Introduction to Matlab

This is to be used in conjunction with some book or article of more substance, e.g. the student edition of Matlab, or the Matlab Primer, which is online in in the Math department's home page. Go to `www.math.ku.edu`; then click on *local users*, and then on *computer system*; it's near the bottom of the page. Also, there's a sort of user manual at *http://en.wikibooks.org/wiki/Matlab*.

And there's a free (share ware) program called *Octave*, presumably available at *sourceforge.net* which does most of the same things MatLab does, and in particular, takes advantage of the same highly accurate public-domain computational matrix tools used by MatLab. Octave has most of the same functionality at MatLab, but it's not identical, and scripts which work in one language may not work the same (or at all!) in the other.

Also, different versions of Matlab have slightly different graphical user interfaces (GUIs). The one in the Math Department lab runs under Linux (Ubuntu, at the moment); this is slightly different from the Mac OSX GUI, and probably different from Windows.

Note: After 30 years of market dominance, Windows is still not a genuine time-sharing system - if you have any choice in the matter, you should avoid it for computational work - for one thing, the entire system crashes too frequently, which is unacceptable if you're doing serious computing. Octave and Matlab both run under Linux. Most Linux distributions include boot loaders and disk partitioning systems that permit you to run Windows (if you must) on one partition, and Linux on another.

1 Preliminaries

If you haven't done much work on computers before, perhaps the main thing to remember is that the computer only knows what you typed, not what you might have *meant* to type. And the computer is very inflexible: if you are supposed to enter something like **x, y, z** and you omit the commas separating the variables, the results may not be what you had in mind. By the same token, if you insert commas because it seems reasonable to do do, you may also be unpleasantly surprised. Matlab on Unix machines is case-sensitive: that means you can have two different variables named **fleep** and **flEep**. Unless there's some particular reason to do otherwise, it is best to type everything in lower case.

(For some reason, Matlab displays most function names in *upper case* in response to the "help" command (see below). Don't get confused; the proper syntax for using the function is lower case.)

Matlab is a "research platform" which can be used to compute nearly anything numerically (e.g., roots of equations, inverses of matrices, Fourier and Laplace transforms). The key word is "numerically". As an example, although we know from class that the general solution to the ODE

$$y'(t) + 2y(t) = 5$$

is $y(t) = Ce^{-2t} + 5$, this is not, in general something that Matlab can tell us. What Matlab *can* do is solve this IVP numerically, using a variety of different numerical methods. Never

mind how this takes place (you'll find out later) — the end result of the computation will be two *vectors* named t and y. And $y(i)$ will be the numerical solution to the IVP at the time $t(i)$. This brings us to the first point, which is that the basic objects in Matlab are *vectors and matrices*, rather than scalars (which are treated as 1×1 matrices). All the algebraic operations (+. -, *, /, and ^) are designed to work with vectors and matrices. Thus if t=[1,2,3.5,7], (a row vector), sin(t)=[sin(1), sin(2), sin(3.5), sin(7)]. But t*sin(t) makes no sense. If you want to form the vector whose components are $t(i)$*sin($t(i)$), then the correct expression is

`y = t.*sin(t)`,

where the "dot" indicates that the operation is to be performed "pointwise" on the elements of the vectors. Where there's no possibility of confusion, Matlab does what you think it shoud: for instance, with `t` as above, `25*t` will be a vector with each element equal to 25 times the corresponding entry in `t`. Siimilarly, `t + 2.4` is a vector with the same dimension as `t` but with 2.4 added to each entry. We'll pick up other details as we proceed.

2 How to plot data in MATLAB

(This is for your information; work through it, but do not hand it in.)

To display 2-dimensional data in MATLAB, use the function **plot()**, whose argument needs to be a vector. MATLAB does not do symbolic math, so a command like **plot(sin(t))** makes no sense unless MATLAB knows what "t" is (It does have its own algorithms for sine, cosine, etc.). Proceed thusly to graph the sine function on the interval [0,4*pi] using 125 equally spaced points. Enter the commands as written to the right of the MATLAB prompt >>:

```
>> t = 0:4*pi/125:4*pi
```

This creates a vector named t with entries [0,4*pi/125,8*pi/125,...,4*pi]. Notice that this entire list of numbers was displayed on the screen, which is not generally what you want. To *SUPPRESS* the display, end the line with a semi-colon(;)!

```
>> x = sin(t);
```

This creates a vector named x with entries [sin(0),sin(4*pi/125),...,sin(4*pi)]

`>> plot(x)` Notice the units ...

`>> plot(t,x)` What's the difference?

