Matlab and Programming
Last updated September 27, 2006

If you encounter any spelling or grammatical errors, please let me know, I am writing this up on the fly!

1 Introduction

The purpose of this document is to keep a concise record of the programming topics and their uses in Matlab that we will cover over the semester. Each time we introduce a new topic in class, I will endeavor to transcribe it here as quickly as possible. It is my hope that this well help you in your programming projects.

It should once again be stressed that Matlab has a very comprehensive and readable Help section which can be accessed by the F1 key. It can also be found under the *Help* tab by selecting the *MATLAB Help*

2 Variables in Matlab

All information in Matlab must be saved to variables. Variables come in many different sizes and flavors. If you ever want to know what flavor your variable is, you can view them with the workspace tab. Variables can be constants, or they can be vectors, arrays or matrices. As a word of caution, there are some names for variables which you should stay away from, such as i, j and pi. Try not to use anything that might refer to a commonly known mathematical constant.

To create a vector v of zeros which is 1 row by m columns, you simply type in the following command:

$$v = \text{zeros}(1, m)$$

If you would like to create a vector with defined elements, you can simply enter them:

$$v = [2, 4, 3, 5, 8]$$

In this case, we can access the second entry by the command $v(2)$. Arrays can be created in a similar fashion. If you wish to create a matrix with n rows and m columns:

$$w = \text{zeros}(n, m)$$

To access the entry in the i^{th} row and j^{th} column, you would use the command $w(i, j)$.

The *linspace* command also comes in handy. For example,

$$t = \text{linspace}(0, 1, 101)$$

creates a vector with 101 entries, starting at a value of 0 at $t(1)$ and a value of 1 at $t(101)$. In general, the command

$$t = \text{linspace}(a, b, n)$$

creates a vector with n entires whose starting value is a, ending value is b, that is $t(1) = a$ and $t(n) = b$.

3 Matrices

We will devote a whole section to matrices. How do you define a matrix in Matlab? How do you enter the entries? How do you perform matrix operations? These are a few of the questions we will answer.

3.1 Creating and Accessing Matrices

As previously stated, if you wish to create a matrix with n rows and m columns you use the command:

$$w = \text{zeros}(n, m)$$

To access the entry in the i^{th} row and j^{th} column, type $w(i, j)$. You will notice that the $w = \text{zeros}(n, m)$ command creates an empty (i.e. each entry in the matrix is equal to zero) matrix. If you already know what goes into your matrix, you can simply enter them in when defining the matrix. As an example, the matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

can stored in the variable W by the command

$$W = [\ 1\ 2\ 3\ 4;\ 5\ 6\ 7\ 8;\ 9\ 10\ 11\ 12\ ]$$

A space is used between entries in each row, and a semi-colon separates rows. If for some reason you do not want to define the matrix by rows, but columns instead, you can use the following command:

$$W = [\ [1\ 5\ 9\ ]'\ [2\ 6\ 10\ ]'\ [3\ 7\ 11\ ]'\ [4\ 8\ 12\ ]'\ ]$$

Notice that an apostrophe was used after each column. This denotes the transpose of a matrix. Remember that the transpose of a matrix A, denoted $A^\top$,is where you switch the rows and columns, hence $a_{ij}^\top = a_{ji}$. Hence, if W is defined as above, then the command

$$X = W'$$

will give

$$X = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}.$$

One must be very careful when using the transpose command. If the matrix you wish to compute the transpose of is real, then the apostrophe will be enough. However, consider the following example:

$$R = [1 + i1 - i; 2 + 3i4 - 6i]$$

which has complex entries. This matrix looks as follows:

$$R = \begin{bmatrix} 1+i & 1-i \\ 2+3i & 4-6i \end{bmatrix}$$

However, performing the command R' yields

$$R' = \begin{bmatrix} 1-i & 2-3i \\ 1+i & 4+6i \end{bmatrix}$$

which is not just the transpose of R, but the *conjugate* transpose of R. To compute the transpose of a complex valued matrix, you use the command $R.'$ (notice the dot that preceeds the apostrophe. Hence,

$$R.' = \begin{bmatrix} 1+i & 2+3i \\ 1-i & 4-6i \end{bmatrix}$$

Now, you can access more than just an entry of a matrix at a time. You can access whole sections. For instance,

$$X(2:3, 1:2)$$

displays the intersection of the second and third rows of X with the first and second columns of X. Hence you will get the result

$$\text{ans} =$$
$$\begin{matrix} 2 & 6 \\ 3 & 7 \end{matrix}$$

printed to the screen. Of course you can always save the result to a variable to create a new two by two matrix.

To generate an identity matrix (ones along the diagonal), the command

$$eye(n, m)$$

comes in very handy. This is useful when you want to initialize matrices of a certain form, for example P in the $PA = LU$ factorization scheme.

One final remark. To determine the dimensions of your matrix, use the *size* command. For instance,

$$\text{size}(X)$$

returns

$$\text{ans} =$$
$$\begin{matrix} 4 & 3 \end{matrix}$$

This comes in handy when you need to know the dimensions of your matrix for loops in your programs.

3.2 Operations on Matrices

We have already discussed one simple operation, the transpose. One of the more common operations dealing with matrices is computing the inverse of a matrix. The command for computing the inverse. The command for this is simply $inv(A)$, where A is a square matrix. If A is not square, Matlab will give you an error stating such. If we consider R defined as in the previous section, then the command

$$\text{inv}(R)$$

yields the result

$$\text{ans} =$$
$$\begin{matrix} 1.1176 - 0.5294i & -0.2353 + 0.0588i \\ -0.0294 - 0.6176i & 0.0588 + 0.2353i \end{matrix}$$

Eigenvalues and eigenvectors can be computed in a similar manner. The command $\text{eig}(R)$ will compute the two eigenvalues of R, while the command $[v, e] = \text{eig}(R)$ will store the eigenvectors of R in v and the eigenvalues of R in e.

Matrix multiplication is the next topic we shall cover. To multiply two matrices together, make sure that the inner dimensions agree. As an example, in the previous section, W was 3×4 and X was 4×3. So to multiply those two matrices together, simply use the command

$$X * W$$

which will return a 4×4 matrix given by the following output:

$$\text{ans} =$$
$$\begin{matrix} 107 & 122 & 137 & 152 \\ 122 & 140 & 158 & 176 \\ 137 & 158 & 179 & 200 \\ 152 & 176 & 200 & 224 \end{matrix}$$

Upon using the command

$$W * X$$

a 3×3 matrix will be returned:

$$\begin{array}{l}\text{ans} = \\ \quad \begin{array}{ccc} 30 & 70 & 110 \\ 70 & 174 & 278 \\ 110 & 278 & 446 \end{array}\end{array}$$

The operations of addition and subtraction are the same. To subtract a matrix B from a matrix A, simply use the command $A - B$. Remember, A and B have to have the same dimension. What will happen if the command $X - W$ or $W - X$ is used? What is the result of the subtraction $X - W.'$ or $W - X.'$? (Here X and W are defined as previous.)

4 Simple Programming Commands

Next we will take a look at some of the simpler programming commands that you can use with Matlab.

4.1 For Loops

The for loop is one of the most fundamental building blocks of programming. In Matlab, the structure is as follows:

```
for j = start : step : stop
  stuff to do each loop
end
```

If the stepsize is not included,, it will assume that it is 1. As an example consider the following very simple loop:

```
for k = 1 : 5 : 29
  k
end
```

The above code simply outputs the value of j each time it goes through the loop. Notice that it should output 1, 6, 11, 16, 21 and 26. Once it gets to 31, it has passed the stop value of 29. An important fact is that the step does not have to be positive, and even more so, it does not have to be an integer. Consider

```
for k = 1 : -0.1 : 0
  k
end
```

This loop starts at the value of 1, and ends at 0 by increments of 0.1. Many times, programs will require nested loops.

```
for k = 1 : 1 : 10
  for l = 5 : -1 : 0
    [k, l]
  end
end
```

Try to determine what the above program will output. Notice that the output is in the innermost for loop. You can download this program from the website, it is called *firstfor.m*.

4.2 The *if .. then* statement

The *if .. then* statement is a standard conditional statement. The simplest standard form is

```
if (object relation object)
  stuff to do
end
```

The condition is usually a relation. The standard relations are <, >, <=, >=, == and ~=, which are less than, greater than, less than or equal to, greater than or equal to, equal to, and not equal to, respectively. As an example:

```
if (x >= 12)
  x = x + 1;
end
```

The above segment of code compares the variable x to the value 12, if x is at least 12 or greater, then the one more will be added to x. If x is not at least 12, then nothing will happen.

As stated at the beginning of the section, the above is the simplest form of the *if .. then* statement. There are more complicated forms, such as the *if .. elseif .. else .. .* As an example:

```
if (x >= 12)
  x = x + 1;
elseif ((x < 12)&(x > 0))
  x = x - 1;
end
```

Notice that in the *elseif* line, there are two relations. You must join these by the & symbol. You can join as many relations as you want this way. The above code does the same as before, however if x is not 12 or larger, then it does not exit, it goes onto the *elseif*. If x happens to be between 0 and 12, then x will be replaced by $x - 1$. The above code used only one *elseif*, but many more can be used.

```
if (x >= 12)
  x = x + 1;
elseif ((x < 12)&(x > 0))
  x = x - 1;
else
  x = 0;
end
```

This last code differs in one small way. There is an *else*, which stipulates that if none of the other conditions are satisfied, then set $x = 0$. The M-file *firstif.m* is an example of the above code. The correct call sequence is *firstif*(*value*), where *value* is any number you choose. Try many different values, and see what output you get. Pay particular attention to the values of 0 and 12.

5 Plotting

Plotting can be simple, plotting can be complicated. We will of course start with the simple. Let us first plot $\sin(x)$ on the interval $(-\pi, \pi)$. To do this, we need to define a linspace over the interval $(-\pi, \pi)$ (see the previous section entitled **Variables in Matlab**). This is done by the command

```
x = linspace(-pi,pi,1000)
```

Then to plot, simply try the command

$$\text{plot}(x, \sin(x))$$

If you want to change the view, simply use the command

$$\text{axis}([\text{-pi, pi, -1, 1}])$$

Plotting another function on the same graph requires the use of the *hold on* command. The *hold on* command needs to be used only once per figure. Any plot command after the *hold on* command will be sent to the current figure. So for example, consider the following sequence of commands:

$x =$ linspace(-pi, pi,1000)
plot$(x, \sin(x),$ '.g')
axis([-pi, pi, $-1, 1$])
hold on
plot$(x, \cos(x),$ 'r')
hold off

Notice that on each of the plot commands, there is an extra command. n particular, the '$.g'$ and $'r'$. The first command is to plot with green dots, the second to plot with a red line. The above program can be downloaded from the course website, it is called *firstplot.m*. The M-file *secondplot.m* combines the for loop with the plot commands. It also gives a few more options on how to plot, such as giving a figure a title and labels on the axes. The *hold on* command tells Matlab to keep the current figure and plot the new data on the same figure. To actually create a new, blank figure, you would simply use the command *figure*.

6 Quick Tips

First and foremost, Matlab's built in Help Menu is excellent. It usually gives you enough examples of the command syntax to get you where you need to go.

Placing a semi-colon at the end of a command stops the output form being displayed to the screen.

To clear all the variables in your workspace, simply use the command *clear*. To clear *Command Window*, use *clc*.