

Getting Started with Matlab

In this course you will be using Matlab to perform numerical computations for the problem sets¹. Matlab is a matrix programming language: a variable is considered a matrix by default, and matrix operations (such as matrix multiplication, transpose, inverse, etc.) are built in. Since it is a programming language, it gives you a lot of control over its calculations, but you must be rather precise in telling it what to do. This short guide is intended to help you get started using Matlab, but it only scratches the surface of what you can do with it. Much more information can be found in the Matlab manual, which also contains a very useful tutorial section.

Getting started

You can use Matlab either through the Unix system by writing the command 'matlab' at the unix prompt, or through the PCs. If you are using the Windows version of Matlab, many operations can be done using the pull-down menus. However, for all platforms, you can type commands directly to the Matlab interpreter at the >> prompt.

A very useful command is *help*. If you just type *help* and hit return, you will be presented with a menu of help options. If you type *help <command>*, Matlab will provide help for that particular command. For example, *help hist* will call up information on the *hist* command, which produces histograms. In addition, you can use the *Help* menu at the top of the Matlab window to get the *Table of Contents* and *Index* of commands.

You might have to change the working directory (using the command *cd*) to tell Matlab where your files are (or will be) located. The commands *pwd* and *dir* may also be useful for file management.

To end a Matlab session, type *exit* or *quit*.

Working with variables

There are two main things to remember: First, Matlab is case sensitive, so the variable **X** is not the same as **x**. Second, variables in Matlab are treated as matrices. For example, the command **X*Y** is a matrix multiplication, and works only if the two matrices are conformable. To perform array multiplication, type **X.*Y**. This multiplies the (a,b) element of **X** with the (a,b) element of **Y** for all (a,b), and so requires the two matrices to be of the same size. **X'** means the transpose of **X**.

Suppose **a** and **b** are scalars (1×1 matrices). You can refer to the (a,b) element of the matrix **X** by **X(a,b)**. In a similar vein, **X(:,b)** refers to the b'th column of **X**, and **X(a,:)** refers to the a'th row of **X**. You can get even fancier: **X(1:10,b)** refers to the 10 by 1 vector comprised of **X(1,b)** through **X(10,b)**. Suppose that **i** is an $m \times 1$ vector of integers between 1 and n (repetitions are ok), and **X** is an $n \times k$ matrix. Then **X(i,:)** is the $m \times k$ matrix formed by selecting rows of **X** according to **i**.

¹ You are free to use any other software, of course, as long as you get the right results. This is just a recommendation and my own personal preference.

You can also perform logical indexing of a matrix. If \mathbf{i} is an $n \times 1$ vector of logical (true-false) values, \mathbf{X} is an $n \times k$ matrix, $\mathbf{X}(\mathbf{i},:)$ returns the matrix formed by only including the rows of \mathbf{X} that correspond to a value of 1 in the vector \mathbf{i} . As an example of a logical vector, suppose that \mathbf{a} is an $n \times 1$ vector of numbers. The expression $(\mathbf{a} > \mathbf{0})$ returns an $n \times 1$ logical vector; the i 'th element of $(\mathbf{a} > \mathbf{0})$ is equal to true if $a(i) > 0$ and is equal to false otherwise. A vector of ones and zeros can be turned into a logical vector by using the *logical* command.

For small matrices, we can define them by typing something like:

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

which will put the following matrix in Matlab's memory:

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

To define a new matrix it will sometimes be useful to use a command along the lines of

```
x = [ones(n,1) z];
```

This will create a matrix \mathbf{X} by putting an $n \times 1$ vector of ones in front of the matrix \mathbf{z} . The variable \mathbf{z} must have exactly n rows.

Note that some commands, including *sqrt*, *log*, and *exp*, perform calculations element-by-element (There are different commands for the matrix counterparts, but you probably won't be needing those).

Getting stuff done

There are two ways to get things done in Matlab: Interactive and Semi-Batch.

Interactive Mode

In this approach, you just sit at the computer, type a command at the `>>` prompt, then, when it is done, type another command, and so on, until you have the answer you want. There are a couple of things to keep in mind when doing this.

First, if you add a semicolon `;` at the end of a command before hitting the return key, Matlab will perform the calculation but not return the result to the screen. This can be very useful. For example, suppose \mathbf{x} is a 1000 by 1 vector of numbers, and you want to create a new vector \mathbf{y} as follows: $\mathbf{y} = \mathbf{3} + \mathbf{2} * \mathbf{x}$. This will take each element of \mathbf{x} , double it, and add 3. If you don't include the semicolon at the end, Matlab will actually list the entire \mathbf{y} vector it calculates on the screen.

Second, as you go along making calculations and defining new variables as in the previous example, Matlab will keep these variables in its memory. To see which variables are currently in Matlab's memory, type *who*. The alternative command *whos* lists the variables in memory plus some additional information about the variables.

Third, you may want to keep track of the operations you perform and the things that get printed to the screen. To do this, type *diary <filename>*. Anything that appears on the Matlab screen after you issue the *diary* command will also get written to the text file *<filename>*, until you type the command *diary off*. If there is already a file called *<filename>* in the working directory, the screen output will get appended to the end of that file.

Fourth, after performing a set of calculations, you may end up with some possibly large variables that you wish to save for later use. The command *save <filename> y Y X* will save the variables *y Y X* to the file *filename.MAT*. The *.MAT* file is a Matlab storage file. It is a binary file, not a text file, so you cannot look at the contents with a text editor. Instead, you will have to issue the Matlab command *load <filename>*, to tell Matlab to read the file and put it back in its memory.

Below are some other commands you may find useful. Remember, you can always use the help facility or consult the manual to see how to use these and any other commands.

randn	log	clear	sqrt	rand	exp	size	plot
if	ones	length	hist	for	zeros	mean	min
while	eye	median	max	kron	diag	cov	sort
chol	cd	std	find	inv	dir	pwd	fmin
fmins							

Semi-Batch

You can automate things by typing in all the commands you want to perform, in order, in a text file that has the extension *.m*. These are called “M-files” and you should use them extensively when working on the problem sets. Suppose you write the following program in an M-file called *prog.m*:

```
% A short sample program
x = randn(100,1);
y = 3+2*x;
hist(y,20)
```

At the Matlab prompt, you would type **prog** and hit return. Matlab would read **prog.m**, then perform the commands in order: first generate 100 independent standard normal draws, then multiply those by 2 and add 3, then graph the histogram of these draws (using 20 bins). The first line is ignored because it starts with *%*; it's a good idea to comment your code extensively with *%*².

Keep in mind that the M-file must be a text file, not, for example, a Word file (If you write the M-file in a word processor, just remember to save it as a text file). When using Matlab for Windows, you can use the **New M-file** and **Open M-file** options in the **File** menu to work with M-files.

When handing in problem sets, you should include a copy of the M-files used to solve them.

² If you are experiencing difficulties printing an M-file that starts with *%*, try adding an extra blank line at the top of the file. Postscript printers may misinterpret the file type if the M-file starts with *%*

Functions

Matlab lets you define new functions, which can then be called from within a Matlab session or from a batch file. The function needs to reside in the current working directory or along Matlab's search path (which can be displayed and modified with the command *path*).

Here is an example of a function in Matlab. The file containing the function should be called **func1.m**.

```
function y = func1(a,b);  
z = a^2;  
y = z - b;
```

This function takes two arguments **a** and **b**, and returns **y**, equal to $a^2 - b$. If we wanted to return both **z** and **y**, we could replace the first line of the function definition with

```
function [y,z] = func1(a,b);
```