Saipritam Rupanagudi

Jay Farrell

EE 105

January 24, 2024

EE 105 Lab 1: MATLAB as an Engineer's Problem Solving Tool

Introduction: This lab is all about understanding the basics of MATLAB and figuring out the different uses for it and how the code works through different graphs and plots. Utilizing matrices and arrays is also a major part of the lab and using the matrix operations is essentially what matters in order to find the values necessary. The final major part of this lab is understanding the Riemann integral equations that are involved and using those to plot the multitude of figures displayed in the lab manual. Overall, this lab is not only an introduction to MATLAB but also a way to quickly dive deep into how MATLAB is used in the world of math in general especially in the larger scale.

Matrices and Arrays Code:

```
clear;
close all;
clc;
% This is the code for the Matrix Multiplication Tool
% The first line defines the variable A and sets a value for it
A = [sqrt(2); 1; exp(pi)];
% We then set the values for variable B in order to later find variable C
B = [3;5;7];
% We set the equation for variable C using matrix multiplication of the two other variables
C = A.' * B;
% We end the code now with displaying the value of the variable C
display(C);
C = 171.2275
```

Explanation: I pretty much use the code provided in order to define the two variables as the two matrices and then set the variable C as a product of those two matrices. I then complete the code by outputting the variable C's value by using the term display to reach a value of 171.2275.

Scripts Code:

```
clear;
close all;
clc;
% We are repeating the code for the matrix multiplication tool but this time for variable D
% We are declaring the variable A now as 3 x 1 matrix
A = [sqrt(2); 1; exp(pi)];
% We are also declaring the variable B now but this time as a 1 x 3 matrix
B = [3;5;7];
% We now are solving for the variable D but we have to start by initializing it by setting it to 0
D = 0;
% We now make the for loop to determine the dot product of the previous two variables A and
B
% We use the length of the variable A to determine how long the for loop goes
for i = 1:length(A)
  D = D + A(i) * B(i);
end
% We end the code now with displaying the value of the variable D that we just solved for
display(D);
D = 171.2275
```

Explanation: I now add the for loop using the length of the variable A as its parameter in order for the for loop to not go on forever but go around enough times in order to achieve a value for the variable D. The rest of the code stays the same from before ending with the variable D getting displayed as the same value as variable C from before. Overall, most of the code stays the same but there are definitely some changes as well.

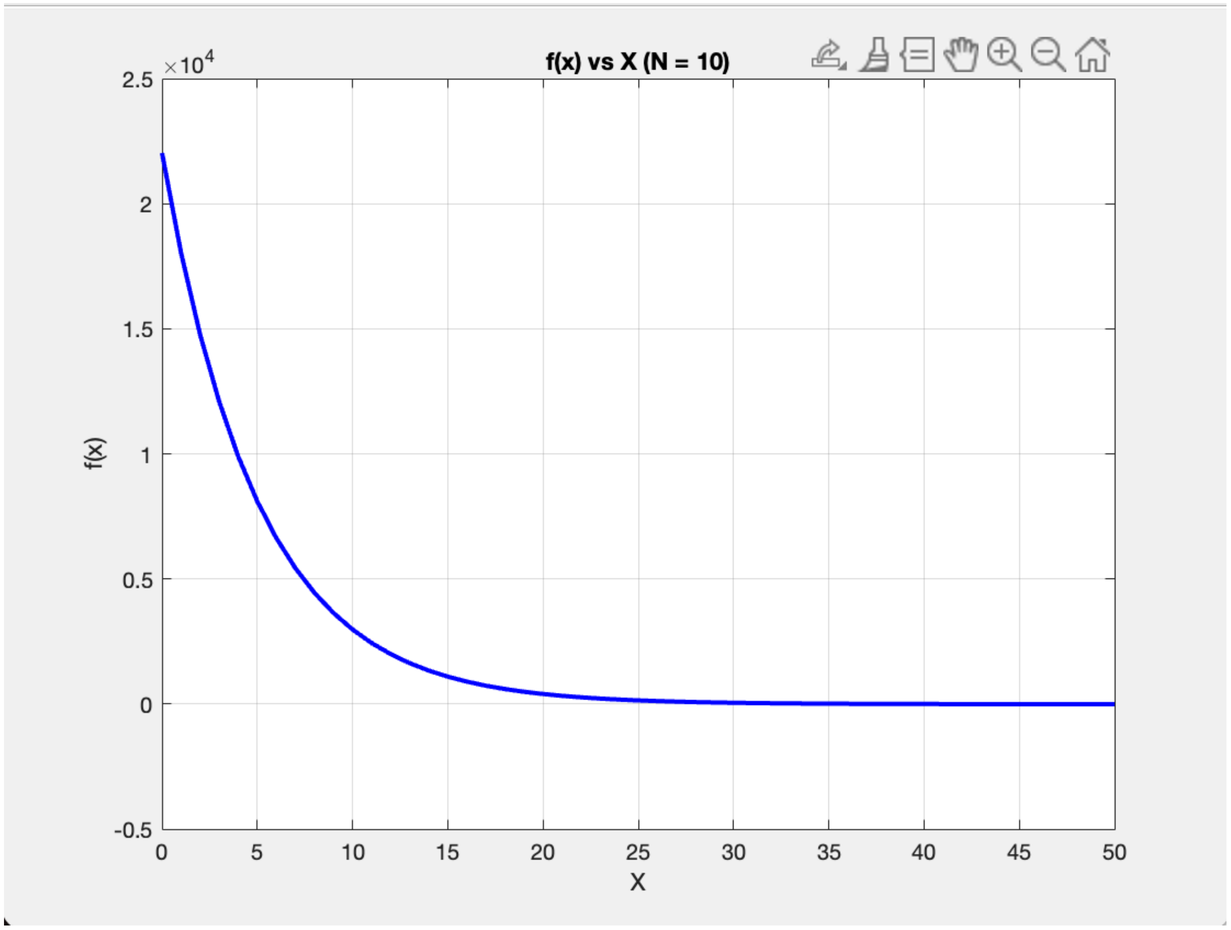
More Advanced Scripts Code:

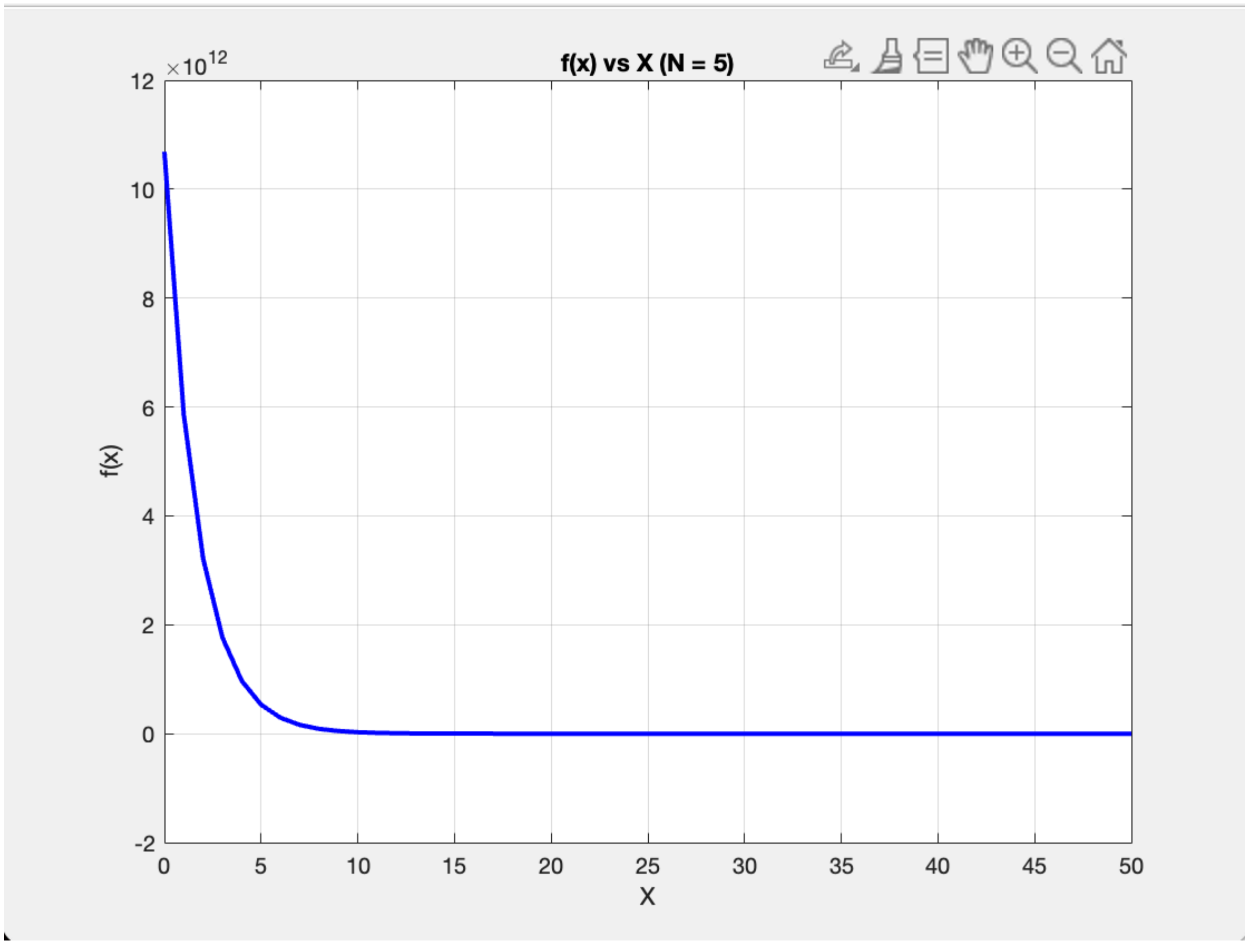
% We pretty much write the code here using the function terminology and adding in the equation provided as f(x) in the lab manual

```
function [y] = fx(x)
  y = cos(x) ./ (1 * exp(3 * x));
end
```

