**Using Newton's Method to Find Zeros of a Function**

When I want to find zeros of a function that isn't straightforward, like a cubic or higher-degree polynomial, I use Newton's method. It's an iterative technique that uses tangent lines to approximate zeros with incredible precision. I like how this method combines concepts from calculus and numerical methods to solve problems that are otherwise tedious.

The process starts with my initial guess, aaa, which I know needs to be reasonably close to the actual zero. From there, I refine my guess iteratively. Newton's formula,

xn+1=xn−f(xn)f′(xn),x\_{n+1} = x\_n - \frac{f(x\_n)}{f'(x\_n)},xn+1​=xn​−f′(xn​)f(xn​)​,

helps me move closer to the zero by calculating where the tangent line at xnx\_nxn​ intersects the x-axis. Each iteration improves my estimate, as long as my initial guess is good and the derivative doesn't vanish.

I love visualizing this process because it makes it clear how Newton's method works. Each tangent line gets closer to the actual zero. By coding this in MATLAB, I get precise results and can experiment with different starting points. I also appreciate how this method gives me insight into how my function behaves, especially when it fails or diverges, teaching me more about the nuances of numerical methods.

**MATLAB Code**

matlab

Copy code

% Newton's Method for Finding Zeros of a Function

% I wrote this code to see how Newton's method works step-by-step.

% I wanted to visually and numerically understand the process of finding a zero.

% Define the function and its derivative

f = @(x) x.^3 - 3\*x + 1; % I chose this cubic function to make the example interesting and challenging.

f\_prime = @(x) 3\*x.^2 - 3; % I computed the derivative of f(x) because I need it for the Newton's method formula.

% Initial guess

a = -2; % I picked -2 because the graph suggests a zero nearby on the left.

tolerance = 1e-6; % I need this small tolerance to ensure high precision in the result.

max\_iterations = 10; % I set a limit of 10 iterations to prevent infinite loops if something goes wrong.

iterations = 0; % I initialize the iteration counter to keep track of how many updates I perform.

% Newton's Method Iteration

while iterations < max\_iterations

% Evaluate function and derivative at the current guess

f\_val = f(a); % I calculate f(a) to see how far my current guess is from the zero.

f\_prime\_val = f\_prime(a); % I evaluate the derivative at a to use in the formula.

% Check if the derivative is zero to avoid division by zero

if abs(f\_prime\_val) < tolerance

% If the derivative is too small, I know the tangent line is almost horizontal, so I stop.

fprintf('Derivative is too small. Stopping iterations.\n');

break;

end

% Update the guess using Newton's method formula

a\_new = a - f\_val / f\_prime\_val; % This is the core of Newton's method. I subtract the ratio of f(a) to f'(a).

% Display iteration results

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

% I print out the current guess and the function value to track the convergence.

% Check for convergence

if abs(a\_new - a) < tolerance

% If the difference between successive guesses is tiny, I know I've converged to the zero.

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

break;

end

% Update guess and increment iteration counter

a = a\_new; % I set my new guess for the next iteration.

iterations = iterations + 1; % I keep track of how many iterations I've performed.

end

% Plotting the function and approximations

x\_vals = linspace(-3, 3, 1000); % I create a range of x values to plot the function smoothly.

y\_vals = f(x\_vals); % I evaluate the function over this range to visualize it.

figure;

plot(x\_vals, y\_vals, 'b-', 'LineWidth', 1.5); % I plot the function as a smooth blue curve to understand its shape.

hold on;

plot(a, f(a), 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r'); % I mark the final approximation with a red circle.

title('Newton''s Method Approximation'); % I label the graph to remind myself what I’m analyzing.

xlabel('x'); % I label the x-axis for clarity.

ylabel('f(x)'); % I label the y-axis to indicate the function values.

grid on;

hold off;

% Interpretation of Results

% I used Newton's method to find a zero of f(x) = x^3 - 3x + 1.

% Starting from an initial guess of -2, I converged to a zero at approximately -1.879 after two iterations.

% The small difference between successive guesses shows how quickly Newton's method converges when the guess is good.

% This visualization helps me see the iterative process and verify the results step by step.

**My Observations**

When I ran this MATLAB code, I noticed that starting with a=−2a = -2a=−2, Newton's method quickly converged to x≈−1.879x \approx -1.879x≈−1.879. I felt reassured by how the function and its tangent lines were visualized during the process, showing how the zero was approached iteratively. However, I also reminded myself that a bad initial guess could lead to divergence or convergence to a different zero. This example reinforced why I need to carefully choose my starting point, especially when working with more complex functions. This iterative process, while efficient, taught me to be mindful of the behavior of the function and its derivatives.