**Symbolic Implementation of Newton's Method in MATLAB**

**Abstract**

In this project, I explored the symbolic implementation of Newton's method in MATLAB to find the zeros of a cubic function. By leveraging MATLAB's symbolic toolbox, I performed algebraic manipulations, including differentiation and substitution, to automate the iterative process of refining approximations to the zeros. This approach provided not only high precision but also an intuitive visualization of the tangent lines converging to the zeros. Through this implementation, I gained deeper insights into the numerical method's efficacy, limitations, and the importance of thoughtful initial guesses.

Newton's method has always fascinated me as a numerical technique for approximating zeros of functions, and I wanted to take this exploration a step further by using MATLAB's symbolic toolbox. Symbolic computation allows me to manipulate functions algebraically, similar to how I would by hand, but with greater efficiency and scalability. This project focused on automating the iterative process of Newton's method symbolically, providing me with precise results and the ability to visualize how the method converges to zeros.

Using symbolic variables in MATLAB gave me flexibility and clarity. Unlike traditional numeric computations, I could explore the structure of the function and its derivative symbolically, enabling a deeper understanding of the iterative steps. Through substitution and differentiation commands, I automated the tedious algebraic steps, which highlighted the mathematical elegance of Newton's method.

I particularly appreciated how symbolic computation allowed me to visualize the tangent lines dynamically converging to the zeros. Each iteration refined the approximation, demonstrating the interplay between the function, its derivative, and the iterative formula. This process reinforced my understanding of the method's strengths, such as its rapid convergence with a good initial guess, and its limitations, like the potential for divergence with a poor starting point.

**MATLAB Code**

matlab

Copy code

% Abstract: This MATLAB script demonstrates Newton's method symbolically.

% I implemented symbolic computation to enhance my understanding of each iterative step

% and automate algebraic manipulations. This approach makes the numerical process clear and precise.

% Declare x as a symbolic variable

syms x; % I declare x symbolically to enable algebraic manipulations.

% Define the function

f = x^3 - 3\*x + 1; % I chose this cubic function to demonstrate Newton's method with symbolic computation.

% Compute the derivative symbolically

df = diff(f, x); % I calculate the derivative symbolically to use it for tangent line computations.

% Initial guess

a = -2; % I start with -2 as an initial guess because it’s near a zero based on the function's graph.

% Set parameters

tolerance = 1e-6; % I set a small tolerance to ensure precise convergence.

max\_iterations = 10; % I limit the number of iterations to prevent infinite loops.

iterations = 0; % I initialize the iteration counter to track my progress.

% Loop for Newton's Method

while iterations < max\_iterations

% Substitute the current guess into the function to evaluate f(a)

y\_a = subs(f, x, a); % I substitute the current guess (a) into f to find the function value at that point.

% Substitute the current guess into the derivative to evaluate f'(a)

slope = subs(df, x, a); % I use the derivative evaluated at a to compute the slope of the tangent line.

% Check for a zero slope

if abs(slope) < tolerance

% If the slope is too small, I stop to avoid division by zero.

fprintf('Slope is nearly zero at iteration %d. Stopping.\n', iterations+1);

break;

end

% Compute the next approximation using the Newton's method formula

a\_new = a - double(y\_a / slope); % I calculate the next guess by subtracting f(a)/f'(a) from the current guess.

% Display the iteration results

fprintf('Iteration %d: a = %.6f, f(a) = %.6f\n', iterations+1, a\_new, double(subs(f, x, a\_new)));

% I print the current guess and the function value to monitor the iterative process.

% Check for convergence

if abs(a\_new - a) < tolerance

% If the difference between successive guesses is small, I consider the method converged.

fprintf('Converged to zero at x = %.6f after %d iterations.\n', a\_new, iterations+1);

break;

end

% Update the guess for the next iteration

a = a\_new; % I set the new guess as the starting point for the next iteration.

iterations = iterations + 1; % Increment the iteration counter.

end

% Visualizing the Function and Tangent Lines

figure;

fplot(f, [-3, 3], 'b-', 'LineWidth', 1.5); % I plot the function over the range [-3, 3].

hold on;

% Plot the tangent line at the final iteration

y\_tangent = subs(f, x, a) + subs(df, x, a) \* (x - a); % I construct the tangent line formula symbolically.

fplot(y\_tangent, [-3, 3], 'r--', 'LineWidth', 1); % I plot the tangent line to visualize its intersection with the x-axis.

plot(a, subs(f, x, a), 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r'); % I mark the final approximation on the graph.

title('Newton''s Method with Symbolic Computation');

xlabel('x');

ylabel('f(x)');

grid on;

hold off;

% Interpretation of Results

% Using this script, I converged to a zero of f(x) = x^3 - 3x + 1 near x = -1.879.

% The symbolic approach allowed me to observe the iterative process and visualize the tangent line converging to the zero.

% The flexibility of symbolic computation made it easy to refine approximations and understand the method's nuances.

**My Observations**

Through symbolic computation, I could understand Newton's method on a deeper level. The ability to manipulate functions algebraically and automate iterations provided clarity and precision. I converged to x≈−1.879x \approx -1.879x≈−1.879 after a few iterations, demonstrating the efficiency of this method. However, this experience also highlighted the importance of a good initial guess, as poor starting points could lead to divergence or convergence to the wrong zero.

Visualizing the function and its tangent lines in MATLAB reinforced my learning. I appreciated seeing how each iteration brought the approximation closer to the actual zero. This approach not only improved my technical skills but also deepened my appreciation for the elegance and power of numerical methods like Newton's.