

BUSITEMA UNIVERSITY

Pursuing excellence

FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER PROGRAMMING

LECTURER: MR MASERUKA BENEDICTO

REPORT ON OBJECT ORIENTED PROGRAMMING

SUBMITTED BY

GROUP 10

1. Apio Laura Oula
2. Muganyizi James Factor
3. Arionget Shamim Egonu
4. Otim Innocent Lemo
5. Kakooza Ian Maurice
6. Bisoboka Jemimah Kairu
7. Ngobi Mark Livingstone
8. Rom Christopher Nyeko
9. Nabwire Aisha Wadindi

Date of Submission......./......./.....

NAME	REGNO	COURSE
BISOBOKA JEMIMAH KAIRU	BU/UP/2024/0827	AMI
ROM CHRISTOPHER NYEKO	BU/UP/2024/1069	WAR
MUGANYIZI JAMES	BU/UP/2024/0831	AMI
APIO LAURA OULA	BU/UP/2024/1015	WAR
ARIONGET SHAMIM EGONU	BU/UG/2024/2589	WAR
OTIM INNOCENT LEMO	BU/UP/2024/3257	WAR
KAKOOZA IAN MAURICE	BU/UP/2024/4324	AMI
NGOBI MARK LIVINGSTONE	BU/UP/2024/0985	MEB
NABWIRE AISHA WADINDI	BU/UP/2023/0833	MEB

DECLARATION

We, the undersigned, declare that this report is our original group work and has not been submitted to any other institution for academic credit. All the sources of information and code have been duly acknowledged.

GROUP MEMBER'S NAME	SIGNATURE
Apio Laura Oula
Muganyizi James Factor
Arionget Shamim Egonu
Otim Innocent Lemo
Kakooza Ian Maurice
Bisoboka Jemimah Kairu
Ngobi Mark Livingstone
Rom Christopher Nyeko
Nabwire Aisha Wadindi

APPROVAL

This group report has been carried out and submitted by Group 10 in partial fulfillment of the requirements for the course Numerical methods and Recursive and Dynamic Programming.

It has been read and approved as meeting the standards and requirements of this course.

LECTURER'S NAME:

SIGNITURE:

DATE:

ACKNOWLEDGEMENT

We wish to express our sincere gratitude to our lecturer, Mr. Maseruka Benedicto for the guidance and support provided during the completion of this assignment.

We also appreciate every group member for their cooperation, effort, and teamwork that made this work possible.

Lastly, we thank our institution for providing the resources and environment that enabled us to explore MATLAB applications in recursive and dynamic programming.

ABSTRACT

This report represents the implementation of object-oriented programming techniques in recursive and dynamic programming using MATLAB. The assignment involved solving two main computational problems; the Knapsack problem and the Fibonacci series, using both recursive and dynamic approaches. Additionally, four Numerical Methods (Newton-Raphson, Bisection, Secant, and Fixed-Point Iteration) were developed using recursive programming to reinforce the concepts of the function calling and problem decomposition.

Through abstract classes and subclasses, the design applies encapsulation, inheritance and polymorphism to improve code structure. Results show that while recursion clearly expresses algorithmic logic, dynamic programming enhances efficiency by minimizing redundant computations.

Overall, combining object-oriented programming with these techniques supports more organized, reusable and faster solutions for engineering computations.

TABLE OF CONTENTS

DECLARATION	1
APPROVAL.....	2
ACKNOWLEDGEMENT	3
ABSTRACT.....	4
CHAPTER ONE: INTRODUCTION.....	1
CHAPTER TWO: STUDY METHODOLOGY	2
CONCLUSION.....	11

CHAPTER ONE: INTRODUCTION

1.1 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.