% We finally set the function y just like we did with the for loop and finish the first m file

Advanced Scripts Plot:





Standard Integration Code:

clear;

close all;

clc;

% We are essentially just defining the standard integration for the variable Q here with the given information

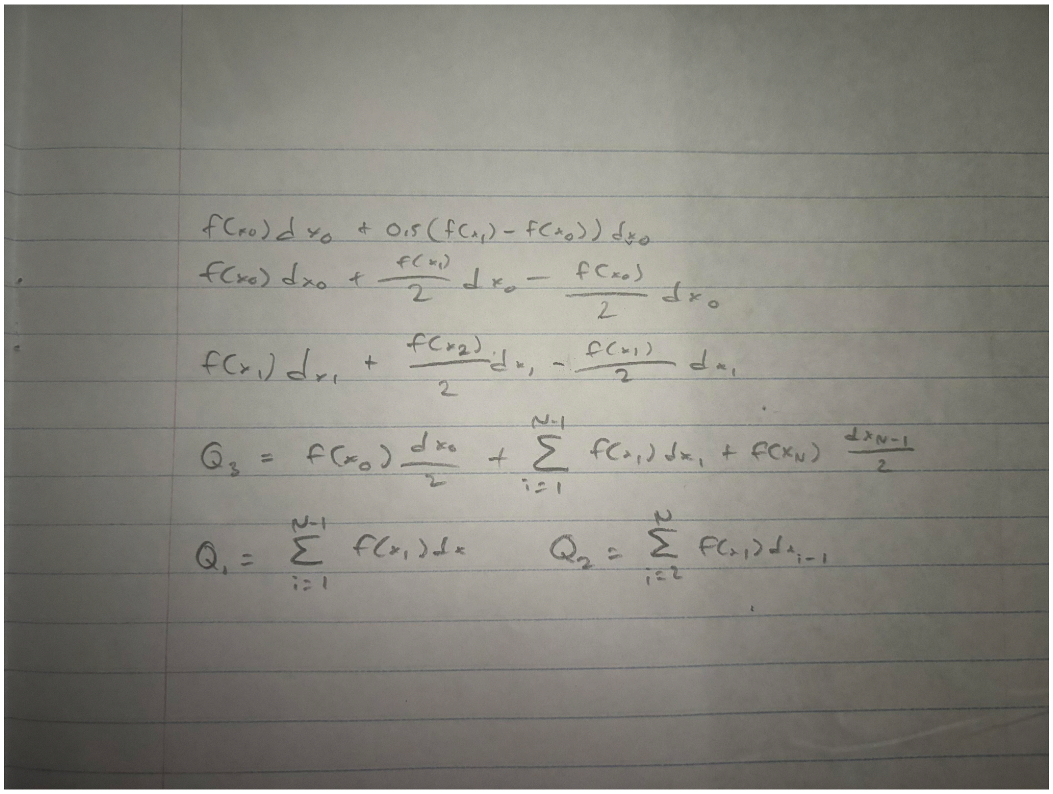
Q = quad(@fx, 0, 5);

