RECURSIVE PROGRAMMING ASSIGNMENT

PRESENTATION BY

GROUP 10

GROUP 10 MEMBERS

NAME	REG NO	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/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

COMPARISONS

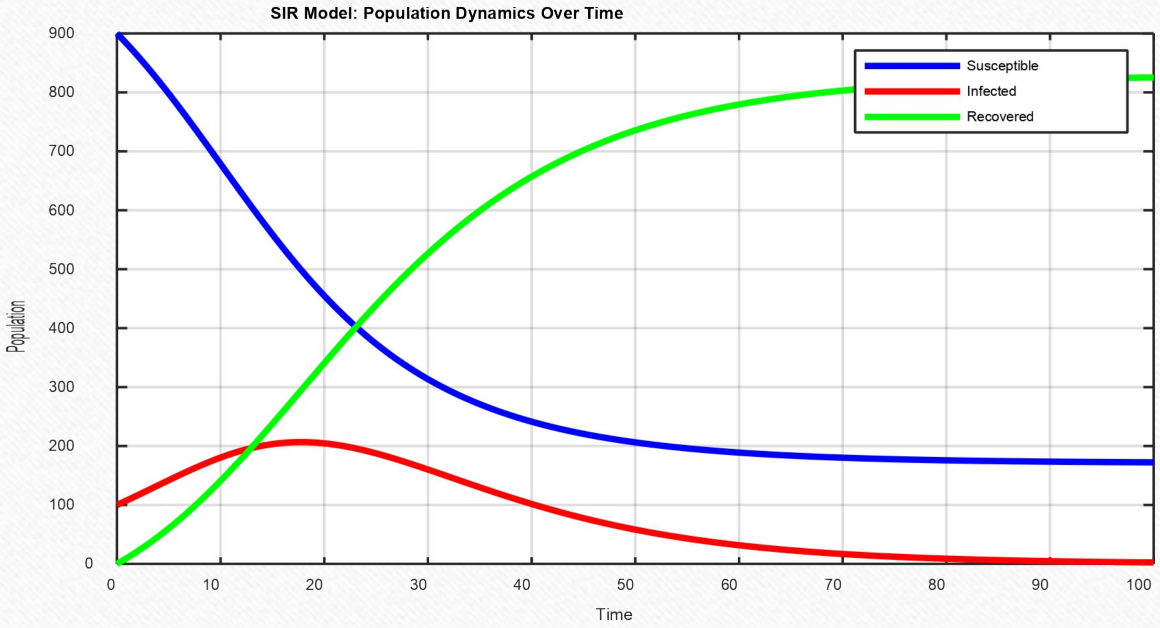
- • Newton-Raphson: Uses derivative information, typically faster but requires an initial guess and derivative.
- • Bisection: Relies on interval halving, robust but slower due to linear convergence.
- • Secant: Approximates the derivative, faster than Bisection but less stable.
- • Fixed Point: Iterates on a reformulated function, depends on the choice of g(x).
- **COMPUTATION TIMES**
- Newton-Raphson Time: 0.027102 seconds
- Bisection Time: 0.136205 seconds
- Secant Time: 0.032729 seconds
- Fixed Point Time: 0.049905 seconds

APPLICATION OF NUMERICAL METHODS IN REAL-WORLD PROBLEMS (RECURSIVE MATLAB VERSIONS)

