# Matlab Tutorial for Computational Methods CE 30125

prepared by
Aaron S. Donahue
adonahu1@nd.edu

This document is a list of basic and key Matlab commands that will be helpful in the Computational Methods course. The document is broken up into sections, each focusing on a particular aspect of Matlab.

*Disclaimer:* This is by no means a complete list of Matlab commands you may wish to use in this course. This is meant to be a guide to help you work towards working with Matlab comfortably.

## 1 Introduction

The name Matlab stands for "Matrix Laboratory", and the program Matlab is ideally suited for matrix and vector manipulations. Matlab is capable of implementing simple and complex codes, as well as simple arithmetic operations, symbolic manipulation, statistical analysis and data visualization. In the Computational Methods course we are most interested in using Matlab to write codes for solving Ordinary and Partial differential equations and visualization of data and results. When used properly Matlab can be a very powerful tool.

## 2 Basic Operations

### 2.1 The Help Window

The help window for Matlab is very useful. If you know a command you want to use or have an idea of what it may be called you can use the search engine and generally it will be found. Once you have your command in the help window the first section will be the syntax. This is the generalized format for how the command can be used, and generally there are a few ways to use a command so you will see many different examples highlighting the different options. At first it can seem a bit confusing, but the more you use the help window the easier it will become to understand.

The next part of the help window will be a full description in words, this is useful for making sure that the command is truly what you need. Following the description is generally a few specific examples. You can copy and paste these examples into your Matlab command prompt to see the command at work. This are extremely useful for understanding how to use a command or function since all you need to do is replace the components in the example with your own components to get the command working for you.

There are two ways to access the help window. The first is to simply type 'doc' in the command prompt as follows

```
>> doc
```

the second is to go to the help menu drop down from the Matlab tool bar and select 'Product Help'. Both will open the help window. Try it out, even with the commands given in this tutorial.

## 2.2 Arithmetic and Symbols

```
>> 3 + 2

ans =
       5

>> 3 - 2

ans =
       1

>> 3 * 2

ans =
       6

>> 3 / 2

ans =
       1.5

>> 3 ^ 2

ans =
       9

>> sqrt(3)

ans =
       1.7321

>> pi

ans =
        3.1416

>> exp(1)

ans =
        2.7183

>> abs(-1)

ans =
```

```
     1
```

This is just a sample of the arithmetic operations one can do in Matlab, in general the syntax is the same as it would be in Excel or on your Calculator. If you can't fiugre it out use the help window.

### 2.3 Naming Variables

The following root code will assign the value of "expression" to the variable with name "name".

```
>> "name" = "expression"
```

For example

```
>> a = 4
```

Now you perform commands on the variable "a" as if it where the number 4.

```
>> a * 2

ans =
       8
```

This is especially useful in programming where a variable may change and you do not want to have to go through your code changing the number 4 to the new value everytime. More on this in the programming section.

*Note:* Names can practiacally be anything as long as it starts with a letter, for example

```
>> im2cool = 4
```

Assigns the value 4 to the variable "im2cool". However

```
>> 2cool = 4
```

Causes Matlab to return an error, because it begins with a numnber. Similarly certain logical commands should can not be assigned names, such as "for" which has a special use in Matlab. If this is the case Matlab will give you an error message explaining this.

### 2.4 Vectors and Matrices

As mentioned above Matlab is ideally suited in working with Matrices and Vectors, and when you become more advanced at programming in Matlab you will want to structure your code to take advantage of this, more on that later. The following are a list of basic operations for constructing a vector or a matrix.

### 2.4.1 Vectors

```
>> a = [0 1 2 3 4]
>> b = [5;6;7;8;9]
```

This will create two vectors, "a" is a row vector that has 5 entries from 0 to 4, while "b" is a column vector with entries from 5 to 9. To create a row vector use a "space", for a column vector use the "semicolon". Another way to quickly create a vector is

```
>> a = 0:1:4
```

This will construct a row vector that starts at 0 at counts up by 1 to until reaching 4

### 2.4.2 Vector Operations

To begin we look at some basic operations for a vector.

```
>> c = a'
```

Will give the transpose of the vector "a" to the name "c", in this case making it a column vector.

```
>> length(a)
```

Will give the length of the vector "a", in this case the answer would be 5.

