Getting started with Matlab

Matlab physically resides on each of the computers in the Math Dept's computer lab. Once you have logged in, find the Matlab option under `Applications->Math Lab ->Matlab`, then drag the icon to your upper toolbar.

You can make Matlab do computations three different ways:

- Type commands directly into the main 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.
- Define your own functions by typing a separate text file (called an **m-file**).

Introductory Computations

Matlab understands the usual mathematical commands (if you've worked with Maple, it is similar). See if you understand what Maple outputs for each of the lines below (the whitespace below is mainly for readability):

Enter	Explanation
`( 2 * 5 ) + 3^2`	Arithmetic Operators
`x = 1 + 2*3;`	Assign operation to x
	Semicolon does what?
`4*x`	Compute $4x$
`t =[1,2,3];`	Assign the array to variable t
`t`	Will show the array
`t = t+1`	Overwrite t with the value $t+1$
`t = t.^2`	Square each element of array in t
`t+2=t;`	This gives an error - Why?
`cos(5*pi/4)`	The constant π uses smallcase p
`exp(4)`	The exponential, e^4
`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)
```

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
```

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';
  ```

  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
```

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);
```

The functions are printed below as well for future reference (You may safely ignore them for the time being- Just download them from the class website).

```
function cobweb(fcn,x0,N,xmin,xmax)
% fcn is the name of the function, x0 is the starting
% value, N is the number of iterates, xmin and xmax
```

```
% give the range of x-values to be plotted, and

dx=(xmax-xmin)/1000;
x=xmin:dx:xmax;
y=feval(fcn,x);

plot(x,y,'b',[xmin xmax],[0 0],'k',[0 0],...
  [min(y)-.1*abs(min(y)) max(y)],'k',[xmin xmax],...
  [xmin xmax],'g');
hold on
Y=iterates(fcn,x0,N);
YY(1)=Y(1);
for i=1:N
    XX(2*i-1)=Y(i);
    XX(2*i)=Y(i);
    YY(2*i)=Y(i+1);
    YY(2*i+1)=Y(i+1);
end;
XX(2*N+1)=Y(N+1);
plot(XX,YY,'r',x0,0,'r*');
```

This uses `iterates.m` shown below.

```
function Y=iterates(fcn,x0,N)
% fcn is the name of the function, x0 is the starting
% value, N is the number of iterates

Y=[x0];
x=x0;
 for i=1:N
        y=feval(fcn,x);
        Y=[Y y];
        x=y;
end
```

Finally, the driver:

```
logistic=@(x) 3.*x.*(1-x);
cobweb(logistic, 0.2, 5, 0, 1);
```