

FACULTY OF ENGINEERING & TECHNOLOGY

##### ASSIGNMENT REPORT ON ALGORITHM DEVELOPMENT, CONTROL STRUCTURES BASED ON RECURSIVE PROGRAMING USING THE KNOWLEDGE OF MATLAB MODULES 1-5

COMPUTER PROGRAMMING

COURSE LECTURER: MR. MASERUKA BENDICTO

By GROUP 18

# ABSTRACT

This report looks at the numerical approximation methods for finding solutions to functions that is Newton Raphson Method, secant method, bisection method and fixed point iteration. It further more looks at the methods for solving differential equations numerically and they include Euler method and Runge-Kutta base on recursive programing.

Using the concept of dynamic and recursive programing, we were able to solve the Knapsack and Fibonacci problems.

The above methods were designed in the MATLAB environment that is the Live Script. Graphs are also added to compare the problems analytical solutions obtained by the different methods. The project demonstrated fundamental skills in data handling, organization, and problem-solving within the MATLAB environment, providing practical experience in a complete data workflow.

# ACKNOWLEGEMENT

By the Grace of GOD we were able to work together as a group to complete the assignment and we acknowledge him for that.

We thank, Mr. Maseruka Bendicto our course lecturer for guiding us in this course which is a vital aspect for our engineering profession.

Appreciation goes to group members for the commitment and team spirit which simplified work and made it easy for us to complete the task and come up with this report.

# DECLARATION

We, Group 18 members hereby declare to the best of our knowledge, that this assignment report is a true record of our unending efforts in applying the knowledge we acquired from modules one through five. It is truly an original creation of our own and it has never been used by any other individual for any academic award in any learning institution.

**MEMBERS OF GROUP 18**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **NAME** | **REG NUMBER** | **PROGRAM** | **SIGN** |
| **1** | AUMA DIANA | BU/UP/2024/1020 | WAR |  |
| **2** | ENAMU REAGAN EGIMU | BU/UG/2024/2672 | APE |  |
| **3** | MUKHOOLI ELIJAH | BU/UG/2024/2586 | MEB |  |
| **4** | NABAWEESI CLAIRE | BU/UP/2024/1046 | WAR |  |
| **5** | NAKAWEESA LINNET | BU/UP/2024/4327 | APE |  |
| **6** | NANDAULA CATHERINE | BU/UP/2024/4322 | AMI |  |
| **7** | OLUK CHRISTIAN GLEN | BU/UP/2024/3842 | WAR |  |
| **8** | OMARA PASCAL KELLY | BU/UP/2024/1063 | WAR |  |
| **9** | TUMUHAISE SARAH | BU/UG/2024/2674 | AMI |  |
| **10** | UHURU DENISH BRIAN | BU/UP/2024/3841 | WAR |  |

# APPROVAL

This is to confirm that this report has been written and presented by Group 18, giving details of the assignment carried out.

Course Lecturer

|  |  |
| --- | --- |
| **Signature** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |
| **Date** |  |
|  |  |

Table of Contents

[ABSTRACT ii](#_Toc211804480)

[ACKNOWLEGEMENT iii](#_Toc211804481)

[DECLARATION iv](#_Toc211804482)

[APPROVAL v](#_Toc211804483)

[CHAPTER 1 : INTRODUCTION 1](#_Toc211804484)

[CHAPTER 2 : CONCLUSION 13](#_Toc211804485)

[CHAPTER 3 : REFERENCES 14](#_Toc211804486)

# : INTRODUCTION

In this assignment we were required to using code within the MATLAB environment. In order to determine a root it is usually essential to have an initial estimate of its value. In some cases you may have more than one root (or none) and you wish to identify which one you are concerned with. The method we will describe now involves user interaction and is used as an illustration. As we are developing the ideas in this report we could consider how they could be generalized to include root selection. In general the methods we have talked about will require either an initial guess for the root or a bracketing interval containing a root. We were required to ensure that the different numerical methods were tested based on recursive programing and that this would help us to plot graphs and compare the problem analytical solutions obtained along with the computation time.