### 2.4.3 Vector to Vector Operations

Now we look at performing operations between two vectors. Something that is very important to remember here is that there is a difference between vector-vector operations and entrywise operations. The following are vector operations.

```
>> d1 = c + b
>> d2 = c - b
```

Note that for the sum and difference the vectors have to be of the same shape, i.e. same number of elements and the same shape, such as both column vectors. The command 'a+b' would produce an error because 'a' is a row vector and 'b' is a column vector.

```
>> d3 = a * b
```

The multiplication symbol does the dot product between the two vectors, in this case the answer would be 80. Note that for the dot product to work it must be a row vector multiplied by a column vector, both with the same number of elements. Which begs the question, what happens if I multiply a column vector by a row vector. Doing this will produce a matrix, which will be covered in more detail in the next section.

What if I want to multiply each entry in one vector by each entry in another vector individually? We call this an entrywise operation and to do this you must place a period, '.', before the operation, as follows.

```
>> d3 = c .* b

ans =

     0
     6
    14
    24
    36

>> d4 = c ./ b

ans =

         0
    0.1667
    0.2857
    0.3750
    0.4444
```

Now the ".*" operation will produce a vector with each entry as the multiplication between the equivalent entries in vector "c" and "b". Note that in order to do entrywise operations the two vectors have to be the same size, i.e. the same number of entries and either both column vectors or both row vectors.

## 2.5 Matrices

The commands to work with matrices are very similar to the commands used with vectors, but given the extent that matrices are used there are many more useful commands to employ. First, to define a matrix

```
>> A = [0 1 2;3 4 5;6 7 8;9 10 11]

A =

     0     1     2
     3     4     5
     6     7     8
     9    10    11
```

Produces the 3 x 3 matrix 'A' with entries from 0 to 11. Notice how the syntax is a combination of the row vector formulation, 'spaces', and the column vector formulation, ';'. This is really like telling Matlab that A is a column vector of row vectors. The following are basic matrix commands.

### 2.5.1 Matrix Operations

```
>> size(A)
```

```
ans =

     4     3
```

Will give the size of A, the first value is the number of rows and the second value is the number of columns.

```
>> B = A'

B =

     0     3     6     9
     1     4     7    10
     2     5     8    11
```

The apostrophe will assign the transpose of A to the label B.

```
>> A(2,2)

ans =

     4

>> A(2,:)

ans =

     3     4     5

>> A(:,2)

ans =

     1
     4
     7
    10
```

The use of the parantheses will select specific components of a matrix. In this case A(2,2) produces the entry of A in the second row and second column, A(2,:) produces the second row of A and A(:,2) produces the second column of A. You can be even more specific, such as

```
>> A(2:4,[1 3])

ans =

     3     5
```

```
     6     8
     9    11
```

Which will produce the entries of the second through fourth row and the first and third column.

```
>> A(:)

ans =

     0
     3
     6
     9
     1
     4
     7
    10
     2
     5
     8
    11
```

If just a colon is used you will get a column vector starting at the first point and tracing down each column of the matrix.

```
>> inv(A)
```

Will produce the inverse of A, note for this to work A must be a square matrix.

```
>> rref(B)

ans =

     1     0    -1    -2
     0     1     2     3
     0     0     0     0
```

Will produce the row-reduced-echelon-form of a matrix. Lastly

```
>> C=sparse(A)
```

will assign to the label C a "sparse" version of the matrix A. A sparse matrix is a matrix where all zero entries have been removed, and the logic of the structure tells Matlab that where an entry is missing there would have been a zero. This is very useful for memory allocation and fast matrix operations.

*Spoiler Alert!!!* Joannes will bring up Sparse matrices often and many of the later homeworks will require that you use the sparse command to get an answer in a reasonable amount of time. In

short, this command is one of the most used commands in the class.

