ECE102, Fall 2020
Signals & Systems
University of California, Los Angeles; Department of ECE

Introduction to MATLAB

Prof. J.C. Kao
TAs: T. Moonsor, A. Ghosh & G. Zhao

This tutorial is a basic introduction to MATLAB. It focuses on creating vectors in MATLAB, performing operations on vectors, plotting figures, and creating functions.. For more references, you can check Mathworks website (see `https://www.mathworks.com/support/learn-with-matlab-tutorials.html`) where you can find more documentation or video tutorials. You can also check this section "Getting Started with MATLAB" (see `https://www.mathworks.com/help/matlab/getting-started-with-matlab.html?s_cid=learn_doc`) or any other online resource.

1. **MATLAB access**

   UCLA have a campus-wide MATLAB license, you can access MATALB from this link:
   https://softwarecentral.ucla.edu/matlab-getmatlab

2. **MATLAB desktop**

   When we open MATLAB, we see three windows (see `https://www.mathworks.com/help/matlab/learn_matlab/desktop.html`):

   - "current folder": it is the current working directory from which you can upload or save files,
   - "workspace": it keeps a list of the variables created,
   - "command window": it is where we type commands (indicated by the prompt >>).

   In general, we'll be dealing with commands which we write in the command window/command line, and also with the editor.

3. **Scalars**

   One simple command is to create a scalar variable (in the command window):

   ```
   a = 3
   ```
   which should give us the output:
   ```
   a =
   ```

```
3
```
You can run the command again, but instead with a semicolon:
```
a = 3;
```
This will assign the value 3 to the variable a, but it will not print the output.

In addition, we can use complex numbers in MATLAB. If we want `b` to equal $1 + 2j$ (or $1 + 2i$), we would write:
`b = 1 + 2j` or `b = 1 + 2i`.
If we want the complex conjugate of a complex number, we'll run:
```
conj(b)
```

4. **Vectors and Matrices**

Next, we'll want to know how to create vectors and matrices. By default, most values in MATLAB will be vectors, matrices, or scalars (It is called *MATrix LABoratory*, after all).
If we want a matrix `A` of all zeros with 3 rows and 2 columns, then we'll run:
```
A = zeros(3, 2)
```
By default, the command `zeros(3)` will return a 3x3 matrix of all zeros. Other useful functions are `ones(m, n)` (mxn matrix of all ones), `eye(n)` (nxn identity matrix), `randn(m, n)` (mxn matrix of values randomly distributed according to the standard normal distribution), such as:
```
B = ones(3, 2)
```
If we want the transpose of a matrix, we'll use the single quotation `'`, which is used as:
```
A'
```
By default, multiplication of vectors or matrices is a matrix multiplication, so the command:
```
A*A
```
will return an error, while the command
```
A*A'
```
will be $A * A^T$.

If we want the elementwise multiplication of two vectors or two matrices, then we'll need to precede `*` with a period, like so:
```
C = A.*B
```
In this case, we'll have $c_{ij} = a_{ij} * b_{ij}$. Other elementwise operations are division (/) and 'to the power of' (^). Multiplication of a scalar and a matrix will cause all elements of the matrix to be multiplied by that scalar.

If we want to read or write to different elements of our matrices or arrays, we need to be able to index them. To get a_{kl}, we would write: `A(k, l)`
If we want the entire kth row of A, we would write
```
A(k, :)
```
Notice the use of the colon to represent the entire range of columns. If we want to get a sub-array of A, for example the 2nd to third row of A, we could run:
```
A(2:3, :).
```
If we want to assign to A, we can do so:

`A(2:3, :) = 1`
or
`A(2:3, :) = A2`
provided that `A2` has the same dimensions as `A(2:3, :)`

5. **Creating a range of numbers and performing operations on them**

   Since we'll be plotting $x(t)$ vs. t, we'll need to know how to create vectors that represent our time t. Say for example we want an array `t1` containing the values [0 1 2 3 4 5]. Then we could run: `t1 = 0:5`
   or
   `t1 = linspace(0, 5, 6)`
   In this case, `0:5` is an array of numbers from 0 to 5 with step size 1, and `linspace(0, 5, 6)` is 6 numbers equally spaced from 0 and 5. In order to vary the step size, e.g. to 0.5, we can instead write:
   `t2 = 0:0.5:5`

   By default, t1 and t2 will be row vectors (1 row and n columns), and can be concatenated as: `[t1 t2]`
   In general, vectors should be treated as matrices with a single row or a single column, and can be manually defined:
   `[1 2 3]` is a row vector and is equivalent to `[1, 2, 3]`, while `[1; 2; 3]` is a column vector equal to `[1, 2, 3]'`. Additionally, `[1, 2; 3, 4]` is a 2x2 matrix.

