THE UNIVERSITY OF ARIZONA®

Summer 2009 REU: Introduction to Matlab

Moysey Brio & Paul Dostert

June 29, 2009

Using Matlab for the First Time

Click on Matlab icon (Windows) or type

$$>> matlab\ \&$$

in the terminal in Linux. Many windows will pop up when Matlab is launched. Some are useful, but generally I close all except the ***Command Window***.

The most important commands to learn in Matlab are ***help*** and ***doc***.

- *help CommandName*: displays help information for *CommandName* INSIDE the Command Window.
- *doc CommandName*: displays html help in a new window.

Try the following in Matlab:

» help log

» doc log

Search for 'exponential' in the Help Navigator

Assigning & Unassigning Variables

We use = to assign a variable a value and we use ***clear*** to clear a variable. To find out what variables are assigned, we use ***who***.

Do the following in the Matlab Command Window:

```
» a = 1
» b = 2; c = pi;
» a
» a,b
» who
» whos
» clear a
» who
» clear all
» who
```

Calculator Work

Nearly everything that would work on a calculator, such as a TI-89, works exactly as expected in Matlab. This includes complex arithmetic.

Do the following in the Matlab Command Window:

```
» a = 1; b = 2; c = pi;
» a*b, a+b, b/2, c^b
» x = 1 + i
» x^2
```

Many other calculator type commands are defined as functions in Matlab, such as the exponential or sine functions.

Do the following in the Matlab Command Window:

```
» exp
» exp(1)
» sin(pi/3)
» format long, x = pi+i
» format bank, x
» format short, x
```

Basic Vectors

Matlab is built on top of linear algebraic routines, so vectors and matrices are very natural. To create a vector, we use brackets [ ]. For a row vector, we enter the elements with either commas or spaces. For a column vectors, we separate the elements by semicolons.

Type the following:

```
» u=[1,3,5,7]
» v = [1 3 5 7]
» w =[1;3;5;7]
```

In many situations, we can use patterns to create large vectors. Suppose I want a vector that contains all odd numbers from 1 to 99. We type:

```
» u=[1:2:99]
```

This means *make u go from 1 by 2 to 99*. Try:

```
» u=[15:-3:0]
```

To change a row vector to a column vector and vice-versa, we use the transpose operator, given by ’. Try:

```
» v=u’
```

Vector Arithmetic

Vector addition and scalar multiplication work exactly as expected for same sized vectors. Row vector-column vector multiplication and column vector-row vector multiplication works as expected as well.

Try the following:

» *u=[1 3 5 7], v=[0 1 2 4], w=[2:2:10]*
» *2*u+v, u-pi*v*
» *u+w*
» *u*v*
» *u'*v, u*v'*

To access any part or subset of a vector, we use parenthesis indexing. To enlarge a vector, we can create a new vector of vectors!

Try the following:

» *u(1), u(2), u(2:4)*
» *w1=[w(1) w(3) w(4) w(5)]*
» *w1=w([1 3 5 2])*
» *w1=w([3:-1:1 4])*
» *u2= [u 2 3]*
» *u= [u v]*

Advanced Vectors

To find the number of entries of a vector, we use the ***size*** or ***length*** commands. To find a norm, we use the ***norm*** command.

Do each of the following:

```
» u = [1:2:11], v=[2:2:12]
» [m n] = size(u)
» k = length(v)
» norm(u)
» norm(u,1)
» norm(u,inf)
```

Instead of treating vectors as objects, we can treat each part as individual numbers, and operate on them, rather than operating on a whole vector. To indicate we are performing an operation on each part of a vector (or matrix) we put a dot, ., before the operator.

Do each of the following:

```
» u^3
» u.^3
» u*v
» u.*v
```

Matrices (Arrays)

Matrices are formed and manipulated exactly the same as vectors. Recall a space or comma separates elements of a row (puts the next element in a new column), while a semicolon separates elements of a column (a semicolon indicated that a new row starts).

Do each of the following:

» *A = [1 2; 3 4]*
» *A = [1 2; 3 4 5]*
» *A =[1 2 3 4; 5 6 7 8]*

The zeros, ones, and rand command work on matrices as well. To create an identity matrix, we use the command ***eye***.

Do each of the following:

» *zeros(4,2)*
» *ones(2,3)*
» *eye(3,3), eye(4)*

Matrix Operations

Addition of same-sized matrices works as expected. Matrix multiplication, when defined, works as expected as well.

