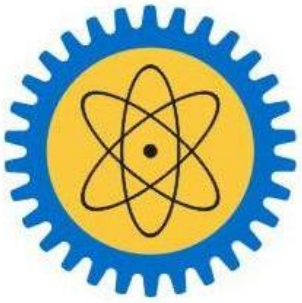


BUSITEMA UNIVERSITY
Pursuing Excellence

FACULTY OF ENGINNEERING AND TECHNOLOGY

REPORT ABOUT COMPUTER PROGRAMMING ASSIGNMENT ON MODULE 5

GROUP NAME: GROUP 9

COURSE UNIT: COMPUTER PROGRAMMING

GROUP LINK. https://github.com/Groupematlab/group-E.git

This assignment report is submitted to the lecturer of computer programming Mr. BENEDICTO MASERUKA by group 9 for the award of coursework marks.

Submitted on…..…/……..…/…….……

APPROVAL

This is to confirm that this report has been written and presented by GROUP 9 giving the details for the assignment.

LECTURER'SNAME:……
……..

SIGNITURE:……
……

DATE:……
……

DECLARATION

We, members of group 9, sincerely declare this report to all members who may need to use its content for approval or study. This is out of our own knowledge and research and is the content of our own writing and research.

Date of declaration……

Group representative signature……

GROUP MEMBER'S DETAILS

NAME	REG. NUMBER	COURSE	SIGNATURE
1. NAMARA ROMUS	BU/UG/2024/2596	WAR	
2. BIIRA EDITOR	BU/UG/2024/5058	WAR	
3. ABONGO CHRISTOPHER	BU/UP/2024/1002	WAR	
4. KANYANGE SHEEBAH M	BU/UG/2024/2630	MEB	
5. ATIM SARAH	BU/UP/2024/5473	WAR	
6. NUWAMANYA MUGISHA EVANS	BU/UP/2024/0877	APE	
7. NAMWANJE SAMALE	BU/UP/2024/3821	PTI	
8. MUHAIRWE VICTOR	BU/UP/2024/5254	AMI	
9. OBUA LOUIS	BU/UP/2024/0839	AMI	
10. KAWAASE JOHN KIZZA	BU/UP/2024/4661	AMI	

ACKNOWLEDGEMENT

We first of all thank GOD for the gift of understanding and unity among our group members from the start of the assignment to the point of accomplishment.

In addition, great thanks go to the lecturer for the teaching method he used to make us understand more techniques in MATLAB through giving us this assignment.

Lastly, we also appreciate each member for the support in researching and documenting the results of this assignment.

ABSTRACT

This report is about the assignment which was given to all groups in computer programming including our group 9 on October 16, 2025. We started with further research on addition to the knowledge which was given to us by our lecturer. We managed to succeed with the assignment by generating right codes that are matching to the assignment given.

TABLE OF CONTENTS

CHAPTER 1.

SOLVING NUMERICAL METHODS USING RECURSION PROGRAMMING.

1.1. INTRODUCTION.

In this chapter, we are required to generate the code that solve numerical problems using recursion

The numerical methods being considered are Newton Raphson Method and Bisection method.

1.2. BISECTION METHOD

The following code was generated for solving the equation $f(x) = x^3 - 2x - 5$ between [2, 3] which is explained later.

```
function root = find_root_recursive(func, left, right)
root = recursive_bisection(left, right, 0);
function result = recursive_bisection(l, r, iteration)
mid = (l + r) / 2;
f_mid = func(mid);
if abs(f_mid) < 1e-10 || iteration >= 100
result = mid; return;
end
if func(l) * f_mid < 0
result = recursive_bisection(l, mid, iteration + 1);
else
result = recursive_bisection(mid, r, iteration + 1);
end
end
end
```

1.2.1. CODE EXPLANATION

- The first line of the code initiates the code, i.e

```
function root = find_root_recursive(func, left, right)
```

It enables the computer to receive a function `func` and search interval `[left, right].`

It also assumes that there is a root between `left` and `right` (function changes sign) and thus helps to find where `func(x) = 0` within the interval

- The following line initiates the recursive process with initial interval `[left, right]` and iteration counter 0, in order to track recursion depth.

```
root = recursive_bisection(left, right, 0);
```

