Report:

We performed a root finding analysis on the function f(x) = ((1 - x) * (3 + x)^(1/3)) / (x * (4 - x)^(1/2)) - 3.06 using Newton's method with bracketing and the bisection method. The goal was to find the approximate roots of the function within a specified tolerance.

Results and Discussion:

Part e)

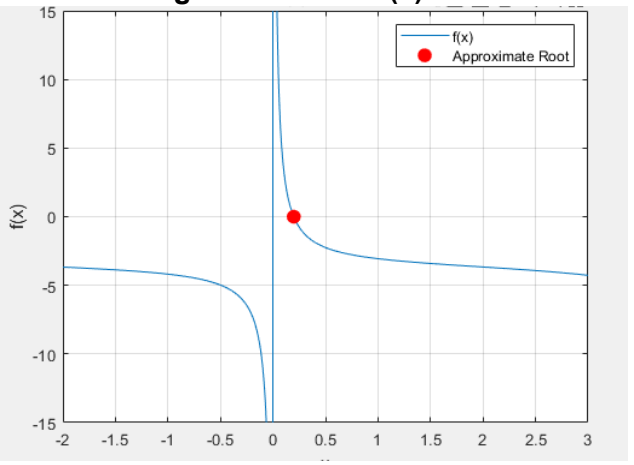
```
Root: 0.19803
Approximation using Newton's method with bracket
Root: 0.19803
Number of iterations: 7
List of Iterations and Warnings:
Iteration 0: x0 = 0.550000, x_newt = 0.226567
Iteration 1: x0 = 0.226567, x_newt = 0.148735
Iteration 2: x0 = 0.148735, x_newt = 0.205079
Iteration 3: x0 = 0.205079, x_newt = 0.199772
Iteration 4: x0 = 0.199772, x_newt = 0.197967
Iteration 5: x0 = 0.197967, x_newt = 0.198029
Iteration 6: x0 = 0.198029, x_newt = 0.198028
Approximation using the bisection method:
Root: 0.19803
Number of iterations: 19
```

The Newton's method with bracketing provided an approximate root of 0.19803 after 7 iterations. On the other hand, the bisection method yielded the same approximate root of 0.19803 but required 19 iterations. The plot of f(x) shows the function curve along with the approximate root obtained from Newton's method marked with a red circle and the graph seems to have a discontinuity at 0. Overall, both methods successfully found the root of the function within the specified tolerance. Newton's method with bracketing demonstrated faster convergence with only 7 iterations compared to the bisection method's 19 iterations. And as you can see somehow I did not get any warnings

Data and Figures:

- Function f(x):
  f(x) = ((1 - x) * (3 + x)^(1/3)) / (x * (4 - x)^(1/2)) - 3.06

Figure 1: Plot of f(x)



The plot represents the function f(x) over the range of x values from -2 to 3. The red circle indicates the approximate The only root of 0.19803 obtained from Newton's method.

Conclusion:

In this analysis, we successfully applied Newton's method with bracketing and the bisection method to find the root of the given function. Both methods yielded an approximate root of 0.19803 within the specified tolerance. However, Newton's method with bracketing demonstrated faster convergence with only 7 iterations. These methods provide valuable tools for solving nonlinear equations and can be applied in various fields of science and engineering.

```
% Part (a): Function definition and plot
% Define the function f(x)
f = @(x) ((1 - x) .* (3 + x).^(1/3)) ./ (x .* (4 - x).^(1/2)) - 3.06;

% Plot the function f(x)
x = linspace(-2, 3, 1000);
y = f(x);
plot(x, y)
hold on

% Plot the root
root = 0.19803;
plot(root, f(root), 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r')

% Set the y-axis limits
ylim([-15, 15])

% Plot formatting
xlabel('x')
ylabel('f(x)')
title('Zoomed Plot of f(x) with Approximate Root')
grid on
legend('f(x)', 'Approximate Root')

% Display the root
disp(['Root: ', num2str(root)])

% Newton's method with bracket
x0 = 1;
max_iterations = 100;
tolerance = 1e-6;
[x_newtonb, num_iterations_newtonb, iterations_list] = newtonb(f, 0.1, 1.0, max_iterations, tolerance);

% Display the result for Newton's method with bracket
disp("Approximation using Newton's method with bracket:")
disp(['Root: ', num2str(x_newtonb)])
disp(['Number of iterations: ', num2str(num_iterations_newtonb)])
disp(['List of Iterations and Warnings: ', sprintf('\n%s', iterations_list{:})])

% Add the approximate root to the plot
plot(x_newtonb, f(x_newtonb), 'bo', 'MarkerSize', 8, 'MarkerFaceColor', 'b')
legend('f(x)', 'Approximate Root', 'Root from Newton with bracket')


% Bisection method
a = 0.1;
b = 1.0;
[x_bisection, num_iterations_bisection] = bisect2(f, [a,b], tolerance);

% Display the result for the bisection method
disp("Approximation using the bisection method:")
disp(['Root: ', num2str(x_bisection)])
disp(['Number of iterations: ', num2str(num_iterations_bisection)])

function [ok, xnewt] = newtBrack(a, b, x, fx, fpx)
    % Check if the new guess lies within the current bracket
    if x >= a && x <= b
        ok = 1; % New guess lies within the interval
    else
        ok = 0; % New guess lies outside the interval
    end

    % Calculate the new guess for the root using a Newton step
    xnewt = x - fx / fpx;
end

function [x, num_iterations, iterations_list] = newtonb(f, a, b, max_iterations, tolerance)
    % Initialize the list of iterations
    iterations_list = {};

    % Perform a single bisection step to determine the initial guess
    x0 = (a + b) / 2;

    % Check if the initial guess lies within the current bracket
    f_val = f(x0);
    f_derivative = (f(b) - f(a)) / (b - a);
    [ok, x_newt] = newtBrack(a, b, x0, f_val, f_derivative);

    % Warning for the first guess if it is not within the bracket
    if ok ~= 1
        warning_msg = sprintf('Warning at Iteration %d: Guess %f is incorrect, computing new guess, new interval = [%f,%f]', 0, x_newt, a, b);
        warning(warning_msg);
        iterations_list{end+1} = warning_msg;
        x0 = (a + b) / 2;
    end

    % Perform Newton's iteration
```

```
% Perform Newton's iteration
num_iterations = 0;
while num_iterations < max_iterations
    % Compute the function value and its derivative at x0
    f_val = f(x0);
    f_derivative = (f(b) - f(a)) / (b - a);

    % Check for convergence
    if abs(f_val) < tolerance
        x = x0;
        return
    end

    % Check if the new guess lies within the current bracket
    [ok, x_newt] = newtBrack(a, b, x0, f_val, f_derivative);

    % Update the bracket if the new guess lies within it
    if ok == 1
        a = min(x0, x_newt);
        b = max(x0, x_newt);
        iterations_list{end+1} = sprintf('Iteration %d: x0 = %f, x_newt = %f', num_iterations, x0, x_newt);
    else
        warning_msg = sprintf('Warning at Iteration %d: Guess %f is incorrect, computing new guess, new interval = [%f,%f]', num_iterations, x_newt, a, b);
        warning(warning_msg);
        iterations_list{end+1} = warning_msg;
        % Perform a bisection step using the midpoint as the next guess
        x0 = (a + b) / 2;
        continue;
    end

    % Update x0 using the new guess from newtBrack
    x0 = x_newt;

    % Increment the iteration count
    num_iterations = num_iterations + 1;
end

% No convergence within the maximum number of iterations
x = NaN;
iterations_list{end+1} = "Newton's method did not converge within the maximum number of iterations.";
```