Riemann Integral Code:

```
clear;

close all;

clc;

% Defined the fx variable with the equation provided in the lab manual

fx = cos(x) ./ (1 * exp(3 * x));

% Defined both the N and dx variables for future use

N = 25;

dx = 5/N;

% Created the x axis and the y axis for the graph

x1= dx:dx:5;

y1 = fx(x1);

x2 = 0:1/1000:5;

y2 = fx(x2);

% Set the labels for the graphs with the x label and y label while plotting the entire graph

figure;

graph(x2 - dx/2, y1, 1);

hold on;

plot(x2, y2, 'LineWidth', 2);

title('f(x) vs X (N = 5)');

xlabel('X');

ylabel('f(x)');

% Add in the legend as well for the graph that adds to the detail

legend('Line Plot');

grid on;
```

Tradeoffs: The key tradeoff with this code is the values we input in as for the variables N and dx. As they both decrease, the graph becomes more and more accurate because the chance of an error goes down. Right now the graph is essentially super simple and it doesn't involve a lot of different values but as we add more complications to the graph, the graph will become more complex and the possibility of errors obviously do increase.

Trapezoidal Approximation Derivation:



$$f(x_0)\,dx_0 + 0.5\,(f(x_1) - f(x_0))\,dx_0$$

$$f(x_0)\,dx_0 + \frac{f(x_1)}{2}\,dx_0 - \frac{f(x_0)}{2}\,dx_0$$

$$f(x_1)\,dx_1 + \frac{f(x_2)}{2}\,dx_1 - \frac{f(x_1)}{2}\,dx_1$$

$$Q_3 = f(x_0)\frac{dx_0}{2} + \sum_{i=1}^{N-1} f(x_1)\,dx_1 + f(x_N)\frac{dx_{N-1}}{2}$$

$$Q_1 = \sum_{i=1}^{N-1} f(x_1)\,dx \qquad Q_2 = \sum_{i=2}^{N} f(x_1)\,dx_{i-1}$$

Q1 Function Code:

```
% This function code is pretty much built off of the given code in the lab manual and its expected to help us in the graphs we are plotting in the near future

function Q_1 = q1_sum(N)

dx = (5 - 0) / (N - 1);

x = 0:dx:5;

y = fx(x);

A = [ones(N - 1, 1); 0];

Q_1 = y * A * dx;

End
```

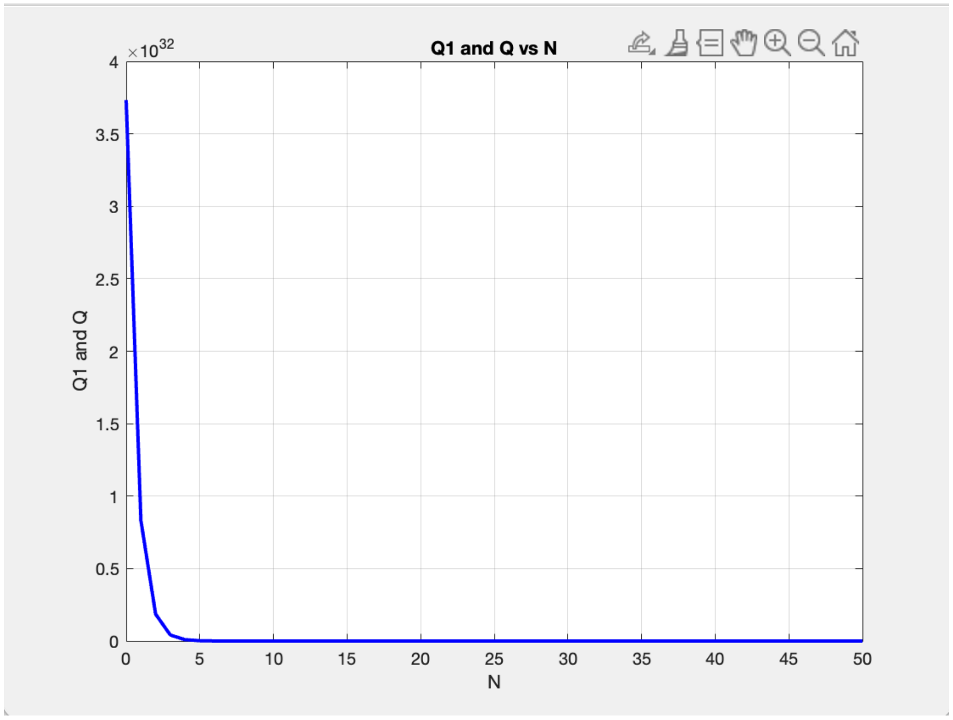
Q3 Function Code:

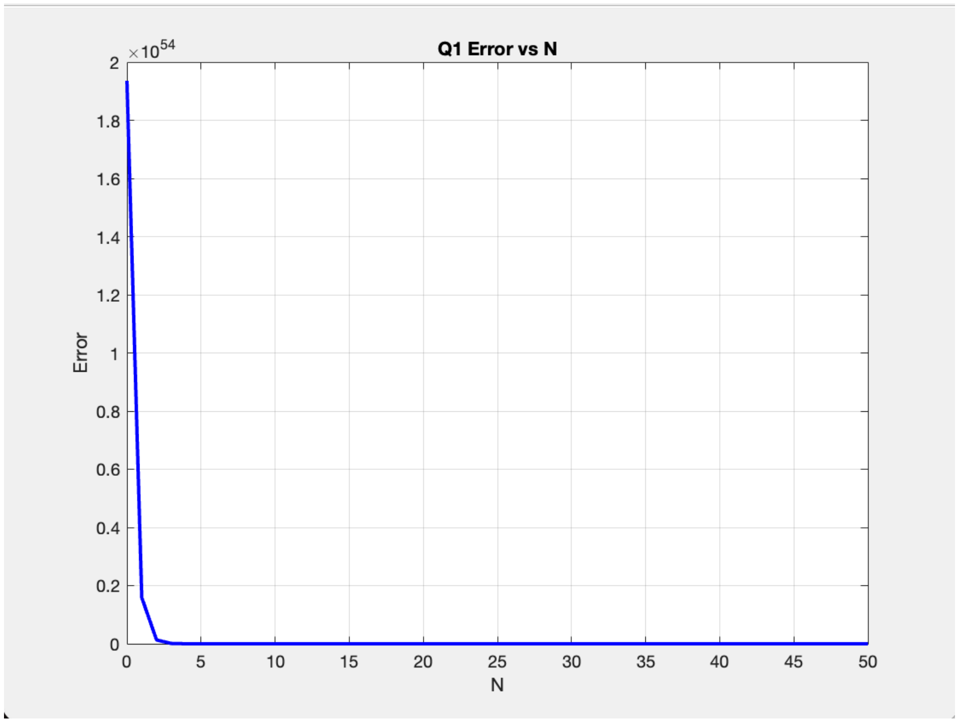
```
% The function Q3 is the one we had to code ourselves and it's pretty similar to the Q1 function we were provided just with a slight change
% The key is that Q3 is including the 0.5 at the beginning and end of the A variable that changes the results drastically
function Q_3 = q3_sum(N)
dx = (5 - 0) / (N - 1);
x = 0:dx:5;
y = fx(x);
A = [0.5; ones(N - 2, 1); 0.5];
Q_3 = y * A * dx;
End
```

Q1 and Q3 Function Code and Plots:

```
clear;
close all;
clc;
N = 2:1:200;
% Creating the necessary 3 different variables in order to graph both the original Q3 graph and the error graph
Q_3N = zeros(1, length(N));
Q_N = ones(1, length(N));
Q_exact = quad(@fx, 0, 5);
for i = 1:length(N)
  Q_3N(i) = q2_sum(N(i));
  Q_N(i) = Q_exact;
End
```

```
% Solving for the variable Q3 error as direct as possible

Q_3error = Q_3N - Q_N;

subplot(2, 1, 1);

% This is the code for plotting the original N and Q3_N variables as well on a graph

figure;

plot(n, x1, 'b-', 'LineWidth', 2);

title('Q1 and Q vs N');

xlabel('N');

ylabel(‘Q1 and Q');

grid on;

% This is the code for Plot Q_3error and plotting it as a simple graph

figure;

plot(n, x1, 'b-', 'LineWidth', 2);

title('Q1 Error vs N');

xlabel('N');

ylabel('Error');

grid on;
```





Conclusion: In conclusion, this lab was definitely a little confusing at first mainly because of the lack of clarity within the instructions but as I broke it down into smaller bits it became simpler to understand. Using MATLAB to solve these different Riemann sums and evolve them into certain graphs as shown in the lab report is definitely something I didn't expect to do at first but got more comfortable with in this lab with some practice. Overall, the different coding involved along with the multitude of different figures involved made this lab tough at first but it was definitely easier as time went on.