HUMBER POLYTECHNIC

Faculty of Applied Science and Technology

Laboratory 1 Introduction to Signals and Systems

Student Name: Michael McCorkell Student Number: N01500049 Date: January 16th 2025

Note 1: This is an individual lab. Please bring your laptop and install required software to complete this lab.
Note 2: For your lab report, please take necessary screenshots and proper captions and include them in the corresponding parts of the lab. Demonstrate your work during the lab time. Submit the report file and .pdf file on the course website.

1. Learning outcome:

1.1 Familiarize with basic MATLAB scripting with functions.
1.2 Familiarize with signal visualization using MATLAB.
1.3 Conduct basic operations on signals using MATLAB.
1.4 Compute numerical and symbolic integration and estimating signal power and energy

2. Background

2.1 Signals visualization in MATLAB:

MATLAB is commonly used to visualize and process signals. However, for continuous-time (CT) signals, it is not possible to store, visualize, and process them directly in MATLAB. In order to resolve this issue, MATLAB uses symbolic variables to represent CT functions. Another common practice of avoiding this is to use part of the time axis as discrete-time points and plot the corresponding values at these discrete points. If the time intervals are small enough, the results displayed can be very close to the actual continuous-time signals.

In this lab, we will explore both methods to work with some common CT signals and signal manipulations.

Important Note:
When plotting to visualize results, a good question is how many points is enough. If too few points are chosen, information is lost. If too many points are chosen, memory and time are wasted. A balance is needed. For oscillatory functions, plotting 20 to 200 points per oscillation is normally adequate.

2.2 Working with function in MATLAB:

Working with functions is fundamental to signal processing. Functions contains one or more sequential commands and can accept inputs and return outputs.

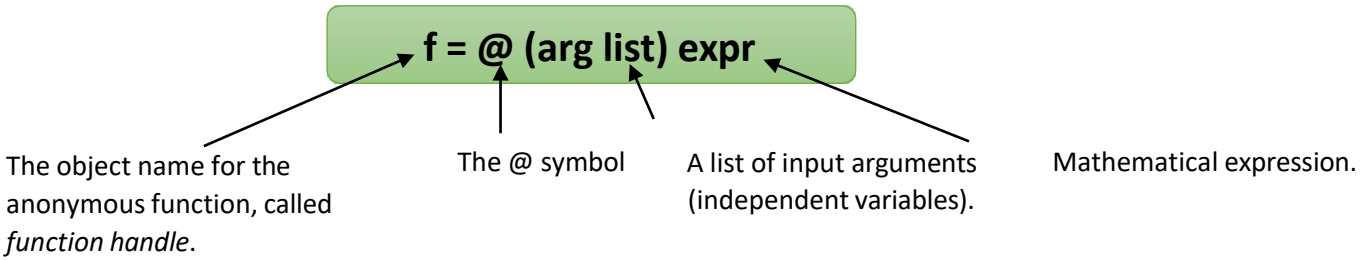
MATLAB has many built-in functions, but we can also create our own functions. MATLAB provides several methods for defining and evaluating functions. We will learn about two methods in this course: anonymous and M-file functions.

Anonymous Functions

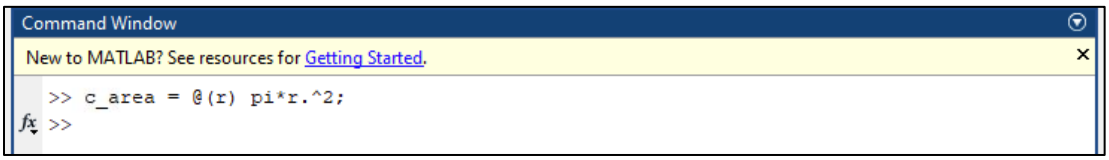
An anonymous function is a simple (one-line) user-defined function that is defined without creating a separate

function file (M-file). Anonymous functions can be defined in the Command Window, within a script file, or inside a user-defined function.

