Matlab Lab, Math 204

Matlab physically resides on each of the computers in the Math Dept's computer lab. We can also use **Octave**, which is a free alternative to Matlab (but most things are compatible with either package). For home use, if you're thinking about going into any kind of engineering, Matlab is probably going to be used, and the student version is available for about $100. We can use Octave in this class, and you can do a quick internet search to find the latest version you can download to your home computer.

There is a convenient alternative to both of these- **Octave Online** is a web-browser based version of Octave (and Matlab) that is free to use. I would encourage you to enroll- that way you can upload the code that I'll give you in class. You can enroll just by using your Google (Whitman) email.

What does Matlab do?

Matlab (short for Matrix Laboratory) was originally designed as a way to do fast and easy linear algebra computations, so it uses vectors and matrices as its basic data type.

We'll be using Matlab primarily for computing and plotting, and you can make Matlab do computations three different ways:

- Type commands directly into the command window (you'll do your computations live- We'll start with this below).
- Have your Matlab commands typed into a separate text file (called a **script file**), and then have Matlab read these commands in. Matlab will then execute those commands as if you were typing them in. These text files are saved with an m suffix (like `script01.m`).
- Define your own functions by typing a separate text file (called an **m-file**). These text files are also saved as with an m suffix. We'll show you how you can distinguish between a function and a script a bit later.

Introductory Computations

Matlab understands the usual mathematical commands. Type the commands given on the left side and see if the output is what you would predict. (the whitespace below is mainly for readability).

The prompt for the command window should look something like this: `[>`

Enter	Explanation
`( 2 * 5 ) + 3^2`	Arithmetic- note the exponent
`3**2`	This is another way to exponentiate.
`x = 1 + 2*3;`	Equal sign is assigning the value of 7 to x.
`x = 1 + 2*3`	The semicolon will suppress the output. Take it out if you want to see the result.
`4*x`	Compute $4x$ (and display the result)
`t =[1,2,3];`	Assign the array to variable t
`t`	Print the array in t
`t = t+1`	Add 1 to each element of t, store the result back in t
`t = t.^2`	Square each element of array in t
`t+2=t;`	This gives an error - Why? (The operation must be on the right. The variable must be on the left.)
`cos(5*pi/4)`	The constant π uses smallcase p
`exp(4)`	The exponential, e^4. To get the value of e, type exp(1)
`clear`	Clear everything from memory.
`clc`	Clear the commands from the command window
`help sin`	Use help: help *function*
`demo`	Lists demos

Helpful tips:

- If you make an error, use the up arrow key to scroll through your commands (so you do not have to type it all over again).
- You can put multiple commands on one line, if you separate them with commas:

```
>> a=7; b=cos(a), c=cosh(a)
```

Arrays

It's easy to make arrays in Matlab. For example, let's enter the following two arrays:

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 3 & 1 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & -1 \\ 1 & 2 & 3 \end{bmatrix}$$

We can enter a matrix in a single line (we'll use the semicolon to denote the end of a row):

```
A=[1,2,-1;3,1,0]
```

By leaving off the semicolon at the end, the array will be displayed at the end. Sometimes you don't want this to happen. For example:

```
C=ones(100,1)
```

will create a column vector with 100 elements in it- We don't want that displayed.

Tip: When dealing with larger arrays, be sure to end the command line with a semicolon so that the array does not get displayed!

Heres the other way to entry an array (use the carriage return to denote the end of a row):

```
B=[1 1 0
0 2 -1
1 2 3]
```

Usually, when you apply a built-in function to an array, the computation is made element-wise. For example, with the array A defined earlier, `sin(A)` is a 2×3 array whose elements are

$$\begin{bmatrix} \sin(1) & \sin(2) & \sin(-1) \\ \sin(3) & \sin(1) & \sin(0) \end{bmatrix}$$

The dot operator

When you see a dot with an operator, it is telling Matlab to compute that operation element-wise. For example, `A^2` would give you an error, but `A.^2` would square each element of the array.

Similarly, if you've had linear algebra, you'll know that $A * B$ is matrix-matrix multiplication that is defined, but in this case, $A. * B$ would not be defined, since the arrays have different sizes.

Some special cases

If you've had linear algebra, you'll know that the only time that addition between two matrices is defined is when they have the same size. However, when working with an array filled with data, often we'll want to add a row to every row of the matrix, or add a column to every column of the matrix.

We can add a column vector to a matrix- it will automatically add the column to each column of the matrix. Similarly, we can add a row vector to a matrix- it will add the row to each row of the matrix. For example:

```
A+[1,2,1]
A+[-2;1]
```

Some special arrays

Some arrays are so common, they have shortcuts for building them. For example, matrix with m rows and n columns, filled with zeros, filled with ones, and filled with random numbers are defined (respectively) as the following (this example sets $m = 3, n = 7$):

```
m=3; n=7;
A=zeros(m,n), B=ones(m,n), C=rand(m,n)
```

In plotting and other constructions, it is common to want a vector of numbers evenly spaced between a minimum number and a max number- the command is `linspace`. For example, the following command gives you a row vector that has 10 numbers that are evenly spaced between -1 and 5:

```
t=linspace(-1,5,10)
```

Plots

For the Matlab documentation, you can type `doc plot`

To get a plot, Matlab needs two arrays (or vectors). One defines the domain points, the other the range points (so they need to have the same number of points in them).

For example, the following series of commands creates a vector (array) of 150 evenly spaced points in the interval $[0, 3\pi]$, then computes the sine of those values, and plots:

```
t=linspace(0,3*pi,150);
y=sin(t);
plot(t,y);
```

You can also plot the "inverse" sine by reversing the t and y: `plot(y,t)`

If we leave off the domain, Matlab assumes it is the integers from 1 to however many elements are in the vector. For example, `plot(y)` or `plot(y,'r*')`

There are other ways of creating the vectors- We'll see more later.

Function Iteration (Script Files)

We will want to perform function iteration to construct orbits and cobweb diagrams. This is done by using what is called **a "for" loop**.

We will want to have Matlab perform several computations at once, and we do not want to have to keep typing everything "live". We will create a **script file**, which is just a text file with Matlab commands. When the name of the script file is typed in the command window, Matlab will execute the commands in the script.

Matlab comes with a nice editor- We'll open it now by typing **edit** in the command window. Type in the following set of commands, and save the result as `sample1.m` (Be sure that your file has the .m suffix):

```
clear
x0=0.3;
X(1)=x0;
f=inline('x.^2-1');

for k=1:30    %k is the index variable for the loop
   X(k+1)=f(X(k));   %This command is repeated
end                  %End of the "for" loop
plot(X,'.')      %Plot using dots
X'               % ': This transposes the array
```

Once saved, back in the command window, type `sample1` to see the commands executed (note that the variables $x0$, X, y and others have been created in the workspace).

Function Iteration (Function Files)

A function file is typed out in text, and has a special first line. For example, we will convert our previous *script* file into a *function* by adding the following:

```
function X=firstfunc(x0)

f=inline('x.^2-1');
X(1)=x0;
for k=1:30    %k is the index variable for the loop
   X(k+1)=f(X(k));   %This command is repeated
end                  %End of the "for" loop
plot(X,'.')      %Plot using dots
X'               % ': This transposes the array
```

Octave-Online note: Copying a file is suprisingly a pain in the neck. The easiest way to do this is to create a new file, then go to the file you want to copy, and copy the text. Then open the new file and paste it in, then you can rename the file. We'll do an example in class.

To run this function, save it as `firstfunc.m`, and in the command window, type:

```
firstfunc(0.3)
```

Difference between Functions and Scripts

- Functions have inputs and outputs, everything else in memory goes away once the function has completed its computations (scripts, being like live computations, keep everything in memory).
- Careful- Both have `.m` suffixes. You can always tell a function file by its very first line.

- Unlike a function in mathematics, functions in Matlab can use multiple inputs and multiple outputs. Here is an example of a function that takes in two numbers stored in x and y, and outputs two vectors, t and s. Type it out and see how it works:

  ```
  function [t,s]=sample2(x,y)
  t=[x,y];
  s=t';
  ```

  Save this file as `sample2.m`, then in the command window, type something like: `[a,b]=sample2(2,3)` You will see the results of the function in the variables a and b.

Chapter 3 Experiment

In Chapter 3, we are asked to do a computer experiment with the doubling function, $D(x)$:

$$D(x) = \begin{cases} 2x & \text{if } 0 \le x < 1/2 \\ 2x-1 & \text{if } 1/2 \le x < 1 \end{cases}$$

It is complicated enough that we don't want to use the `inline` function as in a script, so we'll write our own function to do it (we will introduce the "if-then" statement as well!):

```
function y=doubling(x)
%function y=doubling(x) is the doubling function from Ch 3
idx=length(x);
for j=1:idx
    if x(j)>=0 & x(j)<1/2
        y(j)=2*x(j);
    elseif x(j)>=1/2 & x(j)<1
        y(j)=2*x(j)-1;
    else
        y(j)=NaN;
    end
end
```

Save this file as `doubling.m`

Now we will type a script function that will set up the initial value of our orbit, compute the orbit, and show us the result as a plot:

```
X(1)=1/5;
for k=1:20
  X(k+1)=doubling(X(k));
end
plot(X,'.')
```

Type and save as `ch3.m` In the command window, type `ch3` to see the results. Now, change the initial point to 1/9 and try it again. Increase the number of iterations until you see an error in the plot.

Cobweb Diagrams

The full code will take us too far into coding for right now, so we will simply download the functions we need and use them. Download and save the functions `cobweb.m` and `iterates.m` There is a script file as well in `driver1.m`

To run the cobweb diagram of the logistic function, type `driver1` in the command window.

We can also run the cobweb diagram "live". Look at the cobweb diagram for the doubling function by typing the following into the command window:

```
cobweb(@doubling,1/5,20,0,1);
```

Homework

The homework for today is:

- Ch 3 exercises: 2, 3, 6, 7(b*, d*), 8, 10*, 11, 12.
- Section 3.6 (The Computer May Lie). You only need to do the experiment for the doubling function (*). Note that exercise 14 may help.