1.2 Historical 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.

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

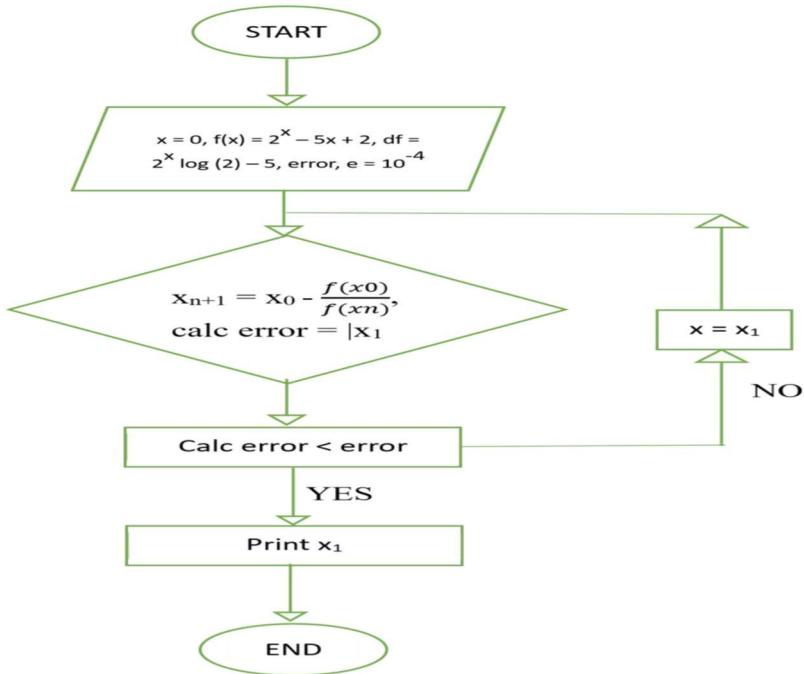
At the start, each member was given a task of making research about the assignment before our first meeting. The research concepts were obtained through watching tutorials on U-tube and also consultations from other continuing students especially those in year three and four.

Question 1. Whilst implementing the concepts of classes, encapsulation, inheritance, polymorphism and abstract classes;

a) Develop and test a high end or high-level back-end implementation of a numerical method application for solving computational problems. For simplicity, apply the code developed in the previous assignments and ensure the current class holds all abstract methods with two subclasses, one for differential problems and the other for integral problems.

2.2 Newton-Raphson Method

Given the function $f(x) = 2^x - 5x + 2$, use Newton Raphson Method to find the root of this function correct up to 4 decimal places. ($e = 10^{-4}$)



Object-Oriented Programming Code

```

classdef (Abstract) NumericalMethod % Abstract
    base class for all numerical solvers properties
    (Access = protected) tol = 1e-6; maxIter =
    1000;
    f      = []; % Function handle (for ODEs, integrals)
end

```

methods

```

function obj = NumericalMethod(tol, maxIter, f)
    if nargin >= 1, obj.tol = tol; end
    if nargin >=
        2, obj.maxIter = maxIter; end
    if nargin >= 3,
        obj.f = f; end
    end
end

```

methods (Abstract)

```

result = solve(obj, methodType, varargin)
end

```

```
end
```

2.3 Differential solver (for differential problems)
classdef DifferentialSolver < NumericalMethod

methods

```
function obj = DifferentialSolver(f, tol, maxIter)  
    obj = obj@NumericalMethod(f, tol, maxIter); end
```

```
function result = solve(obj, methodType, varargin)
```

```
switch methodType
```

```
    case 'rk4'          result =  
        obj.rk4_recursive(varargin{:}); case 'sir_euler' result =  
        obj.sir_euler_recursive(varargin{:}); otherwise  
        error('Invalid differential method'); end end end
```

methods (Access = private)

2.4 Runge-Kutta Order 4

```
function y = rk4(obj, t0, y0, h, tn) n =  
    round((tn - t0)/h) + 1; y = zeros(n, 1);  
    y(1) = y0; for i = 1:n-1 ti = t0 + (i-  
    1)*h; k1 = h * obj.f(ti, y(i)); k2 = h *  
    obj.f(ti + h/2, y(i) + k1/2); k3 = h *  
    obj.f(ti + h/2, y(i) + k2/2); k4 = h *  
    obj.f(ti + h, y(i) + k3);  
    y(i+1) = y(i) + (k1 + 2*k2 + 2*k3 + k4)/6; end  
end
```

