Matlab Intro and Exercises

Professor L.G. de Pillis
Department of Mathematics
Harvey Mudd College
Claremont, CA 91711

1 Matlab Introduction

- On the Linux workstations, start up Matlab with the command "matlab". In the Matlab command window, you will get the Matlab prompt which looks like this: $\gg$. To exit Matlab, type "quit", or exit through the Matlab pull-down menus.

- Make use of the on-line help. Some possibilities:

help help	←	Help on help
help	←	List of help topics
help general	←	List of general purpose commands.
intro	←	An introduction to Matlab, and a good start for those unfamiliar with Matlab.
demo	←	A demonstration of Matlab capabilities.
doc	←	Hypertext documentation.
lookfor	←	Keyword search through help entries.

- Matlab is interactive. You can give commands at the command prompt $\gg$; you can also write script files (programs), called ".m-files". A ".m-file" contains a sequence of standard Matlab commands. The file must always have a ".m" suffix, e.g., *myfile.m*. Remember, when you want to run your own ".m-file", you must either be running Matlab in the same directory in which the ".m-file" sits, or you should change your path setting inside Matlab so Matlab looks in the correct directory. You can also change directories within Matlab by using the "cd" command (the same as you would at the Unix prompt).

- There are many resources available on the Web for helping your learn about Matlab. For example, a google search will lead you to:

  http://www.utexas.edu/cc/math/tutorials/matlab6/matlab6.html

  http://www.engin.umich.edu/group/ctm/basic/basic.html

  http://www.owlnet.rice.edu/~elec241/matlab.html

  More links can be found on the Math 165 website (http://www.math.hmc.edu/~depillis/MATH165/)

- If you want to use a packaged ODE solver in matlab (like ode23, ode45, ode113, ode15s, ode23s, ode23T, ode23tb), here is an example of how to get started: Suppose you wish to solve the ODE

$$dy/dt = K(y - s)$$

a model of heat dissipation using Newton's law of cooling. You must first create a Matlab function file that defines the right hand side of your equation, in this case $K(y - s)$. Call this function FOO, and it should look like this:

```
function yprime=FOO(t,y)
K=-0.1;
s=10;
yprime=K*(y-s);
```

Next, we set our integrating options using the matlab command "odeset." For example, to set the relative error tolerance to 0.5, we type

```
options=odeset('RelTol',0.5);
```

The following script will solve the ODE and plot the computed results against the exact solution. These lines can be typed at the Matlab command prompt, or placed in a ".m-file", and run from Matlab.

```
K=-0.1; s=10;
t0 = 0; tfinal = 60;
y0 = 100;
[t y] = ode23(@FOO,[t0 tfinal],y0,options); %Note the use of the @ sign
trusol = 90*exp(K*t)+s; %get values for true solution
plot(t,y,'r+',t,trusol,'bo');
xlabel('time');
ylabel('y value');
title('Newtons Law of Cooling');
legend('Computed Solution', 'True Solution');
```

- Other useful information:

  % : This symbol always precedes a comment

  A semicolon ; can be used to suppress output

  fprintf : Can be used to print output to the screen or to a file. Examples:

  - To the screen:

    ```
    fprintf('This is a test, x=%g \\n',x).
    ```

    This command prints the number x in either exponential or fixed point formant (whichever is shorter) and then starts a new line.

  - To a file called "output.txt":

    ```
    fid=fopen('output.txt')
    fprintf(fid,'hello %g', x)
    ```

  input : Get input from the keyboard, e.g.,

  ```
  a = input('Left Endpoint a = ?')
  ```

  Logical operators :

  - and: &
  - or: |
  - not: $\sim$
  - exclusive or: xor

  Condition testing : There is an *if ... elseif ... else ... end* construct. Use *help if* for more information.

  Looping : There are for and while loops. Use *help for* and *help while* for more information.

  Defining a function : Use *help function* to see an example as to how to define your own functions.

2 Exercises

Matlab is an interactive program to help you with numeric computation and data visualization. Matlab is built upon a foundation of sophisticated matrix software for analyzing linear systems of equations.

The following is a list of simple exercises to work through which will help you familiarize yourself with Matlab. All exercises will work with Matlab version 6.

Save everything that is typed in your Matlab session in a file called 'MATintroexercises' by using the 'diary' command.

1. Matlab contains all the standard arithmetic operations, and has many built-in functions. These exercises will familiarize you with how to carry out arithmetic calculations in Matlab.

   (a) Create a variable a and set $a = \pi + 3.0$. Create b and set $b = \cos(a)$.

   (b) Type 'format long', and redisplay variable b. Does it look different?

(c) Type 'format bank', and redisplay variable b. Does it look different?

(d) Calculate $a \times b$. Calculate $a + b$. Calculate a/b. Calculate b^a.

2. Since matrices are the basis for all of Matlab, we start with some simple exercises to familiarize you with some of the basic commands available to you. All basic arithmetic operations can be performed on matrices as well.

(a) Create a matrix
$$A = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Find the inverse of A, call it $invA$.

(c) Confirm that $invA$ is an inverse by creating the identity matrix I with $I = A \times invA$.

(d) Add 2 to every element in $invA$.

(e) Find the eigenvalues of A (hint: eig). Save them in a vector called $EigA$.

(f) Find the coefficients of the characteristic polynomial of A (hint: poly).

(g) Find the roots of the characteristic polynomial of A, and place them in a vector called $RootsA$. How do $RootsA$ and $EigA$ differ?

