University of California, Riverside

Bourns College of Engineering

Department of Electrical and Computer Engineering

EE 105 Lab 1 Solution

MATLAB as an Engineer's Problem Solving Tool

Luis Fernando Enriquez-Contreras

Contents

List of Figures

Listings

1 Introduction

This introductory MATLAB laboratory course equips students with fundamental skills in scientific computing. Through hands-on exercises, students will gain proficiency in:

- Matrix algebra: Manipulating and analyzing numerical data represented as matrices
- Function creation: Defining and implementing custom functions for specific math equations
- Data visualization: Generating informative plots and graphs to interpret results
- Computational optimization: Employing techniques to improve the efficiency of numerical algorithms

By applying these acquired skills, students will develop a technical report that clearly communicates the purpose, methodology, and outcomes of the MatLab program. This report will showcase his or her understanding of MATLAB and its capabilities in solving scientific and engineering problems.

2 Matrices and Arrays

```
clea;
clc;
% Matrix Multiplication
% Declare A Matrix (3 x 1 Matrix)
A = [pi;sqrt(2);exp(pi)];
% Declare B Matrix (1 x 3 Matrix)
B = [1;5;7];
% Convert A Matix from a column to a row vector and muliple with column
```

```
% vector b to get scalar value C
C = A.' * B;
```

Listing 1: Matrix Multiplication

3 Scripts

```
clear;
clc;
% Matrix Multiplication
% Declare A Matrix (3 x 1 Matrix)
A = [pi;sqrt(2);exp(pi)];
% Declare B Matrix (1 x 3 Matrix)
B = [1;5;7];
% Set D to zero to start the for loop
D = 0;
% Assumes both vector A and B are of the same length
for i = 1:length(A)
    D = D + A(i)*B(i); % Add previous D value (sum of previous ith
   elemenents) to the cuurent ith elemement multiplication
end
display(D);
```

Listing 2: Matrix Multiplication using a For Loop

The for loop and the matrix multiplication arrive at the same result. However, matrix multiplication requires less code and is less computationally intensive than the for loop.

4 More Advanced Scripts

4.1 $f(x_i)$ Function

```
% Input column vector x and an output column vector y where the i-th
   element of the output vector is y(i) = fx(i)
function [y] = fx(x)
    y = cos(x) ./  (1 + exp(3 * x)); % Outcolumn is calculated use vector
   multiplication
end
```

Listing 3: $f(x_i)$ Function

4.2 Plot $f(x_i)$

```
function plot_export(N, title_text, filename) % Input N for the number
   spacings between the elements, title_text for the name of the plots,
   and the filename in order to export the plot. Note each plot must have
    a unique filename
x = linspace(0, 5, N + 1);
y = fx(x);
plot(x, y); % Creates the scatter plot
title(title_text,'Interpreter','latex'); % Adds title to the plot. Adding
   $$ between the text, and adding 'Interperter', 'latex' to the matlab
   function, creates text with LaTeX formatting
xlabel('$X$','Interpreter','latex'); % Adds xlabel to the plot. Adding $$
   between the text, and adding 'Interperter', 'latex' to the matlab
   function, creates text with LaTeX formatting
```

```matlab
ylabel('$f(x)$','Interpreter','latex'); % Adds ylabel to the plot. Adding $$ between the text, and adding 'Interperter', 'latex' to the matlab function, creates text with LaTeX formatting
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, append(filename, '.pdf'),'ContentType','vector'); % Exports plot as a vector pdf image. (Requires R2020a or later)
end
```

Listing 4: Plot Function for $f(x_i)$

```matlab
clear;
clc;
% This code automatically calculates, labels, and plots
N = [200, 5, 10]; % Declare the N that will be ran
for i = N % iterate through given N for example i = 200
    title_text = {['$f(x)$ vs $X$'] [append('$N = ', string(i), '$') ]}; % Create the the title for each i. Convert the integer to a string and add it to the title
    filename = append('Fig/fx_', string(i)); % Add number to the filename to ensure each plot has a unique filename
    plot_export(i, title_text, filename); % With i, title_text, and filename determined run and export documentex each plot. Vector pdf plots should be in the same folder as the code location ready to be inserted to a tex document
end
```

