EE_105_022_23W Lab 1

Avery Juwan Brillantes

TOTAL POINTS

95 / 100

QUESTION 1

1 Point 3 - Matrices and Arrays 15 / 15

✓ **+ 15 pts** *C is correctly calculated*

QUESTION 2

2 Point 5 - Scripts 20 / 20

✓ **+ 20 pts** *D is correctly calculated using a for loop*

QUESTION 3

3 Point 6 - More Advanced Scripts 25 / 25

✓ **+ 5 pts** *1st script is correct*

✓ **+ 5 pts** *2nd script is correct*

✓ **+ 15 pts** *Correct plot*

QUESTION 4

4 Point 7 - Integral Approximation 35 / 40

✓ **+ 5 pts** *7b.i*

✓ **+ 5 pts** *7.b.ii*

✓ **+ 2.5 pts** *7.b.iii.A*

✓ **+ 2.5 pts** *7b.iii.B*

✓ **+ 10 pts** *7c*

✓ **+ 10 pts** *7d*

7.a asks you to use integral or quad function to calculate Q not your own.

1 Point 3 - Matrices and Arrays **15 / 15**

✓ **+ 15 pts** *C is correctly calculated*

2 Point 5 - Scripts **20 / 20**

✓ **+ 20 pts** *D is correctly calculated using a for loop*

3 Point 6 - More Advanced Scripts 25 / 25

✓ **+ 5 pts** *1st script is correct*

✓ **+ 5 pts** *2nd script is correct*

✓ **+ 15 pts** *Correct plot*

4 Point 7 - Integral Approximation 35 / 40

✓ **+ 5 pts** *7b.i*

✓ **+ 5 pts** *7.b.ii*

✓ **+ 2.5 pts** *7.b.iii.A*

✓ **+ 2.5 pts** *7b.iii.B*

✓ **+ 10 pts** *7c*

✓ **+ 10 pts** *7d*

7.a asks you to use integral or quad function to calculate Q not your own.

EE 105 Lab Report 1

Avery Juwan T. Brillantes

3. Matrices and Arrays

Code:

```
A = [pi; sqrt(2); exp(1)]
B = [1; 5; 7]
C = A'*B   % notice that we use the transpose function to make multiply a 1x3
matrix and a 3x1 matrix
```

Output:

```
C =

   29.2406
```

We notice that we get a scalar value since we used “*” instead of “.*“, This means that the answer will be a summation of the A(i)*B(i) which results in a scalar value.

5. Scripts

Code:

```
help clear
clear all % clears the memory
A = [pi; sqrt(2); exp(1)]
B = [1; 5; 7]
D = 0
for i = 1:3  % this code acts the same way as A’*B
  D = A(i)*B(i) + D
end
```

Output:

```
D =

    3.1416


D =

   10.2127


D =

   29.2406
```

We notice that through the use of for loops we can achieve the same result however we notice that it can be a little longer and more complicated than just “A’ * B”

6. More advanced Scripts

i) first m-file

Code:

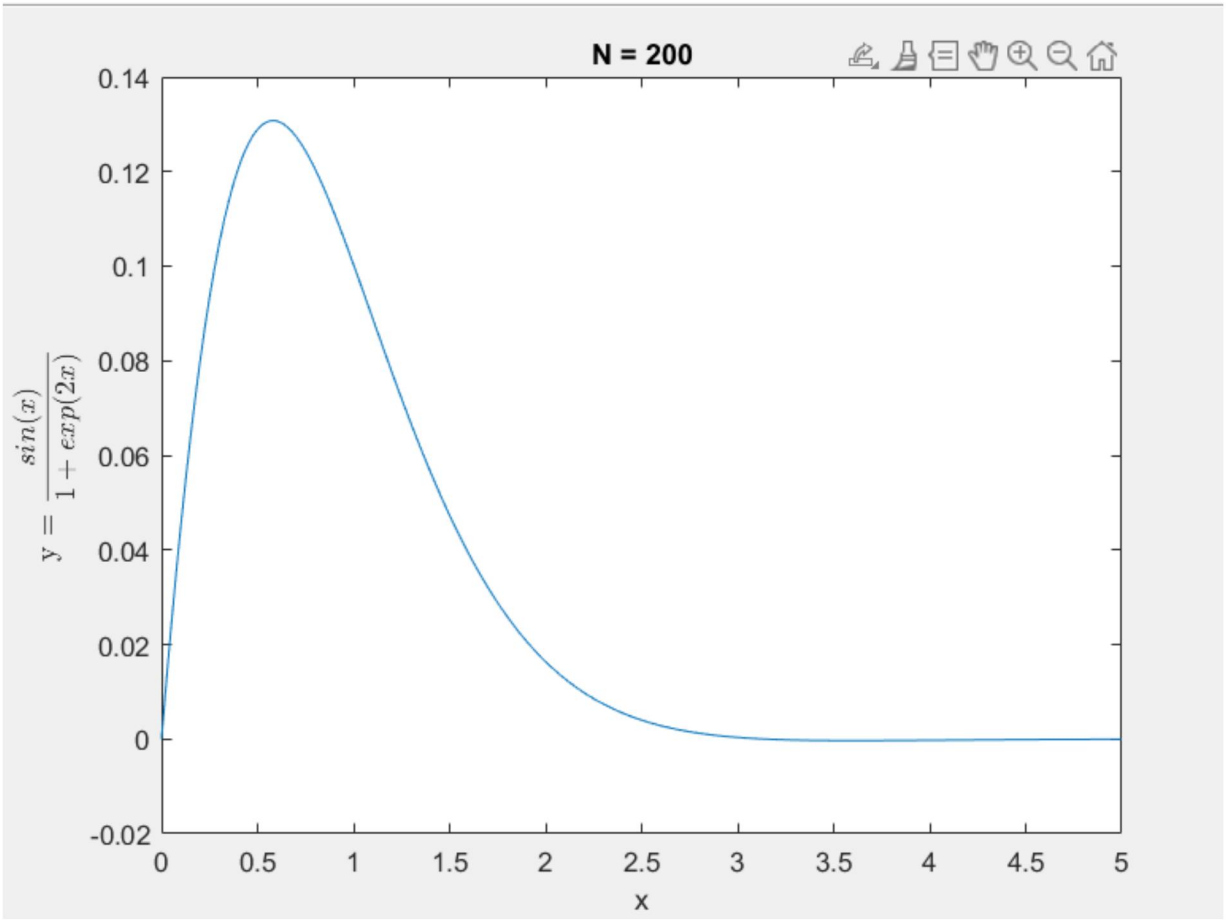
```
function [y] = Myfunc(x)
   y = sin(x) ./ (1+exp(2.*x));  % function given
end
```