An anonymous function is defined by typing the following in the Command Window:



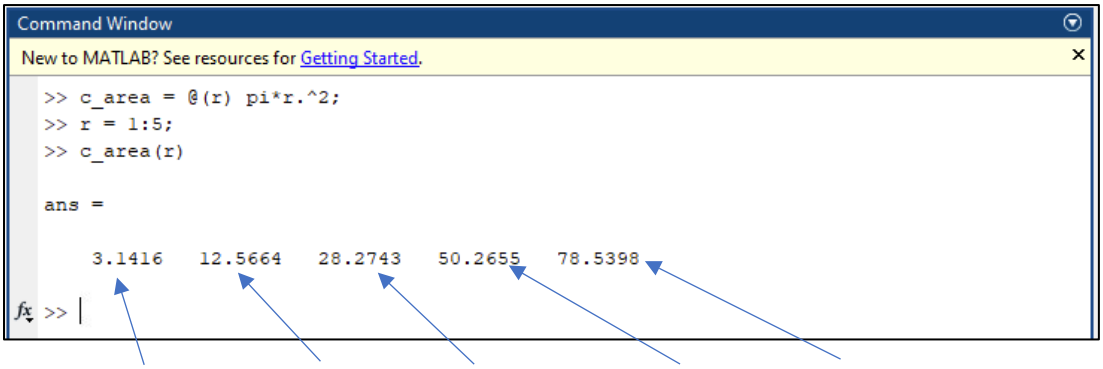
For example, create a handle to an anonymous function that finds the area of a circle:



Variable `c_area` is a function handle. The operator `@` creates the handle, and the parenthesis () immediately after the @ operator include the function input arguments. This anonymous function accepts a single input `r`, an implicitly returns a single output, an array the same size as `r` that contains the area of circles with radius of `r`.

Find the area of a circle with radius of 2cm by passing the value to the function handle, just as you would pass an input argument to a standard function.

Vector inputs allow the evaluation of multiple values simultaneously. The output will be a matrix of the same size as the input argument.
Below is an example to find the area of circles with radius of 1cm, 2 cm, 3 cm, 4 cm, 5 cm.



Area for r=1cm, Area for r=2cm, Area for r=3cm, Area for r=4cm, Area for r=5cm

We can use `plot` command to visualize result in a graph.

```
Command Window
New to MATLAB? See resources for Getting Started.
>>  c_area = @(r) pi*r.^2;
>> r = 1:5;
>> plot(r, c_area(r), '*')
>> xlabel('radius [cm]'); ylabel('area[cm2]'); grid;
>> axis ([0 6 0 90]);
>>
```

3. Procedures

3.1 Anonymous function declaration

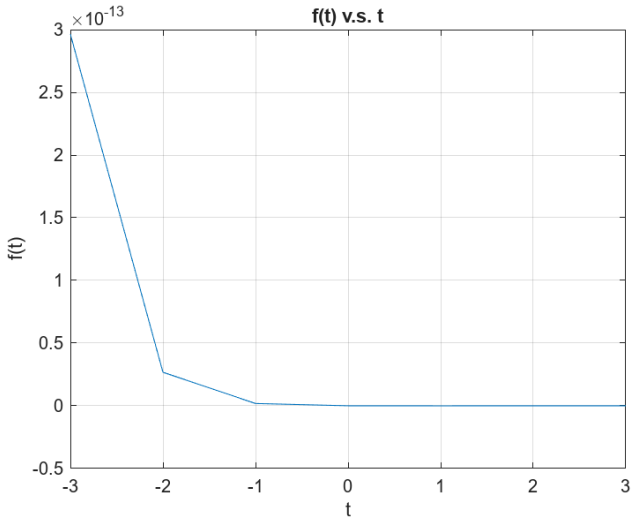
Task 1 (10%): Define the exponentially damped sinusoid function $f(t) = e^{-2t}\sin(2\pi t)$ using anonymous function method. Once defined, (a) evaluate the function $f(t)$ for $t = 0$. (b) Plot $f(t)$ versus t in the given interval [-3,3]. Add grid lines and proper labels to the x and y axis.