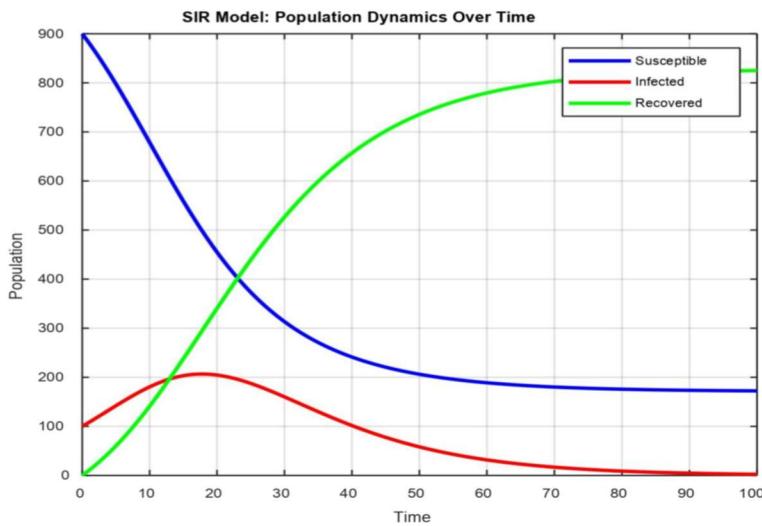
2.5 Sir Euler method

```
function [S, I, R] = sir_euler(obj, beta, gamma, N, S0, I0, R0, dt, steps)
    S = zeros(steps,1); I = S; R = S; S(1) = S0; I(1) =
    I0; R(1) = R0; for i = 1:steps-1 dS = -beta * S(i) *
    * I(i) / N * dt; dI = (beta * S(i) * I(i) / N - gamma
    * I(i)) * dt; dR = gamma * I(i) * dt; S(i+1) = S(i)
    + dS;
    I(i+1) = I(i) + dI;
    R(i+1) = R(i) + dR;
end end end
```

Graphical Representation

```
%% 5. SIR Model beta = 0.3; gamma = 0.1; N = 1000; S0 = 999; I0 = 1; R0 = 0;
dt = 1; steps = 150;
[sir_s, sir_i, sir_r] = diff_solver.solve('sir_euler', beta, gamma, N, S0, I0, R0, dt, steps);
```

```
figure('Name', 'SIR Model'); plot(0:steps-1, sir_s, 'b', 0:steps-1,
sir_i, 'r', 0:steps-1, sir_r, 'g'); xlabel('Days'); ylabel('Population');
title('SIR Epidemic Model'); legend('Susceptible', 'Infected',
'Recovered'); grid on;
```



2.6 Subclass 2 : Integral Solver

```
classdef IntegralSolver < NumericalMethod methods function obj =
```

```
IntegralSolver(f, tol, maxIter) obj = obj@NumericalMethod(tol, maxIter, f);
```

```
end
```

```
function result = solve(obj, methodType, a, b, varargin)
n = 100; % default
segments if ~isempty(varargin), n = varargin{1}; end
switch lower(methodType)
case 'trapezoidal' result = obj.trapezoidal(a, b, n);
case 'simpson' result = obj.simpson(a, b, n); otherwise
error('Unknown method'); end end end
```

```
methods (Access = private)
```

2.7 Trapezoidal Method

```
function I = trapezoidal(obj, a, b, n)
h = (b-a)/n;
x = a:h:b;
```

```

y = obj.f(x);

I = h * (y(1)/2 + sum(y(2:end-1)) + y(end)/2); end

