Lab 1: Introduction to MATLAB and Modules 1 + 2a

Please note that it is recommended that you complete the MATLAB Onramp before attempting this lab! It is not necessary but it will help you greatly. Moreover if you are confused about any syntax that you don't think is adequately explained in this document, input "doc (whatever function you are unsure about) in the command window and it will take you to the official documentation.

Section 1: Introduction

In this introductory section, you will be exposed to the tools necessary to complete this lab. The idea is to follow along and run the code yourself, and perhaps modify it a bit to ensure you understand the content/syntax. Section 2 will be the questions for handin.

Vectors:

To define 1 dimensional row vector in MATLAB, you may use any one of 2 methods:

Method 1: The syntax t = start:step:stop defines a vector titled 't' whose first element has the value 'start' and the last element has the value of 'stop', and increments 'start' in step size equal to 'step' until it reaches 'stop'. These need to be integers. For example run the following code:

```
t1 = [1:0.1:2]
```

```
t1 = 1×11
    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000 ···
```

Just note that the processing time to generate the vector is inversely proportional to the step size, in other words a vector with step size whose order of magnitude is E-12 will take much longer to generate than one with a step size of E-3. Of course this is due to the first one having many more elements than the second. You need to discern for yourself what step size is appropriate for the given question, for instance if you are using 't' as the independent variable for plotting f(t) = sin(t), you need to choose dt small enough (dt = step size) such that there is adequate resolution in your plot.

Method 2: The syntax t = linspace(start, stop) will generate a row vector from start to stop of evenly spaced points, and t = linspace(start, stop, n) will generate a row vector from start to stop spanning 'n' points. The difference between this one and the first method and this one is the first one allows the easy manipulation of the step size while this method allows you to directly control the size of the vector. See the output of using this method below:

```
t2 = linspace(1,2,11)
```

```
t2 = 1×11
    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000 ···
```

You may extract the size of a vector or matrix using the size(t) function, which outputs a row vector with 2 elements, the first being the number of rows and the second being the number of columns:

```
size(t2)
```

```
ans = 1×2
     1    11
```

If you only care about the number of columns, you may use the length(t) function which outputs a scalar equal to the number of columns:

```
length(t2)
```

```
ans =
11
```

You may extract certain elements from a vector by using the following syntax: new_vect = old_vect(row,col), where if you want to extract a whole row or a range from a row or column use ':'. See below:

```
t3 = t2(1,:)
```

```
t3 = 1×11
    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000 ···
```

```
t4 = t2(1, 1)
```

```
t4 =
1
```

```
t5 = t2(1, 2:4)
```

```
t5 = 1×3
    1.1000    1.2000    1.3000
```

To generate a matrix or vector, use square brackets around the element set. For matrices, commas seperate columns and semicolons separate rows:

```
matrix1 = [1,2,3;4,5,6;7,8,9]
```

```
matrix1 = 3×3
     1     2     3
     4     5     6
     7     8     9
```

```
matrix2 = matrix1(1:2, :)
```

```
matrix2 = 2×3
     1     2     3
     4     5     6
```

To create a vector for function outputs, pass a vector into a predefined/built in function or write an algebraic expression using the vector you have already defined:

```
dt = 0.1;
t = 0:dt:10
```

```
t = 1×101
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000 ···
```

```
f = sin(t) %This creates a vector of equal size of t, but point wise has been
passed into sin(x)
```

```
f = 1×101
         0    0.0998    0.1987    0.2955    0.3894    0.4794    0.5646    0.6442 ···
```

On a side note, notice how f and t are very similar for small values of t. For algebraic expressions, you need to place a '.' infront of the operation you are applying. This ensures that it is performed POINT WISE on the vector:

```
g = t.^2 + t.^2
```

```
g = 1×101
         0    0.0200    0.0800    0.1800    0.3200    0.5000    0.7200    0.9800 ···
```

```
k = t.*sin(t)
```

```
k = 1×101
         0    0.0100    0.0397    0.0887    0.1558    0.2397    0.3388    0.4510 ···
```

Finally to create for loops, the syntax is: "for index = start:stop" folllowed by "end":

```
k = 0;
for i = 1:10
    k = k+1;
end
k
```

```
k =
10
```

Symbolic maths:

To create a set of symbols, use the following notation:

```
syms x y z

g = y + z^2
```

g = $z^2 + y$

```
k = diff(g, z) %symbolically differentiate g wrt z
```

k = $2z$

```
h = int(g,z) %symbolically integrate g wrt z
```

h =

$$\frac{z^3}{3} + yz$$

To plug vectors/scalars into the symbols, use the following syntax:

```
h = subs(h, {y,z}, {1,2}) % "plug 1 in for y and 2 in for z into the expression for
h and evaluate"
```

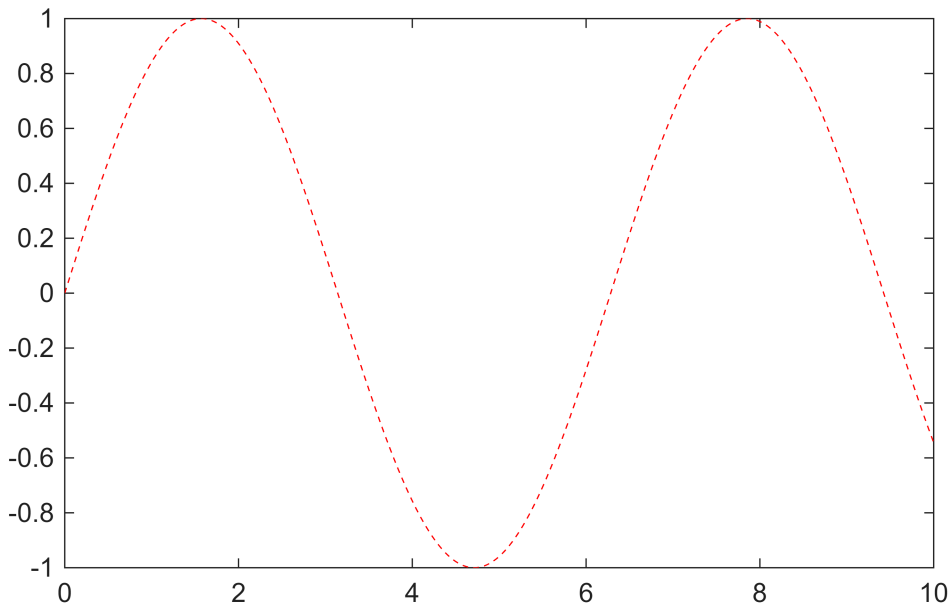
h =

$$\frac{14}{3}$$

Plotting in MATLAB (Important!!)

To plot a function in matlab, the following syntax is used:

```
dt = 0.001;
t = 0:dt:10;
y = sin(t);
plot(t, y, 'r--') %plot y (y-axis) vs t (x-axis) in red (r) dashed line (--)
```



Note MATLAB assumes you are using radians, to convert to degrees use the function rad2deg and deg2rad to go backwards. Also pi is a built in MATLAB variable so you may write it as so

```
t = pi
```

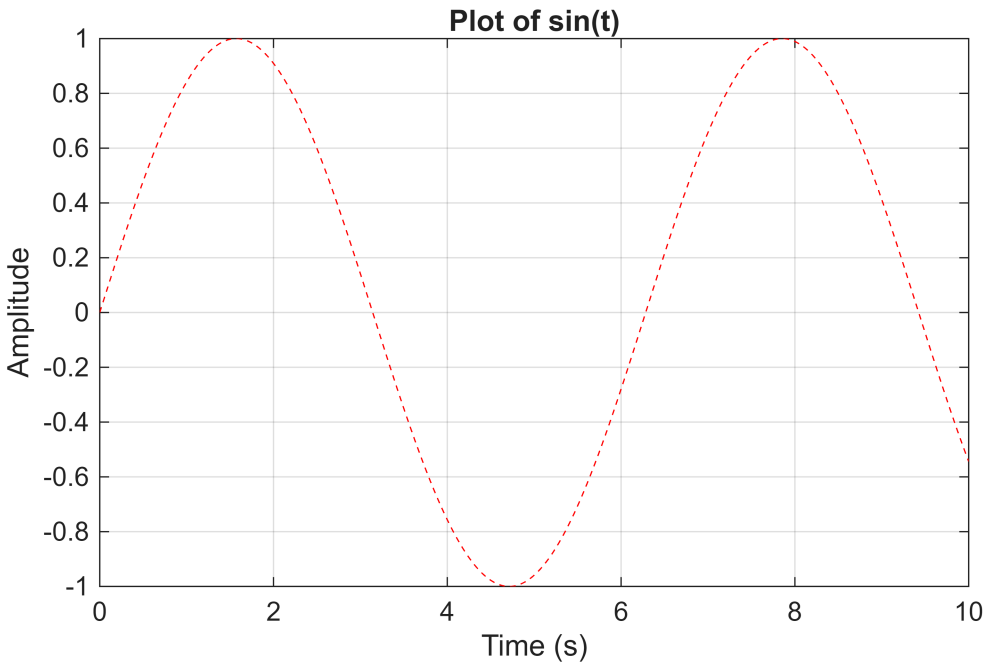
```
t =
3.1416
```

```
u = rad2deg(t)
```

```
u =
180
```