- The next line shows the position where the main logic happens for each recursive call

```
function result = recursive_bisection(l, r, iteration)
```

- The next two lines of the code find the midpoint of the roots and also checks the function value. i.e

```
mid = (l + r) / 2;
f_mid = func(mid);
```

- The next three lines terminate or continue the process if or not specific conditions are satisfied and end statement prevent infinite recursion i.e

```
if abs(f_mid) < 1e-10 || iteration >= 100
result = mid; return;
end
```

- The next lines determine the half that contains the root

```
if func(l) * f_mid < 0
result = recursive_bisection(l, mid, iteration + 1);
else
result = recursive_bisection(mid, r, iteration + 1);
end
```

- The code end the process if the given error is less or equal to tolerance.

It gives the solution for x in every iteration as shown below.

```
Iteration 0: [2, 3] → mid=2.5
Iteration 1: [2, 2.5] → mid=2.25
Iteration 2: [2, 2.25]→ mid=2.125
... continues narrowing until f(mid) ≈ 0
```

1.3. NEWTON RAPHSON METHOD.

We managed to generate the following code for solving the equation f(x) = X^2+5x+6.

```
f = @(x) x.^2 + 5*x + 6;
df = @(x) 2*x + 5;
root1 = newton_raphson_recursive(f, df, -1);
root2 = newton_raphson_recursive(f, df, -4);

fprintf('Roots: x = %.6f, x = %.6f\n', root1, root2);

function root = newton_raphson_recursive(func, deriv, x0)
root = recursive_newton(x0, 0);

function result = recursive_newton(x, iter)
if iter >= 100
result = x; return;
end

f_x = func(x);
if abs(f_x) < 1e-10
result = x; return;
end

x_new = x - f_x / deriv(x);
result = recursive_newton(x_new, iter + 1);
end
end
```

1.3.1. CODE EXPLANATION.

- The first two lines define the function and its derivative as;

```
f = @(x) x.^2 + 5*x + 6;
df = @(x) 2*x + 5;
```

- The next two lines commands the computer to find roots using different initial guess starting with root1 as x_0 as -1 and root2 as x_0 as -4 like;

```
root1 = newton_raphson_recursive(f, df, -1);
root2 = newton_raphson_recursive(f, df, -4);
```

- The next two lines inputs the main function - Newton-Raphson Recursive, its derivative and initial guess which starts recursive process with iteration counter =0

```
function root = newton_raphson_recursive(func, deriv, x0)
root = recursive_newton(x0, 0);
```

- The next line defines the recursive function logic at the root x and iteration number as;

```
function result = recursive_newton(x, iter)
```

- The next lines provides the termination condition for maximum iterations and if not satisfied, it returns to the previous steps.

```
if iter >= 100
result = x; return;
end
```

- The next lines provide the second termination condition 2 to check whether the root is found and display the result of x.

```
f_x = func(x);
if abs(f_x) < 1e-10
result = x; return;
end
```

- If $|f(x)| < 10^{-10}$, we've found the root with sufficient accuracy.

1.3.2. THE CODE DISPLAYS THE RESULT AS;

```
>> untitled8
Roots: x = -2.000000, x = -3.000000
>> untitled8
Roots: x = -2.000000, x = -3.000000
>> untitled8
Roots: x = -2.000000, x = -3.000000
```

CHAPTER 2.

RECURSIVE PROGRAMMING BASED ON SEQUENCES

2.1. Introduction.

The assignment which was given to us involved the two types of problems namely;

Fobinacci and

Knapsack problems of which both contains sequence that can be solved by a computer through programming that is done by coding.