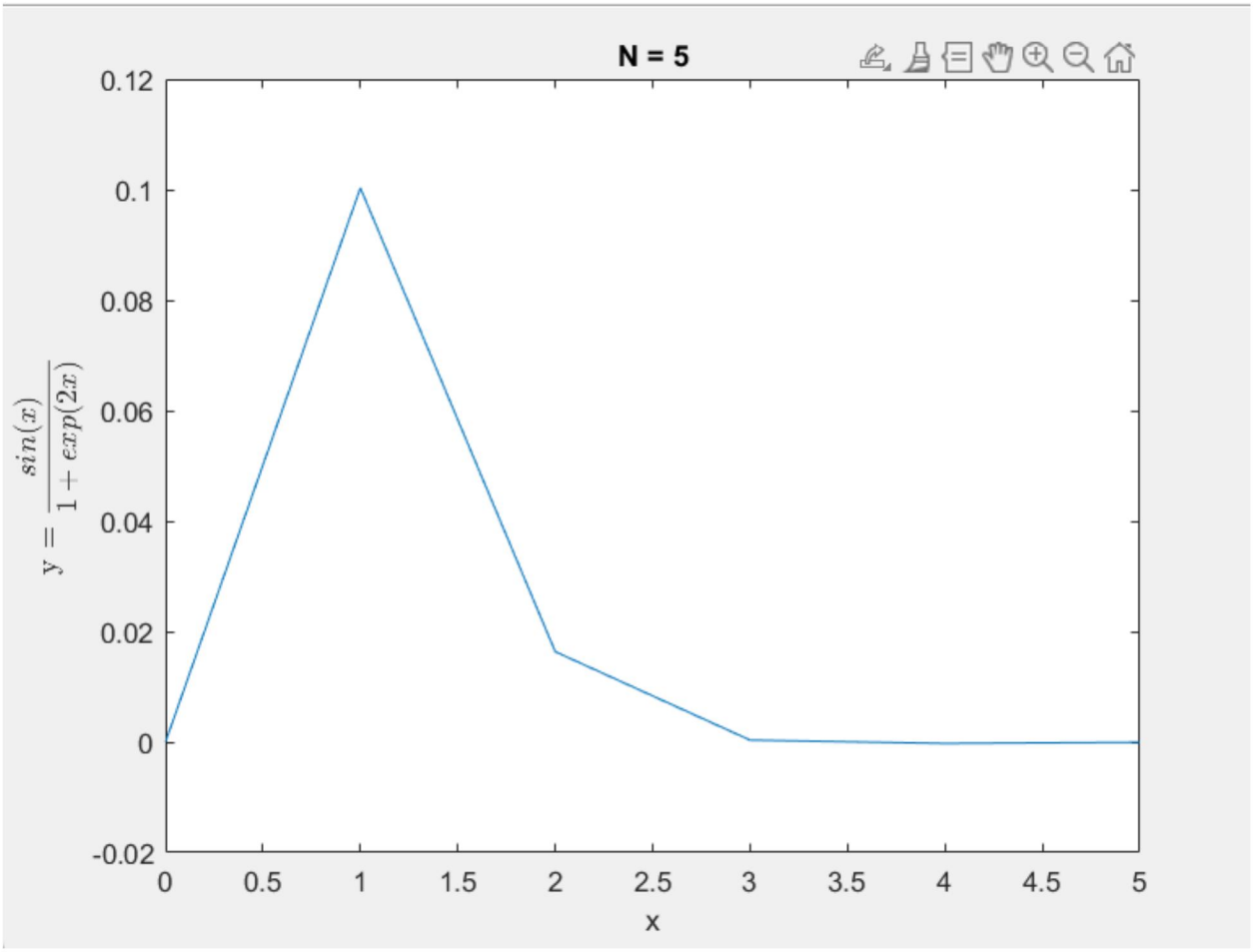
ii) second m-file

Code:

```
function part6plot(N)
   x = linspace(0,5,(N + 1));  % this makes a matrix from 1 to N + 1
   y = Myfunc(x);              % this makes y equal to the function given

   figure
   plot(x,y);  % plots the graph
   xlabel('x'); % labels x-axis
   ylabel('y = $$\frac{sin(x)} {1+exp(2x)}$$','Interpreter','latex') % labels
y-axis
   title(['N = ', num2str(N)]) % shows N as the title of the graph
end
```

Output:





We see that the amount of N determines the amount of accuracy of the graph. We see that with higher N values the function is more of a curve and closer to the actual function than it would be with a lower N value.

7.

a) Code:

```
function antider7a(N)
   x = linspace(0,5,(N + 1));
   y = @Myfunc  % note must use a function handle '@' to be used in integral
   integral(y,0,5) % integral of y between 0 and 5
end
```

Output:

```
ans =

    0.1587
```

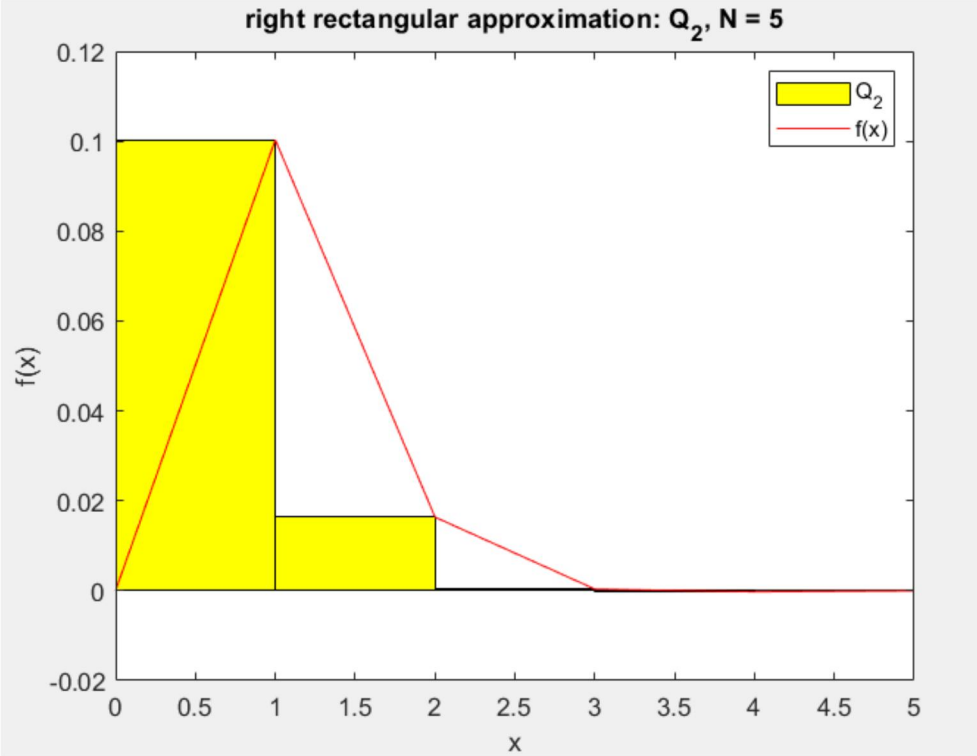
Here we can note that no matter the value of N the integral value will remain the same.

