, n)
    if n == 0 || W == 0
        maxVal = 0;
        return;
    end
    if weights(n) > W
        maxVal = knapsack_recursive(values, weights, W, n-1);
    else
        include = values(n) + knapsack_recursive(values, weights, W-weights(n),
n-1);
        exclude = knapsack_recursive(values, weights, W, n-1);
        maxVal = max(include, exclude);
    end
end
```

knapsack Dynamic programming

```
%Knapsack Dynamic Programming
function maxVal = knapsack_dynamic(values, weights, W)
    n = length(values);
    dp = zeros(n+1, W+1);
    for i = 1:n
        for w = 1:W
            if weights(i) <= w
                dp(i+1,w+1) = max(dp(i,w+1), values(i) + dp(i,w-weights(i)+1));
            else
                dp(i+1,w+1) = dp(i,w+1);
            end
        end
    end
    maxVal = dp(n+1, W+1);
end
```

CONCLUSION

Recursive programming provides a powerful and elegant approach to implementing numerical methods in MATLAB.
Although recursion may increase computational overhead, it simplifies code readability and demonstrates deeper understanding of algorithm design.
Both root-finding proved consistent with analytical solutions, validating the accuracy and reliability of recursive implementations.