Some other matrix commands that come in handy when programming are

```
>> A = zeros(n,m)
```

will produce an n x m matrix of all zeros. (This is useful for memory allocation).

```
>> A = ones(n,m)
```

produces an n x m matrix of .... yep..... ones.

### 2.5.2 Matrix to Matrix Operations

Matrix to matrix operations are similar to the vector to vector operations, in that entrywise operations nessistate the inclusion of a period, '.', before the operation, while operations without a period will be interpreted as true matrix operations. Thus

```
>> A * B

ans =

     5    14    23    32
    14    50    86   122
    23    86   149   212
    32   122   212   302
```

Is the matrix multiplication between the 4 x 3 matrix A and the 3 x 4 matrix B producing a 4 x 4 matrix. On the other hand

```
>> A .* B'

ans =

     0     1     4
     9    16    25
    36    49    64
    81   100   121
```

produces the 4 x 3 entrywise multiplications between the elements of A and the transpose of B. Note that for entrywise operations the two matrices must be of the same size and shape. A.*B would produce an error. Similarly

```
>> A ./ B
```

produces the 4 x 3 entrywise divisions between A and the transpose of B. Often we want to multiply an inverse of a matrix by another matrix, you can use the 'inv' command to produce the inverse and then multiply by that, however Matlab has a trick for multiplying by an inverse quickly. Say you want to do the following $c = M^{-1}S$, you can type

```
>> c= M \ S
```

into Matlab and this will multiply S by the inverse of M.

### 2.5.3 Matrix to Vector operations

It is also possible to work with matrices operating on vectors and vice versa. All the same commands apply, however entrywise operations cannot be used since the matrix and vector will not be of the same shape.

# 3 Supressing Output and Cancelling an Operation

Now that we have worked with numbers, vectors and matrices we may get tired of having Matlab always output what we typed in. Any line written in the command prompt that is finished with a semicolon, ';', will have the output supressed, for example

```
>> a = [1 2 3 4]
```

has the output

```
a =

     1     2     3     4
```

However,

```
>> a = [1 2 3 4];
```

has the output

???? Exactly, no output. This comes in handy when assigning large vectors or matrices that have many many entries. If Matlab printed to the screen for these large matrices it could take forever. Try this, type

```
>> A = zeros(10000,10000)
```

Matlab just goes crazy listing zeros. So in these cases you likely want to supress your output with a semicolon. Also at this point you want to hold down 'cntrl' and press 'c'. This will cancel any operation that Matlab is performing. It is the 'kill' switch. Sometimes your programs can get caught in endless loops that will never end, this command allows you to stop Matlab and maybe find your problem. Or sometimes you just want to end what Matlab is doing.

## 3.1 Figures

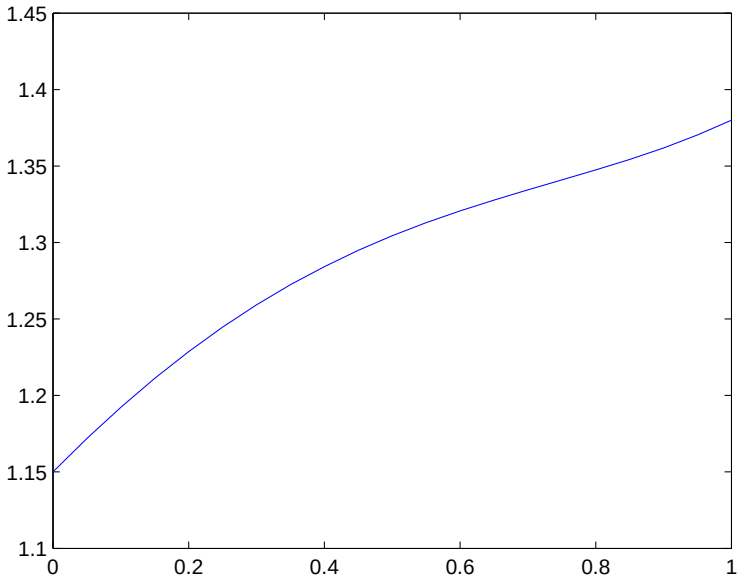
Matlab is extremely well suited for creating professional grade figures. In this class we expect all figures to be produced in Matlab. To plot two vectors use the "plot" command.

```
>> plot(a,b)
```

will plot the points defined by "a" against the points defined by "b". For example

```
>> x = 0:0.05:1;
>> f = 0.15*x.^4-0.1*x.^3-0.27*x.^2+0.45*x+1.15;
>> plot(x,f)
```

