# Lab Workshop #10

Purpose: become familiar with the Matlab software package

 carry out some common numerical calculations using Matlab

 make use of Matlab’s graphics capabilities to create plots

One of the purposes of this course is to introduce you to software tools that you will be using in subsequent courses. Excel will be the tool that you use most often and is that used commonly by practicing engineers. Matlab is yet another example that is used more widely in engineering courses and by engineering professionals in certain industries, e.g., aerospace. One reason that Matlab is less used in industry is its cost; whereas Excel, being part of Microsoft Office is available at no incremental cost.

Matlab’s main strengths are in types of calculations that are beyond the scope of this course (applied linear algebra, differential equations, advanced engineering math); however, there are some features of Matlab that you can take advantage of immediately. It is the purpose of this workshop to familiarize you with those features.

Although the full Matlab package is quite expensive (thousands of $$), there is a student edition of Matlab available for a reasonable price (around $100). You may want to consider purchasing this at some point in the future for your own computer based on the demand for your use of Matlab, but it is not a requirement for this course. The current student edition contains the following features

 

1. Launch the Matlab application from **NAL (Matlab R2009a)**.



 The Matlab window should open and look something like this:



Matlab operates in a very different way than Excel, in sequential fashion as you type in commands line by line. For each command, you get a result. There is also a programming language feature that we will get to later.

 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  and selecting the desired path to your folder of choice. Do that now.

2. Making simple calculations at the Matlab command line

 From the **>>** command line, you can type in a simple (or not so simple) numerical expression, and Matlab will produce the answer.

 For example, enter

 **(sqrt(5)-1)/2** [ this is the formula for the Golden Ratio ]

 and press the Enter key. You should see

 

 You should also notice that the **ans** variable has been automatically created and can be seen in the Workspace window:

 The command you issued is also recorded in the Command History window:

 

 For simple arithmetical calculations, Matlab has similar operator precedence to VBA:

 ^ exponentiation

 - negation, unary minus

 \* / multiplication, division

 + - addition, subtraction

 So, try

 **-3^2** What is the answer? \_\_\_\_\_\_\_\_\_\_\_

 The order of operations is left to right, although that order can be changed by the use of parentheses.

 For example, try

 **2^3^2** What resulted? \_\_\_\_\_\_\_\_\_\_\_

 and **2^(3^2)** The answer? \_\_\_\_\_\_\_\_\_\_\_

 Practice with the exercises on pages 10 and 11 of the Matlab text. Write down the answers in the spaces below. [Note: several of these you can do in your head, but type them in any way for the sake of the practice.]

 1. \_\_\_\_\_\_\_\_\_\_\_ 2. \_\_\_\_\_\_\_\_\_\_\_ 3. \_\_\_\_\_\_\_\_\_\_\_ 4. \_\_\_\_\_\_\_\_\_\_\_

 5. \_\_\_\_\_\_\_\_\_\_\_ 6. \_\_\_\_\_\_\_\_\_\_\_ 7. \_\_\_\_\_\_\_\_\_\_\_ 8. \_\_\_\_\_\_\_\_\_\_\_

 9. \_\_\_\_\_\_\_\_\_\_\_ 10. \_\_\_\_\_\_\_\_\_\_\_

 As you become more experienced with Matlab, you will refer to and use the Workspace window often. Its display can be customized. For example, it might look like

 

 Since **ans** is a single-valued variable (also called a “scalar”), the **Min** and **Max** headings don’t give us any additional information. You can change those by right-clicking in the gray heading area. Do so and deselect **Min** and **Max**, then select **Size** and **Class**. You should then have

 

 This display aligns better with what is shown in your text, although the **Bytes** heading is also included there. You won’t need the **Bytes** heading much. The **1X1** size indicates a scalar, and the **double** class is the same as the Double type in VBA.

3. Assignment of values to variable namesAssignment of values to scalar variables is similar to other computer languages. Try typing

 **a = 4**and then

 **A = 6**

Note how the assignment echoes to confirm what you have done. This is a characteristic of Matlab.

You should also note that the two variables a and A have been added to the Workspace window, and they are distinct.

The echo can be suppressed by following the command line with the semicolon (**;**) character. Try typing

 **b = - 3 ;**

 Matlab treats names in a case-sensitive manner; that is, the name **a** is not the same as the name **A**. This is shown in the Workspace window, but to illustrate it further, enter

 **a**

 and

 **A**

 See how their values are distinct. They are distinct names.

 Variable names in Matlab generally represent matrix quantities. A row vector can be assigned as follows

 **a = [ 1 2 3 4 5 ]** Note: You have to use brackets, [ ], here, not parentheses.

 You could put in commas to separate the numbers, but that’s not required.

 The echo confirms the assignment again. Notice how the new assignment of **a** has taken over from the previous one, and the Workspace window reflects this.

