FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER PROGRAMMING

REPORT ON OBJECT ORIENTED PROGRAMMING

BY

GROUP 10

NAME	REG NO	COURSE
BISOBOKA JEMIMAH KAIRU	BU/UP/2024/0827	AMI
ROM CH RISTOPHER 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/UP/2024/	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

This assignment report is submitted to the lecturer of computer programming Mr

BENEDICTO MASERUKA by Group 10. Submitted on…/……/…….

ABSTRACT

We started our first meeting for research on 26th, October, 2025 in the university library out of which we were exposed to various concepts on how to interact with the matlab interface, tasked to provide solutions for the knapsack problem and Fibonacci sequence using both recursive and dynamic programming in MATLAB, along with graphs to compare computation times and we managed to achieve this through group work and division of tasks.

DECLARATION

We hereby declare that the information in this report is out of our own efforts, research and it has never been submitted in any institution for any academic award

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

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 10.

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

The love of every group member to invest time and provide all they could to see the assignment a success.

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

DEDICATION

We dedicate this report to all Group 10 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 has built our understanding.

APPROVAL

This is to confirm that this report has been written and presented by GROUP 10 giving the details for the assignment.

LECTURER'S NAME: ……
SIGNITURE: ……
DATE: ……

Contents

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.

Qtn 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 methods 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.

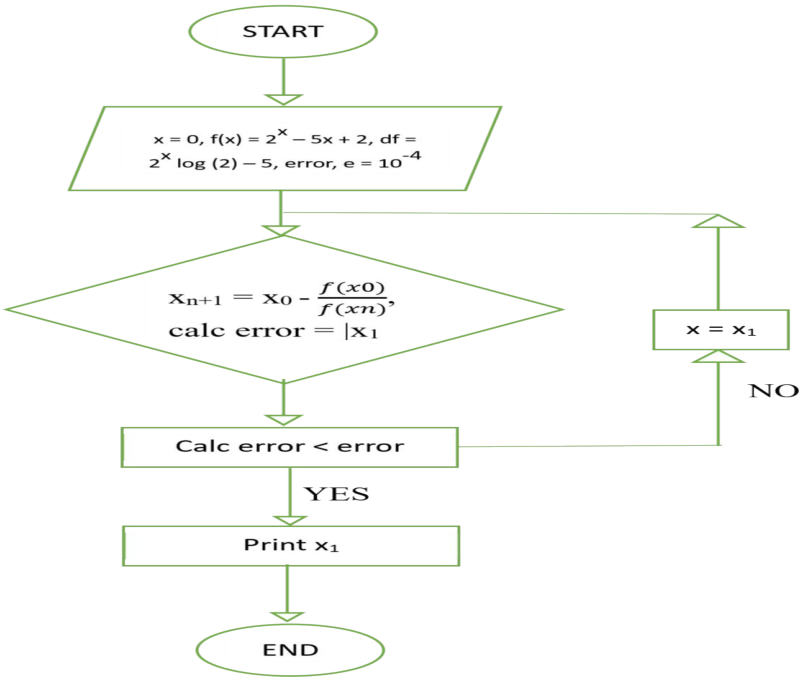
NUMERICAL APPROXIMATION METHODS

3.1 Question

b) All Numerical Approximation Methods for finding the solutions to functions. These include but are not limited to Newton Raphson, Secant, etc.

3.2 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)



NEWTON-RAPHSON METHOD(RECURSIVE)

```
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
```

DIFFERENTIAL SOLVER (FOR DIFFENTIAL 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)
```

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
```

SIR EULER METHOD

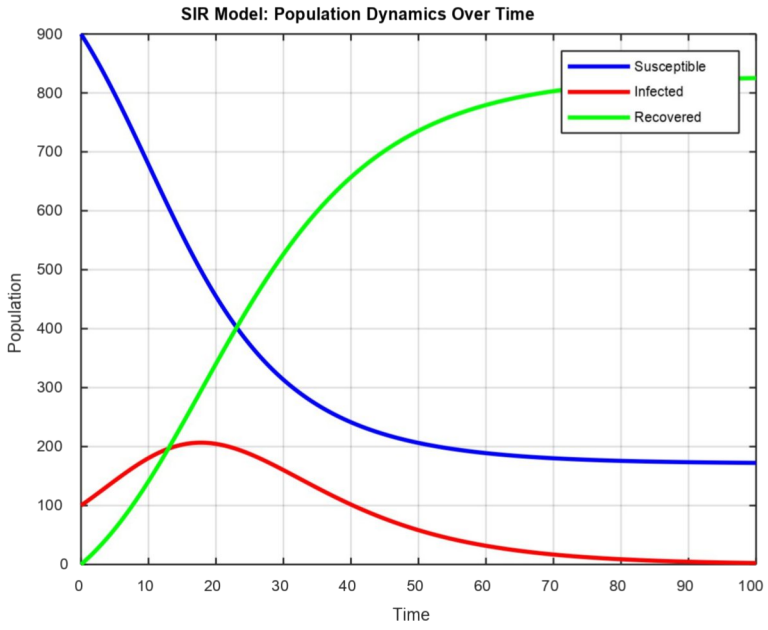
```
function [S, I, R] = sir_euler(obj, beta, gamma, N, S0, I0, R0, dt, steps)
        S = zeros(steps,1); I = S; R = S;
        S(1) = S0; I(1) = I0; R(1) = R0;
        for i = 1:steps-1
            dS = -beta * S(i) * I(i) / N * dt;
            dI = (beta * S(i) * I(i) / N - gamma * I(i)) * dt;
  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



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)
```

