

# A quick Matlab tutorial

Michael Robinson

## 1 Introduction

In this course, we will be using MATLAB for computer-based matrix computations. MATLAB is a programming language/environment that provides easy access to a number of standard libraries: LAPACK, EISPACK, BLAS. These libraries are written in Fortran, and there exist a number of standard interfaces to other languages as well. However, one usually spends a bit of effort interacting with the libraries, which obscures the underlying linear algebra. MATLAB permits us to avoid the formalities and instead focus on the mathematics.

There are a number of ways to run MATLAB, and you can choose whichever suits you best: as a student, you can buy a student license relatively cheaply; you can use in on the SAS computer labs; or you can use the freely available program GNU Octave which is essentially identical. (If you need help with getting Octave installed, I can help. In the places where they differ, Octave is a slightly better language. When I refer to “Matlab” I mean either MATLAB or Octave. I’ll assume that you’ve never used Matlab, but we’ll quickly ramp up your Matlab skills.

## 2 Basic commands, vectors, and matrices

The first command you should be aware of is

```
>> help
```

which is useful for finding out about commands and their arguments. You should be aware that nearly everything in Matlab is self-documenting, so you can figure out much of the language yourself easily.

Everything in Matlab is an vector or a matrix (I’m slightly cheating here. There are more general objects as well, but we won’t encounter them in this course.) Vectors can be named almost anything except certain reserved words. To make a vector come into existence, simply give it a value:

```
>> x=10
```

creates a variable  $x$  and assigns the value 10 to it. You can also make vectors pretty easily:

```
>> y=1:.3:25
```

makes a row vector whose entries (from left to right) start at 1, increment by 0.3 and are no larger than 25. Here's how to specify elements of a matrix:

```
>> a=[1 2 3; 4 5 6; 7 8 9; 10 11 12]
```

This is a 4x3 matrix with those specified values.

Matlab is a useful calculator. Most of the usual operations you might want are available:

```
>> cos(((sqrt(3)+1)/7)^3)
```

What's nice about Matlab is that many of these also work for matrices. You can avoid writing loops in Matlab by working with entire arrays of numbers. For instance, here's how one might compute the squares of the first 10 odd numbers:

```
>> (2.*(1:10)-1).^2
```

Notice two things with the above expression: namely the array multiplication is written (`.*`) and the array exponentiation is written (`.^`). This is to avoid confusion with matrix multiplication (`*`) and powers (`^`). Many other functions are also “vectorized” which means that they work on elements of a matrix or vector, such as sine or cosine.

Matrices can be loaded into Matlab in a variety of ways. The easiest is to simply list the entries with semicolons to separate rows:

```
>> theta=pi/4;
>> a=[cos(theta) sin(theta); -sin(theta) cos(theta)]
```

If you want to know how big a matrix is, use the “size” command:

```
>> size(a)
```

will tell you that  $a$  is 2x2. For small matrices, this isn't very useful, but for larger ones it's handy. If you want a matrix with random entries, for instance a 7x4 matrix, use

```
>> b=rand(7,4)
```

To get a new random matrix of the same size, you can use

```
>> c=rand(size(b))
```

There are other useful matrix-making commands, such as “zeros”, “ones”, “eye”, and “meshgrid”. Read about them! (Also, Matlab can read almost any data file format you can imagine, like JPEG images! Look into the documentation or ask me if you're interested...)

We can multiply two random matrices

```
>> rand(5,3)*rand(3,4)
```

In Matlab, we have easy access to gaussian elimination (solving systems of linear equations). It's essentially matrix division, so Matlab represents it by “\”. In particular, if you want solve  $Ax = b$  for  $x$ , we'd execute

```
>> x=A\b
```

Imagine that we're writing  $A^{-1}b$  as  $\frac{b}{A}$ , and you'll remember to use the backslash. (If you wanted to write  $bA^{-1}$  for  $b$  a row vector, you can use

```
x=b/A
```

which result in  $x$  being a row vector...)

Matlab also provides a way to calculate matrix inverses using the "inv" command. Matrix inversion is not (numerically) the same as gaussian elimination.