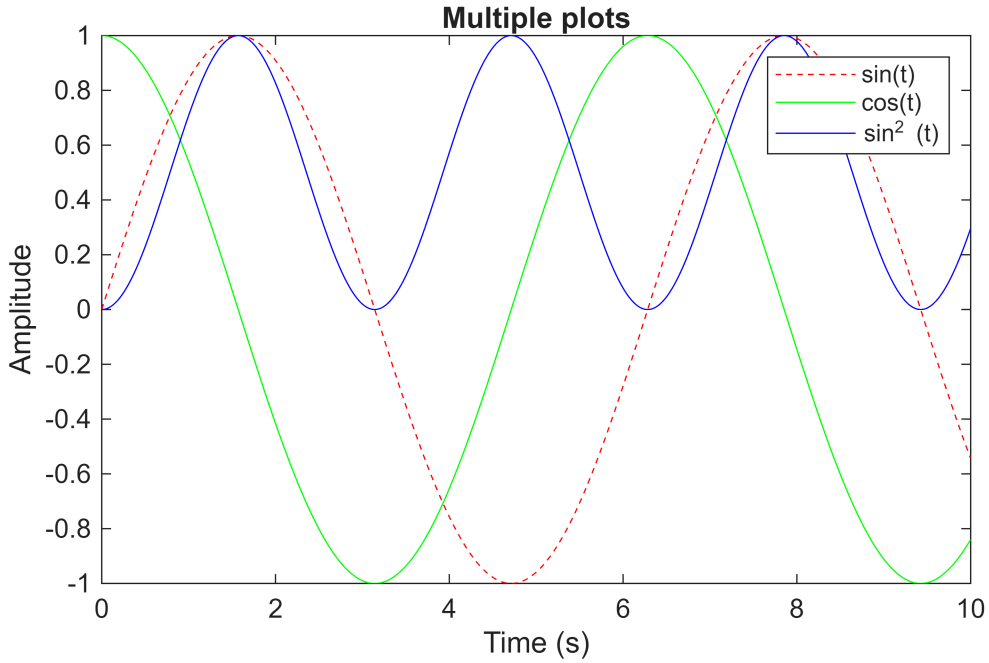
You may annotate the basic plot using the following syntax:

```
dt = 0.001;
t = 0:dt:10;
y = sin(t);
plot(t, y, 'r--')
xlabel('Time (s)');
ylabel('Amplitude');
title('Plot of sin(t)');
grid on;
```



Moreover you may plot multiple functions on the same axes using "hold on". You must use "hold off" when you are done adding plots:

```
dt = 0.001;
t = 0:dt:10;
y = sin(t);
f = cos(t);
g = (sin(t)).^2;
plot(t, y, 'r--')
hold on
plot(t, f, 'g-')
hold on
plot(t,g,'b');
hold off
xlabel('Time (s)');
ylabel('Amplitude');
title('Multiple plots');
legend('sin(t)', 'cos(t)', 'sin^2 (t)');
```



Single line functions

To quickly create a function of 't', use the following syntax:

```
f = @(t) exp(-t.^2).*log(t).^2 %creates function handle with values = RHS.
```

```
f = function_handle with value:
    @(t)exp(-t.^2).*log(t).^2
```

