MatLab Commands That Will Help You Writing Programs For the Final Project

ECE 386, Fall 2000

by Paul Pawola

At this point in the semester, you should be well aquainted with many Matlab commands that you need to use for control systems. However, besides knowing these basic commands, it is helpful to know some of the more advanced and not-always-intuitive commands that will help you when writing script files and more complicated Matlab programs now during the final project, on the ECE 386 class homework, and later in other control systems classes and in your engineering career.

One of the first concepts that you should become familiar with is the concept of a *struct*. If you are familiar with C or with assembly, you might be familiar with structs. Basically, a struct is a data type that contains fields of other other data types. You can set and access these fields using the notation *struct.field*, where *struct* is the name of the struct, and *field* is the name of the field. Note that structs can be nested, i.e., a field of a struct can be a struct itself with its own fields, some of which may be structs, etc.

Example 1:

Suppose you wanted to keep track of the number of various types of fruits and vegetable you bought at the supermarket. Let's say you bought 5 apples, 3 oranges, 4 tomatoes, and 7 carrots. In Matlab, you could create a struct **Food** that contains how much of each type of fruit or vegetable you bought, by doing the following:

```
Food.fruit.apple = 5
Food.fruit.orange= 3
Food.vegetable.tomato = 4
Food.vegetable.carrot = 7
```

With these commands, you have now created a struct **Food** that contains two fields, **fruit** and **vegetable**, both of which are structs. The field/struct **fruit** contains two fields, *apple* and *orange*, which are just equal to numbers. Likewise, the field/struct **vegetable** contains two fields, *tomato* and *carrot*, which are also numbers.

From Example 1, you can see how to create a struct. In Matlab, unlike C, you do not need any special function call to create a data type, you just create it on-line. You can set the fields to be structs, vectors, matrices, strings, numbers, etc, and they all do not need to be the same type. Additionally, you can treat struct fields which are not structs just as you would treat any other variable.

Example 2:

First, create a struct **Class** with two fields: the class name, and the room where you have lab.

```
Class.name = 'ECE 386 Lab'
Class.room = 235
```

Now, you can treat the fields of the **Class** as any other variable:

```
num = 100 + Class.room
```

In this case, num would equal 335.

In addition to having the ability to use structs, you might also want to display text or variables to the screen. This can be done by either using the **disp** command, or by using **fprintf**. The **disp** command stands for display, and is used in the following way:

```
disp(statement)
```

where *statement* is whatever you want to display to the screen, and can be either a string or a number. However, *statement* must be static; that is, *statement* must be set before you display it. The command **fprintf** is like **disp**, only a little more powerful, in the sense that what you display can be dynamic. The usage of **fprintf**:

```
fprintf(statement, args)
```

where *statement* is the statement you want to display, and *args* are the arguments you pass. If you are familiar with C, you should be familiar with **fprintf**. The following example will illustrate how to use this command:

Example 3:

Suppose you are running a program that runs 4 simulations, and you want the program to display what test you are on, and how long each test took. For this example, assume the call to run a simulation is *Time = SIM(in)*, where SIM the simulation file which returns the time it took to run the simulation and *in* is an arbitrary input, stored in th variable Time. Using **disp**, you can type:

```
for i=1:4
Time = SIM(in)
disp('Simulation number:')
disp(i)
disp('Time = ')
disp(Time)
end
```

and the display will look like:

```
Simulation number:
1
Time =
0.25
Simulation number:
2
Time =
0.31
```

and so forth. You can compress this output and make it look better using the **fprintf** command:

```
for i = 1:4
Time = SIM(in)
fprintf('Simulation number %i, Time = %.2f \n', i, Time)
end
```

and the display will look like:

```
Simulation number 1, Time = 0.25
Simulation number 2, Time = 0.31
```

and so forth.

As you will notice, the format of the statement field of **fprintf** contains %i, %.2f, \n. These symbols are used to tell the command what the arguments are. The %i tells the command that the first argument is an integer, so display an integer; the %.2f says to display the next argument as a float going to 2 decimal places; \n signifies new line. Please see the Matlab help for more information. The **sprintf** command is simular to the **fprintf** command, only that **sprintf** returns a string.

Example 4:

```
        name = 'John Doe'
        age = '20'
        str = sprintf('The student named %s is %i years old.', name, age)
        disp(str)
```
This will display:
```
        The student named John Doe is 20 years old.
```
Where %s is used to tell the command that a string is the input.

Knowing how to use the **fprintf** and **disp** commands can make your script or program output more user friendly, but knowing the **sprintf** command can make your program very power when it is used in conjuction with the **eval** command. The command **eval** is used in the following way:
```
                              eval(statement)
```
Where *statement* is a string. The result is that Matlab evaluates what is stated in the string *statement*.

Example 5:
```
        str = '5+5'
        eval(str)
```
This returns:
```
        10
```

Example 5 gives a basic and trivial example of the **eval** command. In the next example you can see how to combine **sprintf** and **eval** into a more powerful program:

Example 6:
```
        Data1 = [1 2 3 4 5]
        Data2 = [2 4 6 8 9]
        Data3 = [1 3 5 7 9]
        Data4 = [4 1 2 3 6]
        Data5 = [9 8 7 6 7]
        Data6 = [5 5 5 5 5]
        ...
        Date1000 = [1 2 3 4 5]
```
Suppose you wanted to find the mean of each data set, and store the values in a matrix called AVERAGE. You could either use the mean command one thousand times, or you could write a Matlab script to do so:
```
        for i=1:1000
        str = sprintf('ave = mean(Data%i)', i)
        eval(str)
        AVERAGE(i) = ave
        end
```
This will automatically find the mean of each data set. Here, you need to use the sprintf command to update which data set you are using, and then use the eval command to evaluated to mean for that particular data set.

All of the commands seen in this file are found in the Matlab Help menu, and can be easily accessed from the Matlab web page or by typing "Help" at the Matlab command line. Please see the Matlab help for more information on these commands, and for more information on other useful Matlab commands.