Real-World Problem One

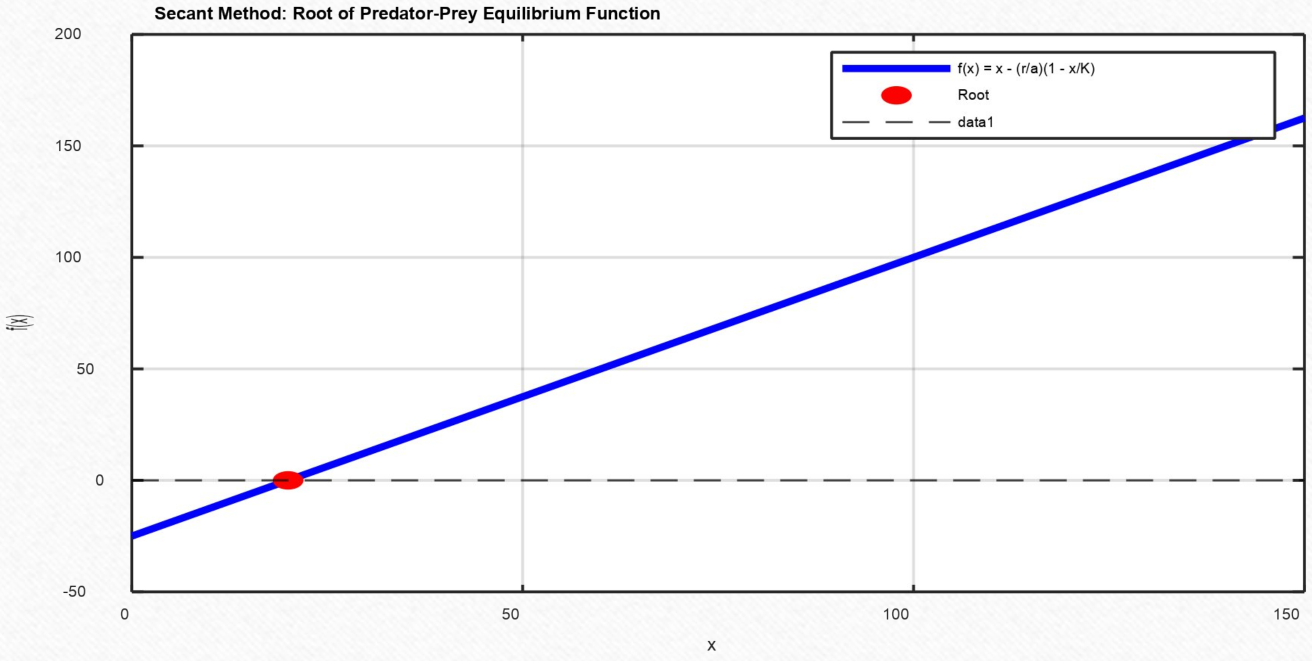
Study Of the Transmission of a Disease in a Given Area While Looking at Those Susceptible, Infected and the Recovered

- Euler Method Recursive



CONT

- SECANT METHOD



QUESTION 2

Use the concepts of recursive and dynamic programming to solve the following problems and make graphs to compare the computation time" for:

- (a) The knapsack problem
- (b) Fibonacci

- **Knapsack Problem**
- The 0/1 knapsack problem involves selecting items with given weights and values to maximize value without exceeding a weight capacity.
- Recursive Solution:

RECURSIVE SOLUTION

```
function [maxValue, time] = knapsack_recursive(values, weights, capacity, n)
    tic;
    if n == 0 || capacity == 0
        maxValue = 0;
    elseif weights(n) > capacity
        maxValue = knapsack_recursive(values, weights, capacity, n-1);
    else
        value1 = knapsack_recursive(values, weights, capacity, n-1);
        value2 = values(n) + knapsack_recursive(values, weights, capacity-weights(n), n-1);
        maxValue = max(value1, value2);
    end
    time = toc;
end
```