We can manipulate matrix entries just as we manipulated vector entries. Suppose we wish to create:

$$A = \begin{pmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 & 2 \end{pmatrix}$$

We can start with an identity matrix and manipulate it instead of trying to type each entry of the matrix:

*» A=2*eye(5); A(5,1)=1; A(1,5)=1; A*

We can increase the size of a matrix, as well:

» A(6,6)=2;

The colon command is very useful with matrices. It indicates that we are to take every entry in a row or column. Try:

» A(:,[1:3]), A([4 1 2],:)

Solving Linear Systems

Suppose we wish to solve $Ax = b$. Matrix-vector multiplication works as expected, so $x = A^{-1}b$ if found by inverting A and multiplying it by b. Try:

```
» A=2*eye(5); A(5,1)=1; A(1,5)=1; b=ones(5,1)*6;
» x = inv(A)*b
```

A better way to do this is to use the left divide (think of us as dividing each side, on the left, by A). We write

```
» x = A \b
```

This actually does Gaussian Elimination, which is much more efficient or course.

Almost every operation on a matrix is defined as expected, with commands that are generally intuitive. Try each of the following equivalent ways to show a system is solvable:

```
» rank(A)
» det(A)
» rref(A)
» null(A)
» [V D] = eig(A)
```

The last command finds the eigenvalues and vectors of A, returning the eigenvectors in the matrix V, with the eigenvalues on the diagonal of the matrix D.

Common Matrix Functions

Almost every function that acts of a scalar would work on each entry of a matrix. Pick a matrix, input it into Matlab, and use the ***abs***, ***sin***, ***sqrt***, and ***exp*** on it. Recall there is a matrix square root and a matrix exponential, these are defined differently. Try:

```
» A = [-1 2; 2 2]
» expm(A)
» C = sqrtm(A), C*C
```

The matrix transpose is the same command as vector transpose. Try:

```
» A = [1 2 3; 4 5 6]
» A', A'*A, A*A;
```

Some other common functions work a bit differently with matrices than vectors. Figure out what each of the following functions do:

```
» A = [1 -1 0; 2 2 -1]
» min(A), min(A'), min(min(A))
» sum(A), prod(A)
» A<0, A>0, A==0
```

Plotting in 2D

Unless we use special add on packages, there is no such thing as an analytic function in Matlab. This makes plotting a bit difficult. Now, how does any other program plot? It takes points, and links them together. You use the same concept in Matlab, but you need to create the points explicitly.

Suppose I want to plot $f(x) = \sin(x)$ for $0 \leq x \leq 2\pi$. I create a vector for x then plot x versus $\sin(x)$:

```
» x=linspace(0,2*pi);
» plot(x,sin(x));
```

The ***linspace(a,b,N)*** command creates a vector from a to b of size N. Without the N, it uses 100 points.

The most common mistakes in plotting is the lack of . when needed. For example, if we wish to plot $y = x^2 + \frac{1}{x}$ for $1 \leq x \leq 3$ you may try:

```
» x=linspace(1,3);
» plot(x,x^2+1/x);
```

The second command does not work, again, because x^2 is not well defined for vectors. Instead, use:

```
» plot(x,x.^2+1./x);
```

Plotting in 3D

Plotting in 3D is the same exact idea as in 2D. To plot a parameterized space curve, we would set up four vectors. One for the parameter, then one for each space dimension. We use the command ***plot3***:

```
» t=linspace(0,10,200);
» plot3(cos(t).*(exp(cos(t))-2*cos(4*t)),sin(t).*(exp(cos(t))-2*cos(4*t)),t);
```

To plot a 3D surface, the general idea is that I make a matrix of x-coordinates, a matrix of y-coordinates, and a matrix of z-coordinates. To plot $z = x + y$ for $0 \leq x, y \leq 1$ we first set up vectors for the x and y coordinates (we set up the axes). Then take each combination of them to make a matrix, using the ***meshgrid*** command:

```
» x=linspace(0,1); y=x;
» [X Y] = meshgrid(x,y);
»surf(X,Y,X+Y);
»mesh(X,Y,X+Y);
»pcolor(X,Y,X+Y); colorbar;
```

This same idea can be used for parametric surfaces:

```
» u=linspace(-1,1); v=u;
» [U V] = meshgrid(u,v);
» surf(U.*V.*sin(15*V),U.*V.*cos(15*V),V);
```

Advanced Plotting

To plot in specific styles and colors, consult the plot documentation. For example, you can plot in red with dotted lines by doing:

```
» x=linspace(0,1); plot(x,sin(pi*x),'r:');
```

To plot multiple functions, you can simply include them in the same plot command, or use the command ***hold on;*** which indicates you want to hold the current plot (you must use ***hold off;*** to stop this):

```
» x=linspace(0,1); plot(x,sin(pi*x),x,cos(pi*x),'r:');
» x=linspace(0,1); plot(x,sin(pi*x)); hold on;
» plot(x,cos(pi*x),'r:'); plot(x,cos(2*pi*x),'g-.');
```