## 2.1 Matrix example

In this example, we examine this a little (courtesy of the MATLAB documentation). Type the following into Matlab (try to figure out what each line does!):

```
>> n = 500;
>> Q = orth(randn(n,n));
>> d = logspace(0,-10,n);
>> A = Q*diag(d)*Q';
>> x = randn(n,1);
>> b = A*x;
```

Now, you've got a matrix  $A$  (yes, it's invertible) with a random vector  $x$ , and  $b$ , their product. The task is to recover an approximation of  $x$ ... Obviously,  $x = A^{-1}b$ , so try

```
>> y = inv(A)*b;
```

Now, using gaussian elimination, we should execute

```
>> y2=A\b;
```

Now these two aren't very different, but compare the error

```
err = norm(y-x)
```

with

```
err2 = norm(y2-x)
```

It's not so different... But the residuals

```
res = norm(A*y-b)
```

and

```
res2 = norm(A*y2-b)
```

differ significantly!

### 3 Visualization

One of the big strengths of Matlab is easy access to visualization. (I would have had you programming in Fortran otherwise!) It's pretty easy to plot things in 2 or 3 dimensions... For instance,

```
>> x=0:0.1:2*pi;
>> figure
>> plot(x,sin(x))
>> title('A sine wave');
>> xlabel('Angle (radians)');
>> ylabel('Value of sine');
```

should give you a sine wave with some axis labels. The “plot” command is a parametric plotting function, so you can also do something like

```
>> t=0:.1:10;
>> x=sin(t.^2);
>> y=cos(t);
>> figure
>> plot(x,y,'r');
```

And 3d plotting is easy too:

```
>> t=0:.1:6*pi;
>> x=sin(t);
>> y=cos(t);
>> z=t;
>> plot3(x,y,z,'+');
```

should plot points on a helix.

### 4 Function files and scripts

If you've done a bit of work in Matlab, sometimes you want to save your commands so that you don't have to type them all in again. (And also to submit your assignments!) The way to do this is to list the commands you want executed one after another in a plain text file with extension “.m”, called a “script”. You can (and should) insert comments with a percent sign to clarify what you've written. MATLAB has a convenient editor for writing scripts. You can start it by typing

```
>> edit
```

and then cutting and pasting commands into the window. You can later execute the commands in a script by typing its filename (without the “.m”) at the Matlab prompt.

Matlab provides the ability to define new functions, which is an important way to manage complicated programs. A function has a number of inputs and

a number of outputs... Like scripts, functions are stored in plain text files with the extension “.m”. However, the first line of a function file must begin with the word “function” and then an input/output specification. For instance, the following could be stored in “helix.m”

```
function [x,y,z]=helix(t)
% Compute points on a parametric curve (a helix)
% Input: t = matrix or vector of parameter values to evaluate
% Output: x,y,z = coordinates of points computed

x=cos(t);
y=sin(t);
z=t;
```

Notice that the function name in the top line must match the filename! If this file is stored in Matlab’s current directory, you can call it:

```
>> [x,y,z]=helix(0:0.1:3*pi);
>> plot3(x,y,z);
```

## 5 Conditionals and loops

Although Matlab can often avoid loops through vectorization, it does have loops and conditionals much like other programming languages. They don’t make too much sense outside of a function definition or a script file, so I’ll assume you’re writing these commands in a script or function.

We’ll start with the “if” statement. Here’s an example:

```
if( a == 1 )
    x=5
else
    x=1
end
```

which says, “if  $a$  is 1, then set  $x$  to 5, otherwise set  $x$  to 1”. Notice that to check equality, we use “==”, but to assign a variable a value, we use “=”. This is a common source of frustration for new Matlab programmers! Instead of “==”, we can check for non-equality by “~=”, or use “<” or “>” for less-than or greater-than. You could also leave off the “else” clause entirely

```
if( x == 1 )
    disp('a must have been something other than 1')
end
```

So much for “if” statements. Loops are introduced with “for” statements. Here’s an example of a “for” loop in Matlab

```
for j=[2 3 5 7 11 13]
    j.^2
end
```

It executes the expression  $j^2$  for the following values of  $j$ : 2, 3, 5, 7, 11, and 13 (in that order). You can put any row vector you like in there, though:

```
for evens=0:2:26
    evens
end
```

Sometimes you just want to repeat something a bunch of times with some random data... For instance, here's an approximation to  $\pi$

```
points_in=0;
for i=1:10000
    x=rand;
    y=rand;
    if( sqrt(x.^2+y.^2) < 1 )
        points_in=points_in+1;
    end
end
disp(4*points_in/10000)
```