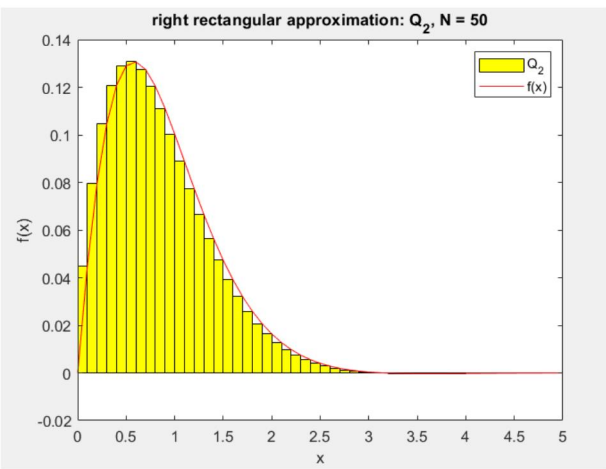
b)

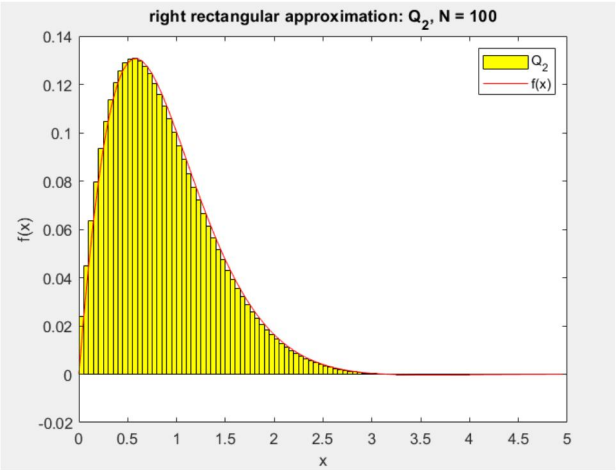
i) Code:

```
function part7Q2
    N = 100;
    dx = 5/N;
    % f(x) bar graph
    x_bar = 0:dx:4;
    y_bar = Myfunc(x_bar);
    % f(x) line graph
    x = 0:dx:5;
    y = Myfunc(x);

    figure
    % plot f(x) bar
    bar(x_bar - (dx/2),y_bar,1,'yellow') % (dx/2) shifts the bar to the left
    xlim([0,5])
    hold on % puts the bar graph on top of the line graph
    % plot f(x) line
    plot(x,y,'red')
    title(['right rectangular approximation: Q_2, N = ', num2str(N)])
    xlabel('x')
    ylabel('f(x)')
    legend('Q_2','f(x)')
end
```

Output:



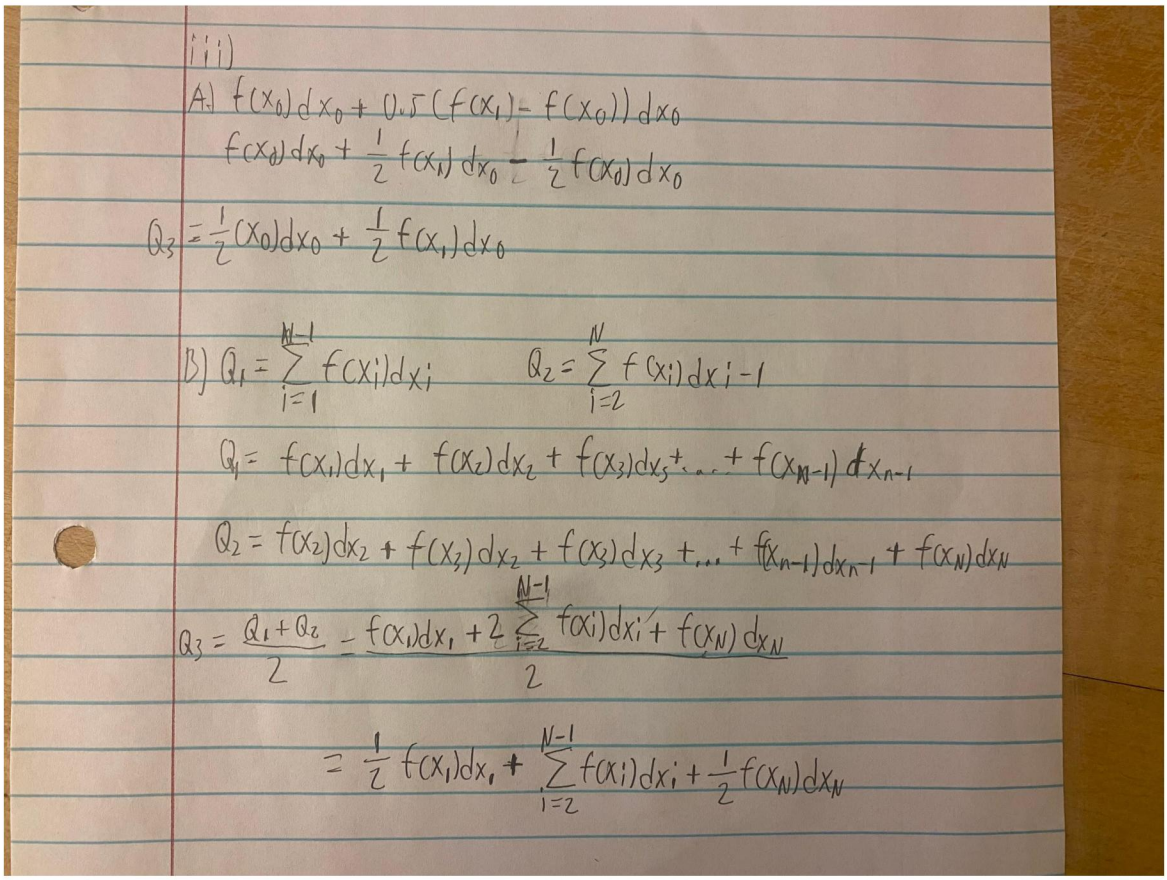




The accuracy is enhanced by decreasing dx because it determines the distance between each rectangle. This means the smaller it gets the more accurate the rectangular approximation graph becomes.

ii) the tradeoff to decreasing dx is that making it smaller by increasing N will result in more computations which means the smaller dx gets the slower the computer runs.

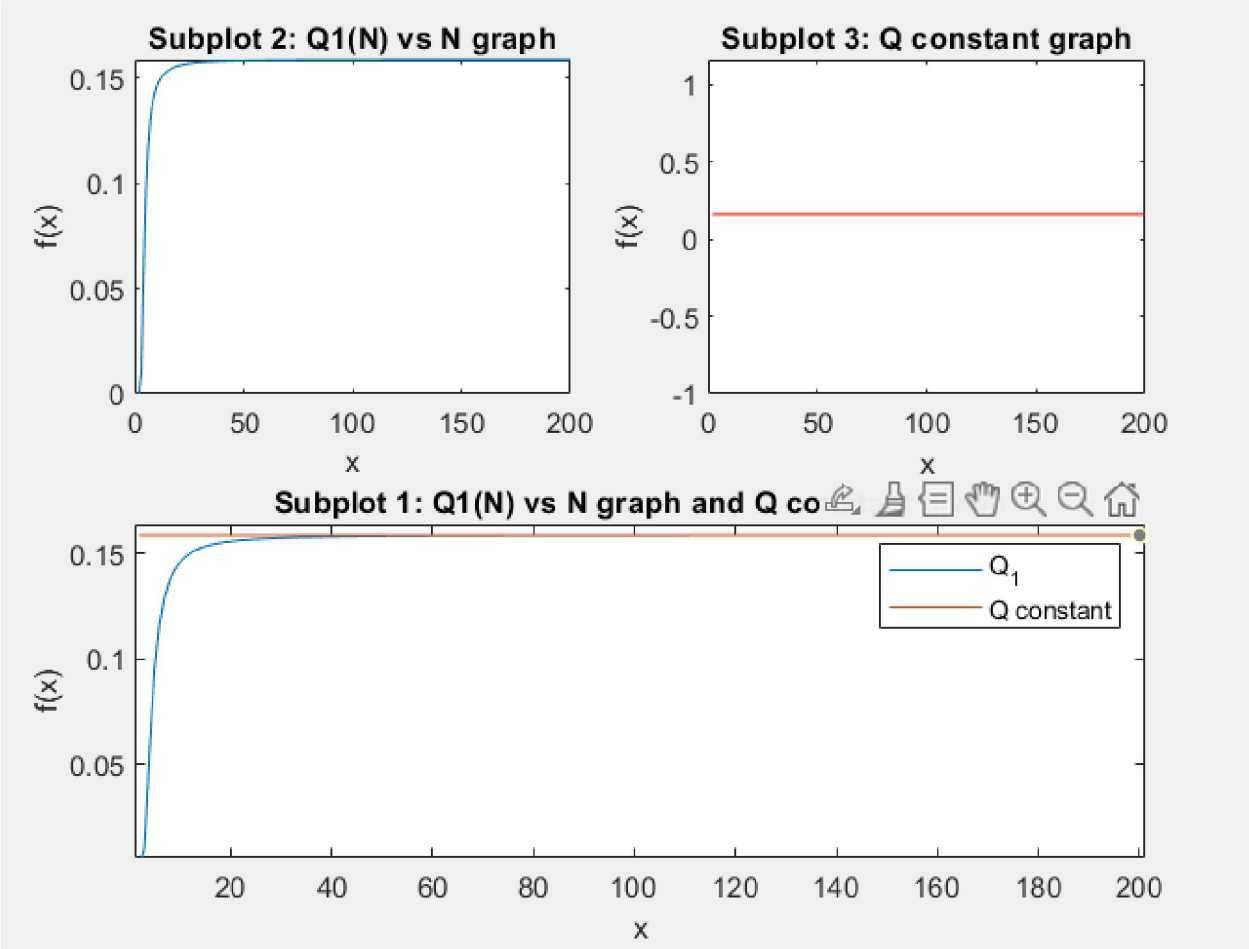
iii)