Task 1 a)

```
expsin = @(t) exp(-2*t)*sin(2*pi*t);
T=0 ;
expsin(t) ;
```

Task 1 b)

```
expsin = @(t) exp(-2.*t).*sin(2.*pi.*t);
t = -3:3;
plot(t, expsin(t)) ;
xlabel("t");ylabel("f(t)");grid;title("f(t) v.s. t");
```



3.2 Heaviside function

In MATLAB, there is a dedicated CT unit step function heaviside:

H = heaviside(x) evaluates the Heaviside step function (also known as the unit step function) at x. The Heaviside function is a discontinuous function that returns 0 for x < 0, 1/2 for x = 0, and 1 for x > 0. Run the following three lines to see the results.

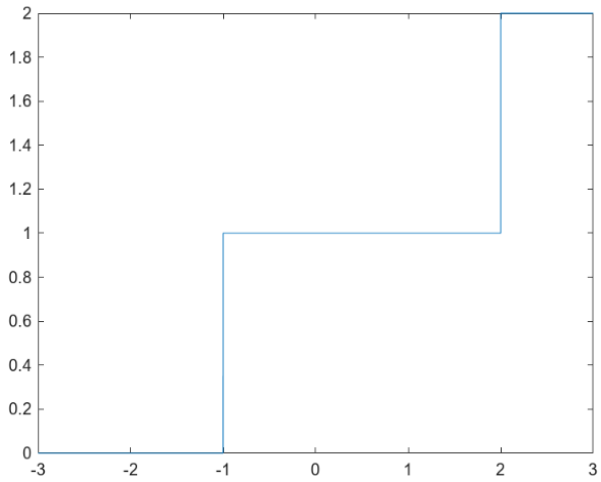
```
H = heaviside(sym(-3))
H = heaviside(sym(4))
H = heaviside(sym(0))
```

You can also plot the heavidside function by using the following lines:

```
syms x
fplot(heaviside(x), [-2, 2])
```

```
fplot(heaviside(x-2), [-2, 4])
```

Task 2 (15%): please use the space below to record the code of plotting a piecewise constant function like the equation below by stacking Heaviside functions, please only plot between [-3,3] and include the resultant MATLAB figure below:



$$f(t) = \begin{cases} 0, t < -1 \\ 1, -1 < t < 2 \\ 2, t > 2 \end{cases}$$

Task 3 (10%): Now please use Anonymous function declaration to define the same piecewise function $f(t)$. Please note that you can use code like below to define a piecewise function:

```
f = @(t) (2.*(t>0)) + (3.*(t<0))
```

```
J = @(t) ( (0.*(t<-1)) + (1.*(-1<t<2)) + (2.*(t>2)) )
```

3.2 Dirac function

Similar to heaviside, there is a dedicated CT unit impulse function called dirac:

d = dirac(x) represents the Dirac delta function of x.

d = dirac(n,x) represents the nth derivative of the Dirac delta function at x.

Run the following three lines to see the results.

```
H = dirac(sym(-3))
H = dirac(sym(4))
H = dirac(sym(0))
```

Similar to Heaviside, we now use fplot to visualize the Dirac function. Run the following scripts to see the result.

```
syms x
fplot(dirac(x), [-3, 3])
```

Question 1 (5%): From the plot, it seems that Dirac function generates a constant zero in this case, is it true? Please explain.

The Dirac delta is zero for all z≠0. Since the plot samples z at non-zero values (even z=0 might not be sampled exactly in numerical evaluation), it will always return zero.

3.3 Relationship between the unit step and unit impulse functions

Use the integral and derivative to examine the relationship between unit step and unit impulse functions.

```
syms x
diff_H = diff(heaviside(x),x)
int_D = int(dirac(x),x)
```

Question 2 (5%): Are the results as expected? If not, please explain why it is different compared to what you have anticipated.

The unit step function is what was expected, but the unit impulse function is not what's expected, probably because of MATLAB simplifying or is using a specific convention for the integral of the Dirac delta function.