(h) Find A^T. Call it At.

(i) Create a 3×1 matrix $b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

(j) If $Ax = b$, find x using the 'inv' command.

(k) If $Ax = b$, find x using '\'.

(l) Find A^3.

(m) If vector $x = [x1\ x2\ x3]$, create in one line vector $y = [x1^2\ x2^2\ x3^2]$. (hint: .^)

(n) In one command line, create a diagonal matrix whose diagonal elements are those of matrix A.

(o) Create a 4×4 magic square (hint: magic). What are the column sums (hint: sum)?

(p) Create a submatrix B of the matrix A so that $B = \begin{bmatrix} 1 & 0 \\ 2 & 4 \end{bmatrix}$, using colon notation.

3. Now we will get some practice visualizing functions.

(a) Draw the graph of the sine function over the interval -4 to 4 by first creating a vector $x = [-4\ -4.1\ -4.2\ \cdots\ 3.9\ 4.0]$ in one command line. Then let $y = \sin(x)$. Use the 'plot' command.

(b) Now use the 'plot' command to plot $\sin(x)$ in red and $\cos(x)$ in blue on the same graph.

(c) Over the same interval in a different graphics window (use 'figure'), plot $e^{-(x+1)^2}$ in a blue line dotted with circles, e^{-x^2} in a magenta line dotted with stars and $e^{-(x-1)^2}$ in grean line dotted with triangles. Use the 'legend' command to label the three curves.

(d) Now plot the following parametrically defined curve: Let t run from 0 to 2π by steps of .001. Let $x = \cos(3t)$, and $y = \sin(2t)$. Plot x versus y.

(e) Give your plot a title. Call it "Cool Parametric Curve". Label the axes of the plot x and y. (hint: title, xlabel, ylabel).

(f) Plot the curve $C(t) = (3\cos(t), \sin(t), 1/t)$ with $0.1 \leq t \leq 4\pi$. Use the 'plot3' command.

(g) Plot the graph of $z = e^{-x^2}e^{-y^2}$ over the square $[-2, 2] \times [-2, 2]$. Use the 'meshgrid' and 'mesh' commands.

(h) Redraw the above surface using the 'surf' command instead of 'mesh'. What has changed?

4. In these exercises you will learn to write Matlab m-files, both script files and function files.

(a) Create a script file called 'plotbump' which carries out the steps in (3g). Test it by typing 'plotbump' at the Matlab prompt.

(b) Create a function file called 'rim' so that the entry 'rim(m,n)' at the Matlab prompt returns a randomly generated integer matrix with entries between zero and nine.

(c) Modify the above function 'rim' so that 'rim(m,n)' still returns a matrix as described above, but additionally, 'rim(m,n,a,b)' returns a randomly generated integer matrix with entries between the integers a and b.

(d) A function can have multiple output arguments. Create a function 'stat' so that for a vector x, stat(x) returns the mean and standard deviation and maximum element of x, and so that for a matrix X, stat(X) returns three row vectors containing, respectively, the mean and standard deviation and maximum of each column of the matrix.

(e) A function can also take another function name as an argument. A function may also contain loops and conditional statements. Write a bisection algorithm to search for the roots of a function. Put the following code into the body of a function named 'bisect' with Matlab file name 'bisect.m':

```
function[root,iter] = bisect(a,b,tol,maxit,fn_str)

for iter=1:maxit
    midpt      = (b-a)/2;
    x          = a+midpt;
    root(iter) = x;
    fx         = feval(fn_str,x);
    fa         = feval(fn_str,a);
    %test for both a small interval and a function value near zero:
    if ((abs(midpt) < tol)&(abs(fx) < tol))
      return;
    elseif (fa*fx > 0)
      a = x;
    else
      b = x;
    end; %if
end; %for
```

Use 'bisect' to search for the root of $f(x) = x^3 - 2x - 5$ for x between one and three. Test the accuracy of the solution by checking how close $f(r)$ is to zero. You will also have to write a function file for $f(x)$.

Note: the name of the function should be passed through to 'bisect' as a string, or as a function reference. For example, at the Matlab prompt, if you have defined a function that you have named 'f3' (and saved in a file called 'f3.m'), and then set values for the input arguments, such as

```
>> a = 1; b = 3; % set the left and right search interval values
>> tol = 1e-8;  %set the small value for the error tolerance
>> maxit = 100;  %set a value for the maximum number of iterations
```

then you can call 'bisect' to find the root of the function named 'f3' in one of these two ways:

```
>> [root_bi,it_bi]=bisect(a,b,tol,maxit,'f3');
```

or

```
>> [root_bi,it_bi]=bisect(a,b,tol,maxit,@f3);
```

When choosing interval endpoints a and b, be careful to choose them so that $f(a) \times f(b) < 0$, or else the answer that 'bisect' returns will be an endpoint, and not necessarily a root. The function 'bisect' can be modified to make sure the user has input appropriate interval endpoint values. How do you think you would do that?

Note: The 'bisect' function will even find roots of built-in Matlab functions. For example, try typing

```
>> [sinroot,itcnt]=bisect(.5,3*pi/2,1e-8,100,@sin);
```

and 'bisect' will return a root of $\sin(x)$, which is stored in the last value of the vector 'sinroot' and will be a number very close to π. The value of 'itcnt' will show how many iterations it took to find the root of $\sin(x)$ to within $1e-8$.