**Calculating Riemann sums (left, right, midpoint, and trapezoidal sums)**

In this project, I’m focusing on figuring out the area under a curve, which is such an essential concept in calculus. This task falls within the realm of integral calculus, and I’m particularly looking at an upside-down parabola. My goal is to approximate the area under this curve over the interval from 0 to 8 seconds.

To start, I decided to use Riemann sums. The first approach I implemented was the left-hand sum. When I run this section of the code, I can see the rectangles I’ve drawn under the curve, and they align with the curve at the left-hand side of each subinterval. For instance, if I break the interval into four parts, the subintervals go from 0 to 2, 2 to 4, 4 to 6, and 6 to 8. The heights of these rectangles are determined by the function’s value at the left endpoints: 0, 2, 4, and 6.

It’s clear that these rectangles underestimate the actual area under the curve. For example, with four subintervals, I calculated an area of 448 units. To improve accuracy, I increased the number of subintervals. By doubling it to eight subintervals, the rectangles became narrower, and the area estimate increased because more rectangles get closer to matching the curve. Increasing further to 16 subintervals, with each rectangle now 0.5 units wide, brought the estimate even closer to the actual area.

It’s satisfying to see how increasing the number of subintervals narrows the gap between the estimated area and the true area. As I continued, I ensured the code could handle more flexible intervals—for example, calculating the area from 2 to 8 instead of 0 to 8. I made adjustments so the number of subintervals could align nicely, like using 24 subintervals to evenly divide the range.

Next, I implemented the right-hand sum method. This is just as valid as the left-hand approach, but here the height of each rectangle aligns with the right endpoint of the subinterval. For example, in the interval from 2 to 8, the first subinterval runs from 2 to 2.5, and I take the height at 2.5. Similarly, the second subinterval goes from 2.5 to 3, and I take the height at 3. This approach tends to overestimate the area under the curve because the rectangles extend above it.

I made sure the code could toggle easily between left-hand and right-hand sums. It’s interesting to see how the area estimates for a given interval tend to fall between the left-hand and right-hand sums. For instance, with 12 subintervals, the area falls somewhere between 456 and 476, depending on the method.

To get a better approximation, I moved on to trapezoidal sums. This method averages the left and right endpoints’ heights for each subinterval, connecting them with a line to form a trapezoid. I like how visually accurate this method looks—it’s clear that the trapezoids hug the curve more closely than the rectangles do. Calculating the trapezoidal area involves averaging the left and right heights and multiplying by the subinterval width. With six subintervals, I found the total area estimate was 491 units, and increasing to 12 subintervals brought it even closer to 492.

For my next step, I added the midpoint sum method. Instead of taking the height at the left or right endpoint, I take it at the midpoint of each subinterval. For example, if a subinterval is from 2 to 4, I calculate the height at 3. This approach seems to balance over- and underestimation really well, and the estimates converge quickly to the true area as I increase the number of subintervals. With just three subintervals, the midpoint sum was already very close to the correct value.

Finally, I made sure my code could handle different functions and intervals. For instance, I tested it with a shifted square root function, and the trapezoidal sums still worked beautifully. I could easily switch back to the original parabola and compare all four methods: left-hand sums, right-hand sums, trapezoidal sums, and midpoint sums.

By running the code and seeing the results, I felt confident in the flexibility of my implementation. Whether I want to calculate the area from 0 to 9 or from 3 to 9, I can do it. Visually displaying the rectangles, trapezoids, and midpoints really helped me understand how each method works and how they approximate the area under a curve. I’m thrilled that I can now explore these approaches in such depth and with such precision.

% MATLAB Code for Calculating Riemann Sums and Visualizing Results

% Define the function and interval

f = @(x) -x.^2 + 8\*x; % Parabolic function

a = 0; % Start of the interval

b = 8; % End of the interval

n = 10; % Number of subintervals (you can adjust this)

% Calculate the width of each subinterval

dx = (b - a) / n;

% Generate x values for the subintervals

x = linspace(a, b, n + 1);

% Left Riemann Sum

x\_left = x(1:end-1); % Left endpoints

left\_sum = sum(f(x\_left) \* dx);

% Right Riemann Sum

x\_right = x(2:end); % Right endpoints

right\_sum = sum(f(x\_right) \* dx);

% Midpoint Riemann Sum

x\_mid = (x(1:end-1) + x(2:end)) / 2; % Midpoints

midpoint\_sum = sum(f(x\_mid) \* dx);

% Trapezoidal Sum

trapezoid\_sum = sum((f(x(1:end-1)) + f(x(2:end))) / 2 \* dx);

% Display results

fprintf('Left Riemann Sum: %.2f\n', left\_sum);

fprintf('Right Riemann Sum: %.2f\n', right\_sum);

fprintf('Midpoint Riemann Sum: %.2f\n', midpoint\_sum);

fprintf('Trapezoidal Sum: %.2f\n', trapezoid\_sum);

% Plot the function and approximations

figure;

hold on;

% Plot the function

fplot(f, [a, b], 'LineWidth', 2);

title('Riemann Sums Visualization');

xlabel('x');

ylabel('f(x)');

grid on;

% Plot Left Riemann Sum rectangles

for i = 1:n

rectangle('Position', [x\_left(i), 0, dx, f(x\_left(i))], 'FaceColor', [0.8, 0.8, 1], 'EdgeColor', 'b');

end

% Plot Right Riemann Sum rectangles

for i = 1:n

rectangle('Position', [x\_right(i) - dx, 0, dx, f(x\_right(i))], 'FaceColor', [1, 0.8, 0.8], 'EdgeColor', 'r');

end

% Plot Midpoint Riemann Sum rectangles

for i = 1:n

rectangle('Position', [x\_left(i), 0, dx, f(x\_mid(i))], 'FaceColor', [0.8, 1, 0.8], 'EdgeColor', 'g');

end

% Overlay Trapezoidal Areas

for i = 1:n

fill([x(i), x(i+1), x(i+1), x(i)], [0, 0, f(x(i+1)), f(x(i))], [0.9, 0.9, 0.5], 'FaceAlpha', 0.5, 'EdgeColor', 'k');

end

legend({'f(x)', 'Left Sum', 'Right Sum', 'Midpoint Sum', 'Trapezoid Area'});

hold off;