Listing 5: Code to run function for $f(x_i)$

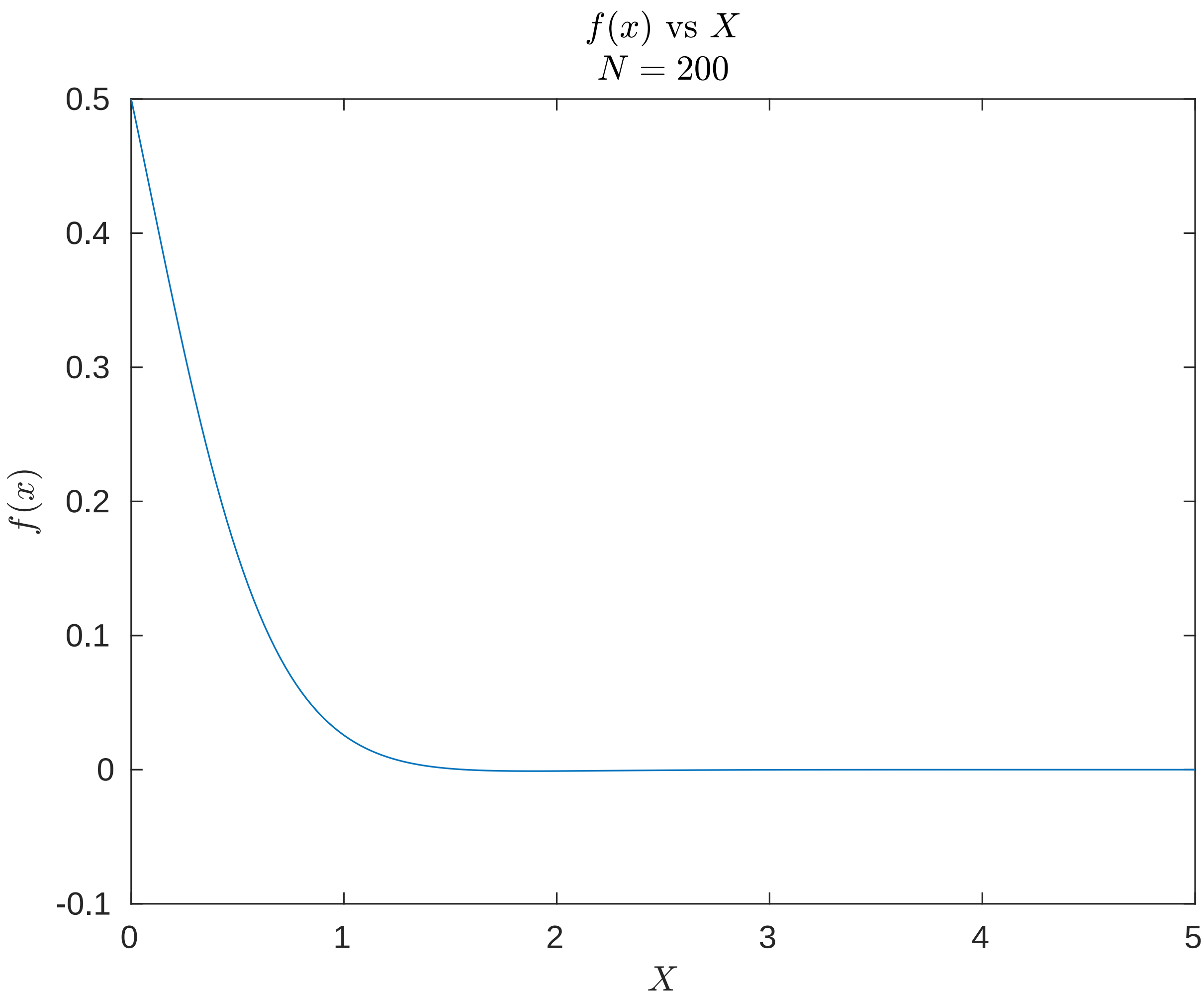

Figure 1: $f(x_i) = \frac{cos(x_i)}{1+e^{3x_i}}$ for $N = 200$

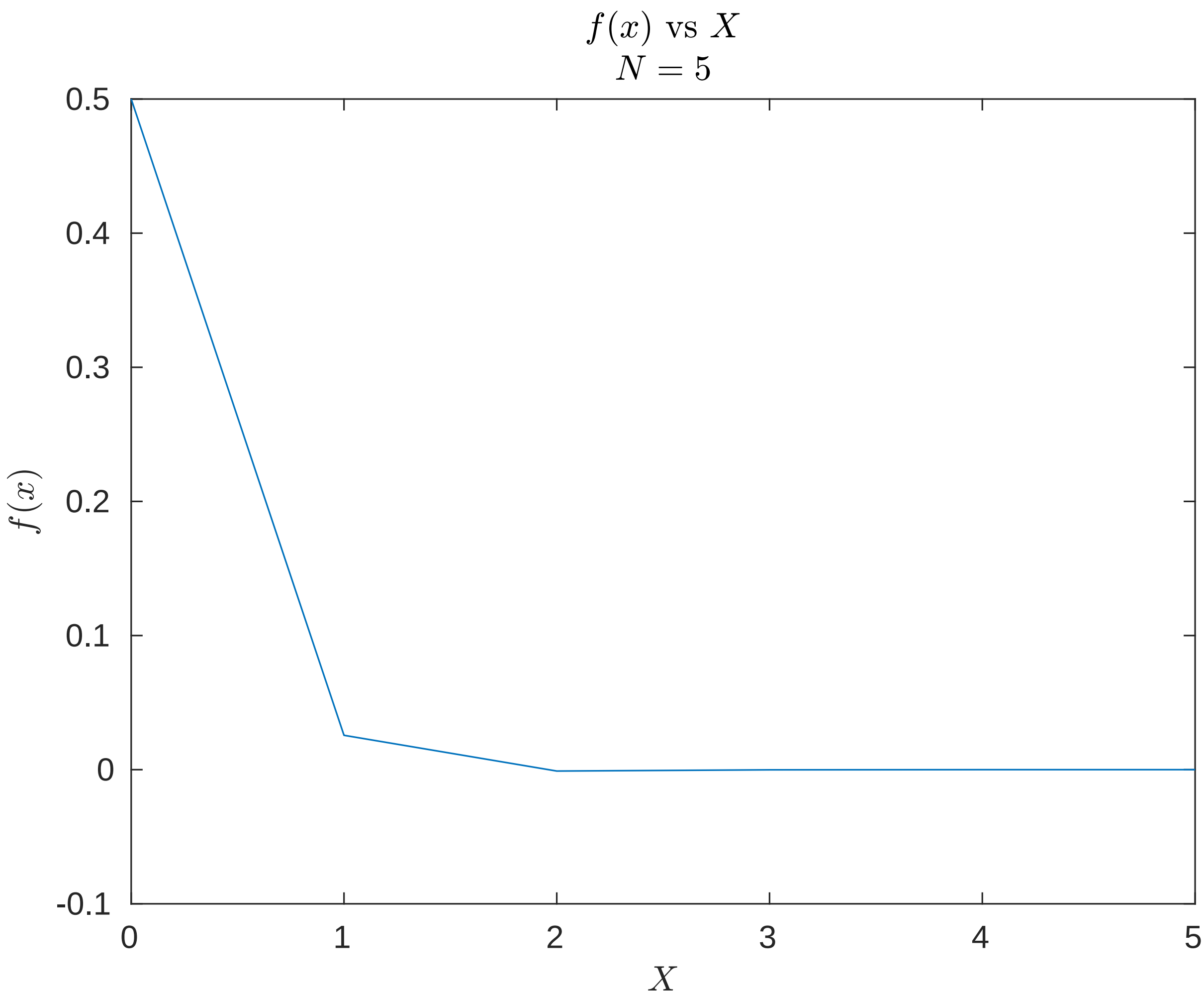

Figure 2: $f(x_i) = \frac{cos(x_i)}{1+e^{3x_i}}$ for $N = 5$

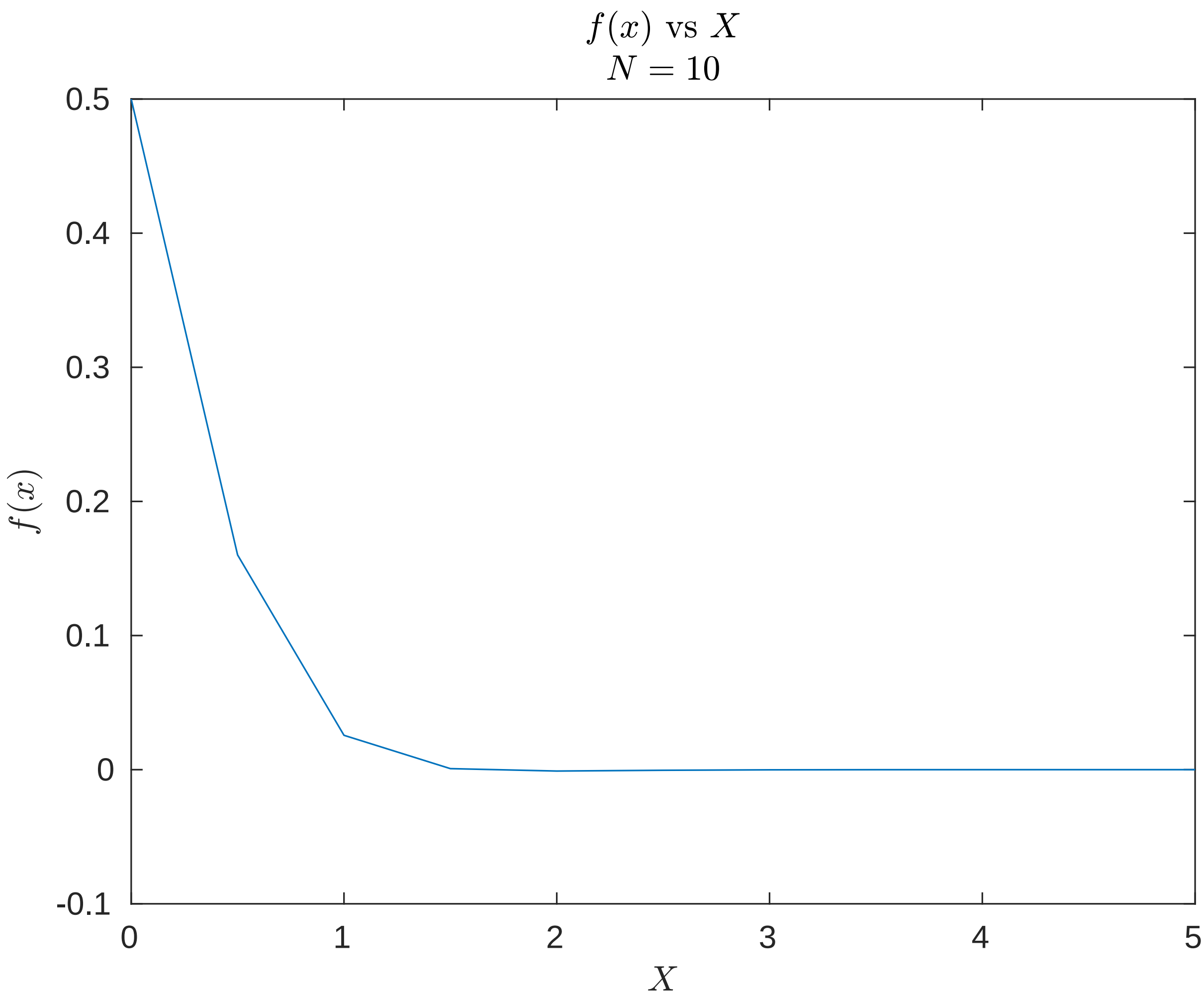

Figure 3: $f(x_i) = \frac{cos(x_i)}{1+e^{3x_i}}$ for $N = 10$

Enhanced resolution of the domain (N) yields an increase in the overall plot resolution. This relationship is visually evident in the juxtaposed figures. Figure 1 exhibits a high domain resolution, resulting in a smooth and continuous plot devoid of discernible edges. Figures 2 and 3 conversely, demonstrate the effects of a lower domain resolution, characterized by visually apparent edges and a less refined plot appearance.

5 Area under the Curve

5.1 Standard Integration

```
clear;
clc;
% Find the integral using quad
