
MATLAB Tutorial

This tutorial will give an overview of MATLAB commands and functions that you will need in ECE 366.

1. **Getting Started:** Your first job is to make a directory to save your work in.

Unix and PC: Create a directory called "matlab" or "ece366". Now that we have a place to save files, we need to start Matlab. The procedure used depends on where you are logged on.

Unix: Open up your favorite editor (say, for example, "axe" or "emacs") and an xterm; type "tap matlab" in the xterm window, then the command "matlab". Another window should open with a plot in it. This might take a while depending on how many other things are running. Be patient.

PC: Run matlab by double clicking on its icon. MATLAB has its own editor that you can open from the command window by going to File → New → M-File. You need to verify which directory Matlab is currently operating in. Some DOS and UNIX commands can be used in the MATLAB command window. For example, you can type 'pwd' to see your current working directory. If it is not correct, use the 'cd' command to change your directory to the one we created previously. Matlab allows us to type commands in directly, though sometimes this can become tedious. Generally we will write the commands to a file, and ask Matlab to execute the commands in the file. This is why we use the text editors. Matlab first looks in the directory it is currently in for these special files, so make sure you operate Matlab in your directory and that you save the files you wish Matlab to run in the same directory. Files Matlab run end in a '.m' extension and are called m-files.

2. **How to Quit and Get Help:** Whenever you want to quit Matlab, type 'quit' or 'exit' on the command line. You might want to try this to make sure you can exit and restart Matlab properly. Whenever you need help concerning a Matlab command, you may type 'help command-name' at the prompt, where 'command-name' is the function you are having trouble with. For example, standard operations like addition and multiply have a help page, under ops. So type 'help ops' to see what the on line help has to say. Finally, you can type 'demo' to visit the various features of Matlab.
3. **Basic Operations:** The basic unit of data for Matlab is the floating point number. These are stored either as scalars, vectors, or matrices. Matlab can be used much like a calculator. For example, try typing $(45 + 7)/3$

and you should get back the answer,
ans = 17.3333

While Matlab does not usually operate on symbolic quantities like

bels to numbers, vectors, and matrices generated. For example, suppose you type $a = 34/4$

```
we get in return
a = 8.5000
```

Now Matlab has a stored variable 'a'. This variable can be recalled at any time by typing its name at the prompt. To find out all the variable names currently in use, type 'who' at the prompt. Now type in the line, $b = 2.0$;

and notice how Matlab does not report back the value of b. When a semi-colon is placed on the end of the expression, output is suppressed. Check to make sure that b is stored in the memory. Now type, $a * b$ which returns with 17. Besides multiplication, all standard operations are defined. To find out which standard operations are available, type 'help ops'. Complex numbers can be generated by using 'i' and 'j'. For example, type $c = a + b * j$

To generate the complex number 'c'. We can look at the real and imaginary components of the complex number 'c' by using the functions 'real' and 'imag'. Other special numbers include 'pi'. Some other common operators we will use in this course include 'sqrt', 'abs', 'conj', 'angle', 'cos', 'sin', 'log', 'exp'. The following example shows how to find the magnitude and phase of a complex number:

```
>>z=-3-j*4;
>>z_real=real(z);
>>z_imag=imag(z);
>>z_mag=sqrt(z_real^2+z_imag^2)
z_mag=5
>>z_mag=abs(z)
z_mag=5
>>z_rad=angle(z) %Angle in radians;
z_rad=-2.2143
>>z_deg=angle(z)*180/pi %Angle in degrees;
z_deg=-126.8699
```

4. **Vectors:** Now we consider vectors. Vectors in Matlab can be typed directly in as follows, $x = [1 \ 2 \ 3]$; to generate a row vector, while $y = [4; 5; 6]$; generates a column vector. Check to see what x and y look like when they are reported by Matlab. We can address an individual component of a vector by using its index. For example, type $y(2) = 41/7$ at the prompt to change the value of the second element of the vector y. There are many ways to generate vectors. One method we will typically use to generate a vector for time. Type at the command line, $t = 0:0.01:2$; which generates a vector with the first element a 0, the last element 2, with the difference between neighboring elements 0.01. Hence the total length of the vector t is 201. If you wish to check the length of a vector use the command 'length'. Multiplication and other ops are defined where they make sense for vectors as well. So

