Matlab Tutorial II

CS 310 — Fall 1999

1 Plotting

An especially useful feature of Matlab is that the graphical display of data can be done quite easily. If `x` and `y` are vectors of equal length, then the Matlab command `plot(x,y)` will open up a graphics window showing a curve connecting the points defined by the vectors `x` and `y`. The first point on the curve is `(x(1), y(1))`, which is connected with a line to `(x(2), y(2))`, and so on.

For example:

Matlab Command	Explanation
`x = [1 2 3 4 5 6]`	Create the vector of x-coordinates.
`y = [5 2 0 1 5 8]`	Create the vector of y-coordinates.
`plot(x,y)`	Now make the graph.

If you want to plot a function, you must create the `x` and `y` vectors first. Here is an example where we plot the function $y = \sin(x)$ from $x = 0$ to $x = 3$.

Matlab Command	Explanation
`x = 0:.1:3`	Create the vector of x-coordinates. This is a vector starting at 0, and ending at 3, and incrementing by 0.1
`y = sin(x)`	This creates a vector `y`, by taking the sine of each element of `x`.
`plot(x,y)`	Now make the graph.

Since some of these vectors can be quite long, you might want to suppress the printing of the results by adding a ";" to the end of a command. Here is another example, where we graph the function $y = \sin(x) + \sin(3x)$ in the interval $(-\pi, \pi)$. This time we'll add labels to the graph, and print it.

Matlab Command	Explanation
`x = -pi:.1:pi;`	Create the vector of x-coordinates. This is a vector starting at $-\pi$, and ending at π, and incrementing by 0.1. The last point in the vector won't be exactly π, but slightly less — do you see why?
`y = sin(x)+sin(3*x);`	This creates a vector `y`, by applying the function to each element of `x`.

(...continued from previous page)

Matlab Command	Explanation
`plot(x,y)`	Now make the graph.
`title('Test graph')`	Add a title to the graph.
`xlabel('x')`	Add a label on the x-axis.
`ylabel('y = sin(x)+sin(3x)')`	Label the y-axis.
`print test`	Creates a file called `test.ps` in the matlab directory. This is a postscript file that contains the graph you just made. To print it, double-click on it and print it from the PostScript viewer.
`close`	This will get rid of the graph.

The `title` command can be useful for labeling your plots with your name, so that when you print them out you can easily find which one is yours. We print the plot with the `print` command. The print command doesn't actually do any printing; it creates a postscript file in the current directory, which you can print later with `lpr`.

☞ Plot the function

$$y = 2x - 3\cos(5x)$$

in the interval $(-\pi, \pi)$ with Matlab.

Try plotting the function using `x = -pi:.1:pi`, and again using `x = -pi:.5:pi`. What is the difference between these two cases?

☞ Consider the vector `x = -pi:.1:pi`. The first element, `x(1)`, is -3.1416, which is very close to $-\pi$. The last element is 3.0584, which is not very close to π. We get this error because $-\pi$ and π aren't separated by a multiple of 0.1.

Figure out how you could fix this problem. That is, create a vector whose first element is very close to $-\pi$, and whose last element is very close to π.

☞ What do the following commands do?

```
t = 0:.01:2*pi;
plot(cos(t),sin(t));
```

In this example, we used the `plot` command a little differently: the x values don't need to be strictly increasing. The x values here are determined by `cos(t)`.

If you want to put more than one curve on the same graph, you can supply additional (x, y) vectors to the plot statement, like this: `plot(x1,y1,x2,y2)`. We'll do an example like this later, in Section 3.

If you would like more information about plotting, use Matlab's `help` command:

```
help plot
```

.

2 More about Multiplication

When we write an expression like `a*b`, there are two ways we can interpret this. The first is using normal matrix multiplication, the second is the "element-wise" multiplication. In Matlab, regular matrix multiplication is used whenever you type `a*b`. If you want to use element-wise multiplication, you must use `a.*b`. For `a.*b` to work, `a` and `b` must have the same size.

The same is true for division and exponentiation. `a./b` divides each element of `a` by the corresponding element of `b`. `a^2` computes the matrix product `a*a`, but `a.^2` squares each element of `a`.

Here are some examples:

Matlab Command	Explanation
`a = [1 2; 3 4]`	
`b = [5 6; 7 8]`	
`a*b`	This computes the matrix product of `a` and `b`. For example, the $(1,1)$ entry of the answer is $1*5+2*7$.
`a.*b`	This computes the element-wise product of `a` and `b`. For example, the $(1,1)$ entry of the answer is $1*5$.
`a^2`	Computes the matrix product `a*a`.
`a.^2`	Computes the square of each element of `a`.
`a^(-1)`	Computes the inverse of the matrix `a`.
`a.^(-1)`	Computes a matrix in which every element is the reciprocal of the corresponding element in `a`.
`a./b`	This divides each element of `a` by the corresponding element in `b`.

Most of the time, you won't want to use the element-wise operation. However, it is useful in cases where we want to compute something like $t\sin(t)$, where t is represented by a vector `t`. If `t` is a row vector with length 100, then `sin(t)` is also a row vector with length 100, or a 1×100 matrix. To calculate $t\sin(t)$, you must use element-wise multiplication (`t.*sin(t)`), since you cannot multiply a 1×100 matrix by a 1×100 matrix.

☞ Plot the function

$$f(x) = \frac{2x\sin(x^2)}{\sqrt{1-\left(\frac{x}{2\pi}\right)^2}}$$

in the interval $(0, 2\pi)$ with Matlab.

