# Signals and systems Lab 03

Manqad Raza
Aimen Asim

## Task a

$A=1$, $f_0=1$

$$y(t) = y(0) + \int_0^t x(\lambda)\,d\lambda$$

$$\int_0^t x(\lambda)\,d\lambda = \int_0^t A\sin(2\pi f_0 t)\,dt$$

$$\int_0^t \sin(2\pi t)\,dt$$

$$= \left.\frac{-\cos(2\pi t)}{2\pi}\right|_0^t$$

$$= \left(\frac{-\cos(2\pi t)}{2\pi}\right) - \left(-\frac{\cos(2\pi(0))}{2\pi}\right)$$

$$= \frac{-\cos(2\pi t)}{2\pi} + \frac{1}{2\pi}$$

$$= \frac{1}{2\pi} - \frac{\cos(2\pi t)}{2\pi}$$

$$= \frac{1-\cos(2\pi t)}{2\pi}$$

$$= \frac{1-(\cos^2(\pi t) - \sin^2(\pi t))}{2\pi}$$

$$= \frac{1-(1-2\sin^2(\pi t))}{2\pi}$$

$$= \frac{2\sin^2(\pi t)}{2\pi}$$

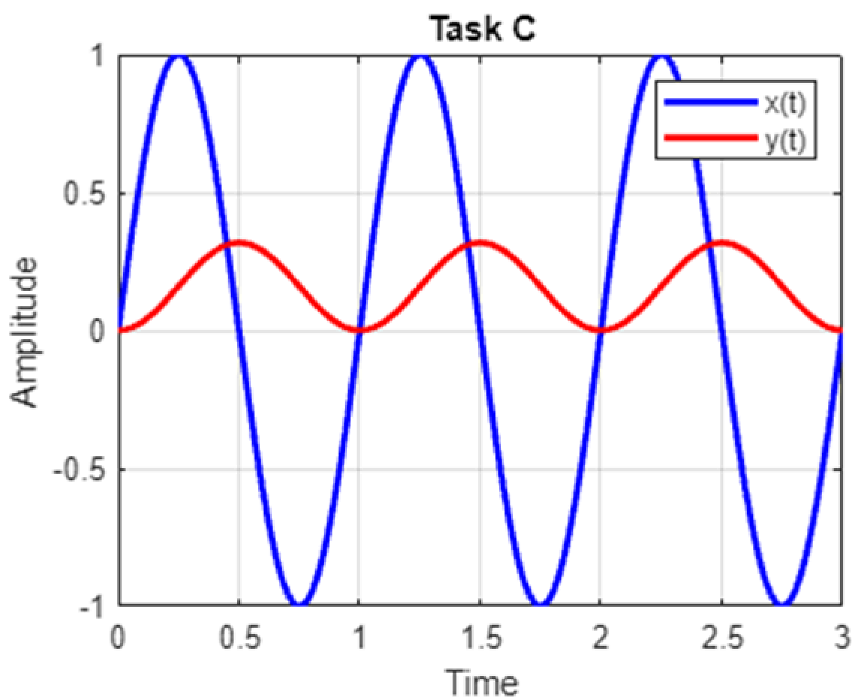
$$= \frac{\sin^2(\pi t)}{\pi}.$$

## ***Task b***

The value of yt obtained is the same as calculated in part(a). This explains the use of the int function and how we can integrate a function and set parameters in that as well

## ***Task c:***

```
clc

clear

T = 0:0.01:3;

xt = sin(2*pi*T);

yt = (sin(pi*T).^2)/pi;

figure;

plot(T, xt, 'b-', 'LineWidth', 2);

grid on;

hold on;

plot(T, yt, 'r-', 'LineWidth', 2);

title('Task C');

xlabel('Time');

ylabel('Amplitude');

legend('x(t)', 'y(t)');

hold off;
```


Task C
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
2.5
3
Time
Amplitude
x(t)
y(t)

## ***Task d:***

```
clc;
clear all;
close all;

% Creating a vector
a = [1 2 3 4 5]
```

```
a = 1×5
     1     2     3     4     5
```

```
% Calculating the cumsum of a
b = cumsum(a)
```

```
b = 1×5
     1     3     6    10    15
```