c) Code:

```
function part7Q1approx
   N = 2:200;
   Q1 = NaN(length(N), 1);
   for i = 1:length(N)
       dx = (5-0) / (N(i)-1);
       x = 0:dx:5;
       y = Myfunc(x);
       A = [ones(N(i)-1,1); 0];
       Q1(i) = y * A * dx;
   end
   figure
   subplot(2,1,2)
   plot(N,Q1) % Q1(N) vs N graph
   hold on
   Q = 0.1587 % integral answer from 7a
   QQ = ones(1,length(N)) * Q
   plot(N,QQ) % Q constant graph
   xlabel('x')
   ylabel('f(x)')
   legend('Q_1','Q constant')
   title('Subplot 1: Q1(N) vs N graph and Q constant graph')
   subplot(2,2,1)
   plot(N,Q1) % Q1(N) vs N graph
   xlabel('x')
   ylabel('Q_1')
   title('Subplot 2: Q1(N) vs N graph')

   subplot(2,2,2)
   plot(N,QQ) % Q constant graph
   xlabel('x')
   ylabel('Q constant')
   title('Subplot 3: Q constant graph')
end
```

Output:



Here we see that the graph of Q1 approaches the Q constant as it goes to infinity.

d.
Code:

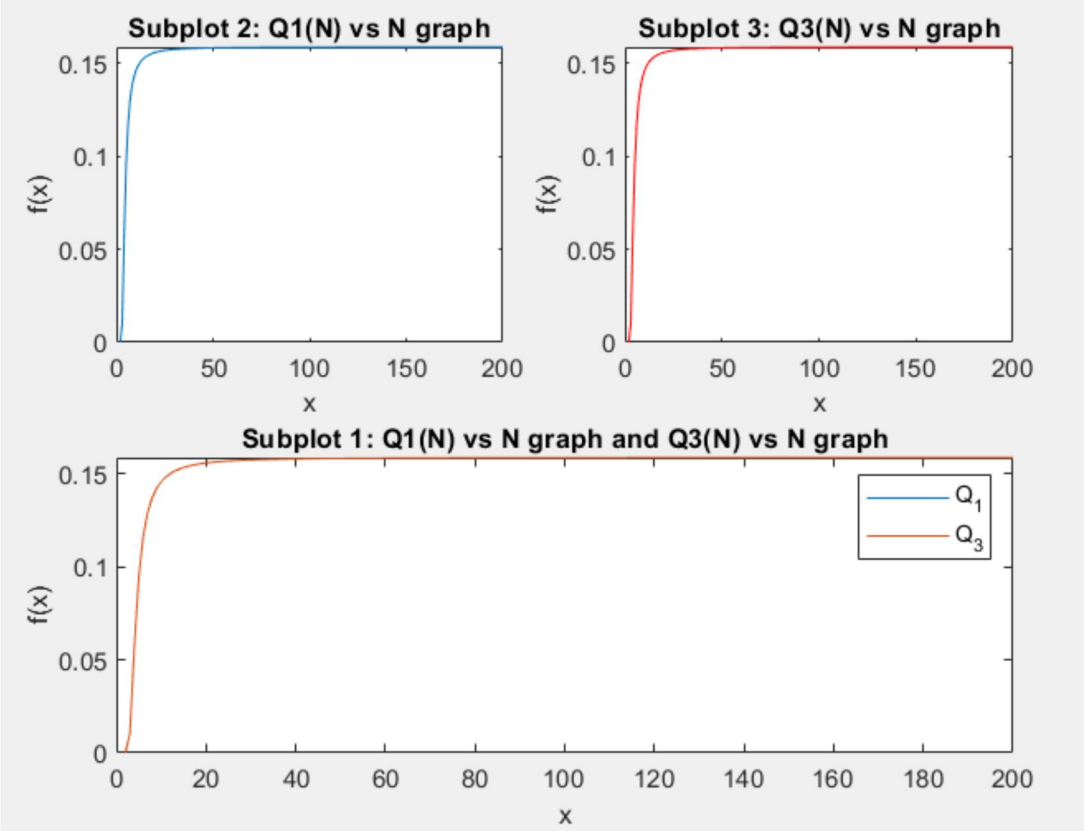
```
function part7d
   N = 2:200;
   Q1 = NaN(length(N), 1);

   for i = 1:length(N)
       dx = (5-0) / (N(i)-1);
       x = 0:dx:5;
       y = Myfunc(x);
       A = [ones(N(i)-1,1); 0];
       Q1(i) = y * A * dx;
   end
   for i = 1:length(N)
       dx = (5-0) / (N(i)-1);
       x = 0:dx:5;
       y = Myfunc(x);
```

```
    Q1A = [ones(N(i)-1,1); 0;];
    Q2A = [ones(N(i)-1,1); 1;];
    Q3A = (Q1A + Q2A)/2;  % note this equation is given
    Q3(i) = y * Q3A * dx;
end

figure
subplot(2,1,2)
plot(N,Q1) % Q1(N) vs N graph
hold on
plot(N,Q3) % Q1(N) vs N graph
xlabel('x')
ylabel('f(x)')
legend('Q_1','Q_3')
title('Subplot 1: Q1(N) vs N graph and Q3(N) vs N graph')
subplot(2,2,1)
plot(N,Q1) % Q1(N) vs N graph
xlabel('x')
ylabel('f(x)')
title('Subplot 2: Q1(N) vs N graph')

subplot(2,2,2)
plot(N,Q3,'red') % Q3(N) vs N graph
xlabel('x')
ylabel('f(x)')
title('Subplot 3: Q3(N) vs N graph')

%------------------Error------------------%
Q = 0.1587; % integral answer from 7a
error_Q1 = Q1 - Q;
error_Q3 = Q3 - Q;
figure
subplot(2,1,2)
plot(N,error_Q1) % error_Q1(N) vs N graph
hold on
plot(N,error_Q3) % error_Q3(N) vs N graph
xlabel('x')
ylabel('f(x)')
legend('Error Q_1','Error Q_3')
title('Subplot 1: Error Q1(N) vs N graph and Error Q3(N) vs N graph')
subplot(2,2,1)
plot(N,error_Q1) % error_Q1(N) vs N graph
xlabel('x')
ylabel('f(x)')
title('Subplot 2: Error Q1(N) vs N graph')

subplot(2,2,2)
plot(N,error_Q3,'red') % error_Q3(N) vs N graph
xlabel('x')
```