**TASK GIVEN**

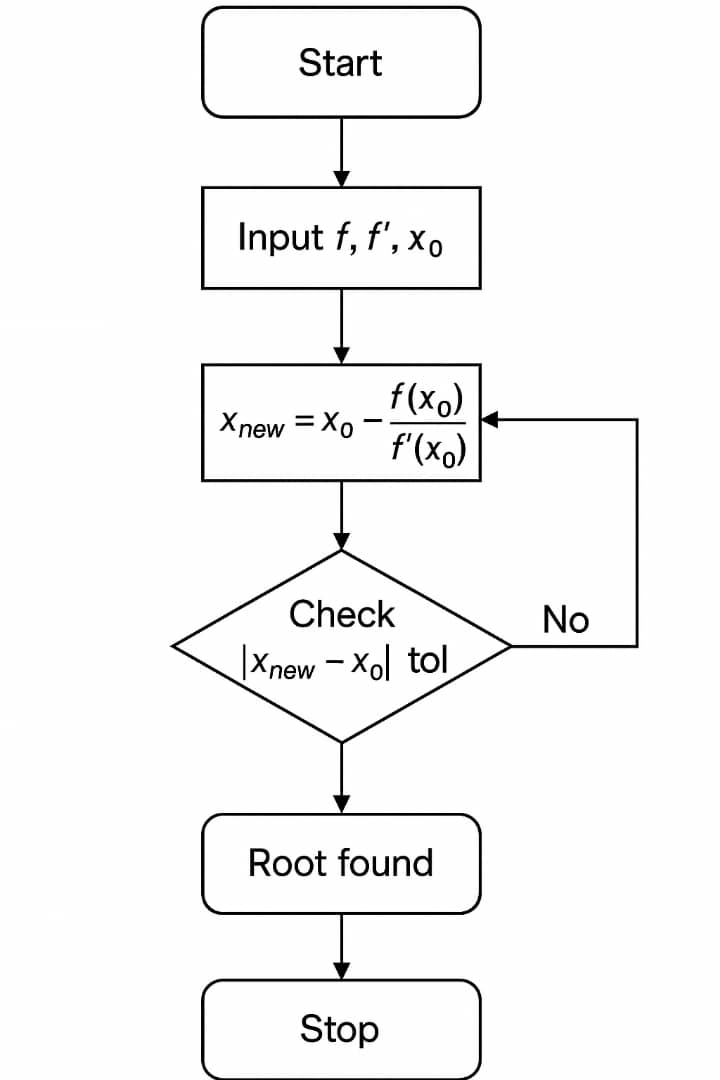
From the assignment of numerical method make an equivalent code based on recursive programing

**Solution to the question**

Here is the question we are trying to solve using Newton’s Raphson method, secant method, Bisection method and then fixed iteration based on recursive programing.

Solve

Below are the flow charts;

Newton’s Raphson Method

**Pseudo Codes for the flow Chart**

*FUNCTION NewtonRecursive(f, f', x0, tol, maxIter)*

*IF maxIter == 0 THEN*

*RETURN x0*

*ENDIF*

*x1 = x0 - f(x0) / f'(x0)*

*IF |x1 - x0| < tol THEN*

*RETURN x1*

*ELSE*

*RETURN NewtonRecursive(f, f', x1, tol, maxIter - 1)*

*ENDIF*

*END FUNCTION*

**The actual code**

function root = recursiveNewton(f, df, x0, tol, maxIter)

if maxIter == 0

root = x0;

return

end

x1 = x0 - f(x0)/df(x0);

if abs(x1 - x0) < tol

root = x1;

return

else

root = recursiveNewton(f, df, x1, tol, maxIter - 1);

end

end

f = @(x) x.^3 - x - 2;

df = @(x) 3\*x.^2 - 1;

root = recursiveNewton(f, df, 1.5, 1e-6, 50);