Q = quad(@fx, 0, 5); % Function, lower bound, upper bound
```

Listing 6: $\int_0^5 \frac{cos(x)}{1+e^{3x}} dx$

$$\int_0^5 \frac{cos(x)}{1+e^{3x}} dx \approx 0.201$$

5.2 Riemann Integral Approximation Equations

5.2.1 Right Rectangular Approximation

```
clear;
clc;
N = 25; % 5 recantangles for every number
dx = 5/N; % divide by the range
x_bar = dx:dx:5; % x domain for bar plot. Instead of starting at 0, right
    rectangular approximation starts i + 1, 0 + dx
y_bar = fx(x_bar); % range for bar plot
x_line = 0:1/1000:5; % Add a high resolution domain
y_line = fx(x_line); % range for scatter plot
bar(x_bar-dx/2, y_bar, 1, 'green'); % Creates the bar plot
hold on; % Add a second line to the plot
plot(x_line, y_line, 'red'); % Creates the scatter plot
```

```matlab
title('Right rectangular approximation $Q_{2}$','Interpreter','latex'); % Adds title to the plot. Adding $$ between the text, and adding 'Interperter', 'latex' to the matlab function, creates text with LaTeX formatting
xlabel('$X$','Interpreter','latex'); % Adds xlabel to the plot. Adding $$ between the text, and adding 'Interperter', 'latex' to the matlab function, creates text with LaTeX formatting
ylabel('$f(x)$','Interpreter','latex'); % Adds ylabel to the plot. Adding $$ between the text, and adding 'Interperter', 'latex' to the matlab function, creates text with LaTeX formatting
legend('$Q_{2}$', '$f(x)$','Interpreter','latex'); % Adds legend to the plot. Adding $$ between the text, and adding 'Interperter', 'latex' to the matlab function, creates text with LaTeX formatting
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, 'Fig/q2_bar_plot.pdf','ContentType','vector'); % Exports plot as a vector pdf image. (Requires R2020a or later)
```

