MATLAB Fundamentals

MATLAB is an abbreviation of Matrix Laboratory, which is a mathematics software package optimized for working with vectors and matrices. It is a powerful tool for data analysis and plotting, and much more. It is available on all our lab's PCs, and is very worthwhile to learn, since it can be applied to a tremendously broad range of scientific applications. We'll start by learning to manipulate MATLAB vectors and create plots.

0 MATLAB Help

Almost the most important and useful thing in MATLAB is its "help" feature. To learn about the details of using any command whose name you know, type:

```
help 'commandname';
```

At the end of the help information, you'll also get a very useful list of related commands. If you don't know the command name, and need to try finding it out using a related keyword, use

```
lookfor 'keyword' -all;
```

If you're using a Windows-based implementation of MATLAB , you can also access the same information via the Help menu.

1 Vectors and Matrices

Try the following commands and observe the results:

```
> a = 1:10

> b = 0:0.4:1.5

> c = [2 5 8 2 ; 2 5 0 0 ; 3 4 2 1]

> d = a'*b;

> e = ones(100,1);
```

Note the difference between placing a semicolon at the end of a command, and leaving it out — in the first case output is suppressed, which is crucial for working with large matrices. Other useful commands to try are `linspace` and `logspace`.

Example: Generate a sine function with an amplitude of 1.5 and a frequency of 5kHz, and lasting 0.1sec. You'll first need a vector representation of time. This should be closely-spaced enough to have at least 10 points per cycle of the sinusoid.

Pay attention to operations defined both as matrix operations and element-by-element. To specify an element-wise operation, put a dot in front of the operator. If the four commands below are run on the variables defined above, the 1st and 3rd will not work, but the 2nd and 4th will:

```
> c*b;

> c*b';

> c.*b;

> c(1,:).*b;
```

2 Indexing

Indexing, or selecting the desired elements of a vector or matrix, is crucial in this environment. The entries of MATLAB matrices are always numbered starting with 1, and some have special names. Some examples follow:

`> c(:,3)` is the third column of matrix `c`.

`> c(3,:)` is its third row.

`> c(4,1)` will produce an error, since `c` only has three rows, as we've defined it.

`> d(2:7,:)` gives rows 2-7 of `d`

`> d(:,2:end)` refers to columns 2 through the last one.

These expressions can be used within other operations and calculations, letting you have powerful access to various subsets of your data.

Example: Assign a small section of your sinusoid (about one or two cycles) to a new variable, and make an equal-length variable section of the time vector. The "Workspace" pane in MATLAB will show you the size of various variables, or you can use `size(variable)` or `length(variable)`.

3 Plotting

For making plots, the most basic syntax is:

```
> plot (x_vector,y_vector);
```

This must be done with vectors of the same length and plots them one vs. the other.

`> plot(x);` simply plots the data in vector `x` vs. the index of each data point.

`> loglog(x,y);` and `semilogy(x,y);` produces non-linear plotting axes.

`> hold;` prevents a plot from being re-drawn when the next `plot` command is issued, letting you plot multiple curves on the same set of axes.

`> plot (x,y,'r.:');` plots using a red dotted line, placing a dot at each data point.

```
> axis ([-3 2 -1 1);
```

sets the limits of the current plot such that the range of the x-axis is from -3 to 2, and the y-axis from -1 to 1.

```
> figure(2);
```

creates a blank new figure window numbered 2, or if the window already exists, makes it active.

Example: Plot your full-length 0.1sec sinusoid (vs. its time vector), and the one- or two-cycle segment, in two different plot windows. Adjust the axes of the full-length plot such so you can see the oscillations of the sinusoid.

Now, see what happens if you re-generate the sinusoid using fewer time points. How many points per cycle do you need to be able to recognize it as a sinusoid? How many points do you need for it to appear at the correct frequency?

Here are some example commands for adding labels, gridlines, titles, etc. to your plots:

```
> grid;
> title('figure title');
> xlabel('label for x-axis');
```

As always, use `help` to get the details of usage for any of these commands (in this case `help plot`).

4 m-files and Functions

Any time you have a series of MATLAB commands that you are likely to re-use, it's worth saving them to an m-file, so named because their file names always end with "`.m`". Such files can be run by simply by typing the filename (without the `.m`) at the MATLAB prompt.

A special case of an m-file is a MATLAB ***function***. This is analogous to functions in other programming languages, and can be called from other functions, m-files, or the MATLAB prompt with parameters passing to and from it. If you are unfamiliar with the concept of functions in programming, that is beyond the scope of this introduction, but help is available from your lab instructor.

A MATLAB function must always adhere to the following pattern:

```
function [output_par]=myfunction(input_par_1, input_par_2)
 command1;
 result1=input_par_1+input_par_2;
 command2;
 output_par=result1;
```

The file containing this function must be named `myfunction.m`. As written, it requires two input arguments, and returns one output argument (i.e. you'd use it something like this: `outvar=myfunction(invar1, invar2)`. The numbers of intput/output arguments can, of course, be changed.

5 More Examples

To get more practice with MATLAB , work through these few examples related to Lab Module 1. These don't need to be turned in, but it's highly worth your while to do them.

5.1 Voltage Divider

1. Create a vector of resistance values R_1, in the range 1-10kΩ.
2. If R_1 is used to make a voltage divider with a second resistor R_2 having a value of 5kΩ, write down two equations representing the output voltage in the two cases, where R_1 is "upstairs" and "downstairs."
3. Plot both curves on the same set of axes. This should give you more insight into the sensitivity of resistive divider setups?

5.2 Photodiode

The analytical equation for the *i-v* curve of a diode is

$$I = I_0(e^{qV/k_BT} - 1) \ ,$$

in which V is the applied voltage, q, k_B and T are the elemental electrical charge, Boltzmann's constant, and absolute temperature, respectively (the values are $q = 1.6 \times 10^{-19}$C, $k_B = 1.38 \times 10^{-23}$J/K, and $T = 300$K). I_0 is the *saturation current* in reverse bias, which depends on the diode material parameters and how the p-n junction is made.

1. Start with a vector of voltages, using a range and spacing that will include the transition region of a diode *i-v* curve, and generate a vector of current values based on those voltages, assuming a value for I_0 of 1μA. Plot this curve and set the plot axes to be able to see clearly the region where all the "action" is.
2. Input your measured data for a diode in darkness either by directly entering the vectors, or by first entering them into a file, and using the `load` command. Plot this on the same figure with the ideal curve from above and compare. What is the key difference from the theoretical diode equation? What is the physical cause?
3. How can you change the diode equation to model the series resistance R_s of the diode? Try using different values of R_s to see if you can get the ideal curve to more closely approximate the data you took.
4. (time permitting) Create an m-file with a function that computes the ideal diode *i-v* curve, given a vector of voltage values, and a series resistance R. Show it to your lab instructor.