**Derivative Reveal**

Alright, so here’s the derivative reveal code that I need to replicate. I’m planning to break this down into two parts: one where the graph is static and another where it’s animated. I like the idea of starting simple with the static version first, to get everything working smoothly, and then layering in the animation after that.

To begin, I need to set up some key parameters. First, I have to define the domain over which I’ll calculate the derivative. This means specifying values for xmin and xmax. I also need to specify a function as a string—something flexible enough to handle any function I want to input later. For now, I’ll start with something straightforward, , which has a nice bell curve shape with a twist because of the product with xxx.

The function is key, so I’ll make sure it’s passed in as a string. MATLAB uses exp for exponentiation, so I need to account for that when setting it up. Also, I need to allow for flexibility in choosing a specific point in the domain to analyze the function. This point will be where the tangent line gets plotted.

Now, about generating the curve. I need to decide how many points to use—I think I’ll start with 1,000 points to ensure the graph looks smooth. It’s important to make the code adaptable so a teacher could use it for different scenarios. The top graph will show the function, while the bottom one will plot its derivative up to the point of interest.

For the tangent line, I like the idea of keeping it as a segment rather than an infinitely long line. Visually, this makes the graph cleaner. When I run the code, I need to see two things happening: the tangent line appearing on the function, and the derivative graph being plotted up to the point of interest. I’ll also include a vertical line that connects the two graphs, keeping my focus aligned between the function and its derivative.

Now for testing—I should be able to run the code for different points. For example, if I analyze the function at x=−1, I should see a downward-sloping tangent line, indicating a negative derivative. Then, if I change the point to x=0, the tangent line should flatten out, showing a derivative of zero. I’ll repeat this for x=1 and so on, to see how the derivative behaves.

Moving to the animated version, I need to introduce a speed factor. Animation requires plotting multiple times, and I don’t want it to take forever, so the speed parameter will control how many points are skipped during plotting. This will make the animation smoother and faster.

One thing I’ll watch out for is ensuring the graph always includes the x-axis. For example, if the maximum value of the graph is negative, I’ll adjust it to zero so the axis is visible. Similarly, if the minimum value is above zero, I’ll reset it to zero. This will help keep the visualization clear and aligned with calculus concepts.

If I want to push this project further, I can add an option to plot the second derivative instead of the first. This would give the user a choice to either analyze .

I’ll need to make sure only one derivative is plotted at a time to avoid overcrowding the graph.

To summarize, my code will:

1. Define a domain and a function to work with.
2. Plot the function and its tangent line at a point of interest.
3. Display the derivative graph, aligned with the function graph.
4. Allow for animation by progressively plotting points.
5. Handle edge cases to ensure the graph includes the x-axis.
6. Optionally, let the user choose between the first and second derivatives.

That’s the plan! Time to dive into coding this step by step. I’ll make sure each part works perfectly before moving on to the next.

% Derivative Reveal Code

% Parameters

xmin = -5; % Minimum x value in the domain

xmax = 5; % Maximum x value in the domain

n = 1000; % Number of points to plot

function\_string = 'x .\* exp(-x.^2)'; % Function as a string

point\_of\_interest = 0; % Point to analyze

show\_second\_derivative = false; % Flag for first or second derivative

% Generate domain and function values

x = linspace(xmin, xmax, n); % Generate domain

f = eval(['@(x) ', function\_string]); % Convert string to function

y = f(x); % Calculate function values

% First derivative using symbolic differentiation

syms xsym;

f\_sym = str2func(['@(xsym) ', function\_string]); % Symbolic function

df\_sym = diff(f\_sym(xsym), xsym); % First derivative

d2f\_sym = diff(df\_sym, xsym); % Second derivative (optional)

% Evaluate derivatives

df = matlabFunction(df\_sym); % Convert first derivative to numeric function

d2f = matlabFunction(d2f\_sym); % Convert second derivative to numeric function

dy = df(x); % First derivative values

if show\_second\_derivative

d2y = d2f(x); % Second derivative values

end

% Tangent line at point of interest

tangent\_slope = df(point\_of\_interest); % Slope of tangent line

tangent\_line = @(x) tangent\_slope \* (x - point\_of\_interest) + f(point\_of\_interest);

% Plot Function and Derivative

figure;

subplot(2, 1, 1);

plot(x, y, 'LineWidth', 2); % Plot original function

hold on;

plot(x, tangent\_line(x), '--r', 'LineWidth', 1.5); % Plot tangent line

scatter(point\_of\_interest, f(point\_of\_interest), 80, 'r', 'filled'); % Highlight point

title('Function with Tangent Line');

xlabel('x');

ylabel('f(x)');

grid on;

hold off;

subplot(2, 1, 2);

plot(x, dy, 'LineWidth', 2); % Plot first derivative

if show\_second\_derivative

hold on;

plot(x, d2y, '--g', 'LineWidth', 1.5); % Plot second derivative (optional)

legend('First Derivative', 'Second Derivative');

else

legend('First Derivative');

end

title('Derivative Graph');

xlabel('x');

ylabel('Derivative Value');

grid on;

% Animation (optional)

animate = true; % Set to false to disable animation

if animate

figure;

for i = 1:20:n

clf; % Clear the figure

% Function graph

subplot(2, 1, 1);

plot(x, y, 'LineWidth', 2);

hold on;

plot(x, tangent\_line(x), '--r', 'LineWidth', 1.5);

scatter(x(i), f(x(i)), 80, 'r', 'filled');

title('Function with Moving Tangent Line');

xlabel('x');

ylabel('f(x)');

grid on;

hold off;

% Derivative graph

subplot(2, 1, 2);

plot(x(1:i), dy(1:i), 'LineWidth', 2);

if show\_second\_derivative

hold on;

plot(x(1:i), d2y(1:i), '--g', 'LineWidth', 1.5);

legend('First Derivative', 'Second Derivative');

else

legend('First Derivative');

end

title('Animated Derivative Graph');

xlabel('x');

ylabel('Derivative Value');

grid on;

pause(0.05); % Pause to control animation speed

end

end