Listing 7: $\int_0^5 \frac{cos(x)}{1+e^{3x}} dx$

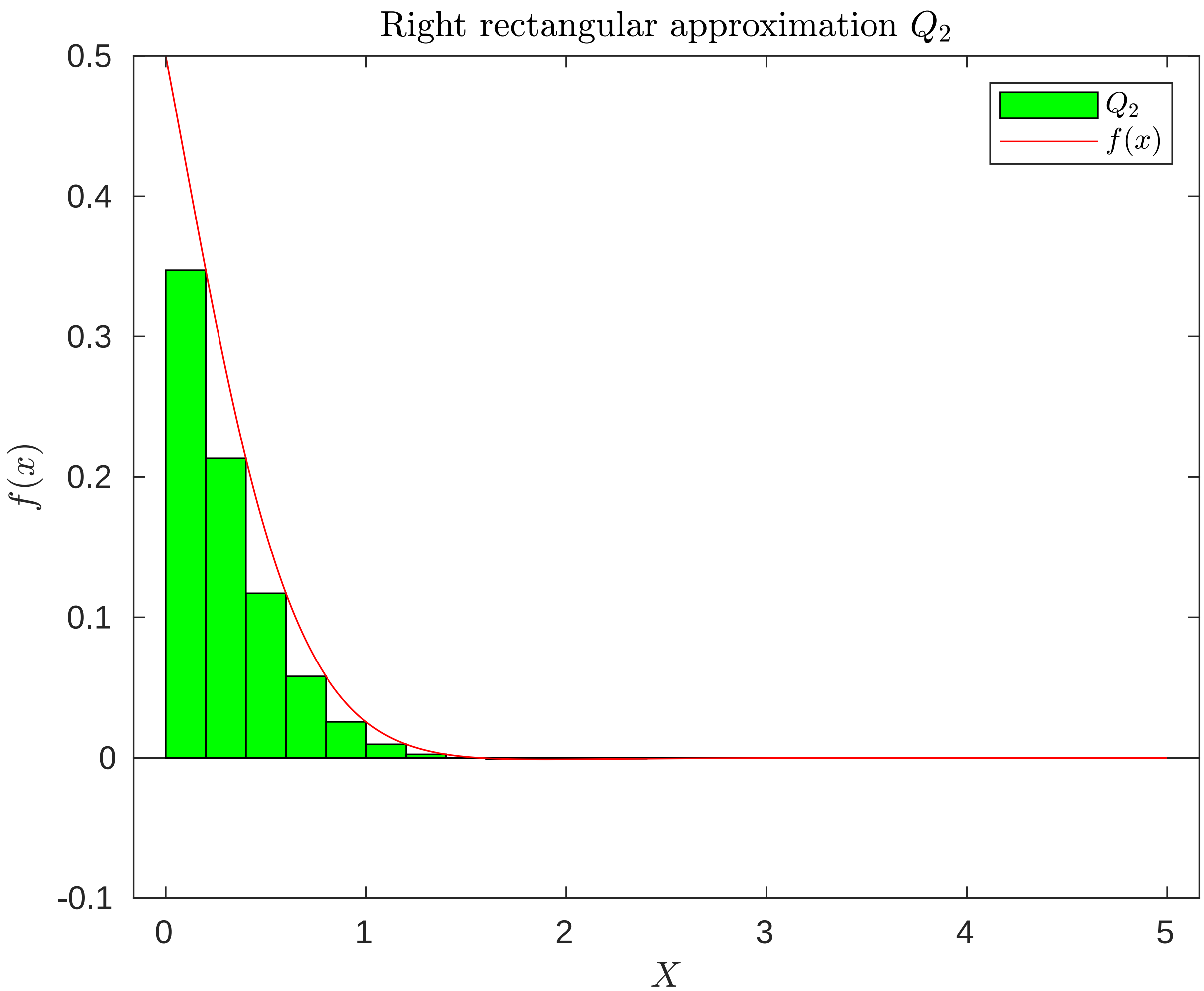

Figure 4: Right rectangular approximation for Q_2

5.2.2 Tradeoffs

Deceasing dx increases the resolution of the domain and the accuracy of the approximation. However, a higher resolution require more computational power. This is akin to higher resolution images requiring more space.