* t ', since the vectors x and t are of different lengths. Many standard functions will operate on vectors. Vector representations provide the foundation to rapidly create and explore various signals. Consider the simple 10Hz sinusoid described by $f(t) = \sin(2\pi 10t + \pi/6)$. Two cycles of this sinusoid are included in the interval $0 \leq t < 0.2$. A vector t is used to uniformly represent 500 points over this interval and the function $f(t)$ is evaluated at these points.

```
>>t=0:0.2/500:0.2-0.2/500;
>>f=sin(2*pi*10*t+pi/6);
```

The value of $f(t)$ at $t = 0$ is the first element of the vector and is obtained by

```
>>f(1)
ans=0.5000
```

Multiplication of two vectors is one important issue in MATLAB. There are three standard ways to multiply vectors: inner product, outer product, and element-by-element product. The multiplication operator '*' in MATLAB refers to regular vector multiplication. For example, $1 \times N$ row vector times an $N \times 1$ column vector results in the scalar inner-product and $N \times 1$ column vector times a $1 \times M$ row vector results in the outer product which is an $N \times M$ matrix. In most cases where we define signals using vectors we are interested in multiplying these different functions. This multiplication can be implemented by the element-by-element multiplication '.*' which requires the two vector to be the same size. Consider the following example:

```
>>x=[1 2 3];
>>y=[2 3 4];
>>z=x*y' %y' is the vector transpose of y;
z=20
>>A=x'*y
A=2 3 4
4 6 8
6 9 12
>>v=x.*y
v=2 6 12
```

5. **Matrices:** Matrices are entered in a similar manner to vectors. For example, to create a 2x3 matrix A, type in $A = [1\ 2\ 3; 4\ 5\ 6]$

which returns with the result, $A =$

```
[1 2 3
4 5 6]
```

Of course, you can apply multiplication, addition, and other typical operations (for example, $A * x$) so long as the dimensions agree. The size of a matrix can be checked with the function 'size'. Individual

and check what this does to the Matrix A. There are many ways to generate a matrix. For later interest, you may wish to check the functions 'ones' (a matrix filled with ones), 'zeros' (a matrix filled with zeros), and 'eye' (identity matrix). Other matrix generation functions include 'rand' (random matrix), 'compan', 'hilib', and so on.

6. **Plotting and MATLAB m-files:** Now we are ready to use the file editor. Create a file called t1.m and type in the following text,

```
>>t = 0:0.01:5;
>>y = sin(4*t);
>>plot(t,y);
```

The first two are familiar commands, creating vectors of the same length. The last command will ask Matlab to create graph with t on the x-axis and y on the y-axis. However, all of these commands are in the text file. To have Matlab execute these commands, save the file as t1.m, and then type t1 at the command line. When typed, Matlab first looks for a variable called t1, and failing that, looks for a function or m-file with this name. Provided Matlab's current directory is the same as the directory where you saved the file, it will find t1.m and execute the commands in it. A plot should come up on the screen. It is the graph of a sinusoid. You can change the style the style of the line by using a third option to "plot", such as `plot(t,y,':')`.

Note that the colon is enclosed in single quotes and separated from the other arguments by a comma. You should get a dotted line. You can make it red by using `plot(t,y,'r:')`.

Other options that you might want to try are: solid red 'r', dashed green 'g :', dashdot blue 'b-.' or try `help plot` for more information. We can label the plot for more clarity. After the plot command in your m-file, enter the lines:

```
>>xlabel('Time in seconds');
>>ylabel('Sine of 4 time t');
>>title('A tutorial exercise for ECE 366');
```

Once these lines are entered, save the file and run it once again. To save a post script file version of this plot so you may print it out, use the command 'print', in particular, use 'print -dps filename'. Such a file may now be sent to one of the laser writers.

You can also use the 'subplot' command to plot more than more function in the same figure. For example,

```
>>t = 0:0.01:5;
>>y = sin(4*t);
>>z=y.^2;
>>subplot(211);
>>plot(t,y);
>>subplot(212):
```

will plot the sine function in the upper panel, and the ' \sin^2 ' function in the lower panel.

7. **Writing Functions in MATLAB:** MATLAB functions are very similar to MATLAB script files. In this section, we will create a *text* file that will be a MATLAB function. The function file you create should be named `example2.m`. The name of the function and the name of the file must be the same. This file should contain the following MATLAB commands:

```
function y=example2(x)
%
% example2:  this is where the help entry goes
%
y = exp(-abs(x)).*sin(2*pi*x);
return;
```

Start MATLAB and enter the following commands at the MATLAB prompt:

```
>> help example2
>> t=0:.01:6;
>> plot(t,example2(t));
```

If you did everything correctly, you should see the help text in response to “`help example2.`” Notice that we’ve now added a new command to MATLAB that can be used as if it were a built-in function. If you get errors, you must correct the file `example2.m`, save it, and then rerun the commands above.