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
```

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
```

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)
```

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
```

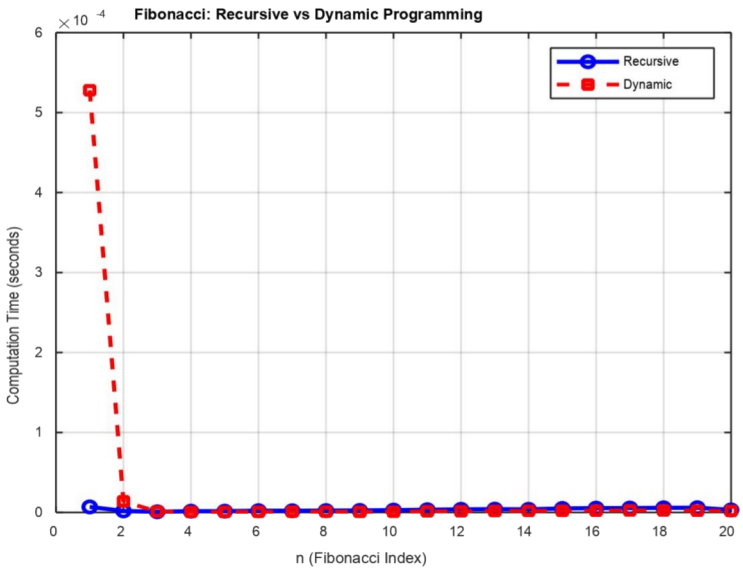
GRAPHICAL REPRESENTATION

```
%% 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;
```



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
            end
            mv = dp(end,end);
        end
    end
end
```

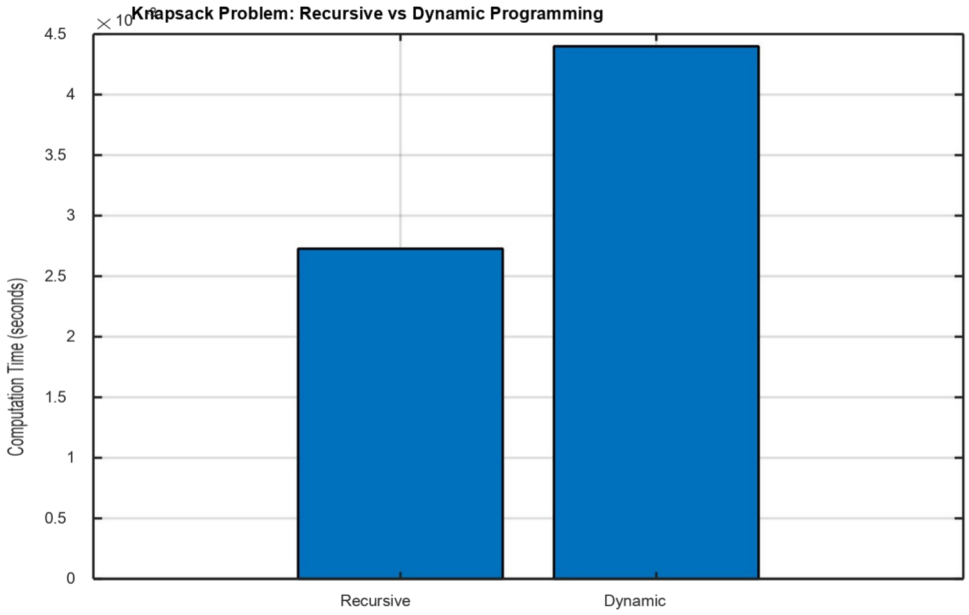
GRAPHICAL REPRESENTATION

```
%% 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 project applied numerical methods like Newton Raphson, Euler, Runge-Kutta to solve functions and differential equations for real-world problems.

Comparing analytical and numerical solutions showed the importance of choosing the right method considering accuracy, stability, and computation time.