5.2.3 Trapazoidal approximation derivation

Given that the area under the first line segment can be approximated:

$$f(x_0)\,dx_0 + 0.5\big(f(x_1) - f(x_0)\big)\,dx_0$$
$$f(x_0)\,dx_0 + \frac{f(x_1)}{2}dx_0 - \frac{f(x_0)}{2}dx_0$$

This means that the next area can be approximated:

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

This can continue so forth until the N-th area can be approximated as:

$$f(x_{N-1})\,dx_{N-1} + \frac{f(x_N)}{2}dx_{N-1} - \frac{f(x_{N-1})}{2}dx_{N-1}$$

Summing these terms up yields the equation Q_3

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

5.2.4 $Q_3 = \frac{Q_1}{Q_2}$

Given that:

$$Q_1 = \sum_{i=1}^{N-1} f(x_i)\,dx_i$$

$$Q_2 = \sum_{i=2}^{N} f(x_i)\,dx_{i-1}$$
$$Q_3 = f(x_1)\,\frac{dx_1}{2} + \sum_{i=2}^{N-1} f(x_i)\,dx_i + f(x_N)\,\frac{dx_{N-1}}{2}$$

All dx_i are equal to each other. Substituting in Q_1 and Q_2 in to Q_3 gives:

$$Q_3 = \frac{1}{2}\left(\sum_{i=1}^{N-1} f(x_i)\,dx_i + \sum_{i=2}^{N} f(x_i)\,dx_{i-1}\right)$$
$$Q_3 = \frac{1}{2}\left(f(x_1)\,dx_1 + \sum_{i=2}^{N-1} f(x_i)\,dx_i + \sum_{i=2}^{N-1} f(x_i)\,dx_i + f(x_N)\,dx_{N-1}\right)$$
$$Q_3 = f(x_1)\frac{dx_1}{2} + \sum_{i=2}^{N-1} f(x_i)\,dx_i + f(x_N)\frac{dx_{N-1}}{2}$$

5.2.5 Q_1 and Q vs N

```
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
```

Listing 8: Q_1 Function

```
clear;
clc;
% N starts at 2 and ends at 200
N  = 2:1:200;
% Domain for Q1
Q_1N = zeros(1, length(N));
% Populate each ith value with the furutre q1 N calculation
for i = N
    Q_1N(i-1) = q1_sum(i);
end
% Create the domain for for the integral calcualtion
```

```matlab
Q_N = ones(1, length(N));
% Find the integral using quad
Q = quad(@fx, 0, 5); % Function, lower bound, upper bound
% Convert the integral value from a scalar value to line of a constant
% value
Q_N = Q_N*Q;
% Create the domain for for error calculation
Q_1error = zeros(1, length(N));
% Subtract the summation from the actual value to calculate the error
Q_1error = ((Q_1N - Q_N) / Q) * 100;
% Declare the figure
figure;
% Declare the subplot
% First subplot
subplot(2, 1, 1);
% Plot the approximation
plot(N, Q_1N, 'blue');
hold on;
% Plot the integral line
plot(N, Q_N, 'red');
% Add tilte
title('$Q_{1}$ and $Q$ vs $N$','Interpreter','latex');
% Add x axis label
xlabel('$N$' ,'Interpreter','latex');
% Add y axis label
ylabel('$Q_{1}$ and $Q$','Interpreter','latex');
```

```matlab
% Add legend
legend('$Q_{1}$', '$Q$', 'Interpreter','latex' );
% Second subplot
subplot(2, 1, 2);
% Plot the error line in a seprate subplot
plot(N, Q_1error, 'blue');
title('$Q_{1}$ Error vs $N$','Interpreter','latex');
% Add x axis label
xlabel('$N$' ,'Interpreter','latex');
% Add y axis label
ylabel('% Error','Interpreter','latex');
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, 'Fig/q1_sum_error_plot.pdf','ContentType','vector'); %
    Exports plot as a vector pdf image. (Requires R2020a or later)
```