3 Using Files

Although Matlab works well interactively, for larger problems it is often more convenient to write the Matlab commands in a file and then have Matlab execute them. Then, if you want to change a few things, you won't need to enter in all of the commands again — you can just change the file and re-execute it.

Create a file (use Notepad) called `first.m` with the following lines in it. (Or, you can copy the file from the course Web page, found under the Matlab link)

```
% This file contains commands to plot the function t*sin(t^2)
% along with its derivative, in the interval 0 to pi.
clear;
t = [0:.01:1] * pi;
f = t.*sin(t.^2);
d = sin(t.^2) + 2*t.^2.*cos(t.^2);
plot(t,f,t,d);
title('The function t*sin(t^2), and its derivative.');
```

The lines beginning with a `%` are comments. Notice that the file begins with "`clear`", which wipes out previously existing variables. It is a good idea to always begin your Matlab programs with a `clear` statement.

All Matlab command files must have a "`.m`" extension. Save this file in your matlab folder.

To execute the commands in this file in Matlab, go to the `File` menu and select `Run Script`. Click on the `Browse` button, and you should see the first.m. Select it and click on `Open`, then click OK in the `Run Script` window.

4 Programming in Matlab

4.1 For loops

Here is an example of a Matlab `for` loop. It creates a vector `y` with the squares of the first five integers.

```
for i = 1:5
   y(i) = i^2
end
```

You can either type this into a file and load it into Matlab, or you can type it in directly to Matlab. When you press return after typing the first line, Matlab waits for you to continue typing since it knows that the `for` loop hasn't been finished yet.

Notice that `1:5` is a vector, and that `i` got assigned to each element in the vector. You can use a `for` loop with any vector:

```
for j = 1:2:10
   x(j) = j^3
end
```

In this loop, only the odd numbers get cubed. It is often a good idea to use a ";" at the end of lines inside the loop to suppress printing. Here is a more complicated example:

```
y = [ 1 3 1 6 8 6 4 1 6 ];
w = zeros(1,10);
for i = y
   w(i) = w(i) + 1;
end
w
```

Can you figure out what this loop is doing? In the loop, the values of `i` are successively set to the elements of `y`. So, the first time through the loop, `i` is 1, so `w(1)` is incremented by 1. The second time through, `w(3)` is incremented by 1. The third time, `w(1)` is incremented by 1 again!

The last statement (where `w` appears on a line by itself) displays the vector `w` on the screen.

Loops can be nested, as in the following example, which creates this matrix:

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

```
for i = 1:5
  for j = 1:5
     H(i,j) = 1/(i+j-1);
  end
end
H
```

☞ Before executing the following Matlab commands, can you figure out the result?

It might help if you remove the semicolon from the statements inside the loop, to get a better idea of what is going on at each step. First just look at `M` — how is it being created? Then look at `B`.

If you're going to be doing lots of experimenting, you should enter the commands into a file, so that you don't need to keep re-typing them.

```
B = zeros(6,6);
for k = 1:5
   M = [ones(k,k) zeros(k,6-k); zeros(6-k,k) ones(6-k,6-k)];
   B = B + M;
end
B
```

4.2 If statements

The following example shows how to use an `if` statement in Matlab.

```
if (x == 1)
  x = 2;
  y = 1;
else
  x = 1;
  y = 2;
end
x, y
```

In this example, if the value of `x` is 1, then we do two things: set `x=2` and `y=1`. Otherwise (if `x` is not equal to 1), then we set `x` to 1, and set `y` to 2. The last line prints the values of `x` and `y`. For this example to work, the variable `x` must already exist, or Matlab will complain.

Be sure to notice that you must use `==` to test for equality. You use a single equals sign for assignment, as in `x=1;`

It is possible to put the entire if statement onto a single line, like this:

```
if (x==1) x=2; y=1; else x=1; y=2; end
```

Usually, you will want to use the original multi-line style, since that makes your Matlab programs easier to read. When typing these directly into Matlab, though, it is often easier to put everything on a single line.

Here is another example. See if you can determine what `x` will be before you type it all in.

```
for i = 1:10
  if (i > 7)
    x(i) = 7;
  elseif (i < 3)
    x(i) = 3;
  else
    x(i) = i;
  end
end
x
```

This example also introduced `elseif`, which you can use if you have more than two alternatives you want to test for. Matlab first tests to see if `i` is greater than 7. If so, then it executes `x(i)=7;`, and then it skips the rest of the `if` statement. If `i` is not greater than 7, then Matlab checks the next test: if `i` is less than 3. If so, Matlab executes `x(i)=3;`. If both `i>7` and `i<3` have failed, only then will Matlab execute the statement `x(i)=i;`.

In the next example, we'll create a matrix with non-zero elements on the main diagonal, and the diagonal above the main diagonal.

```
for k = 1:5
  C(k,k) = 1;
  if (k < 5)
     C(k,k+1) = k
  end
end
```

The above can be done more easily with this one-line command:

```
B = eye(5) + diag(1:4,1)
```

You can learn more about the `eye` and `diag` commands using `help eye` and `help diag`.

☞ See if you can write a loop in Matlab to create a 10×10 matrix with 1's along the diagonal and 2's above *and* below the diagonal. (A solution is in the file `answer.m`, located on the course Web page under the Matlab link.)

☞ As a final programming example, copy the Matlab program `slow_program.m` from the course Web page. Open up this file in an editor and try to understand what the program is doing. Run the program within Matlab.

Which method is faster? How much faster?