The MATLAB on-line help system has a nice write-up of functions and how to handle various things like returning more than one value, checking the number of arguments, etc. The following commands will give you more help with writing scripts and functions.

```
>>> more on
>>> help function
>>> help script
```

8. **For loops:** A for loop is a construction of the form
for `i=1:n`, `<program>`, end

Here is a sample program for matrix addition.

```
>>function c=add(a,b)
```

```
>>% c=add(a,b) This is the function which adds
```

```

>>% function a+b.
>>[m,n]=size(a);
>>[k,l]=size(b);
>>if m~=k | n~=l,
>>    r='ERROR using add: matrices are not the same size';
>>    return,
>>end
>>c=zeros(m,n);
>>for i=1:m,
>>    for j=1:n,
>>        c(i,j)=a(i,j)+b(i,j);
>>    end
>>end

```

9. **While loops:** A while loop is a construction of the form

while <condition>, <program>, end

where condition is a MATLAB function. The program program will execute successively as long as the value of condition is not 0. While loops carry an implicit danger in that there is no guarantee in general that you will exit a while loop. Here is a sample program using a while loop.

```

function l=twolog(n)

% l=twolog(n). l is the floor of the base 2
% logarithm of n.

l=0; m=2; while m<=n
    l=l+1;
    m=2*m;
end

```

Examples

Try the following examples on your own.

1. Perform the following operations in MATLAB:

(a) Generate the following *column* vectors as MATLAB variables:

$$x = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \quad y = \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$

(b) Using the computer, issue the following MATLAB commands:

- i. $x * y'$
- ii. $x' * y$
- iii. $x .* y$

(c) Answer the following questions:

- i. Justify the dimension of the matrix produced by each of the

ii. Illustrate your answer by computing each result by hand.

2. In this exercise, we will learn to plot a continuous-time function by creating a MATLAB vector of *samples* of the function.

Let $x(t) = A \cos(2\pi f_0 t)$, where $A = 20$ volts and $f_0 = 1$ MHz. We wish to form the MATLAB vector of *samples* of $x(t)$ as

$$y[n] = x(nT_s),$$

where T_s is chosen such that we get exactly 12 samples/period of $x(t)$.

Do the following in MATLAB:

- (a) Create a MATLAB vector y containing samples (at 12 samples/period) of 10 periods of $x(t)$. (Therefore, your vector should have a total of 120 elements.)
- (b) Plot the vector y . Label both axes and title the plot.

3. Generate new vector representing the signal

$$z(t) = x^2(t)$$

Using the MATLAB `subplot` command, produce a single page containing the following two plots:

- (a) $x(t)$ (Top plot)
- (b) $z(t)$ (Bottom plot)

Again, be sure to label all axes of both plots.

4. One of the strengths of MATLAB is that most of its commands work with complex numbers. Perform the following computations in MATLAB:

- (a) Compute the value of j^j . Is the result what you expect?
- (b) Using the `polyval` command, evaluate the polynomial

$$P(x) = 4 + 3x^3 - (1 - j)x$$

at $x = j^j$.

5. Create new MATLAB functions for each of the following functions:

- (a) `step(t)` function:

$$\text{step}(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases}$$

Your `step(t)` function should take a single argument, which may be either scalar- or vector-valued. Note: you will need to be clever in order to handle vector-valued inputs. The MATLAB command `find` may be useful in writing this function. Alternatively, you may decide to write a loop inside your function. See the MATLAB `for` command for help on writing loops. Other commands that may be useful: `length`, `size`, `zeros`, and `ones`. These are

(b) `ramp(t)` function:

$$\text{ramp}(t) = \begin{cases} t & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases}$$

Your `ramp(t)` function should take a single argument, which may be either scalar- or vector-valued. The same comments as for the `step(t)` command apply here as well. Note: MATLAB commands may call each other, so a clever way to implement the `ramp` function would be to construct it from the `step` function you wrote in part 5a.

(c) To test your functions, plot the following:

$$f(t) = 5r(t + 3) - r(t + 1) - 3r(t) + 5r(t - 1) + 3u(t - 2)$$

for $t \in [-5, 5]$. In MATLAB, use `t=-5:.01:5` for the time-axis, use `ramp(t)` for $r(t)$ and `step(t)` for $u(t)$.