Listing 9: Plot Q_1 and Q vs N

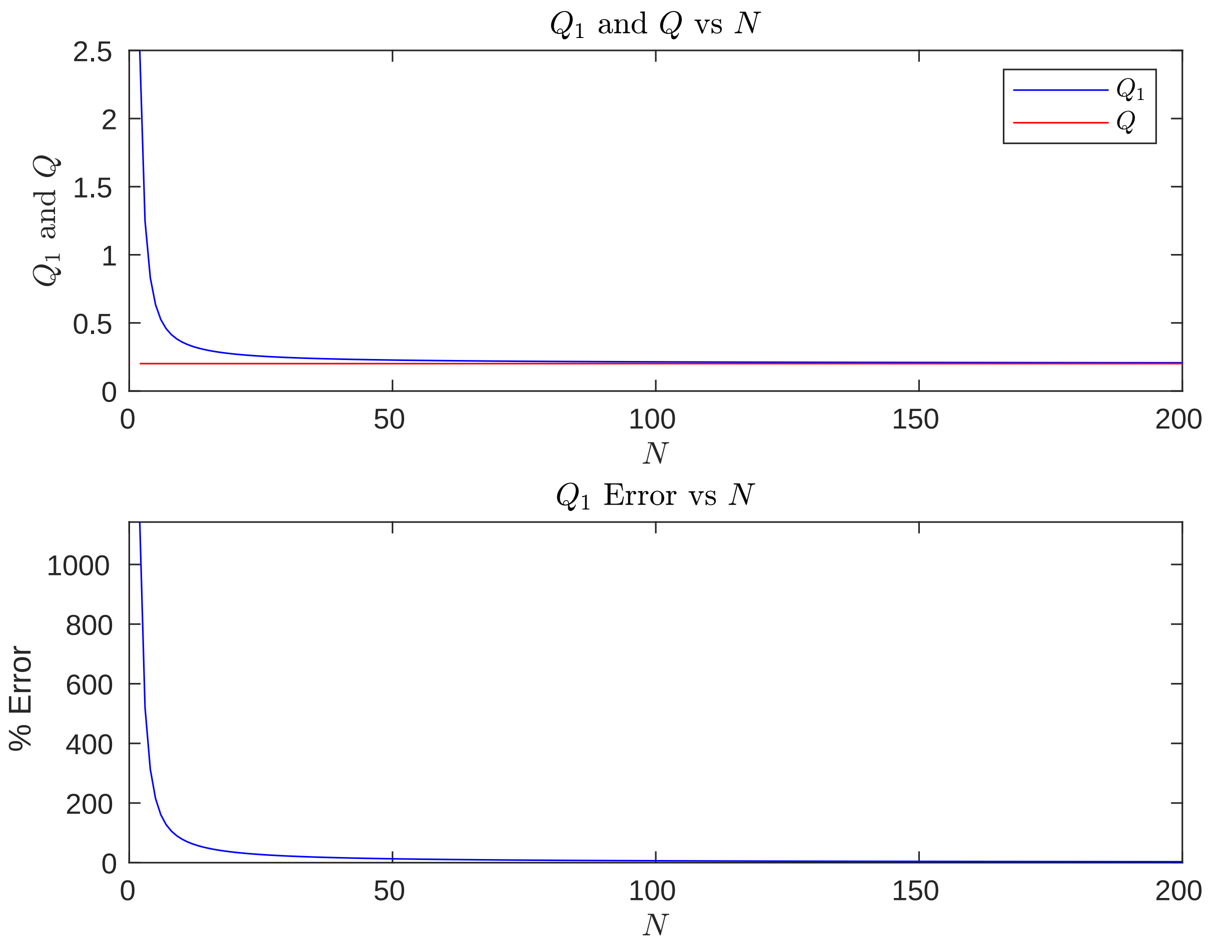

Figure 5: Q_1 and Q vs N and the % Error for Q_1

Q_1 in Figure 5 converges to Q at around $N/=/75$

5.2.6 Q_2 and Q vs N

```
function Q_2 = q2_sum(N)
dx = (5-0)/(N);
x = dx:dx:5;
y = fx(x);
A = [ones(N-1, 1); 0];
Q_2 = y*A*dx;
end
```

Listing 10: Q_2 Function

```
clear;
clc;
% N starts at 2 and ends at 200
N  = 2:1:200;
% Domain for Q1
Q_2N = zeros(1, length(N));
% Populate each ith value with the furutre q1 N calculation
for i = N
    Q_2N(i-1) = q2_sum(i);
end
% Create the domain for for the integral calcualtion
Q_N = ones(1, length(N));
% Find the integral using quad
Q = quad(@fx, 0, 5); % Function, lower bound, upper bound
% Convert the integral value from a scalar value to line of a constant
% value
Q_N = Q_N*Q;
% Create the domain for for error calculation
Q_2error = zeros(1, length(N));
% Subtract the summation from the actual value to calculate the error
Q_2error = ((Q_2N - Q_N) / Q) * 100;
% Declare the figure
figure;
% Declare the subplot
```

```
% First subplot
subplot(2, 1, 1);
% Plot the approximation
plot(N, Q_2N, 'blue');
hold on;
% Plot the integral line
plot(N, Q_N, 'red');
% Add tilte
title('$Q_{2}$ and $Q$ vs $N$','Interpreter','latex');
% Add x axis label
xlabel('$N$' ,'Interpreter','latex');
% Add y axis label
ylabel('$Q_{2}$ and $Q$','Interpreter','latex');
% Add legend
legend('$Q_{2}$', '$Q$', 'Interpreter','latex' );
% Second subplot
subplot(2, 1, 2);
% Plot the error line in a seprate subplot
plot(N, Q_2error, 'blue');
title('$Q_{2}$ Error vs $N$','Interpreter','latex');
% Add x axis label
xlabel('$N$' ,'Interpreter','latex');
% Add y axis label
ylabel('% Error','Interpreter','latex');
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, 'Fig/q2_sum_error_plot.pdf','ContentType','vector'); %
```

```
Exports plot as a vector pdf image. (Requires R2020a or later)
```