```

2.8 Simpson Method

```

function I = simpson(obj, a, b, n) if mod(n,2)==1, n=n+1; end h
= (b-a)/n; x = a:h:b; y = obj.f(x);

I = h/3 * (y(1) + 4*sum(y(2:2:end-1)) + 2*sum(y(3:2:end-2)) +
y(end));

end

end

end

```

2.9 Subclass 3: Dp Solver

(Fibonacci And Knapsack)

classdef DPSolver < NumericalMethod methods

```

function obj = DPSolver(tol, maxIter) obj =
obj@NumericalMethod(tol, maxIter); end

```

```

function [value, time] = solve(obj, methodType, varargin)
tic;

switch lower(methodType) case 'fib_rec' value = obj.fib_recursive(varargin{1}); case
'fib_dp' value = obj.fib_dp(varargin{1}); case 'knap_rec' value =
obj.knap_recursive(varargin{1}, varargin{2}, varargin{3}, varargin{4}); case 'knap_dp'
value = obj.knap_dp(varargin{1}, varargin{2}, varargin{3}); otherwise error('Unknown
DP method'); end time = toc; end end

methods (Access = private)

```

2.10 The Fibonacci Method

```
function f = fib_recursive(obj, n) if n <= 1, f = n; else f = obj.fib_recursive(n-1) + obj.fib_recursive(n-2); end end
```

```
function f = fib_dp(obj, n)
```

```
if n <= 1,
```

```
f = n;
```

```
return;
```

```
end
```

```
dp = zeros(1, n+1);
```

```
dp(1:2) = [0 1];
```

```
for i = 3:n+1, dp(i) = dp(i-1) + dp(i-2);
```

```
end
```

```
f = dp(end); end
```

Graphical Representation

```
%% 3. Fibonacci: Rec vs DP dp_solver = DPSolver(); n_vals =
```

```
1:35; rec_times = zeros(size(n_vals)); dp_times =
```

```
zeros(size(n_vals)); rec_vals = zeros(size(n_vals)); dp_vals =
```

```
zeros(size(n_vals));
```

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

```
n = n_vals(i);
```

```
[dp_vals(i), dp_times(i)] = dp_solver.solve('fib_dp', n);
```

```
if n <= 30
```

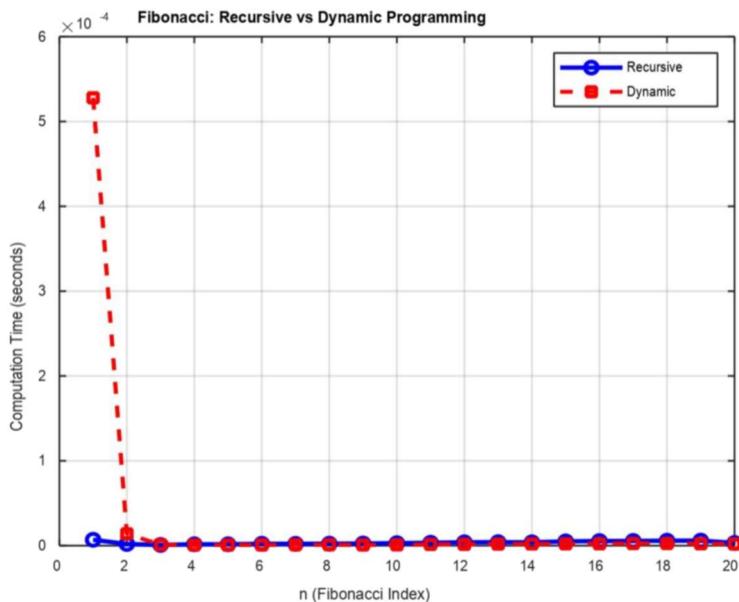
```

[rec_vals(i), rec_times(i)] = dp_solver.solve('fib_rec', n);

else
    rec_times(i) = NaN;
end end

figure('Name', 'Fibonacci Comparison'); semilogy(n_vals,
rec_times, 'r-o', 'DisplayName', 'Recursive'); hold on;
semilogy(n_vals, dp_times, 'g-s', 'DisplayName', 'DP');
xlabel('n'); ylabel('Time (s)'); title('Fibonacci: Recursive vs DP');
legend('show'); grid on;