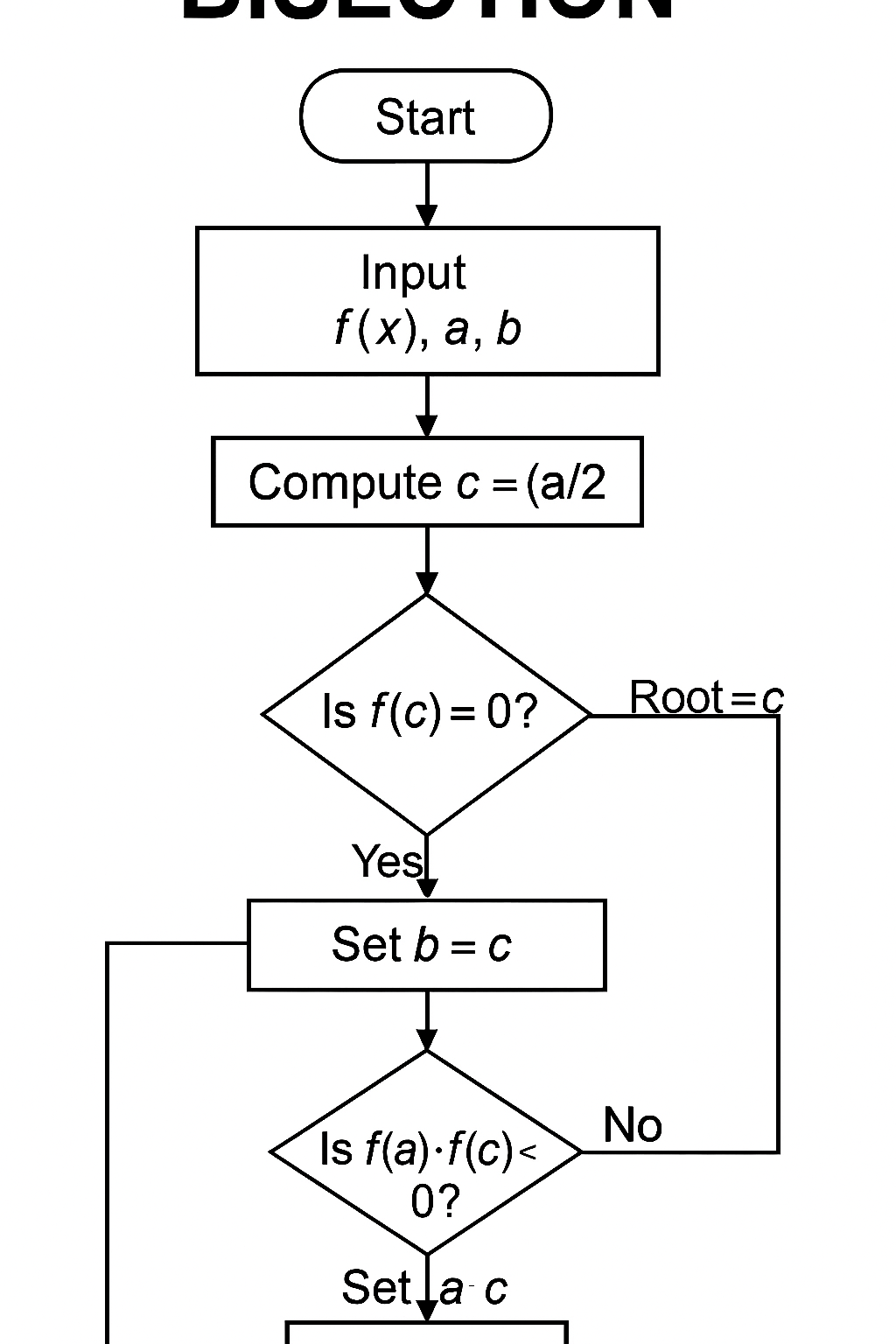
fprintf('Recursive Newton-Raphson Root: %.6f\n', root);

**Code Description:**

* f(x) is your function.
* df(x) is its derivative.
* You start with an initial guess x0.
* Each iteration improves the estimate using the slope at that point.
* The process stops when the change between iterations is very small (tol) or the maximum number of iterations is reached.

Bisection Method

Flow Chart



***Pseudo code***

*FUNCTION BisectionRecursive(f, a, b, tol, maxIter)*

*IF maxIter == 0 OR |b - a| < tol THEN*

*RETURN (a + b)/2*

*ENDIF*

*c = (a + b)/2*

*IF f(a)\*f(c) < 0 THEN*

*RETURN BisectionRecursive(f, a, c, tol, maxIter - 1)*

*ELSE*

*RETURN BisectionRecursive(f, c, b, tol, maxIter - 1)*

*ENDIF*

*END FUNCTION*

**Actual Code**

function root = recursiveBisection(f, a, b, tol, maxIter)

if maxIter == 0 || abs(b - a) < tol

root = (a + b)/2;

return;

end

c = (a + b)/2;

if f(a)\*f(c) < 0

root = recursiveBisection(f, a, c, tol, maxIter - 1);

else

root = recursiveBisection(f, c, b, tol, maxIter - 1);

end

end

f = @(x) x.^3 - x - 2;

root = recursiveBisection(f, 1, 2, 1e-6, 50);

fprintf('Recursive Bisection Root: %.6f\n', root);