## ***Observation:***

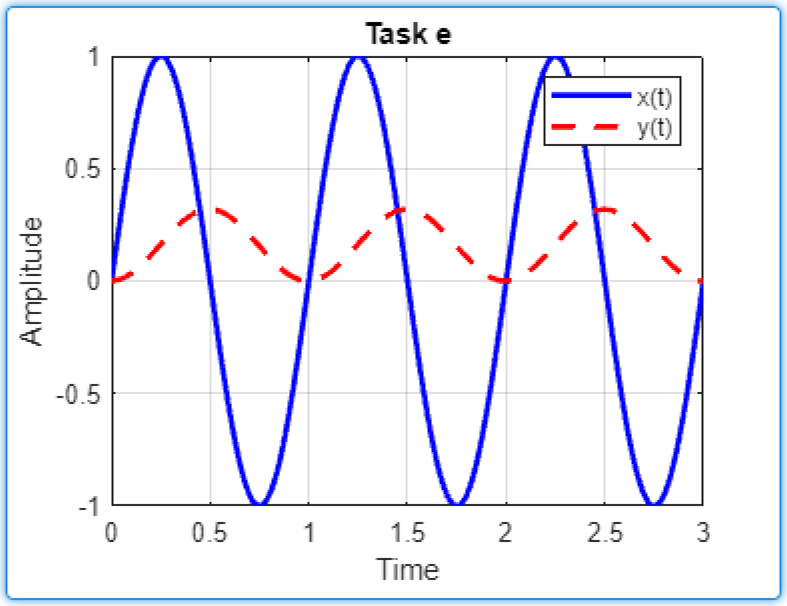
It adds the individual members of the array, one by one, giving the cumulative sums.

## ***Task e:***

**Code:**

```
clc

A = 1;

f0 = 1;
```

```
dt = 0.01;

t = 0:dt:3;

xt = A*sin(2*pi*f0*t);

yt = cumsum(xt) * dt;

figure;

plot(t, xt, 'b-', 'LineWidth', 2);

hold on;

plot(t, yt, 'r--', 'LineWidth', 2);

grid on;

xlabel('Time');

ylabel('Amplitude');

title('Plot of x(t) and y(t)');

legend('x(t)','y(t)');

hold off;
```




## ***Task f:***

Utilizing the built-in cumsum operation for numerical integration gives the same result as symbolic integration using the built-in int function. This conclusion is supported by the identical graphs obtained in part C, derived from expressions in parts A and B, and part E

## ***Task g:***

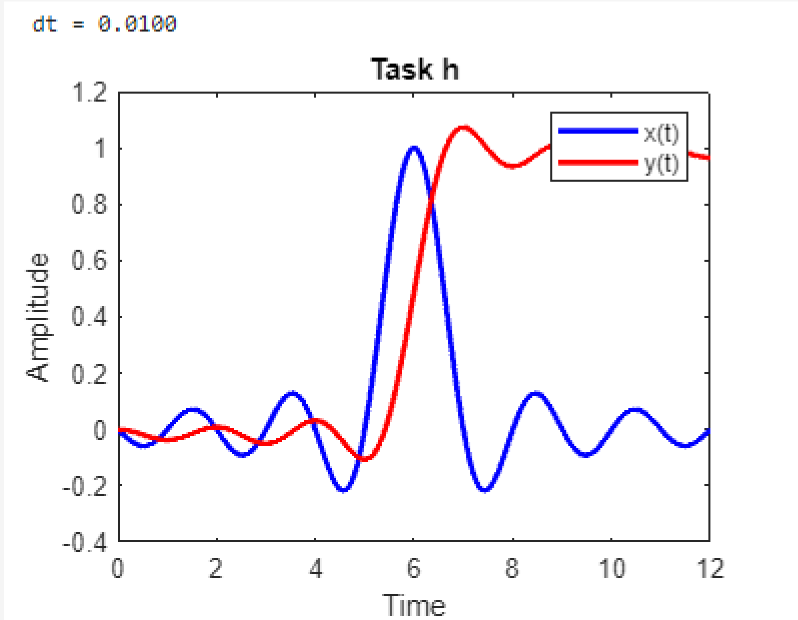
A key advantage is evident when working with real-world data, like sensor inputs. As these inputs are discrete and may lack a precise mathematical function, numerical integration allows for direct processing and analysis, making it highly practical for real-world applications

## ***Task H***:

clc

clear

```
dt=0.01

t=0:dt:12;

xt=sinc(t-6);

yt = cumsum(xt) * dt;

figure;

plot(t, xt, 'b-', 'LineWidth', 2)

hold on;

plot(t, yt, 'r-', 'LineWidth', 2)

xlabel('Time');

ylabel('Amplitude');

title('Task h');

legend('x(t)','y(t)');

hold off;
```




## ***Task i:***

```
b =

     1     1     1     1
```

calculates differences between adjacent elements of X along the first array dimension whose size does not equal 1: If X is a vector of length m , then Y = diff(X) returns a vector of length m-1 . The elements of Y are the differences between adjacent elements of X .