`>> y = sin(t+.1);` Phase shift

`>> z = sin(t+2);` Another

`>>hold on` The next graph drawn will not erase the present one.

```
>> plot(t,y,'r')
```

```
>> plot(t,z,'o')
```

```
>> title('Sine function with phase shift')
```

```
>> xlabel('t')
```

```
>> ylabel('sin(t+a)')
```
Self-explanatory

```
>> hold off
```
("releases" the held plot)

```
>> plot(x,z)
```
What is this?

```
>> subplot(3,1,1); plot(t,x)
```

```
>> subplot(3,1,2); plot(t,y)
```

```
>> subplot(3,1,3); plot(t,z)
```

Subplot(m,n,p) creates an m by n array of plots on a single screen, and refers to the pth one, counting from the upper left. Experiment with this a bit, till you're sure how it works. Notice that the semi-colon can also be used to separate distinct statements, so that we can write more than one on a line if that's convenient.

To get a printout from MATLAB:

```
>> print
```

Sends whatever's displayed to the printer; once something's gone from the screen, it can't be printed.

```
>> print filename.ps
```

Command to save the current image as a text file with the name "filename.ps". At some later time, this can be sent to the printer. Matlab prints postscript (.ps), rather than PDF files. This will not be a problem for the printer.

VERY IMPORTANT!! MATLAB has an extremely useful "help" facility. If you don't know how the print command works, do

```
>> help print
```

and similarly for everything else. In the next section, you may want to type help eul at some point....

Before beginning the next section, do

>> **clf**

to get back to a single figure. There are other ways to do this, including just killing the figure. Matlab will create another when it needs to.

3 Housekeeping and getting around

To save your favorite variable(s) so that they'll be available to you the next time:

```
>> save myfile x y z
```

saves x, y, and z in a file named myfile.mat. You can quit, and then log in again, and recover the variables by doing

```
>> load myfile
```

To leave Matlab, type

```
>> quit
```

Last but not least, YOU CAN RECALL AND EDIT THE PREVIOUS COMMANDS. Using the up-arrow will reproduce the command just entered; pressing the key 3 times will bring back the command entered 3 lines back. You can also recall things by typing just enough info to give Matlab the information it needs. For instance, a vector named 'xdot' could be recalled to the command line and edited by just typing 'xd', or even 'x' if that identifies it uniquely.

4 User defined functions and procedures

MatLab is also a programming language, not too different in principle from BASIC. In addition to simply executing single commands, it can execute sequences of them (called "scripts"), and feed the results of one computation into another. There are two forms of scripts; both are written in plain text and stored as text files in the form "filename.m". The first is just a list of commands to be executed in order. It's called by simply typing its name at the command prompt. The second is a function, to which arguments can be passed. Here's an example in which there are two functions: the first is *rhs.m* which stands for the "right hand side" of the DE; it needs to be edited each time you change the DE. For example the file rhs.m might contain the following:

```
function x = rhs(t,y);
x = t*y^2;
```

This would be, for instance, the right hand side of the DE $y' = ty^2$. We could use this function in another program (function) called euler, which implements Euler's method, and might look like this (note: the percent sign denotes a comment to Matlab; anything following the percent sign is ignored.)

```
function [t, y] = euler(y0, ti, tf, n);
%This function will have two vectors t and y as outputs; it will have as inputs
%the numbers y0 (initial position), ti (initial time), tf (final time),
%and n (number of time steps).  This simple example does
```

```
%no error checking!!!!

dt = (tf-t1)/n;                    % this is the step size
t = zeros(n+1); y = zeros(n+1);   % it saves time to initialize
t(1)=ti; y(1)=y0;             % index in MatLab goes from 1 to wherever, not 0
for i=1:n
  t(i+1) = t(i) + dt;
  y(i+1) = y(i) + dt*rhs(t(i),y(i));
end
```

By construction, the variables t and y are returned by the function. They can be called something else (see below). Variables internal to the function which are not returned, like dt are not accessible. They're "private" rather than public.

So, you can plot the numerical solution given by Euler's method to the IVP $x' = tx^2$; $x(2) = 3$ on the interval $2 \leq t \leq 5$, using 50 data points, by doing

```
>> [t,x] = euler(3, 2, 5, 50);
>> plot(t,x)
```

Both files euler.m and rhs.m must be in the MatLab path.

That should be enough to get you going.....You'll learn more as you proceed through the labs. Be aware that Matlab is a professional research tool, used daily by thousands of working scientists. It will be a useful addition to your collection of skills to become proficient in its use.

There is also a student edition of MatLab, which has some relatively reasonable limitation on the size of vectors and matrices it will process, and includes a user manual, and is (last time I checked) reasonably priced. And there is also Octave, as mentioned above. The last time I used Octave, it needed gnuplot (available also at sourceforge.net) to plot the data; the current version may not.