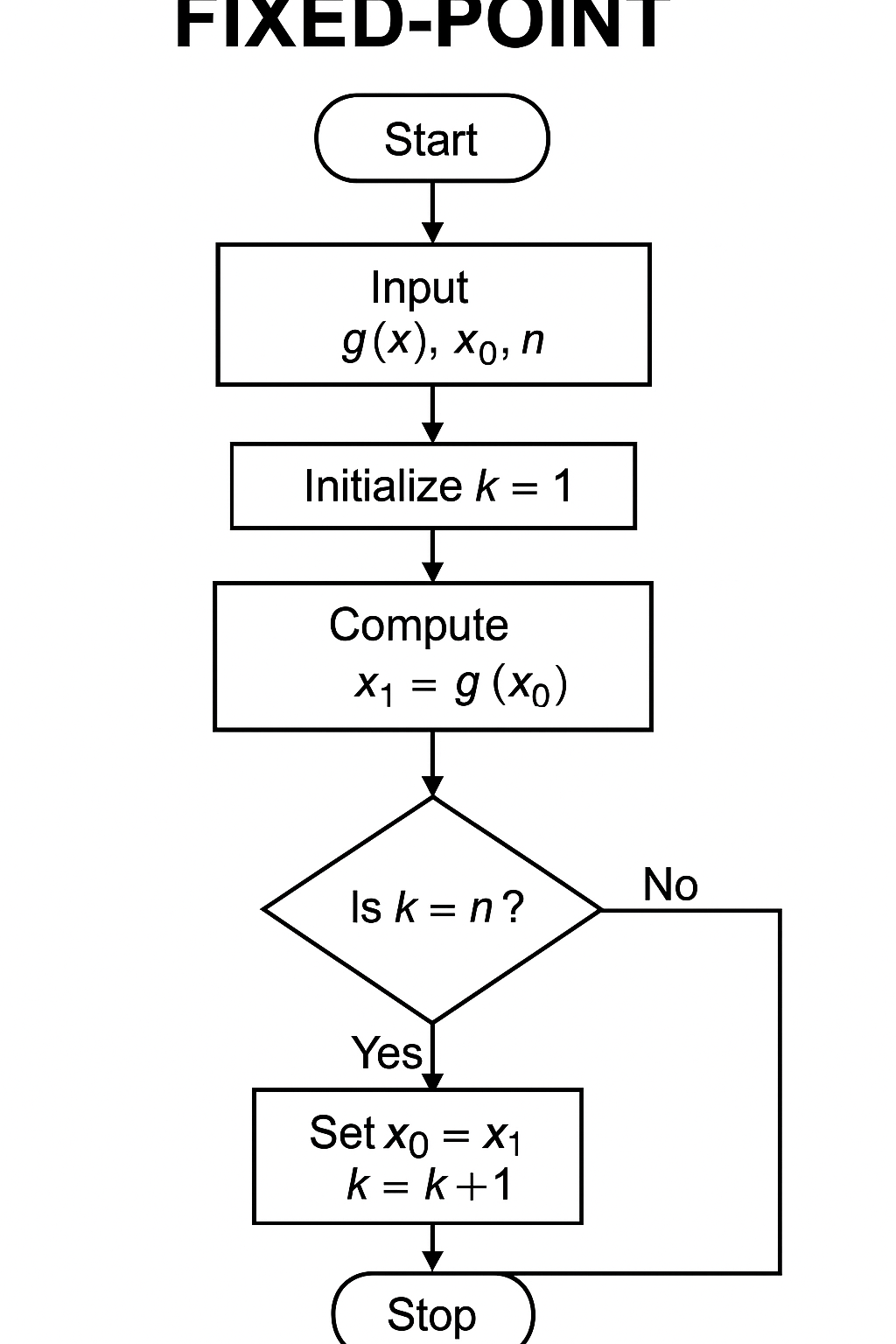
**Concept:**  
The method repeatedly halves an interval [a, b] that contains a root.

**Code Description:**

* Checks if initial interval is valid (sign change).
* Repeatedly divides the interval until the difference between a and b is smaller than tol.
* The midpoint c becomes the approximate root.

**Recursive version:**  
Instead of looping, the function keeps calling itself with smaller intervals until the root is found or iteration limit is reached.