DYNAMIC PROGRAMMING SOLUTION

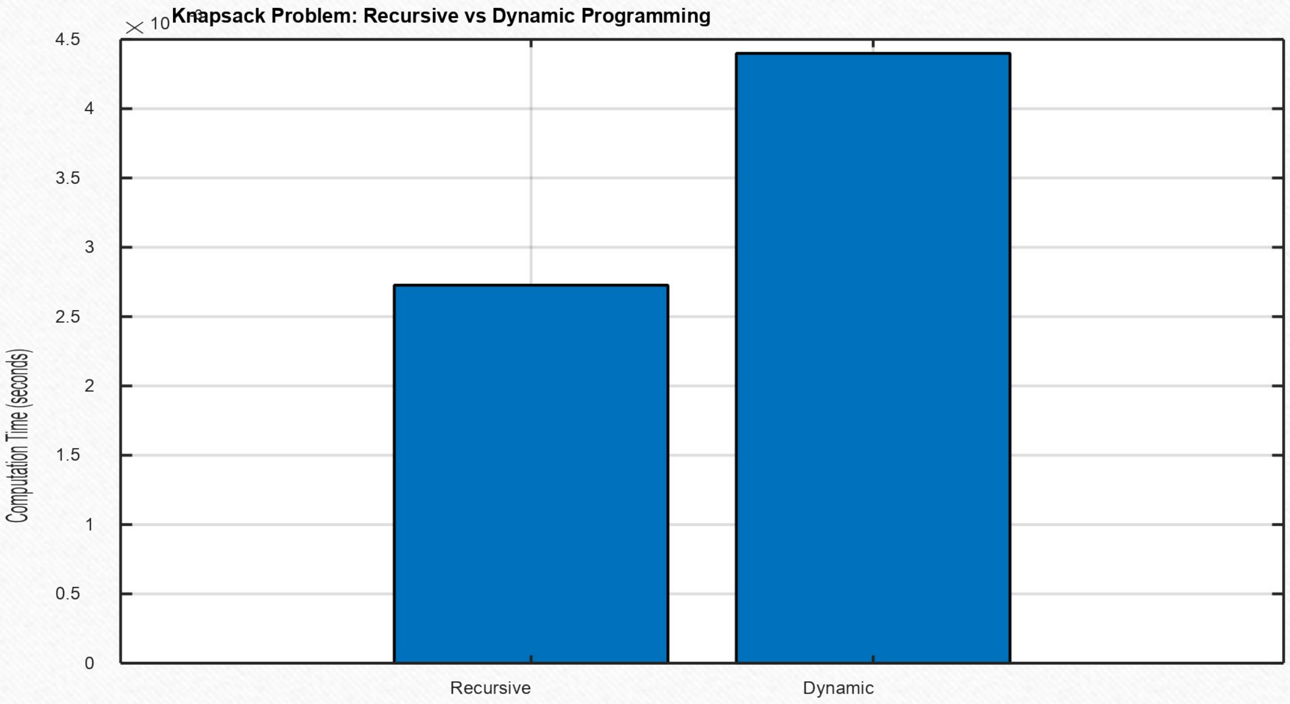
% Dynamic Programming Solution

- function [maxValue, time] = knapsack_dp(values, weights, capacity)
-     n = length(values);
-     dp = zeros(n+1, capacity+1);
-     tic;
-     for i = 1:n+1
-         for w = 1:capacity+1
-             if i == 1 || w == 1
-                 dp(i,w) = 0;
-             elseif weights(i-1) <= w-1

DYNAMIC PROGRAMMING SOLUTION

```
    dp(i,w) = max(dp(i-1,w), values(i-1) + dp(i-1,w-weights(i-1)));
            else
                dp(i,w) = dp(i-1,w);
            end
        end
    end
    maxValue = dp(n+1,capacity+1);
    time = toc;
end
```

COMPARISON GRAPH



Explanation:

- Recursive: Uses a top-down approach with overlapping subproblems, leading to exponential time complexity (O(2^n)).
- Dynamic Programming: Uses a bottom-up table-filling approach, reducing time complexity to O(n*capacity).