***Task j:***

$$x(t) = \frac{d(y(t))}{dt}, \text{ where} : 2te^{-t}$$

$$x(t) = \frac{d}{dt}(2te^{-t}) \qquad u = 2t \quad u' = 2$$

$$v = e^{-t} \quad v' = -e^{-t}$$

$$x(t) = vu' + uv'$$

$$x(t) = \cancel{2t}\ 2e^{-t} - 2te^{-t}$$

$$x(t) = 2e^{-t}(1-t)$$

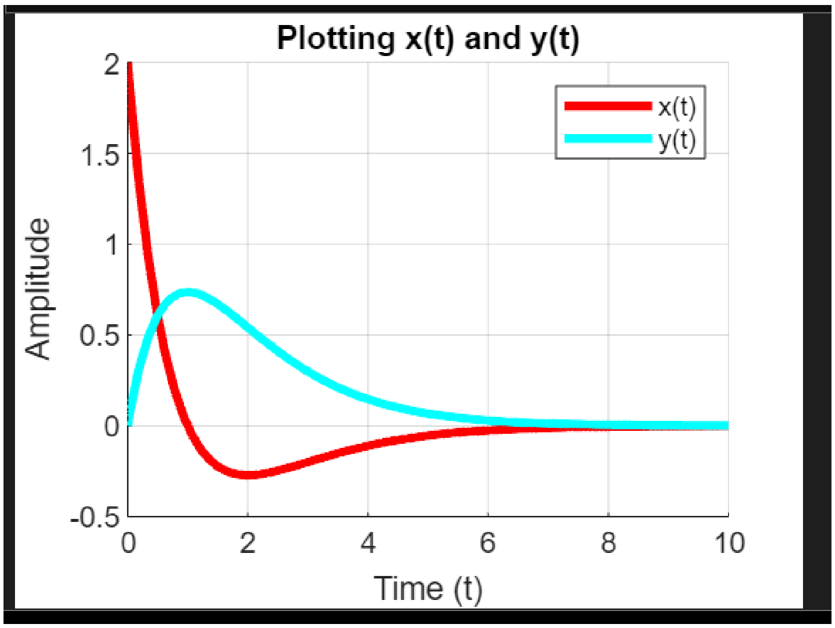
***Task k:***

## Code:

```
dt = 0.01;

t = 0: dt: 10;

xt = (2 * exp(-t)) .* (1-t);

yt = 2 * (t .* exp(-t));

figure; hold on; grid on;

plot(t, xt,'Color','Red', 'linewidth', 3)

plot(t, yt,'Color', 'Cyan', 'linewidth', 3)

title('Plotting x(t) and y(t)');
```

```
xlabel('Time (t)');

ylabel('Amplitude');

legend('x(t)', 'y(t)');

hold off
```

## Graph:




***Observation***:

The MATLAB code plots two functions, x(t) and y(t), over the time interval [0, 10] with a time step of 0.01. The functions are defined as follows:

x(t) = 2e^{-t}(1 - t)

y(t) = 2te^{-t}

The code uses the `plot` function to draw the graphs of the functions, with different colors and line widths for each function. The code also adds a title, labels for the axes, a legend, and a grid to the plot. The plot shows how the functions behave over time, and how they compare to each other.