Fixed Iteration



**Pseudo Code**

*FUNCTION FixedPointRecursive(g, x0, tol, maxIter)*

*IF maxIter == 0 THEN*

*RETURN x0*

*ENDIF*

*x1 = g(x0)*

*IF |x1 - x0| < tol THEN*

*RETURN x1*

*ELSE*

*RETURN FixedPointRecursive(g, x1, tol, maxIter - 1)*

*ENDIF*

*END FUNCTION*

**Actual Code**

function root = recursiveFixedPoint(g, x0, tol, maxIter)

if maxIter == 0

root = x0;

return;

end

x1 = g(x0);

if abs(x1 - x0) < tol

root = x1;

return;

else

root = recursiveFixedPoint(g, x1, tol, maxIter - 1);

end

end

g = @(x) (x + 2).^(1/3);

root = recursiveFixedPoint(g, 1.5, 1e-6, 50);

fprintf('Recursive Fixed Point Root: %.6f\n', root);

Code description

* Start from an initial guess x0.
* Keep updating with x1 = g(x0) until the values converge.
* Works only if ∣g′(x)∣<1|g'(x)| < 1∣g′(x)∣<1 in the region of interest (ensures convergence).

**Recursive version:**  
Reuses the same formula, but function calls itself with new x until tolerance is satisfied.

**QN**. The rate of change of a population y with respect to time t is proportional to the population itself.  
This can be modeled by the first-order differential equation:

* 1. Derive the recursive formula for Range kutta 4and Euler’s method to approximate y(t).  
     (b) Using a step size of h=0.1, write a **MATLAB recursive program** to compute the population .

Below is the Range Kutta 4 method using recursive programming

% Runge-Kutta 4th Order (Recursive)

r = 0.5;

y0 = 100;

t0 = 0; tf = 10; h = 0.1;

t = t0:h:tf;

y = rk4\_recursive(r, y0, t, h, 1);

y\_exact = y0 \* exp(r\*t);

% Recursive Function

function y = rk4\_recursive(r, y0, t, h, i)

y = zeros(size(t));

y(i) = y0;

if i == length(t)

return;

else

k1 = r\*y0;

k2 = r\*(y0 + 0.5\*h\*k1);

k3 = r\*(y0 + 0.5\*h\*k2);

k4 = r\*(y0 + h\*k3);

y\_next = y0 + (h/6)\*(k1 + 2\*k2 + 2\*k3 + k4);

y\_rest = rk4\_recursive(r, y\_next, t, h, i + 1);

y(1:i) = y0;

y(i+1:end) = y\_rest(i+1:end);

end

end

Below is the Euler’s Method using recursive programming

% Euler's Method (Recursive)

r = 0.5;

y0 = 100;

t0 = 0; tf = 10; h = 0.1;

t = t0:h:tf;

y = euler\_recursive(r, y0, t, h, 1);

y\_exact = y0 \* exp(r\*t);

% Recursive Function

function y = euler\_recursive(r, y0, t, h, i)

y = zeros(size(t));

y(i) = y0;

if i == length(t)

return;

else

y\_next = y0 + h\*r\*y0;

y\_rest = euler\_recursive(r, y\_next, t, h, i + 1);

y(1:i) = y0;

y(i+1:end) = y\_rest(i+1:end);

end

end

For the Euler’s method loop, we considered the steps below;  
 Start at initial condition y(0)=y0

 Compute next value using yi+1=y0 + h\*r\*y0;

 Call the same function again for the next time step (i + 1).

 Continue recursively until the end of the time vector is reached.

**Solution to question 2**

* 1. **Fibonacci Problem**

Below is the code showing how to solve Fibonacci Problem using recursive and dynamic approach and its comparison as the graph**.**