To annotate plots, there are only a handful of commands we need to use. We use ***xlabel, ylabel, zlabel*** to label the axes, and ***title*** to put a title on a figure. We use the ***legend*** command to label individual plots on one single figure. Try:

```
» x=linspace(0,2*pi); plot(x,sin(x),x,cos(x),'r:');
» xlabel('x'); ylabel('y'); title('sin(x) vs cos(x)'); legend('sin(x)','cos(x)');
```

Creating and Running a Script

The way most serious people interact with Matlab is through functions and scripts. A script is simply a file which contains a series of commands you wish to be executed. Matlab functions and scripts are files saved with a .m extension, often called 'm-files'.

Do the following in Matlab

1. Create a .m file by going to *File -> New -> Blank M-File* (or type ***edit*** in the command window).
2. Save this file (anywhere is fine) as *test.m* .
3. Type the following lines in the script:

   a=4; b=3;

   c=a+b, d=c+sin(b)

   *e=2*d, f=exp(-d)*
4. Resave the file by going to File->Save or typing Ctrl-s.
5. Run the script by (i) Clicking the white box with a green arrow at the top of the editor window, (ii) Hitting F5, (iii) Go to Debug->Run test, or (iv) Typing test at the Matlab Command Window (if you are in the right directory)
6. In the Command Window, type: »who
7. Add *echo on;* at the start of your script and *echo off;* at the end and rerun.

Creating and Using a Function

A function is essentially a script file which allows for input or output. It keeps all of its variables locally.

1. Open a new file in the editor, and start with the following line:
   function myfunction(a)
2. Save this file as *myfunction.m* (you must always save a function as its name). Make the next line of the file:
   b = sin(a) + cos(a)
3. Resave the file and run twice as:
   » clear all;
   » myfunction(0), myfunction(pi/4)
4. Type *who* in the Command Window. Now change the first line of *myfunction.m* to
   function b=myfunction(a)
5. In the Command Window type
   » x=myfunction(pi/4)
6. In the .m file, change *b=myfunction(a)* to *[b1 b2] =myfunction(a1,a2)* and *b=sin(a)+cos(a)* to:
   b1 = sin(a1); b2 = cos(a2);
7. Now run:
   [x1 x2] = myfunction(pi/2,pi/4)

Programming Basics

Matlab is very similar to other modern languages, such as C, but makes many assumptions which C or Fortran do not make. The trick to using Matlab effectively is to use built in vector or matrix operations when available. For now, let us start with the usual first program. Create a .m file containing the following:

```
fprintf('Hello World! \n');
```

Save the file as *HelloWorld.m*, and run. Run by changing the command window to the correct directory and typing:

```
» HelloWorld
```

Before getting much further, let us discuss data types. There is no explicit declaration of data types (in general) in Matlab. Type:

```
» clear all; a=5; whos
```

and note that a is treated as a double. Note also that a is treated as an array, of size 1×1. If we were to use it as an integer, it would be treated as an integer:

```
» i=1:a
```

Programming: Newton's Method

Our goal is to create a program which uses Newton's method to find the roots of a function. Create a .m file, saved as *myprogram.m* containing:

```
function x=myprogram(x0,N)
f = inline('x.^2-2*x-8');
fp = inline('2*x-2');
x(1)=x0;
for n=1:N
    x(n+1) = x(n)-f(x(n))/fp(x(n));
    if(abs(f(x(n+1))) < 1e-6)
        fprintf('Newton"s Method converged in % d iterations \n',n);
        break;
    end
end
```

Run this program as *x=myprogram(0,10)*.

Note that a ***while*** loop works the same way as a do or if, with the addition conditional in parenthesis (*while(condition)*).

Programming: Tips and Tricks

Here are just some miscellaneous guidelines for using Matlab: s

- Avoid multiple loops, if possible. Look for a vector way to do this.
- Vectorize all functions. For example *f=inline('x^2')* and *f=inline('x.^2')* would return the same result for a scalar, but the second function works on vectors as well.
- Always look for a function before programming it. For example, if you need to compute a discrete Fourier transform, while it may be easy to code, Matlab's built in implementation will be MUCH faster than anything you write.
- Matlab has many nice GUI interfaces, such as the optimization tool *optimtool*. Use them for testing purposes, but they will be tedious for repeated operations.

In general, I use Matlab mostly for testing ideas. For something that absolutely needs to run fast and efficiently, I almost always write in C or C++.