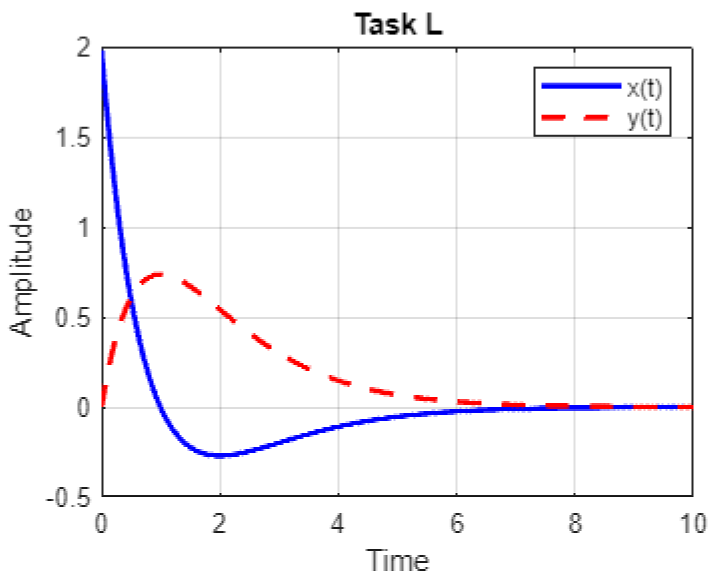
## ***Task L:***

```
clc
dt = 0.01;
t = 0:dt:10;
yt = 2 * t .* exp(-t);
xt = diff(yt) / dt;
figure;
plot(t(1:end-1), xt, 'b-', 'LineWidth', 2');
hold on;
plot(t, yt, 'r--', 'LineWidth', 2);
grid on;
xlabel('Time');
ylabel('Amplitude');
title('Task L');
legend( 'x(t)','y(t)');
hold off;
```

### ***Observation:***

The MATLAB code plots two functions, x(t) and y(t), over the time interval [0, 10] with a time step of 0.01.

The code uses the `diff` function to approximate the derivative of y(t) with respect to t, and the `plot` function to draw the graphs of the