%Recursive approach

function f = fib\_recursive(n)

if n <= 1

f = n;

else

f = fib\_recursive(n-1) + fib\_recursive(n-2);

end

end

%Dynamic approach

function f = fib\_dynamic(n)

fib = zeros(1, n+1);

fib(1) = 0; fib(2) = 1;

for i = 3:n+1

fib(i) = fib(i-1) + fib(i-2);

end

f = fib(n+1);

end

N = 5:30;

time\_rec = zeros(size(N));

time\_dyn = zeros(size(N));

for i = 1:length(N)

tic; fib\_recursive(N(i)); time\_rec(i) = toc;

tic; fib\_dynamic(N(i)); time\_dyn(i) = toc;

end

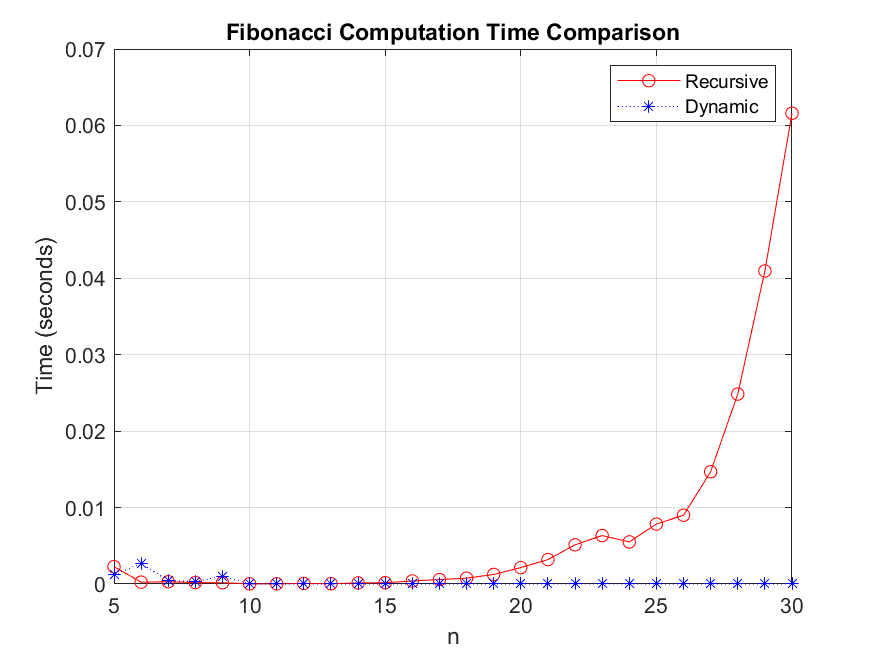
plot(N, time\_rec, 'r-o', N, time\_dyn, 'b:\*');

xlabel('n'); ylabel('Time (seconds)');

legend('Recursive', 'Dynamic');

title('Fibonacci Computation Time Comparison');

grid on;



The graph above shows a graph comparing the recursive approach and dynamic approach in terms of time against n, where n is the position in the Fibonacci sequence

* 1. **knapsack problem**

Below is the code solving the knapsack problem using recursive approach and dynamic approach and its comparison as time.

%Recursive approach

function val = knapsack\_recursive(W, wt, val\_arr, n)

if n == 0 || W == 0

val = 0;

elseif wt(n) > W

val = knapsack\_recursive(W, wt, val\_arr, n-1);

else

include = val\_arr(n) + knapsack\_recursive(W - wt(n), wt, val\_arr, n-1);

exclude = knapsack\_recursive(W, wt, val\_arr, n-1);

val = max(include, exclude);

end

end

%Dynamic approach

function val = knapsack\_dynamic(W, wt, val\_arr)

n = length(val\_arr);

K = zeros(n+1, W+1);

for i = 1:n+1

for w = 1:W+1

if i == 1 || w == 1

K(i, w) = 0;

elseif wt(i-1) <= w-1

K(i, w) = max(val\_arr(i-1) + K(i-1, w-wt(i-1)), K(i-1, w));

else

K(i, w) = K(i-1, w);

end

end

end

val = K(n+1, W+1);

end

W = 10;

wt = [2, 3, 4, 5];

val\_arr = [3, 4, 5, 6];

N = length(val\_arr);

tic; r = knapsack\_recursive(W, wt, val\_arr, N); t1 = toc;

tic; d = knapsack\_dynamic(W, wt, val\_arr); t2 = toc;

fprintf('Recursive value: %d, time = %.6f s\n', r, t1);

fprintf('Dynamic value: %d, time = %.6f s\n', d, t2);

# : CONCLUSION

By the end of the assignment, we were able to know that recursive programming provides a simple way of solving different mathematical equations and the associated problems in the daily life. We also found out the dynamic programming is simplifies the understanding especially in practical computations by solving different mathematical problems such as Fibonacci and knapsack problems.

# : REFERENCES

* MATLAB Documentation
* MATLAB lecture notes by Mr. Maseruka Bendicto
* You tube MATLAB tutorials