 

 An interesting feature of Matlab allows you to display a variable, scalar or matrix, in the form of a spreadsheet. To do that, double-click the little yellow boxes to the left of **a** in the Workspace window.

 This opens the Variable Editor window. It is not a true spreadsheet, but it would allow you to modify particular values in a matrix. Close that window by clicking on the X in the upper-right corner.

 A column vector can be entered in several ways. Try them all.

 **b = [ 1 ; 2 ; 3 ; 4 ; 5 ]**

 or

 **b = [ 1;**

 **2;**

 **3;**

 **4;**

 **5 ]**

 or, by transposing a row vector with the  **'** operator [the apostrophe key on the right side of the keyboard next to the Enter key]

 **b = [ 1 2 3 4 5 ] '**

 A two-dimensional matrix of values can be assigned as follows

 **A = [ 1 2 3 ; 4 5 6 ; 7 8 8 ]**

 or

 **A = [ 1 2 3 ;**

 **4 5 6 ;**

 **7 8 8 ]**

 The values stored by a variable can be examined at any time by typing the name alone, e.g.

 **b**

 or

 **A**

 Of course, you can always double-click the variable in the Workspace window and see the values in the Variable Editor window too.

 Also, a list of all current variables can be obtained in the Command Window by entering the command[[1]](#footnote-1)

 **who**

 or, with more detail, enter

 **whos**

 There are several predefined variables, e.g. **pi** . Try typing

 **pi**

 It is also possible to assign complex values to variables, since Matlab handles complex arithmetic automatically[[2]](#footnote-2). A complex quantity contains two values, the real part and the imaginary part. A complex quantity is often represented as, for example,

  where 

 To see how Matlab handles and represents such quantities, enter

 **2 + i \* 3** What does Matlab display as a result? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Then a complex value can be assigned

 **x = 2 + i \* 3**

Notice in the Workspace window that the Class of **x** is designated as **double(complex)**.

4. Mathematical Operations

 Matlab’s operators work in calculator fashion. Try

 **2 \* pi**

 Also, scalar real variables can be included.

 **y = pi / 4**

 **y ^ 2.45**

Results of calculations can be assigned to a variable, as in the next-to-last example, or simply displayed, as in the last example.Calculations can also involve complex quantities. Using the **x** defined above, try

 **3 \* x**

 **1 / x**

 **x ^ 2**

 **x + y**

 The real power of Matlab is illustrated in its ability to carry out matrix calculations. In order to illustrate vector-matrix multiplication, first redefine **a** and **b** ,

 **a = [ 1 2 3 ]**

 and

 **b = [ 4 5 6 ] '**

 The inner product of two vectors (dot product) can be calculated using the  **\*** operator,

 **a \* b**

 and likewise, the outer product

 **b \* a**

 Now, try

 **a \* A**

 or

 **A \* b**

 What happens when dimensions are not those required for the operation? Try

 **A \* a**

 Matrix-matrix multiplication is carried out in likewise fashion

 **A \* A**

 Mixed operations with scalars are also possible.

 **A / pi**

 It is important to remember always that Matlab will apply the simple arithmetic operators in vector-matrix fashion if possible. At times, you will want to carry out calculations item-by-item in a matrix. Matlab provides for that too. For example, **A ^ 2**results in matrix multiplication of **A** with itself. What if you want to square each element of **A**? That can be accomplished with

 **A .^ 2**

The  **.**  preceding the operator **^** signifies that the operation is to be carried out item-by-item. The Matlab manual calls these *array operations*, and this format is similar to that of array formulas in Excel .

5. Use of Built-in Functions

 Matlab has a rich collection of built-in functions. You can use on-line help to find out more about them. One of their important properties is that they will operate directly on vector and matrix quantities. For example, try

 **log(A)**

 and you will see that the natural logarithm function is applied in array style, element by element, to the  **A** matrix. Most functions, like *sqrt* , *abs* , *sin* , *acos* , *tanh* , *exp* , operate in array fashion, item by item. Certain functions, like exponential and square root, have matrix definitions also. Matlab will evaluate the matrix version when the letter  **m** is appended to the function name. Try

 **sqrtm( A )**This should yield a matrix result which, when multiplied by itself, results in the original **A**  matrix.

 A common use of functions is to evaluate a formula for a series of arguments. Create a column vector  **t** , which contains values from 1 to 101 in steps of 5,

 **t = [ 1 : 5 : 101 ] '**

 Check the number of items in the  **t**  array with the *length* function,

 **length(t)**

 Now, say that you want to evaluate a formula  *y = f(t)* , where the formula is computed for each value of the  **t** array, and the result is assigned to a corresponding position in the  **y**  array. For example, (caution: watch the spacing around the periods)

 **y = t .^ 0.34 - log10(t) + 1 ./ t**

 Done! [Note the use of the array operators with the decimal point.] This is similar to creating a column of the  **t**  values on a spreadsheet and copying a formula down an adjacent column to evaluate  **y** values.