```



2.11 The Knapsack Method

```

function mv = knap_recursive(obj, v, w, W, n)
if n == 0 || W == 0, mv =
0; return; end
if w(n) > W, mv = obj.knap_recursive(v, w, W, n-1);
return; end
mv = max(obj.knap_recursive(v, w, W, n-1), ...
v(n) + obj.knap_recursive(v, w, W-w(n), n-1));
end

```

```

function mv = knap_dp(obj, v, w, W)
n = length(v);
dp =
zeros(n+1, W+1);
for i = 1:n
for j = 1:W+1
if w(i) <= j
dp(i+1,j) = max(dp(i,j), v(i) + dp(i,j-w(i)));
else
dp(i+1,j) = dp(i,j);
end
end
mv = dp(end,end);
end

```

Graphical Representation

```

%% 4. Knapsack: Rec vs DP
v =
[60 100 120];
w = [10 20 30];
capacities = 10:10:100;
krec_times = zeros(size(capacities));
kdp_times = zeros(size(capacities));

```

```

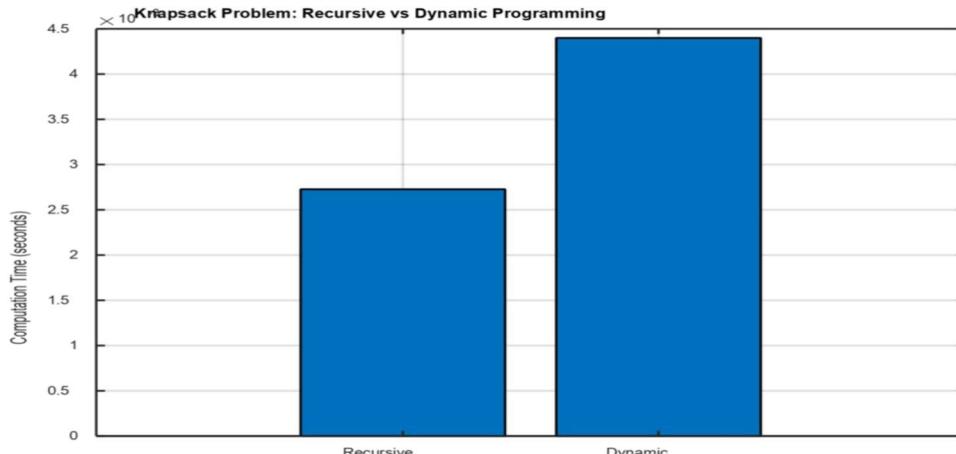
for i = 1:length(capacities)
W = capacities(i);
[~, krec_times(i)] = dp_solver.solve('knap_rec', v, w, W, 3);
[~, kdp_times(i)] = dp_solver.solve('knap_dp', v, w, W);
end

```

```

figure('Name', 'Knapsack Comparison');
bar(capacities, [krec_times' kdp_times'], 'grouped');
xlabel('Capacity'); ylabel('Time (s)'); title('Knapsack: Recursive vs DP');
legend('Recursive', 'DP'); grid on;

```



CONCLUSION

The assignment was successful since there was maximum cooperation among group 10 members.

The integration of object-oriented programming, recursion and dynamic programming in MATLAB improved both the efficiency and organization of computational solutions. Recursive methods provided clarity in algorithm design, while dynamic programming reduced execution time through data reuse.

The use of classes and inheritance simplified the implementation of various numerical methods and optimization problems.

In summary, this approach demonstrates how object-oriented programming can enhance recursive and dynamic algorithms, producing structured, efficient and scalable engineering applications