After defining the "x" and "f" vectors the "plot" command will plot the x values along the horizontal axis matched to the f values along the vertical axis. Note that here in defining "f" we have used the elementwise operation ".∧" for the powers of 'x'. The graph would look like



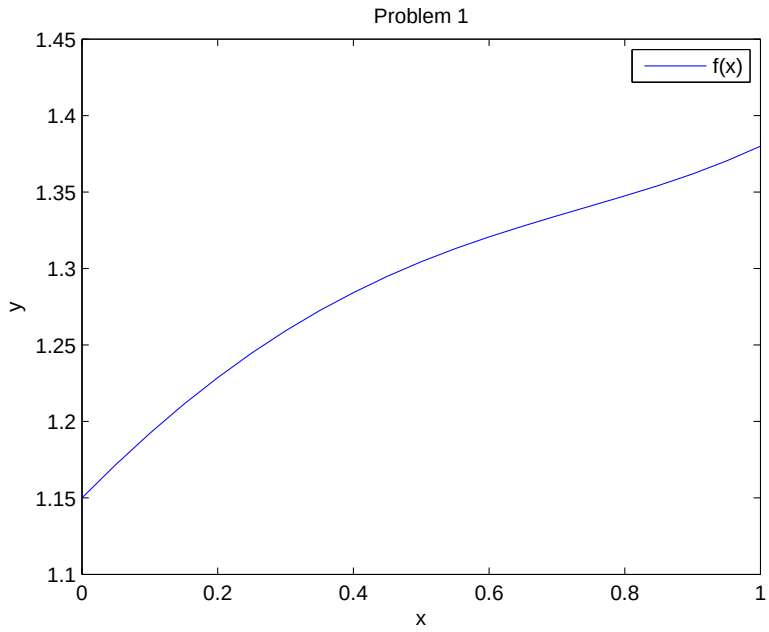

Notice that this plot is very non-descriptive. There are no labels for the axes, no title and no legend. These can all be added to the figure using the following commands

```
>> xlabel('.....')
>> ylabel('.....')
>> title('.....')
>> legend('......')
```

Where the "....." is the name you want typed. For example

```
>> xlabel('x')
>> ylabel('y')
>> title('Problem 1')
>> legend('f(x)')
```

produces the following figure




Now we have constructed a basic figure. There are a variety of options to control the appearance of the figure. For instance the type of line plotted, the color of the line, the thickness of the line, etc. Similarly we can change the font of the labels and titles. There are many options, the following are some key options.

```
>> set(gca,'fontsize',16)
```

Changes the fontsize for the axes values to 16, for example.

```
>> xlabel('x','fontsize',16)
>> ylabel('y',fontsize,16)
```

Changes the fontsize for the two axes labels to 16

```
>> title('Problem 1','fontsize',16)
```

Changes the fontsize for the title to 16

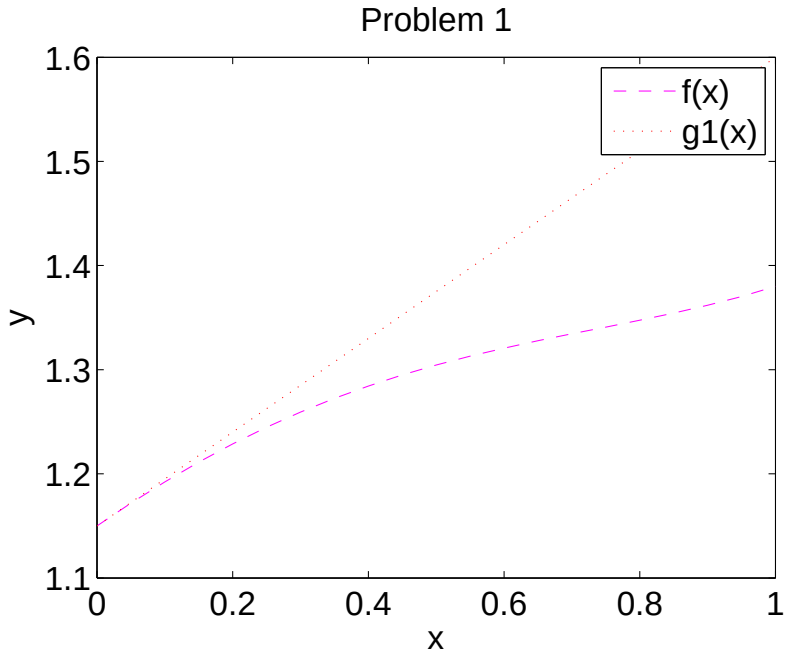
```
>> plot(x,f,'--m')
```

Changes the plot of the line to a dashed line that is magenta instead of blue (the default color). For a complete list of line styles and colors to use in plots go to the help window and type "lineseries properties".

We can plot multiple lines on the same figure as follows.

```
>> g1 = 0.45*x+1.15;
>> plot(x,f,'--m',x,g1,':r)
>> legend('f(x)','g1(x)')
```

This will produce the following plot with two lines, one for $f(x)$ the other for $g_1(x)$.




### 3.2 Saving a Figure

You can save a Matlab figure in a variety of formats. I do not recommend saving a Matlab figure as a jpeg. They tend to be of poor quality when printed and basically cannot be resized well. In my experience the best format is a '.png' file. It saves at a higher quality and will not become too corrupted if resized. For those of you familiar with the program LaTeX, or in dealing with postscripts the '.eps' extension is even better for manipulating. The Matlab '.fig' extension creates a figure only readable by Matlab, and is only useful if you want to save a figure for working on later.

Printing from Matlab directly often does not go very well. The best advice for printing a figure created in Matlab is to save it as discussed above and open that file with your computer, using the built in software to print. Another technique is to import the figure into Powerpoint or Word and print from there.

### 3.3 Advanced Plotting

Without going into explanation the following commands will help you to plot for advanced figures. I suggest consulting the help window for more information about these commands.

```
>> figure(n)
```

This will open up a blank figure with the number n, or if that figure is open it will access it. This is useful when you have mutliple figures open and you want to change around which ones you are messing with, or if you want to produce a new plot without overriding the ones you already have.

```
>> subplot(n,m,j)
```

Produces a an n x m array of subplots within the figure accessing the jth one for the current set of commands.

```
>> loglog(x,y)
>> semilogy(x,y)
>> semilogx(x,y)
```

Produces plots where one or both axes are on a log scale. loglog makes both axes on the log scale, semilogy just the vertical axis is on a log scale and semilogx just the horizontal axis is on a log scale. This is useful for error plotting.

```
>> contour(X,Y,Z)
```

Creates a contour plot of the matrices X, Y and Z. To construct a contour plot with the contours filled in by color exhange 'contour' with 'contourf'.

```
>> colorbar
```

Places the colorbar axis on the plot for reference to the values of each color.

```
>> zlabel('z')
```

has the same function as 'xlabel' and 'ylabel' for 3d plots.

```
>>  text(xp,yp,'.....')
```

Will write the text ..... at the point (xp,yp).

```
>> hold on
```

will hold a figure, so that any plot commands you give will be plotted on top of the plot, instead of replacing the current plot.

```
>> hold off
```

Turns the hold off.

```
>> grid
```

Turns the grid lines on

```
>> xlim([xmin xmax])
>> ylim([ymin ymax])
>> axis([xmin xmax ymin ymax])
```

Each of these controls the limits of the plot. Both xlim and ylim control just the horizontal and vertical axis respectively. The axis command controls both at the same time. Of course 'xmin' is the minumum desired x value, 'xmax' is the maxiumum desired x value, etc.

```
quiver(x,y,u,v)
```

Produces a plot of (x,y) points with arrows(u,v). Useful for plotting velocity diagrams.