Functions are important for analytical computations as MATLAB's integral function requires function handles as inputs.

```
q = integral(f, 0, Inf) %q = integral of f wrt t from t = 0 to infinity
```

```
q =
1.9475
```

To symbolically take the integral/derivative and then substitute the values you want, use the following syntax:

```
syms x
f = x^2 + log(x) + exp(x)
```

f = $e^x + \log(x) + x^2$

```
F = int(f)
```

```
F =
```

$e^x - x + x\log(x) + \frac{x^3}{3}$

```
F_at_x_equals_3 = double(subs(F, x, 3)) %subs x = 3 intp F, and converts symbol to
double
```

```
F_at_x_equals_3 =
```

```
29.3814
```

You can also make a vector of the integral:

```
F_vector = double(subs(F, x, 1:1:10))
```

```
F_vector = 1×10
10^4 ×
    0.0002    0.0009    0.0029    0.0077    0.0193    0.0480    0.1218    0.3160 ···
```

Complex Numbers:

```
x = 2 + j*3
```

```
x =
2.0000 + 3.0000i
```

You can extract the real and imaginary components via:

```
Realx = real(x)
```

```
Realx =
2
```

```
Imaginaryx = imag(x)
```

```
Imaginaryx =
3
```

```
abs(x) %output magnitude of x
```

```
ans =
3.6056
```

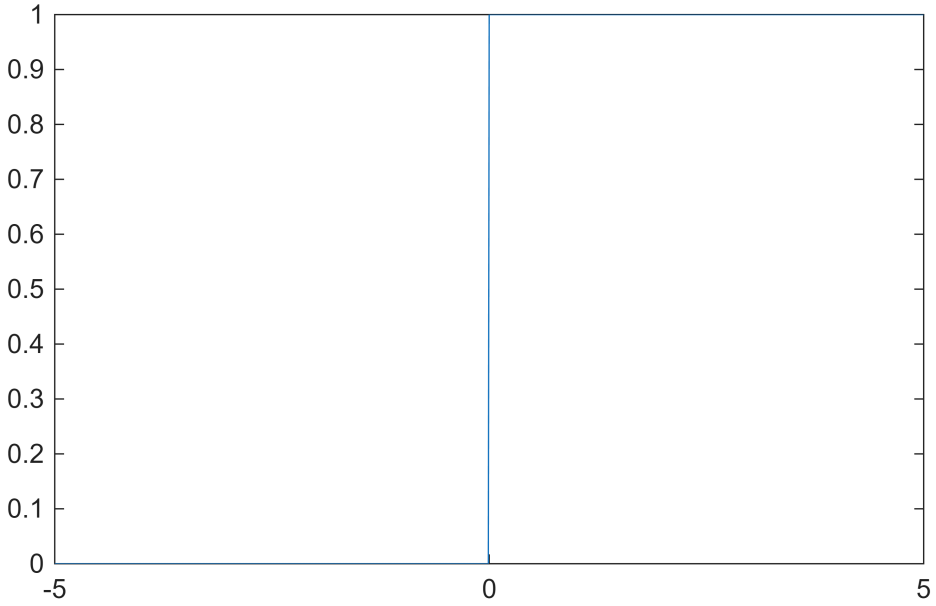
```
angle(x) %phase of x RADIANS
```

```
ans =
0.9828
```

Special functions

To define a unit step, you may use symbolic processing or analytical methods. A simple analytical method is:

```
u = @(t) double(t >= 0);
t = -5:0.01:5;
plot(t,u(t))
```