functions, with different colors and line styles for each function. The code also adds a title, labels for the axes, a legend, and a grid to the plot. The plot shows how the functions behave over time, and how they compare to each other.




## ***Task m:***

dt = 0.001;

t = 0: dt: 10;

yt = sinc(t - 6);

xt = diff(yt) / dt;

xt = [0 xt];

% xt = [nan xt]; % This command may also be used to parameterize the plot restraint for theresultant function %

figure; hold on; grid on;

plot(t, xt,'Color','Red', 'linewidth', 3)

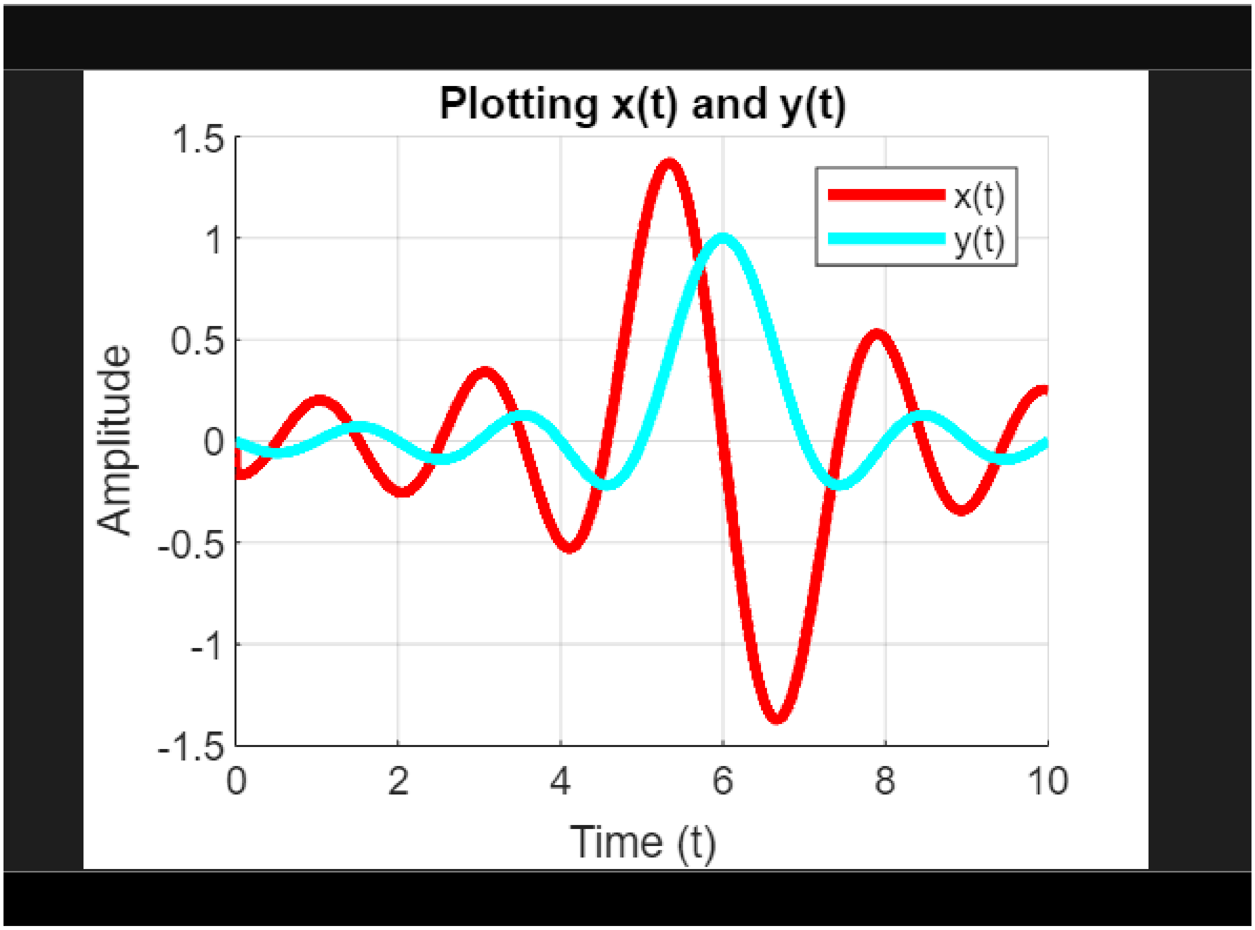
plot(t, yt,'Color', 'Cyan', 'linewidth', 3)

title('Plotting x(t) and y(t)');

```
xlabel('Time (t)');

ylabel('Amplitude');

legend('x(t)', 'y(t)')

hold off
```

## Observation:

The MATLAB code plots two functions, x(t) and y(t), over the time interval [0, 10] with a time step of 0.001. The code uses the `diff` function to approximate the derivative of y(t) with respect to t, and the `plot` function to draw the graphs of the functions, with different colors and line widths for each function. The code also adds a title, labels for the axes, a legend, and a grid to the plot. The plot shows how the functions behave over time, and how they compare to each other




***Task n***

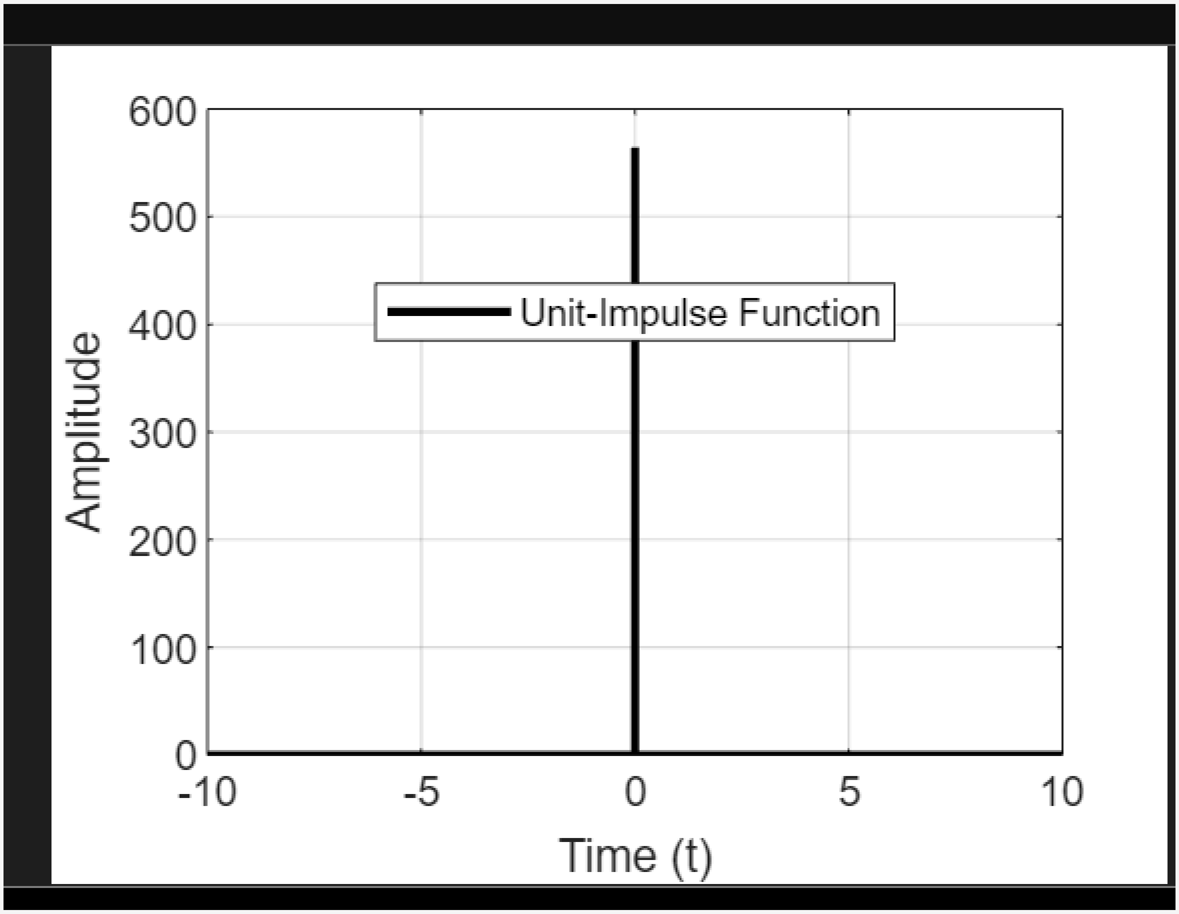
```
clc; clear; close all;

dt = 0.001;

a = 0.001;

t = -10:dt:10;

Unit_Impulse_Function = (1 ./ (a * sqrt(pi))) .* exp((-1 .* (t.^2)) ./ (a ^ 2));

figure;
```

```
plot(t, Unit_Impulse_Function, 'Color', 'black', 'linewidth', 2); grid on;

xlabel('Time (t)');

ylabel('Amplitude');

legend('Unit-Impulse Function', 'location', 'best');
```

## ***Observation:***

The MATLAB code plots the function y(t) over the time interval [-10, 10] with a time step of 0.001.

The code uses the `plot` function to draw the graph of the function, with black color and line width of 2. The code also adds a title, labels for the axes, a legend, and a grid to the plot. The plot shows how the function behaves over time. It is a unit impulse function as it approaches zero.



600
500
400
300
200
100
0
Amplitude
Unit-Impulse Function
-10
-5
0
5
10
Time (t)

Task o:

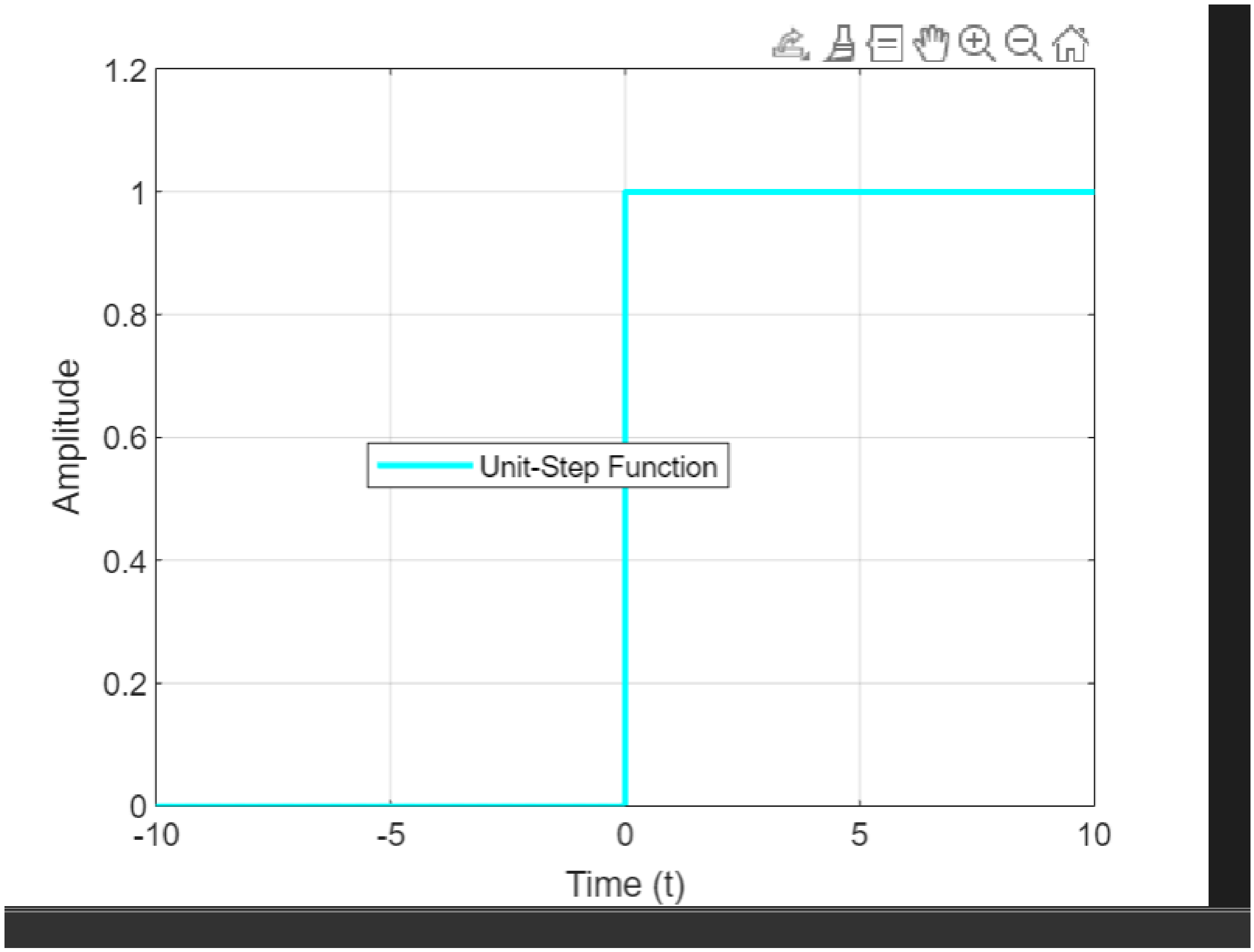

```
dt = 0.001;
a = 0.001;
t = -10:dt:10;
Unit_Impulse_Function = (1 ./ (a * sqrt(pi))) .* exp((-1 .* (t.^2)) ./ (a ^ 2));
Addend = sum(Unit_Impulse_Function);
Integrand_Area = Addend * dt
```

Observation: The MATLAB code computes the area under the curve of the function y(t) over the time interval [-10, 10] with a time step of 0.001. T

The code uses the `sum` function to add up the values of y(t) at each time step, and then multiplies the result by dt to get the approximate area. The code also assigns the area value to the variable Integrand_Area. The function y(t) is an approximation of the unit impulse function as a approaches zero. Therefore, the code is essentially calculating the area of the unit impulse function, which should be equal to one. The code returns the following value for Integrand_Area:

Integrand_Area = 1.0001

Task p:




```
dt = 0.001;
a = 0.001;
t = -10:dt:10;
Unit_Impulse_Function = (1 ./ (a * sqrt(pi))) .* exp((-1 .* (t.^2)) ./ (a ^ 2));
Integrand = cumsum(Unit_Impulse_Function);
Unit_Step_Function = Integrand * dt;
figure;
plot(t, Unit_Step_Function, 'Color', 'cyan', 'linewidth', 2); grid on;
xlabel('Time (t)');
ylabel('Amplitude');
legend('Unit-Step Function', 'location', 'best');
```