6. **Plotting**

   Say we define:
   `x = t2.^2`
   Then we can plot it by running: `plot(t2, x)`,
   where t2 will end up on the x axis and x will be on the y axis.
   If we want the line to be red, we can use `plot(t2, x, 'r')` instead. We can add a grid by running:
   `grid` or `grid on`
   We can also add a title by running:
   `title('this is the title')`
   and similarly for x and y axis labels: `xlabel('this is the xlabel')`,
   and `ylabel('this is the ylabel')`.
   If we want to plot two figures on the same plot, we'll use the command:
   `hold on`
   and if you want to create a new figure, run: `figure` Running figure(1) will tell MATLAB to plot new figures on figure 1 instead of the most recently created figure.

7. **Functions**

   MATLAB has many builtin functions, including `sin`, `cos`, `exp`, `log`, `sqrt`, `min`, and `max`. To use them, simply try:
   `sin(pi)`
   or

```
y = sin(t2)
```
or
```
min(3, t2)
```
Here `pi` is a built-in constant in MATLAB.

What if we want to create our own functions? Take a look at the file "cube.m", which contains:
```
% returns the element-wise cube of x
function out = cube(x)
out = x.^3;
end
```
Here the % turns the rest of the line into a comment, `out` is the output variable (we can also have multiple), `cube` is the name of the function, and `x` is the input variable (there can be multiple). In MATLAB, whenever we create a function `fnc`, we need to create a corresponding file called "fnc.m" in the current directory (unless we add it to MATLAB'S `PATH`).

Then when we run:
```
cube(3)
```
we'll get the output `9`, and if we run:
```
cube(1:3)
```
we'll get the output `[1 8 27]`.

8. **Figure examples**

   We will illustrate all these plotting options with the following three examples.

   (a) **Example 1**

   In this first example, we are going to plot the following two functions on the same plot.

$$x(t) = \cos(2\pi t)$$
$$y(t) = \cos(2\pi t + \pi/3)$$

   We are going to plot them for $-3 \leq t \leq 3$.
   The code is:
```
t=-3:0.05:3;
x=cos(2*pi*t);
y=cos(2*pi*t+pi/3);
plot(t,x,'r',t,y,'b--o');%r: red color, b: blue color,
%'--':dashed line, o: circle marker
grid on;
title('Example 1'); xlabel('t(sec)'); ylabel('x,y');
```
   It generates the plot shown in Fig. 1.

   (b) **Example 2**

   In this second example, we are going to plot a piecewise function. Consider the following

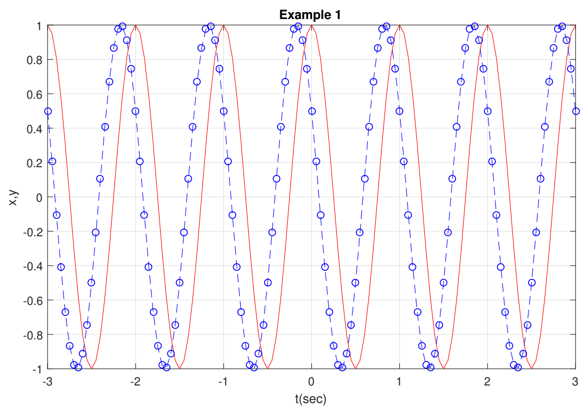

Figure 1: Example 1

function:

$$x(t) = \begin{cases} 1 + e^{-t}, & \text{if } t \geq 0 \\ 1, & \text{if } -1 \leq t < 0 \\ 0, & \text{otherwise} \end{cases}$$

We are going to plot the function for $-3 \leq t \leq 10$. To do this, we are going to define for each time interval a vector that represents the function for that particular time interval and, at the end, we are going to concatenate the vectors.