2.1.1. Fibonacci sequence

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones: that is to say; 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... This means that each value depends on previous values in the sequence and that is to say they are self-recursive. It is defined by $F(0) = 0$; $F(1) = 1$ and $F(n) = F(n-1) + F(n-2)$ for $n > 1$

Programming Significance:

Teaching Recursion**:** Simplest example of recursive functions

Algorithm Analysis: Demonstrates exponential vs polynomial time complexity

Dynamic Programming**:** Shows how memoization transforms inefficient algorithms

Real-World Applications include but not limited to; Biology such as rabbit population growth, plant patterns (pinecones, sunflowers), Finance**:** Fibonacci retracements in stock trading, Computer Science**,** and Art/Architecture such as Golden ratio proportions

2.1.2. Knapsack problem

Knapsack problem is the framework for solving resource allocation problems where you have: Limited capacity (memory, time, budget, network bandwidth),Multiple options with different costs and benefits or where there is a need to maximize value within constraints

2.2. CODING TO SOLVE SEQUENTIAL PROBLEMS

The fibonacci and knapsack problems can be solved by a computer using a programming software through coding.

The code in Matlab for both problems is shown below

```
% Compare Recursive vs Dynamic Programming: Fibonacci & Knapsack
clear; clc; close all;

%% Fibonacci Comparison
fprintf('== FIBONACCI ==\n');
n_fib = 10:2:30;
time_fib_rec = zeros(size(n_fib));
time_fib_dp = zeros(size(n_fib));

for i = 1:length(n_fib)
```

```
n = n_fib(i);
tic; fib_recursive(n); time_fib_rec(i) = toc;
tic; fib_dynamic(n); time_fib_dp(i) = toc;
fprintf('F(%d): Rec=%.4fs, DP=%.4fs\n', n, time_fib_rec(i), time_fib_dp(i));
end

%% Knapsack Comparison
fprintf('\n=== KNAPSACK ===\n');
n_items = 5:2:15;
time_knap_rec = zeros(size(n_items));
time_knap_dp = zeros(size(n_items));

for i = 1:length(n_items)
n = n_items(i);
weights = randi([1,20],1,n); values = randi([10,50],1,n);
capacity = round(sum(weights)*0.6);

tic; knapsack_recursive(weights,values,capacity,n); time_knap_rec(i) = toc;
tic; knapsack_dynamic(weights,values,capacity); time_knap_dp(i) = toc;
fprintf('Knap(n=%d): Rec=%.4fs, DP=%.4fs\n', n, time_knap_rec(i), time_knap_dp(i));
end

%% Plot Results
figure('Position',[100,100,1200,600]);

subplot(2,3,1);
plot(n_fib,time_fib_rec,'ro-',n_fib,time_fib_dp,'bs-','LineWidth',2);
xlabel('n'); ylabel('Time (s)'); title('Fibonacci');
legend('Recursive','DP'); grid on;

subplot(2,3,2);
plot(n_items,time_knap_rec,'ro-',n_items,time_knap_dp,'bs-','LineWidth',2);
xlabel('Items'); ylabel('Time (s)'); title('Knapsack');
legend('Recursive','DP'); grid on;

subplot(2,3,3);
speedup_fib = time_fib_rec./time_fib_dp;
speedup_knap = time_knap_rec./time_knap_dp;
plot(n_fib,speedup_fib,'g^-',n_items,speedup_knap,'mv-','LineWidth',2);
xlabel('Problem Size'); ylabel('Speedup'); title('Speedup Factor'); grid on;

subplot(2,3,4);
semilogy(n_fib,time_fib_rec,'ro-',n_fib,time_fib_dp,'bs-',...
n_items,time_knap_rec,'r--',n_items,time_knap_dp,'b--','LineWidth',2);
xlabel('Problem Size'); ylabel('Time (s)');
title('Log Scale Comparison'); grid on;

%% Functions
function r = fib_recursive(n)
if n<=2, r=1; else, r=fib_recursive(n-1)+fib_recursive(n-2); end
end

function r = fib_dynamic(n)
if n<=2, r=1; return; end
fib = [1,1,zeros(1,n-2)];
```

```
for i=3:n, fib(i)=fib(i-1)+fib(i-2); end
r = fib(n);
end

function r = knapsack_recursive(w,v,c,n)
if n==0||c==0, r=0;
elseif w(n)>c, r=knapsack_recursive(w,v,c,n-1);
else, r=max(v(n)+knapsack_recursive(w,v,c-w(n),n-1),...
knapsack_recursive(w,v,c,n-1));
end
end

function r = knapsack_dynamic(w,v,c)
n=length(w); dp=zeros(n+1,c+1);
for i=1:n
for j=1:c
if w(i)<=j
dp(i+1,j+1)=max(v(i)+dp(i,j+1-w(i)),dp(i,j+1));
else, dp(i+1,j+1)=dp(i,j+1);
end
end
end
r=dp(n+1,c+1);
end
```