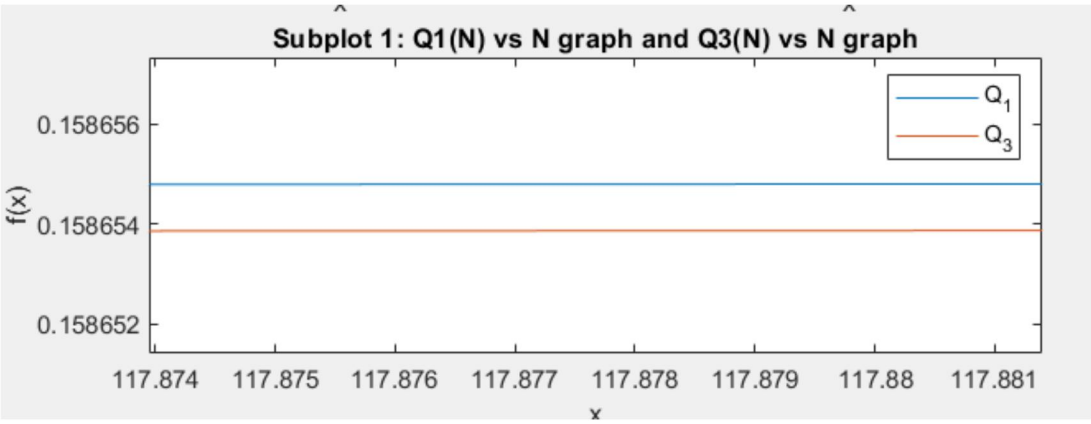
```
    ylabel('f(x)')
    title('Subplot 3: Error Q3(N) vs N graph')
    %-------------Rate of Convergent-------------%
    rate_Q1 = (Q1(end)-Q) / (Q1(end-1)-Q)
    rate_Q3 = (Q3(end)-Q) / (Q3(end-1)-Q)
end
```

Output:

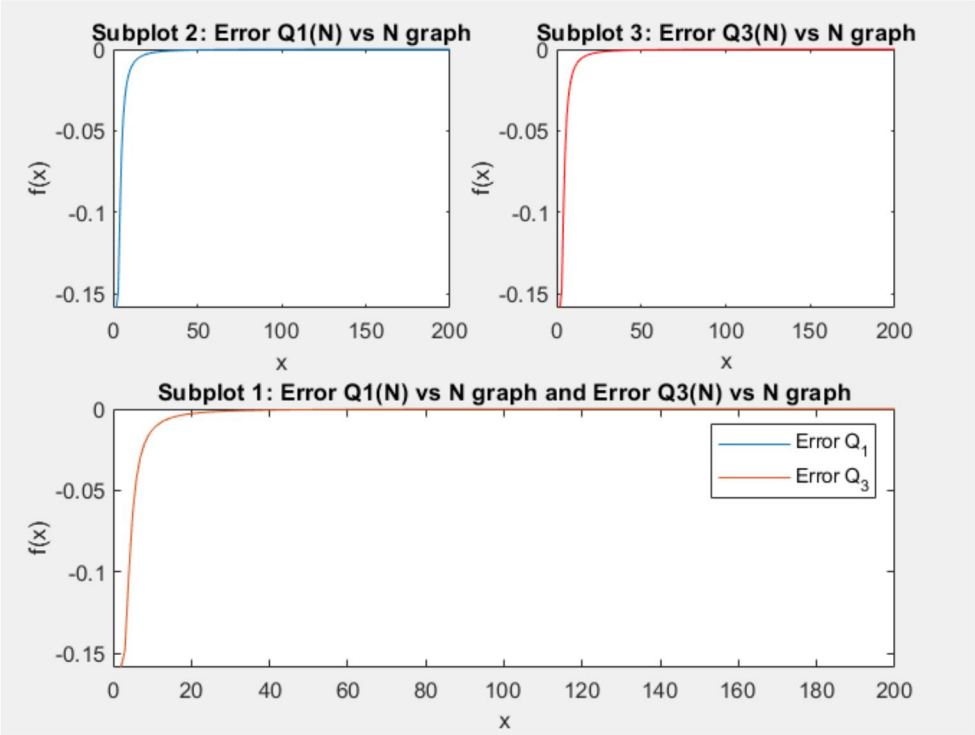
Graph 1



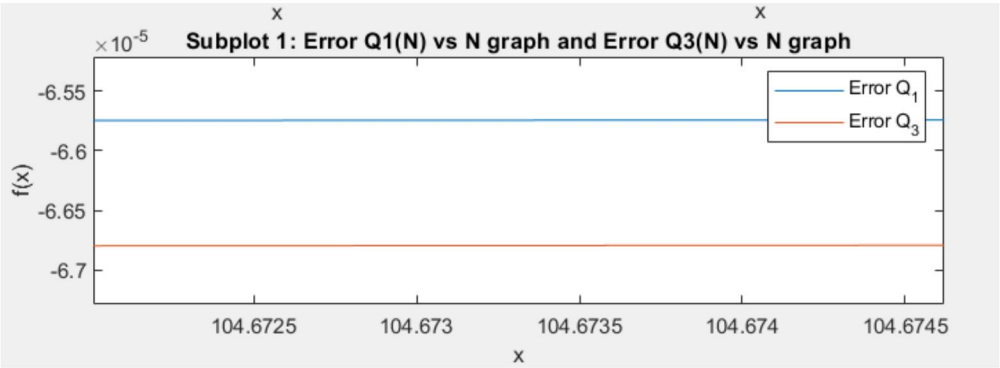
Zoomed in



Graph 2



Zoomed in



Rate of Convergence

```
rate_Q1 =

    1.0643


rate_Q3 =

    1.0751
```

Using graph 1, we can see that Q1 and Q3 have very similar shapes and both values are very close to one another

Using graph 2 we can see that the error for both Q1 and Q3 are very small and both values are almost the same
The rate of Convergence for both Q1 and Q3 are very close to one another. Also note that RateQ1 < RateQ3
This means that Q3 would be the better algorithm since it would require fewer iterations before yielding a useful approximations, however both values are very much the same and it might not be worth the extra computational power to use Q3

8. Read the article
1st Code:

```
function part8code1
   clear x
   tic
   for i = 1:1000000
       x(i) = 1;
   end
   toc
end
```

Output:

```
Elapsed time is 0.135804 seconds.
```

2nd Code:

```
function part8code2
clear x
   x = zeros(1,1000000);
   tic
   for i = 1:1000000
       x(i) = 1;
   end
   toc
end
```

Output:

```
Elapsed time is 0.008387 seconds.
```

3rd Code:

```
function part8code3
   clear x
   tic
   x(1:1000000) = 1;
   toc
end
```

Output:

```
Elapsed time is 0.003084 seconds.
```

4th Code:

```
function part8code4
   clear x
   tic
   x = ones(1,1000000);
   toc
end
```

Output:

```
Elapsed time is 0.001919 seconds.
```

Comparing the 4 code segments we can see that the 4th code segment is the fastest and this means that using matlab functions will always be faster than for loops and other code that doesn't use matlab functions