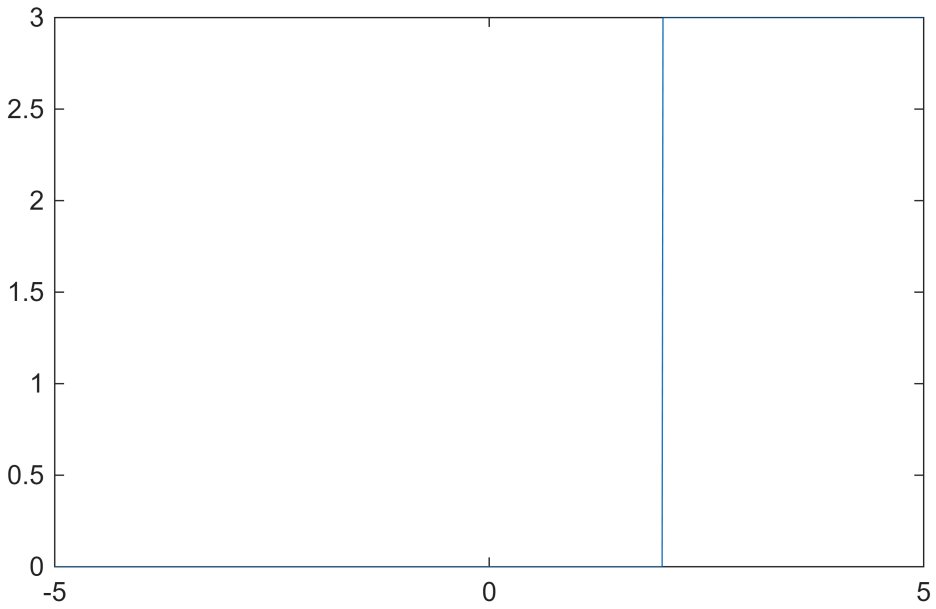


And for a magnitude change and shift:

```
u_shift = @(t,a, b) b*double(t >= a)
```

```
u_shift = function_handle with value:
    @(t,a,b)b*double(t>=a)
```

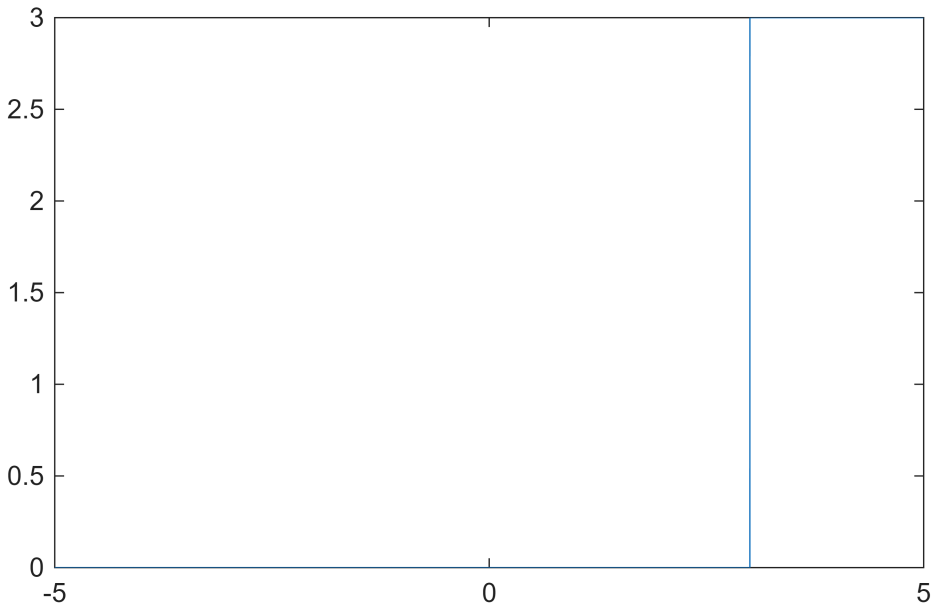
```
plot(t, u_shift(t, 2, 3)) %plug in a = 2 and b = 3 and t = -5:0.01:5 (predefined)
```



Another method is to explicitely define it using symbolic processing:

```
syms t
u = heaviside(t);
```

```
u_shift = heaviside(t-3);
fplot(3*u_shift, [-5,5]); %fplot takes in a symbolic function (first input) and
second is the domain
```



To define the Dirac Delta function, you just use the function dirac(x):

```
x = -5:5;
f = dirac(x)
```

```
f = 1×11
     0     0     0     0     0   Inf     0     0     0     0     0 ···
```