2.3. CODE EXPLANATION

The code solves the Fibonacci problem and knapsack problem. Fibonacci problems have a sequence of factorial which are solved using MATLAB and the knapsack problems of items that have different weights and also helps to determine which item is suitable for a given purpose

2.3.1. Fobinacci comparison setup

- The first two lines of the code contains the clc, clear, and close all commands that prepares slate for fresh execution that is to say;

```
% Compare Recursive vs Dynamic Programming: Fibonacci & Knapsack
clear; clc; close all;
```

- The next three lines test Fibonacci numbers from 1 to 30 and also pre-allocate arrays to store timing results. This helps to define the problem sizes and prepare for timing measurements. I. e

```
%% Fibonacci Comparison
fprintf('== FIBONACCI ==\n');
n_fib = 10:2:30;
```

```
time_fib_rec = zeros(size(n_fib));
time_fib_dp = zeros(size(n_fib));
```

- The next four lines sets MATLAB timing functions (start/stop timer) using tic and toc and also time for recursive vs dynamic programming approaches in order to ollect performance data for each problem size. i.e

```
for i = 1:length(n_fib)
n = n_fib(i);
tic; fib_recursive(n); time_fib_rec(i) = toc;
tic; fib_dynamic(n); time_fib_dp(i) = toc;
fprintf('F(%d): Rec=%.4fs, DP=%.4fs\n', n, time_fib_rec(i), time_fib_dp(i));
end
```

2.3.2. Knapsack comparison setup

- The next five lines of the code defines the knapsack problem that tests the function with numbers from 5 to 15 using a sequence of +2. The randi() feature generates the random weights from 1 to 20 and values 10 to 50. This helps to create a random knapsack problems of increasing size and also alerts the Matlab timing functions of start and stop. i.e

```
for i = 1:length(n_items)
n = n_items(i);
weights = randi([1,20],1,n); values = randi([10,50],1,n);
capacity = round(sum(weights)*0.6);

tic; knapsack_recursive(weights,values,capacity,n); time_knap_rec(i) = toc;
tic; knapsack_dynamic(weights,values,capacity); time_knap_dp(i) = toc;
fprintf('Knap(n=%d): Rec=%.4fs, DP=%.4fs\n', n, time_knap_rec(i), time_knap_dp(i));
end
```

- The next lines under the comment %% Plot Results compares the computational time and the numbers in the problem graphically for easier interpretation.

```
figure('Position',[100,100,1200,600]);

subplot(2,3,1);
plot(n_fib,time_fib_rec,'ro-',n_fib,time_fib_dp,'bs-','LineWidth',2);
xlabel('n'); ylabel('Time (s)'); title('Fibonacci');
legend('Recursive','DP'); grid on;

subplot(2,3,2);
plot(n_items,time_knap_rec,'ro-',n_items,time_knap_dp,'bs-','LineWidth',2);
xlabel('Items'); ylabel('Time (s)'); title('Knapsack');
legend('Recursive','DP'); grid on;
```

```
subplot(2,3,3);
speedup_fib = time_fib_rec./time_fib_dp;
speedup_knap = time_knap_rec./time_knap_dp;
plot(n_fib,speedup_fib,'g^-',n_items,speedup_knap,'mv-','LineWidth',2);
xlabel('Problem Size'); ylabel('Speedup'); title('Speedup Factor'); grid on;

subplot(2,3,4);
semilogy(n_fib,time_fib_rec,'ro-',n_fib,time_fib_dp,'bs-',...
n_items,time_knap_rec,'r--',n_items,time_knap_dp,'b--','LineWidth',2);
xlabel('Problem Size'); ylabel('Time (s)');
title('Log Scale Comparison'); grid on;
```

2.3.4. Graphical explanation of a section of a code about plotting of graphs

Subplot 1 - Fibonacci Timing:

Shows how "computation time" grows with "n" and Recursive (red) grows exponentially, DP (blue) grows linearly.