Listing 11: Plot Q_2 and Q vs N

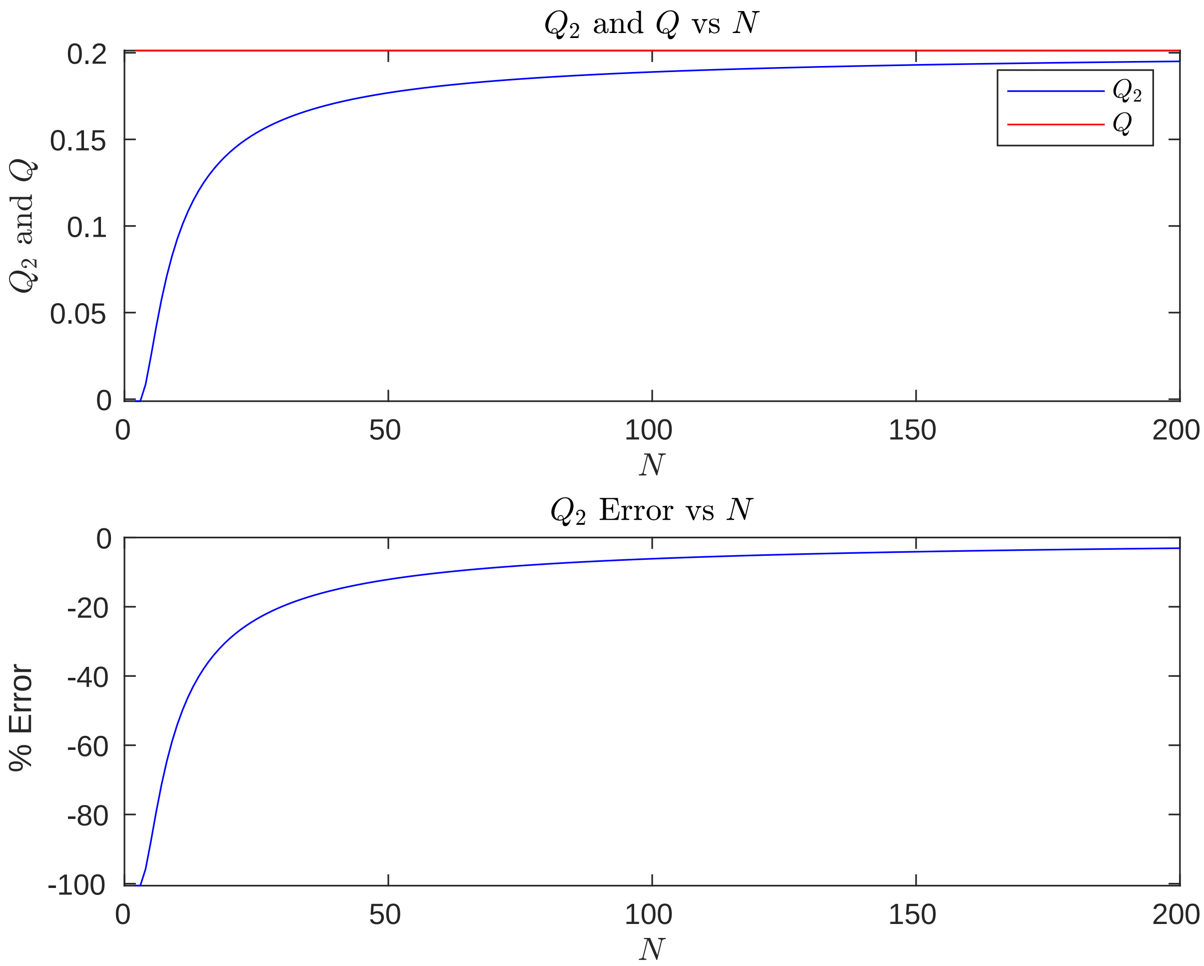

Figure 6: Q_2 and Q vs N and the % Error for Q_2

Q_2 in Figure 6 converges to Q at around $N/=/100$

6 Conclusion

This laboratory exercise showcases MATLAB's versatility across diverse scientific and engineering domains. It notably highlights the fundamental concept of trade-offs in engineering.

Riemann summation serves as a poignant example, empowering students to grapple with the delicate balance between accuracy and computational efficiency—a fundamental skill for effective engineering design.