And to be able to symbolically define it:

```
syms x
f = dirac(x)
```

f = $\delta(x)$

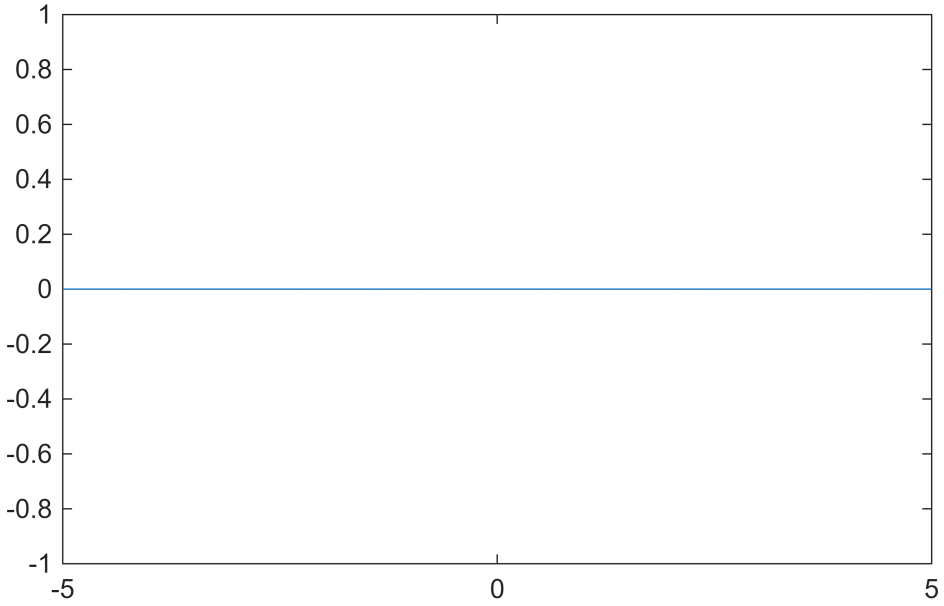
OR you can use the fact that u'(t) = dirac(t):

```
f = diff(heaviside(x))
```

f = $\delta(x)$

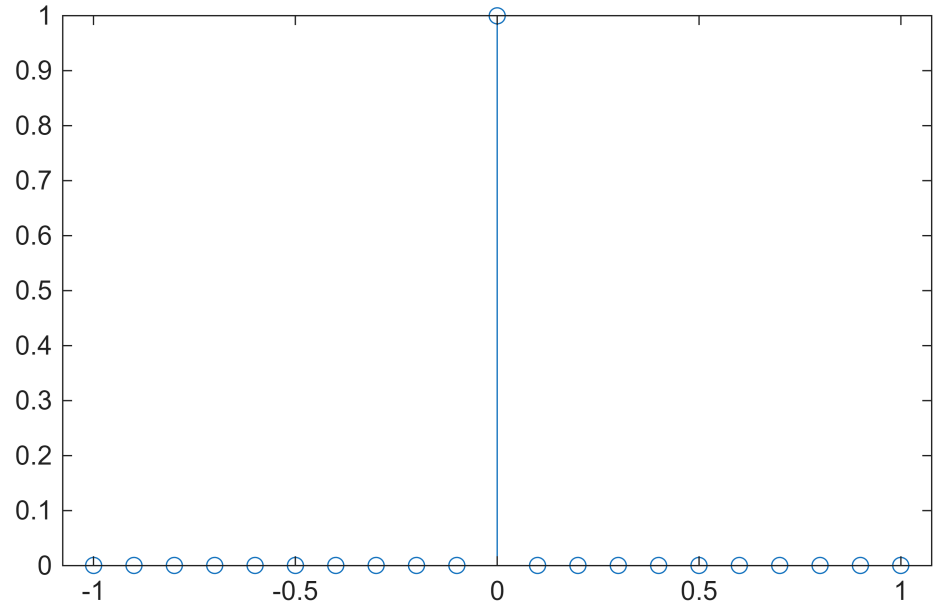
And you can plot it:

```
fplot(f, [-5, 5])
```



Note how the infinity at x = 0 does not show on fplot. Instead use numeric values paired with a STEM plot to show this:

```
x = -1:0.1:1;
y = dirac(x);
idx = y == Inf; % find Inf
y(idx) = 1;     % set Inf to finite value
stem(x,y)
```



To deal with integral involving the dirac delta, recall the sifting property and observe it below:

```
syms x a
f = int(dirac(x-a) * sin(x) , x, -Inf, Inf)
```

f = $\sin(a)$

And this concludes the introduction! You are now ready to attempt the questions below: