**Lab Workshop #11**

Purpose: exploring Matlab’s built-in functions

matrix manipulations in Matlab

using Matlab's 2D graphics and engineering plots

1. Launch the Matlab application from **NAL**. You should eventually see the Matlab window with an entry prompt, **»** , on the left.

At the start of a Matlab session, you will want to change the default folder used by Matlab for saving and retrieving files. You can change this to the D: drive, your USB drive, or another folder. You can do this conveniently by clicking the browse button browsebutton and selecting the desired path to your folder of choice. Do that now.

2. Matlab is based largely on the use of built-in functions. The standard functions available in Matlab are augmented by hundreds of specialized functions that are part of the many toolboxes that may be part of a given Matlab installation. Many mathematical functions will be familiar to you because they are identical to those used in Excel or VBA.

First, let’s review the common mathematical functions. It is important to recognize that Matlab’s functions will work automatically on arguments that are arrays, one- or two-dimensional. Start by setting up some test variables:

**a = 5**

**b = [ 16 36 81 ]**

**c = [ 2 4 6 ; 8 10 12 ; 14 16 18 ]**

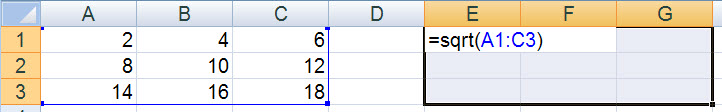
Remember, if you want to suppress Matlab’s echo of your assignments, press the **;** key before you press the *Enter* key.

Now, try **sqrt(a)**

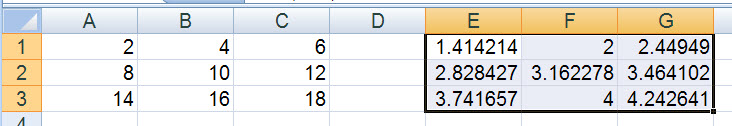
and **sqrt(b)**

and **sqrt(c)**

Here, of course, you wouldn’t want to use the **;** key because you do want to see the results. Notice that, when the square root function is applied to a vector or a matrix, it computes the square root of each item. That is called an *array operation* in Matlab. This is similar to array formulas in Excel. See the example below (you don’t have to launch Excel and do it though):



***Ctrl-Shift-Enter***



You can assign a function result to a variable,

**Cs = sqrt(c) ;**

Notice though that

**Cs \* Cs**

does not yield **c** back. This is because Matlab carries out matrix multiplication when it sees the **\*** operator. Instead, try

**Cs .\* Cs**

and you should see that **c** is regenerated. This is due to the fact that the **.\*** operator causes the multiplication to take place item-by-item, as an array operation.

You have to get used to the distinction between item-by-item array operations and matrix operations in Matlab. You have to deal with those differences all the time.

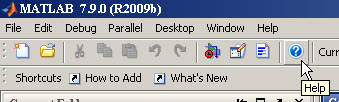
Try **Cm = sqrtm(c)**

This yields a different result from **Cs**, right? To see how this is different, execute

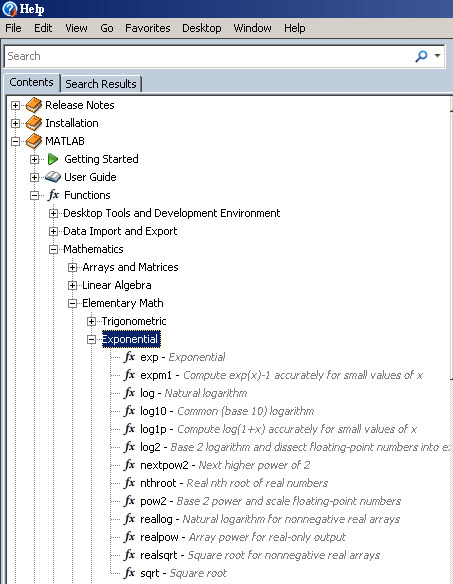
**Cm \* Cm**

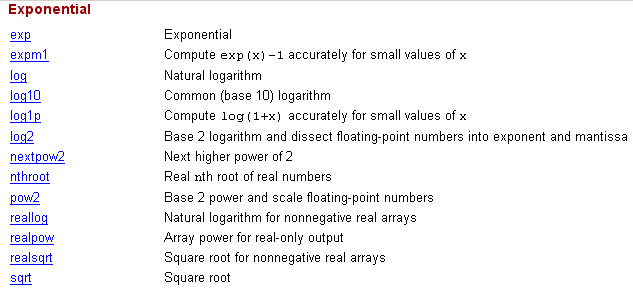
and you should see that, by matrix multiplication, **c** is regenerated. So, **Cm** is the matrix square root of **c**; whereas **Cs** is the array square root of **c**. This distinction is very important. Excel does not have a corresponding function to **sqrtm**, and to carry out matrix multiplication in Excel, you have to use the array function MMULT.

Other common functions used in engineering and scientific calculations are

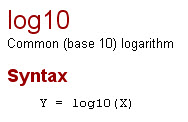
* + exponential/logarithmic
  + trigonometric/hyperbolic

You should get used to Matlab’s Help facility in finding out information about functions. So, click the Help button and expand the list on the left of the Help window to show the display on the following page.

 To the right, you should see



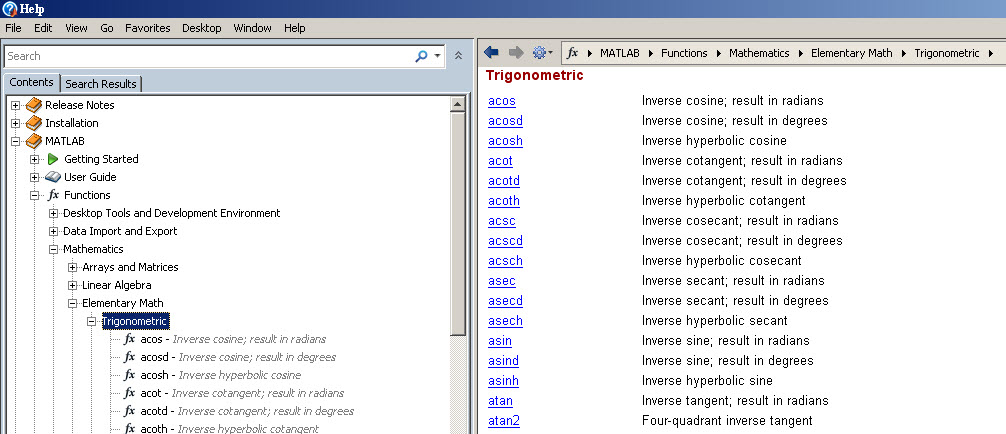
and you can get more detail on a specific function by clicking the link, e.g., for the **log10** function,



Notice that the natural logarithm function is **log**, as in VBA, but not Excel, and that there is a separate **log2** function for the base-2 logarithm.

Try **log10(b)**

Take a look at the trigonometric functions on the Help screen.



Notice that there are functions that require an argument in radians, like **sin** (or inverse functions that produce a result in radians, like **asin**), and there are alternate functions, like **sind** and **asind**, that involve degree quantities.

Let’s carry out a case study of one of the inverse trigonometric functions that occurs frequently in engineering calculations, the inverse tangent. One way to picture a tangent and inverse tangent calculation is via the following diagram.



Using your understanding of trigonometry and triangles you can see that the tangent of the angle  is **b/a**. And likewise, the tangent of the angle  is **–b/(-a)** or the same **b/a**. Matlab will have no problem getting the correct answer.

Try **theta = pi/6**

**tan(theta)**

and **tan(theta+pi)**

You should get the same result and, for a distance from the origin of unity (1),

**a = sin(theta)**

and

**b = cos(theta)**

But, what happens when we try the inverse tangent function, **atan** ?

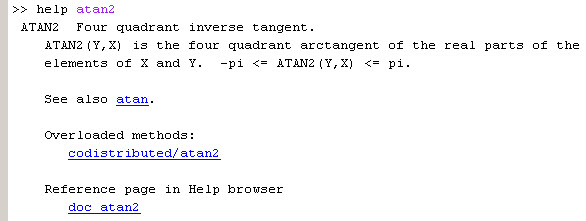
**atan(a/b)**

And you can see that **atan(-a/(-b))** will give the same result and not  because the negative signs will cancel. Matlab’s **atan** is called a 2-quadrant function because it will return angles only in quadrants I and IV (as shown on the diagram below) or between  and . But, as you will notice from Matlab Help, Matlab also has an **atan2** function that handles a 4-quadrant inverse tangent. By the way, Excel also has such a function.

Another way to get quick help for a function, when you already know or remember the function’s name, is to type

**help atan2**

From the command line prompt **>>**.



Notice that the function does cover the four quadrants, but it returns angles between  and , and its arguments are in the order **y** then **x**.

Try these.

**atan2(1,2)**

and

**atan2(-1,-2)**

Do these make sense to you according to the figure on the previous page?

Whereas there is an **atand** function, there is no **atan2d** function, but it’s easy to convert the results to degrees by dividing by  and mulitplying by **180**. Do that by using the up arrow, **🡩**, repeatedly until you get back the **atan2(1,2)** command and then append **\*180/pi** to it. You can repeat that for the **atan2(-1,-2)** command. What are the results?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A frequent task in Matlab is to create a vector of evenly spaced values. For example, let’s say you wanted to create a vector of 10 evenly-spaced values from 0 to 10. The **linspace** function can be used to do this.

**linspace(0,10,10)** where the last argument is the number of values.

Try this and notice the spacing of the values. What is the spacing of the values? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Perhaps you thought that this would generate the integer numbers 0 through 10. Not so. What command would do that?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Try it to make sure.

Another way to do that would be

**0:1:10** [ read “ 0 through 10 in steps of 1” ]

You can actually leave the 1 out and type **0:10** Try that too.

Another spacing function that comes up in engineering calculations is **logspace**, even spacing by the logarithm of the number.

Try **x = logspace(0,1,10)**

Note that this gives 10 values spaced between 1 and 10. The spacing is not even, but execute

**log10(x)**

and you should note that the spacing is even, but in the logarithm of the **x** values.

Review the functions in Table 3.3 on page 68 of the text. Use Matlab commands to solve the Practice Exercises 3.3 on page 69. Enter your results below.

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 5. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. \_\_\_\_\_\_\_\_\_ 6. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. \_\_\_\_\_\_\_\_\_ 7. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. \_\_\_\_\_\_\_\_\_

It is important for you to know how to import data into Matlab from external sources and export data too. We’ll review the importing here and consider that you have a set of data in Excel. From the Spreadsheets folder on the course CULearn site, download and open the ViscosityData.xlsx workbook file in Excel. You should see a single column of data.

How many data are there? \_\_\_\_\_\_\_\_\_

From Excel, press the F12 key to get the Save As window. Change the Save As Type: field to Text (Tab-delimited) (\*.txt) and File name: field should change automatically to ViscosityData.txt . Make sure the folder is the same as your default folder in Matlab. Click the Save button. Click Yes when the compatibility warning message comes up, then close out Excel. You can click No for the final warning message. The ViscosityData.txt file is in a format that can be imported directly into Matlab.

In Matlab, issue the following command

**load ViscosityData.txt –ascii**

If the command executes successfully, you should see a variable in the Workspace window that has the name ViscosityData and shows the same number of values you determined in the Excel workbook.

Now that you have this data set in Matlab, you will explore a few of the data analysis functions applied to the data set.

It will save you some typing if you create a variable equivalent to ViscosityData, so start with

**mu = ViscosityData;**

Then, let’s compute some sample statistics. Write down the values you compute.

count **length(mu)** \_\_\_\_\_\_\_\_\_\_

average **mean(mu)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

median **median(mu)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

sample standard deviation **std(mu)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mode **mode(mu)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

maximum max(mu) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

minimum min(mu) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

sum sum(mu) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

There are some enhanced sample statistics functions you should be aware of. They are reviewed in your text in Table 3.5 on pages 74-75. Try the following:

**[ maxmu locmu ] = max(mu)**

How do you interpret the **locmu** result? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Try **mus = sort(mu);**

Double-click on **mus** in the Workspace to open up the Variable Editor.

How many of the **maxmu** values are there in the data set? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The **locmu** result above found the location of one of the **max** values. Where are the other values located?

To determine that, try **find(mu == max(mu))**

What is the result? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Which of these did **locmu** find? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The double equals ( **==** ) may look strange to you, since you just used the **=** sign in Excel. The operators in Matlab for relational comparisons are close to those in VBA and Excel, but not quite the same. See the table below.

**>** greater than **==** equal to

**>=** greater than or equal to **~=** not equal to[[1]](#footnote-1)

**<** less than

**<=** less than or equal to

Now, you will generate two graphical depictions of the data. First, you can produce a simple plot.

**plot(mu)**

**grid**

**xlabel(‘Sample Number’)**

**ylabel(‘Viscosity – cp’)**

**title(‘Viscosity Data’)**

Note: You could also have typed all that onto one line, separating the statements by semicolons (;).

**plot(mu);grid;xlabel(‘Sample Number’);ylabel(‘Viscosity – cp’); title(‘Viscosity Data’)**

Although the plot shows a noisy series of viscosity data, comment on any trends you observe:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Next, you will create a histogram plot of the data. In this plot, the range of the viscosity data is divided up into a given number of intervals, and the number (or frequency) of data is counted for each interval.

**figure(2)**

**n = length(mu)**

**nintervals = fix(log2(n))-1**

**hist(mu,nintervals)**

**xlabel(‘Viscosity – cP’)**

**ylabel(‘Frequency’)**

**title(‘Histogram of Viscosity Data’)**

The **figure(2)** command opens a new plot window; otherwise, your histogram plot will replace your previous plot. The **nintervals** calculation determines the number of intervals (also commonly called “bins”) in which the data will be counted.

Based on the histogram plot, what is the interval, approximately, that contains the most data?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

and which interval(s) contains the mean, median and mode values you determined earlier?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

There are other aspects of data analysis, such as regression, that we will explore in upcoming lab exercises.

3. Since much of the work done in Matlab has to do with matrices, it is important to know how to create and manipulate matrix-based information. To become proficient, you have to practice methods and techniques over and over. You will do some of that here.

First, assign a matrix of values to a variable:

**X = [ 7.2 -9.1 0.8 14.6 ;**

**-4.3 10.4 2.5 5.5 ;**

**0.1 4.7 11.1 -21.4 ;**

**1.9 2.3 6.2 -8.5 ]**

This assignment command should echo a neatly formatted **X** matrix[[2]](#footnote-2).

Now, try **size(X)**

and it should return a vector with the numbers of rows and columns in **X**.

Let’s say you wanted to extract the 3rd row of **X** into a vector variable **x3**. Try this.

**x3 = X(3,:)**

You should read the indices in the parentheses as “3rd row, all columns.” That’s the meaning of using the colon ( **:** ) there.

So, what command would you use to extract the 2nd column into a variable **y2**?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Be sure to try that and confirm that it works.

Another way to use the colon ( **:** ) is for a range of row or column indices. For example, try

**X(2:3,3:4)**

Make sure you understand your result. You can also use the keyword **end** for the last index value, as in

**X(2:3,3:end)** which should yield the same result.

What if you wanted to add a column to the **X** matrix?

**[ X , [ 12.1 ; -22 ; 0.78 ; 6.5 ] ]**

How would you add those same numbers to **X**, but as an extra row?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The transpose of **X** is computed by

**X’** rows become columns and columns become rows

When you want to multiply a matrix, like **X**, by a scalar quantity, like **π**, simple multiplication works fine in Matlab:

**pi \* X** or **X \* pi** should yield the same result.

The same would work for division by a scalar, like **X/pi**

Also, when you want to add or subtract a scalar quantity to/from a matrix, Matlab recognizes this as a simple operation:

**2 + X** or **X – 5**

But what if we wanted to divide each element of the **X** matrix into a scalar like **π** ?

Matlab will have a problem with **pi/X**

because Matlab wants to carry out a vector/matrix operation, not an item-by-item divide. Instead, type

**pi ./ X**

and you will see that Matlab divides each element of **X** into **π** and produces a matrix result.

Create another 4x4 matrix, this one with random numbers, by entering

**Y = rand(4)**

If you enter the command

**X \* Y**

Matlab will carry out the matrix multiplication of **X** and **Y**.

Does **Y \* X** yield the same result? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If you wanted instead to multiply **X** and **Y** element-by-element and not by the rules of matrix multiplication, what command would you enter?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

and, if you reverse the order of the multiplication, is the result the same? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. One of the common activities we studied with Excel was carrying out case studies. We want to study the behavior of a result, for example, one or more formula calculations, when one or more inputs to the calculation are varied. It is important to know how to do the same with Matlab.

For a simple example, let’s study the behavior of the function, between 0 and .

First, set up a vector **x** with 100 values over that range.

**x = linspace(0,pi/2);**

Note: the default number of values is 100, if you don’t set it as the third argument in the **linspace** function.

Now, compute the function values, assigning them to a vector variable **y**:

**y = x .^ 2 .\* abs(sin(x));**

Note carefully the use of the period ( **.** ) operator here. The **x** vector is one row by 100 columns

( 1x100 ), so, to square each item, you have to use **.^ 2** ; otherwise, if you tried to use **x ^ 2** , Matlab would try to square **x** matrix-wise, i.e., it would try to multiply **x** by itself using the rules of matrix multiplication. Multiplying a 1x100 vector by a 1x100 vector does not conform to the rules of matrix multiplication, so that wouldn’t work. Try it, if you would like to see the error message.

Likewise, the result of **x .^ 2** is a 1x100 vector. Matlab automatically applies the sine function and absolute value function to **x** item-by-item, so **abs(sin(x))** is also 1x100. That again means that using the **\*** operator alone would not work because Matlab would want to perform matrix multiplication, so we use **.\*** instead for item-by-item multiplication.

Create a simple plot of **y** versus **x** with the following commands:

**plot(x,y,’k’) ; grid** The **‘k’** argument specifies that the line be black.

**title(‘Ralphie V’)**

Carry out a similar case study of the function  over the range .

Put your name as the plot title instead of Ralphie. Make a print-out of the plot and hand it in with your lab worksheet.

You should remember that, in Excel, we could carry out a case study where we varied two input cells using the Data Table feature. We then could make a surface plot of the study, although Excel’s surface plotting capability is limited.

Let’s do a two-variable case study using Matlab. We need an example function of two variables.



We’ll pick the ranges  and .

First, create 100-element vectors for **x** and **y** using the **linspace** function.

**x = linspace(0,1);**

**y = linspace(0,1);**

What we would now like to be able to do is evaluate for function every combination of **x** and **y** values in a grid, as shown in the figure below.



We do this in Matlab by creating two matrices that overlay each other with the appropriate combination of **x** and **y** values at each intersection. The **meshgrid** function is used for this.



Create these two matrices as follows:

**[ X Y ] = meshgrid(x,y)**

Then, you can calculate the function values for every combination of **x** and **y** using

**W = sin(2\*pi\*X) .\* sin(3\*pi\*Y)**

You will now see that Matlab’s surface plotting capability is superior to that of Excel. Enter the command

**surfc(X,Y,W)**

Rotate3DButton.jpg

Experiment with the Rotate 3D tool, by clicking on that command button in the plot window and then drag the plot around. You should be able to look under the surface, rotate it around, look straight down on it, etc. Pretty neat.

This gives you an example of a 2-variable case study in Matlab.

5. Graph commands in Matlab can be grouped into three categories:

* graphics management
* curve and surface generation
* annotation and graph characteristics

The following table lists the commands in their respective categories.

|  |  |  |
| --- | --- | --- |
| Management | Generation | Characteristics |
| figure | **2D** | xlabel |
| subplot | plot | ylabel |
| zoom | polar | zlabel (3D) |
| hold | fill | text |
| view (3D) | plotyy | text3 (3D) |
| rotate3d (3D) | **3D** | legend (2D) |
|  | plot3 | box |
|  | surf,surfc | set |
|  | mesh,meshz | grid |
|  | contour | axis |
|  | contour3 | axis equal |
|  | contourf | axis off |
|  | waterfall | colorbar (3D) |
|  | cylinder | clabel |
|  |  | colormap (3D) |

As noted, some commands are restricted to 2D or 3D plots only.

Graphs are created in "figure" windows. A command like "plot" will create a figure window, if one is not available.

Clear out the Matlab workspace with the **clear** command. You can also select *Clear Workspace* from the *Edit* menu. Delete any current figure windows[[3]](#footnote-3). Create a new, empty figure window with

**figure(1)**

Place a single, red asterisk at the coordinate location (2,4) with the command

**plot(2,4,'r\*')**

Draw a straight line that goes from (0,1) to (1,2) using the default line type (solid) and the default color (blue) with

**plot([0 1],[1 2])**

Notice that the (*offensive*) red asterisk is gone. To add the asterisk to the current figure, type

**hold on**

**plot(2,4,'r\*')**

Look carefully to find the asterisk. It's there.

The **axis** command can be used to rescale the axes manually:

**axis([ -1 4 0 5 ])**

The syntax here is **axis([ xmin xmax ymin ymax ])**. You can always review this with the **help axis** command.

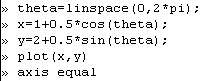
To draw a circle, we make use of the relationships



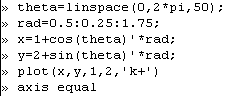
To create a vector of 100  angles from 0 to , it is convenient to use the **linspace** command, instead of **theta=0:2\*pi/100:2\*pi';**

**linspace(0,2\*pi);**

First, delete the current plot, then, create the circle plot with



To create a plot of a family of circles, we make a vector of radius values, and, with careful use of vector/matrix multiplication, create **x** and **y** matrices for plotting. Try the following:

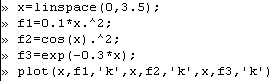


By adding the 50 to the **linspace** command, only 50 points are generated. The addition of the **1,2,'k+'** arguments to the **plot** command adds the **+** sign at the center of the circles.

Often, you will want to plot a family of functions. For example,

in the range . This is easy to do:



Try these alternative **plot** commands:

Multi_function_plot2

and

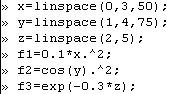
Multi_function_plot3

The **'k'** command sets the color to black. The various color codes are:

 You can always find these by typing **help plot**.

Delete the current plot.

It is also possible to combine plots with different independent variable (abscissa) ranges:

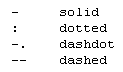


Multi_function_plot5

Instead of plotting with different colors, you can use different line styles:

Multi_function_plot6

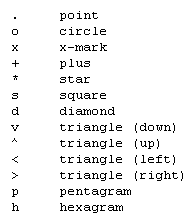
The various line styles are



Instead of using line styles, you can use different markers:

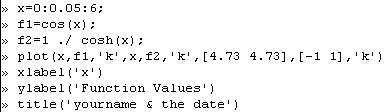
Multi_function_plot7

The various marker codes are:



Clear your workspace and delete any figure windows.

Now, let's get into enhancing graphs with annotations (titles, axes labels, curve labels, etc.).



Annotate2

Show this graph to your TA and have them initial the worksheet or make a print-out of your graph and hand it in with this lab assignment.

It is also possible to add a legend if you use different linestyles, colors or markers.

6. Close out Matlab and return to Windows.

**End of Lab Workshop #11**

1. The tilde ( **~** ) precedes the = sign. The tilde is usually located as the upper-case character on the key to the left of the numeral **1** key. Some people, mainly from states neighboring Colorado, call this a “squiggle.” [↑](#footnote-ref-1)
2. It is customary in mathematics to use upper-case letters for matrices and lower-case letters for vectors, but that is not mandatory in Matlab. We do follow the convention here. [↑](#footnote-ref-2)
3. To delete a figure window, select the window and click on the **X** in the upper right-hand corner. [↑](#footnote-ref-3)