3.4 Odd and even signals

Let's define a piecewise function g(t):

$$g(t) = \begin{cases} 2+t, & -3 \le t < -1 \\ -t, & -1 \le t < 1 \\ -3+2t, & 1 \le t \le 3 \end{cases}$$

You can use one of the two methods:

(1) Use the following function as is and save the function in the same working directory.

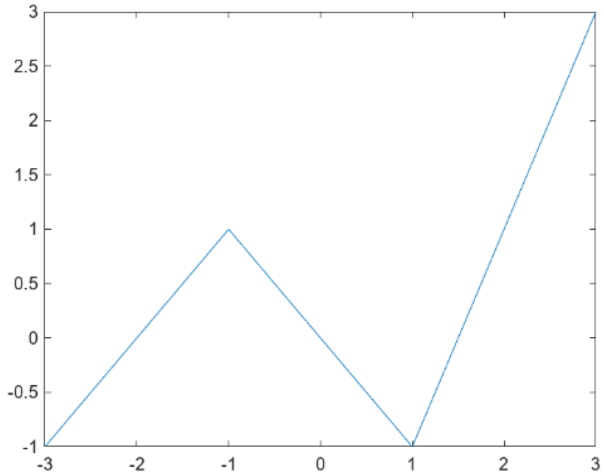
(2) Convert the following MATLAB script into an anonymous function to create a function g(t) in MATLAB.

```
function y = g(t)
    y = (2+t).*(-3<=t & t<-1) + -1*t.*(-1<=t & t<1) + (-3+2*t).*(1<=t & t<=3) ;
```

Then run the visualization by setting up the range of the time axis [-3,3] and plot the trace.

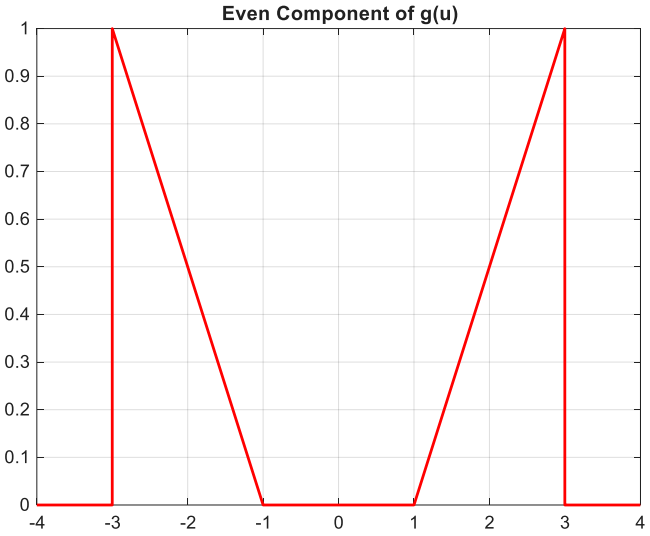
Task 4 (10%): Record your code and plot below.

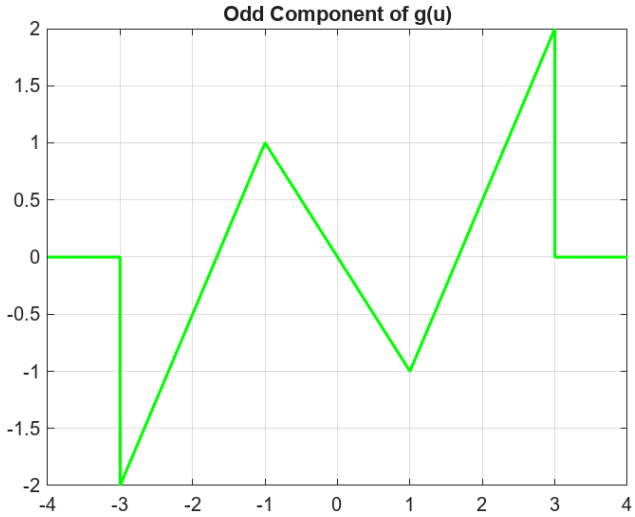
```
g = @(u) ( ((2+u).*(-3<=u & u<-1)) + ((-u).*(-1<=u&u<1)) +
( (-3+2.*u).*(1<=u&u<=3)) )
fplot(g, [-3,3])
```



Task 5 (10%): As mentioned in the lecture, we can break any signal into an odd and an even signal. Please implement the process in MATLAB and decompose g(t) into an odd and an even signal, visualize the odd and even signals individually. Please record your code and the screenshots of the signals below.