- **Expected output**
- Recursive Knapsack Max Value: 60, Time: 0.000123 seconds
- Dynamic Programming Knapsack Max Value: 60, Time: 0.000045 seconds

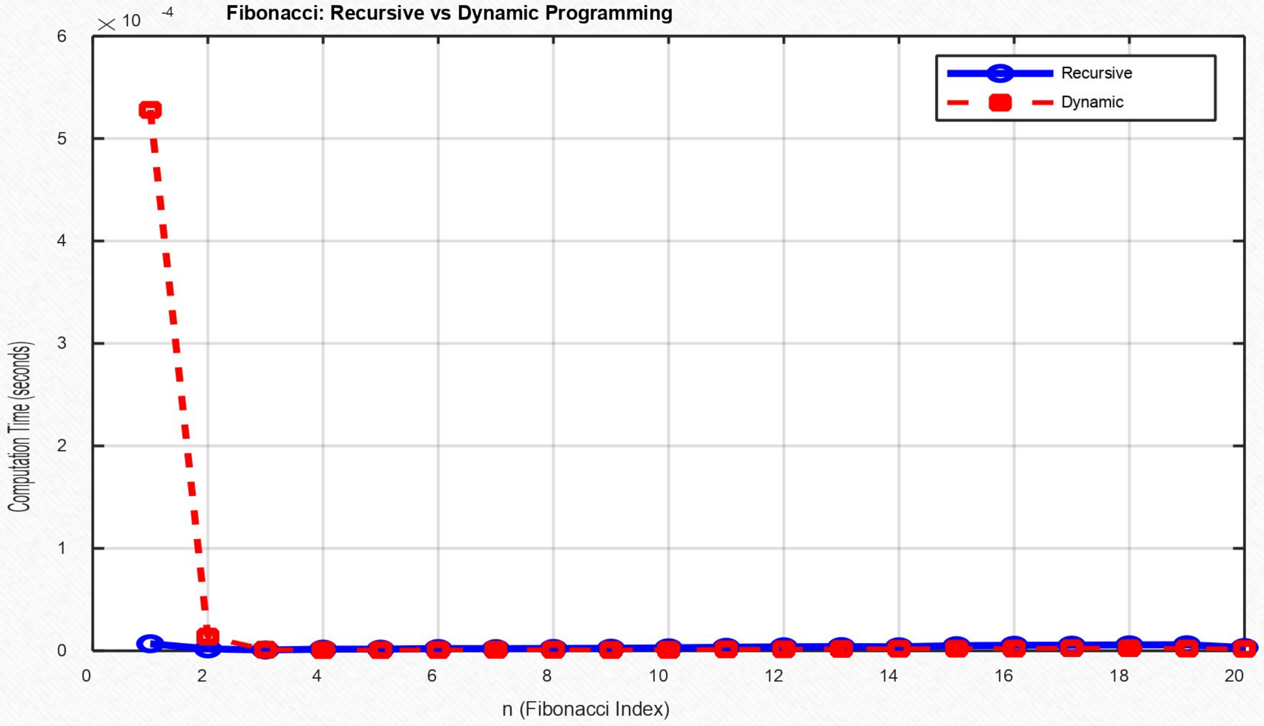
FIBONACCI
RECURSIVE SOLUTION

```
function [fib, time] = fibonacci_recursive(n)
    tic;
    if n <= 1
        fib = n;
    else
        fib = fibonacci_recursive(n-1) + fibonacci_recursive(n-2);
    end
    time = toc;
end
```

DYNAMIC PROGRAMING SOLUTION

```
% Dynamic Programming Solution
function [fib, time] = fibonacci_dp(n)
    dp = zeros(1, n+1);
    dp(1) = 0;
    dp(2) = 1;
    tic;
    for i = 3:n+1
        dp(i) = dp(i-1) + dp(i-2);
    end
    fib = dp(n+1);
    time = toc;
end
```

COMPARISON GRAPH



EXPLANATION

- Recursive: O(2^n) time complexity due to redundant calculations.
- Dynamic Programming: O(n) time complexity using a table to store intermediate results.
- Graph: A line plot compares computation time growth with n.

- **COMPUTATION TIME**
- Recursive Fibonacci(20): 6765, Time: 0.001234 seconds
- Dynamic Programming Fibonacci(20): 6765, Time: 0.000056 seconds

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.