The code is:

```
t1=-3:0.01:-1.01; x1=zeros(1,length(t1));
t2=-1:0.01:-0.01; x2=ones(1,length(t2));
t3=0:0.01:10; x3=1+exp(-t3);

t=[t1 t2 t3]; x=[x1 x2 x3];
plot(t,x,'g');grid on;
title('Example 2'); xlabel('t(sec)');ylabel('x');
```

It generates the plot shown in Fig. 2.

(c) **Example 3**

In this third example, we are going to plot multiple figures next to each other. Consider the following three functions:

$$x(t) = \cos(2\pi t)$$
$$y(t) = \cos(6\pi t)$$
$$y(t) = t\cos(2\pi t)$$

The code is as follows:

```
t=-3:0.01:3;
x=cos(2*pi*t);y=cos(4*pi*t);z=t.*cos(2*pi*t);
```

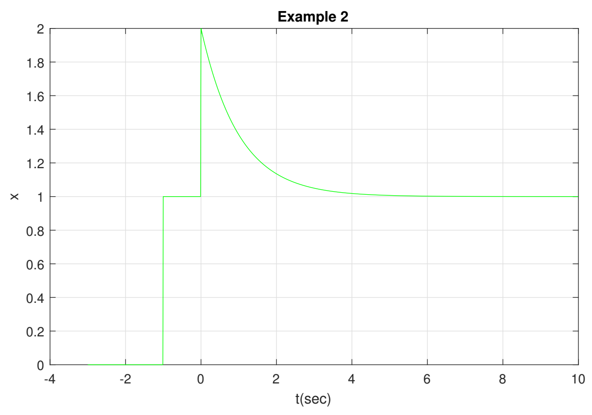

Figure 2: Example 2

```
subplot(1,3,1);%this divides the figure into 1 by 3 grid
%the third argument 1 means to plot in the first location
plot(t,x);grid on;
title('Plot of x(t)=cos(2\pit)'); xlabel('t(sec)');ylabel('x');
axis([-3 3 -3 3]);%this sets the limits of the axes
% the first two are for the x-axis, the second two are for the y-axis
%setting the limits for the x-axis and y-axis is important especially
% when we want to compare between different signals

subplot(1,3,2);
plot(t,y);grid on;
title('Plot of y(t)=cos(4\pit)'); xlabel('t(sec)');ylabel('y');
axis([-3 3 -3 3]);

subplot(1,3,3);
plot(t,z);grid on;
title('Plot of z(t)=tcos(2\pit)'); xlabel('t(sec)');ylabel('z');
axis([-3 3 -3 3]);
```

The code generates the plot shown in Fig. 3.

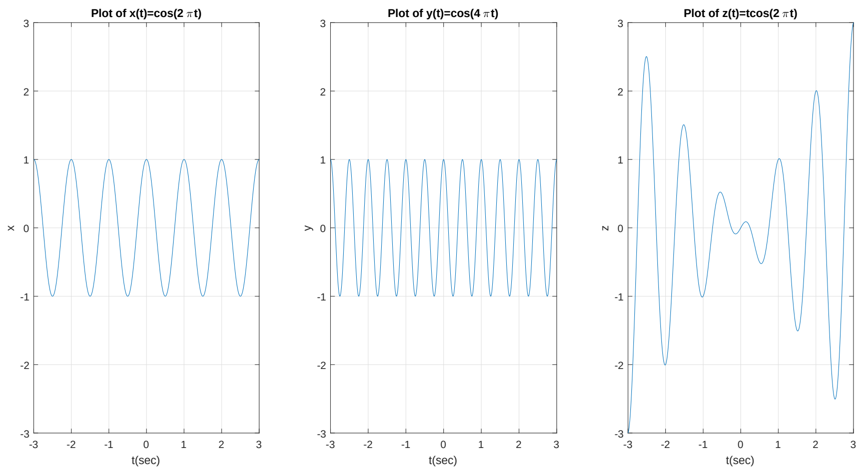

Figure 3: Example 3