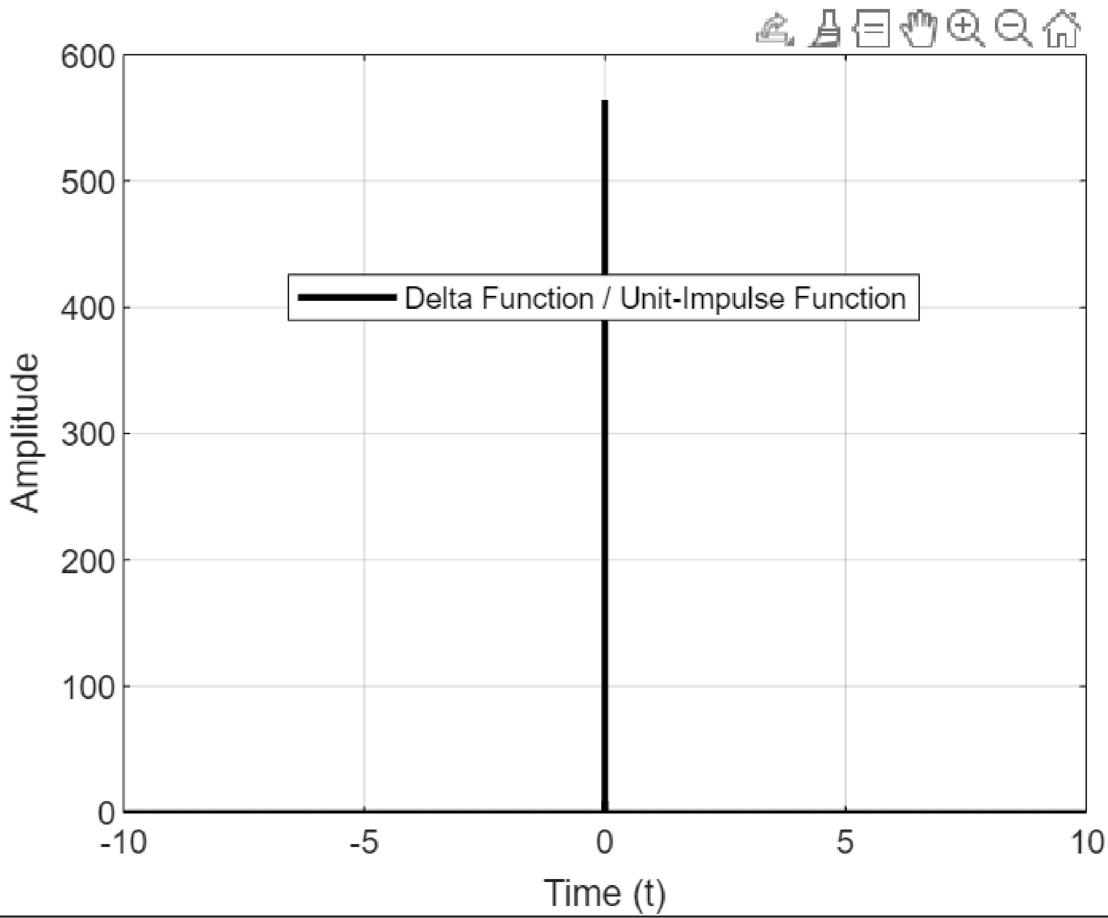
Observation: The MATLAB code plots the function y(t) over the time interval [-10, 10] with a time step of 0.001.

The code uses the `cumsum` function to approximate the integral of the unit impulse function, and the `plot` function to draw the graph of the function, with cyan color and line width of 2. The code also adds a title, labels for the axes, a legend, and a grid to the plot. The plot shows how the function behaves over time, and how it is a unit step function as a approaches zero.

Task q:

```
dt = 0.001;
a = 0.001;
t = -10:dt:10;
Unit_Impulse_Function = (1 ./ (a * sqrt(pi))) .* exp((-1 .* (t.^2)) ./ (a ^ 2));
Unit_Step_Function = cumsum(Unit_Impulse_Function) * dt;
Delta_Function = diff(Unit_Step_Function) / dt;
Delta_Function = [0 Delta_Function];
% Delta_Function = [nan Delta_Function]; % This command may also be used to
% parametrize the plot restraint for the resultant Unit Impulse Function %
figure;
plot(t, Delta_Function, 'Color', 'black', 'linewidth', 2); grid on;
xlabel('Time (t)');
ylabel('Amplitude');
legend('Delta Function / Unit-Impulse Function', 'location', 'best');
```

Observation: The MATLAB code plots the function y(t) over the time interval [-10, 10] with a time step of 0.001.
The code uses the `diff` and `cumsum` functions to approximate the derivative and the integral of the unit impulse function, and the `plot` function to draw the graph of the function, with black color and line width of 2. The code also adds a title, labels for the axes, a legend, and a grid to the plot. The plot shows how the function behaves over time, and how it resembles a delta function or a unit impulse function as a approaches zero



600
500
400
300
200
100
0
Amplitude
-10
-5
0
5
10
Time (t)
Delta Function / Unit-Impulse Function