 There are many, many functions in Matlab. Refer to the help facility.

6. Graphics

 Matlab's graphics capabilities have similarities with those of a spreadsheet program: graphs can be created quickly and conveniently, however, there is not much flexibility to customize them.

 For example, to create a graph of the  **t , y**  arrays from above, enter

 **plot ( t , y )**

 That's it! You can customize the graph a bit with commands like the following

 **title ( 'Plot of y versus t by *yourname*' )**

 Your plot is created in a separate window. You can move back and forth from the plot window to the main Matlab window by clicking in it or using the ***Alt-Tab*** key combination. **xlabel ( 'Values of t' ) ylabel ( 'Values of y' ) grid**There are other features of graphics which will become useful, plotting objects instead of lines, families of curves, plotting on the complex plane, multiple graph windows on the screen, log-log or semilog plots, 3-D mesh plots, and contour plots. We will get to these.

7. PolynomialsThere are a number of functions in Matlab which allow you to operate on arrays as if their entries were coefficients or roots of polynomials. For example, enter

 **c = [ 1 1 1 1 ]**

 and then

 **r = roots(c)**

 and the roots of the polynomial  should be displayed and are also stored in the **r** array. The polynomial coefficients can be formed from the roots with the *poly* function,

 **poly(r)**

 and a polynomial can be evaluated for a given value of  **x** . For example,

 **polyval(c,1.32)** What value results? \_\_\_\_\_\_\_

 If another polynomial,  **** , is represented by the array **d d = [ 2 -0.4 -1 ]**The two polynomials can be multiplied symbolically with the convolution function *conv* to yield the coefficients of the product polynomial

 **cd = conv(c,d)**

 Write out your result as a polynomial in **x**:

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

8. Statistical Analysis

 You can generate a series of (pseudo)random numbers with the *rand* and *randn* functions. Either a uniform (*rand*) or normal (*randn*) distribution is available. For example, to generate 1000 random numbers with a normal distribution,

 **n = 1 : 1000 ;** (Did you forget the **;** ?!) **num = randn(size(n)) ;**

 You probably understand why using a semicolon at the end of the commands above is important, especially if you neglected to do so.

 If you would like to see a plot of noise, try **plot(num)**

 These are supposed to be normally distributed numbers with a mean of zero and variance (and standard deviation) of one. Write in the values you get for

 **mean(num)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_

 and

 **std(num)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_

 No one is perfect! You can find minimum and maximum values,

 **min(num)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_

 **max(num)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_There is a convenient function for plotting a frequency diagram or histogram of the data

 **hist(num,20)**

 where 20 is the number of bins.

 If you would like to fit a polynomial to some data by least squares, you can use the *polyfit* function. Try the following example:

 **t = 0 : 5**

 **y = [ -0.45 .56 2.34 5.6 9.45 24.59 ]**

 **coef = polyfit(t,y,3)**

 The values in **coef** are the fitted polynomial coefficients. To generate the computed values of **y** ,

 **yc = polyval(coef,t)**

 and to plot the data versus the fitted curve

 **plot(t,yc,t,y,'o')**

 The plot of the continuous curve is piecewise linear; therefore, it doesn't look very smooth. Improve this as follows:

 **t1 = [ 0 : 0.05 : 5 ] ;**

 **yc = polyval(coef,t1) ;**

 **plot(t1,yc,t,y,'o')**

 *Now isn't that special!* Add your name to the title of the plot with

 **title('*your name*')**

 Make a print-out of your plot and attach it to your worksheet.

9. Matlab has an optional Symbolic Math toolbox (which is installed in the CU labs) that allows for symbolic operations. This toolbox is included in the current student edition of Matlab.

Try the following exercises. **syms x n f = x^n dfdx = diff(f,x)**The **syms** command identifies **x** and **n** as symbolic variables.

Use this technique to find the derivative of $$\frac{\sin(x)}{x^{n}}$$

What is the symbolic derivative?

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

Now, try

 **f = x^n**

**intf = int(f,x)simplify(intf)**

Use this method to find the integral of

$$e^{-x}\sin(nx)$$

What is the indefinite integral?

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

Here is another example.

 **f = x^3 - 6\*x^2 + 11\*x - 6; factor(f)**

Use this technique to factor

$$x^{6}-1$$

 
What is your result?

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

These are only a few of the symbolic capabilities of Matlab, but they are useful.

10. Finally, execute the ***demo*** command and see that there are numerous video tutorials that can help you get going with Matlab. Use your earphones, if you have them, and watch the *Getting Started with Matlab* video, only about 5 minutes long. If you don’t have earphones, do this at a later time when you have them.

***End of Lab Workshop #10***

1. The **who** and **whos** commands are inherited from early versions of Matlab when there was no Workspace window. [↑](#footnote-ref-1)
2. By the way, Excel also handles complex quantities, a little-known fact to most Excel users. [↑](#footnote-ref-2)