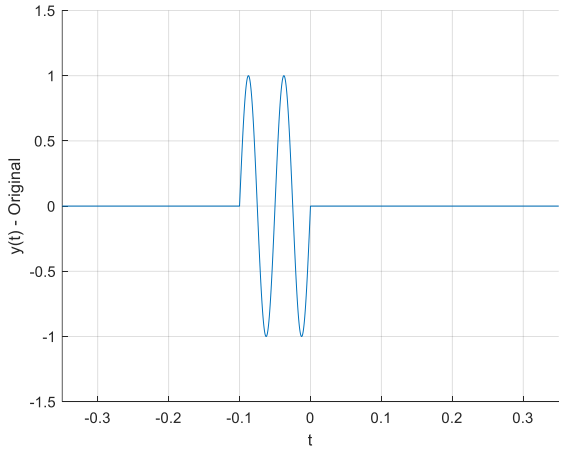
```
g = @(u) ( ((2+u).*(-3<=u & u<-1)) + ((-u).*(-1<=u&u<1)) + ( (-3+2.*u).*(1<=u&u<=3)) )
fplot(g, [-3,3])
g_even = @(u) (g(u) + g(-u)) / 2; % Even component
g_odd = @(u) (g(u) - g(-u)) / 2;  % Odd component
fplot(g_even, [-4, 4], 'r', 'LineWidth', 1.5); title('Even Component of g(u)'); grid on;
fplot(g_odd, [-4, 4], 'g', 'LineWidth', 1.5); title('Odd Component of g(u)'); grid on;
```

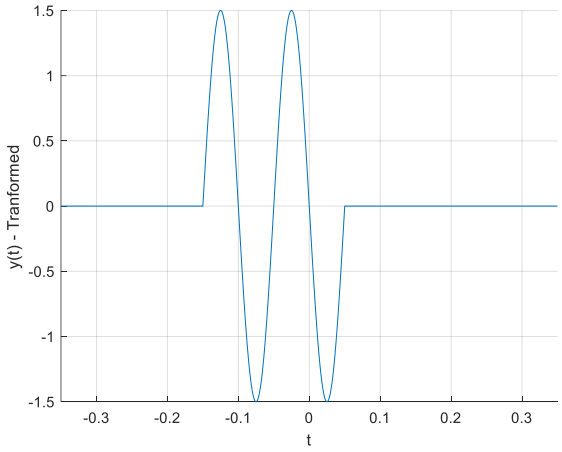




3.5 Shifting and scaling

Below are two signals. The one on the left is the original signal, and the one on the right is the transformed signal using scaling and shifting.





Based on the information given, please do the following tasks.

Task 5 (15%): Recreate the original signal in the MATLAB. Store your original signal in a separate Matlab function file. Please record your code below:

```
x_o=-0.4:0.0001:0.4;
y_o= original_signal(x_o);
plot(x_o,y_o); grid;

function y = original_signal(o)
   y = ( (heaviside(o+0.1) - heaviside(o-0)) .* sin(20*pi*2*o) )
end
```

Task 6 (20%): Recreate the variable transformation in MATLAB. Please record your transformation using MATLAB code below.

```
y_t=transfigure_signal(x_o);
plot(x_o,1.5*y_t); grid;


function y=transfigure_signal(o)
   y = ( (heaviside(o+0.05) - heaviside(o-0.05))  .* sin(20*pi*2*o) )
end
```