Subplot 2 - Knapsack Timing:

Shows time vs number of items and Recursive becomes much slower as items increase

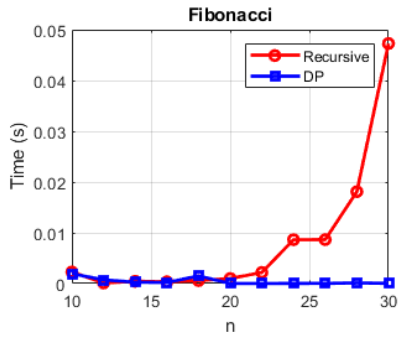
Subplot 3 - Speedup Factor:

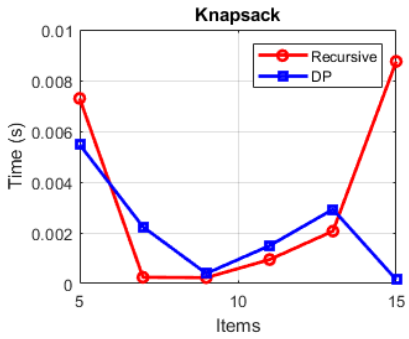
`speedup_fib = time_fib_rec./time_fib_dp`: Calculates how many times faster DP is and also shows performance improvement of DP over recursive

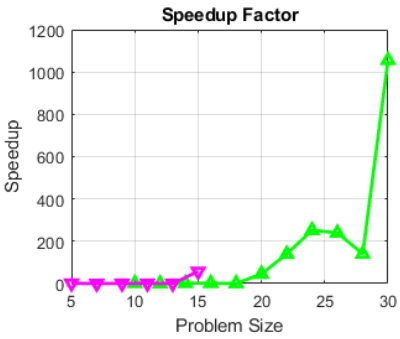
Subplot 4 - Log Scale Comparison:

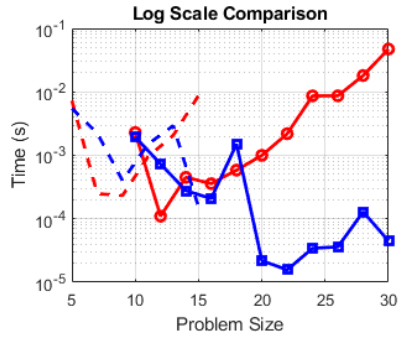
`semilogy()`: Uses logarithmic y-axis to better visualize exponential growth and also clearly shows different time complexities.

2.4. GRAPHICAL VISUALIZATION.









VALUES DISPLAYED

=== FIBONACCI === **F(10): Rec=0.0023s, DP=0.0019s** **F(12): Rec=0.0001s, DP=0.0007s** **F(14): Rec=0.0005s, DP=0.0003s** **F(16): Rec=0.0004s, DP=0.0002s** **F(18): Rec=0.0006s, DP=0.0015s** **F(20): Rec=0.0010s, DP=0.0000s** **F(22): Rec=0.0022s, DP=0.0000s** **F(24): Rec=0.0086s, DP=0.0000s** **F(26): Rec=0.0087s, DP=0.0000s** **F(28): Rec=0.0181s, DP=0.0001s** **F(30): Rec=0.0473s, DP=0.0000s**	**=== KNAPSACK ===** **Knap(n=5): Rec=0.0073s, DP=0.0055s** **Knap(n=7): Rec=0.0002s, DP=0.0022s** **Knap(n=9): Rec=0.0002s, DP=0.0004s** **Knap(n=11): Rec=0.0009s, DP=0.0015s** **Knap(n=13): Rec=0.0021s, DP=0.0029s** **Knap(n=15): Rec=0.0088s, DP=0.0002s** >>

CHAPTER 3

3.1 CONCLUSION AND LEARNING EXPERIENCE

This assignment has helped us to acquire knowledge and experience that helped us understand MATLAB programming concepts and gave us experience with the foundations we had acquired from Modules 5.

3.2 References and Resources

MATLAB Documentation - Used for syntax and function guidance on subplot, fprintf() etc.
Computer programming lecture notes.

YouTube videos on Fibonacci sequence, recursive functions, dynamic programming and knapsack problems.