A Tour of Matlab for CSCE/Math 441

Thomas Shores
Department of Mathematics
University of Nebraska
Fall 2009

What is Matlab?

Matlab is96, Section 6

1. An interactive system for numerical computation.
2. A programmable system for numerical computation.
3. A front-end for state-of-the-art libraries, currently LAPACK and optimized BLAS.

History:

- Late 70s: Cleve Moler writes Matlab under an NSF grant as a front-end for the current state-of-the-art numerical linear algebra packages LINPACK and BLAS.
- Early 80s: Moler founds the MathWorks and introduces a commercial version of Matlab running on a PC and written in Fortran.
- Early 90s: Matlab is rewritten in C and "handle graphics" is added into Matlab.
- Late 90s: Matlab evolves into an OOP supporting LAPACK and optimized BLAS.

An excellent reference for this material is the superb text (recommended but not required) by Desmond Higman and Nicholas Higman, *Matlab Guide*, SIAM, Philadelphia, 2000.

Note: Except for graphics, you can do most of the Matlab work for this course with Octave, an open source program available under Linux – there is a Windows version, too, but it may require some work to set it up.

A Tutorial Introduction to Matlab

Here we go.... Our procedure in all the lectures will be to work our way through the commands that are listed in the narrative that follows. Separately one should open a Matlab window for command execution. The prompt sign "> " and typewriter text indicate a command to be typed in by the instructor or reader at the Matlab command prompt. Comments will not be preceded by the prompt and will appear in normal text. I will make comments on the commands that are executed as we read through the lecture, and students are invited to ask questions as we proceed. In the body of the Matlab command listing there will be occasionally a "%" sign. This is Matlab's way of starting a comment, and all text until the next end of line is ignored by the Matlab interpreter. It isn't necessary to type in these comments in order to work through this lesson in a Matlab session. They are inserted to give a little more explanation to the meaning of the commands for those who are reading this file and executing the commands themselves.

How to handle a Matlab assignment

Well, do it, of course. The real question is: how do I turn it in? I suggest that you start up Matlab and do the work you need to do. Once you actually knew what you're doing, redo it with an eye to recording your work and turning it in to me. To keep a recording of your work, you issue the following command to Matlab

```
> diary 'myfile'
```

(Caution: make sure your current working directory is a place where you can write files – use the icons above the workspace area to change working directories.) Matlab will then send a copy of all your typed input and the output to a file called 'myfile.' For clarity, use descriptive names for your files, such as 'jsmithasgn1' so that when I save the files that you email me, I can tell what it's about by the title. If at any point you want to stop the diary feature, issue the command

```
> diary off
```

To resume the diary feature, simply type

```
> diary on
```

This will cause input to be appended to myfile. You can make comments in your homework file by typing % at the command line and this too will be recorded. For example

```
> % This is a comment.
```

Be sure to start your file with the comments

```
> % Name:  yourname
> % Email:  your email address
```

When you end your session, the file will be closed and you can view it and even edit it with a text editor. As a matter of fact, you can even edit and view it with the Matlab Editor. Just type

```
> edit myfile
```

When you have finished the assignment, you can sow this text into another document.

Starting and stopping

Start up Matlab and quit after a few calculations and getting some help.

```
> 1+1
> 1+1;
> pi
> % to get help on a command or built-in variable just type:
> help pi % get help on built-in variable pi
> help quit % get help on built-in common
> helpwin % or get help browser style
> quit
```

A calculator

Restart and do the following:

```
> x=0.3
> sin(x)
> exp(x)*sin(x)/log(x)
> 2^24
```

```
> (1+sqrt(5))/2
> x
> format long
> x
> format
> z = 2 -3*i % complex numbers are no problem either
> z^2
> exp(z)
```

Simple plotting

Matlab's plotting, unlike a CAS like Maple, does not plot a function.Rather, it does a dot-to-dot on vectors of x- and y-coordinates. Most Matlab built-in scalar functions know to handle a vector argument by applying the function to each coordinate.

(Correction: used to not plot a function. There is a function called 'ezplot' whose expanded capabilities does just that.)

```
> x = 0:  0.1:  3;
> y = sin(x);
> plot(x,y);
> z = x - x.^2/2 + x.^3/6;
> plot(x,z)
> % close the plot window, then
> plot(x,y)
> hold on
> plot(x,z)
```

Functions and scripts

There are two kinds of text files that can be "run" under Matlab, both of which have the suffix ".m" The first kind is a script file. This file contains a list of commands that one could execute at the command line. It is rather like a program in C or Fortran, but not as capable. For one thing, there is no division of program into a "main" section and various subroutines. You cannot include function definitions in a script file. Functions, that is, value returning subroutines are what function files, the second kind of executable text file, are good for.

Make sure a copy of chebyfcns.m is in your working directory and type

```
> edit chebyfcns
```

This file shows some of the things you can (and can't) do in Matlab. Multiple functions are a no-no, unless some are private, but this file would actually run just fine under Octave.

Numbers

Start Matlab. Matlab is a fine calculator. But as with any calculator, one has to be careful:

```
> ((2/7+1000)-1000)-2/7
```

Multiple commands are possible

```
> x=sin(.3),y=cos(.3);x^2+y^2
```
And of course help is always available, specific or general:
```
>help sqrt
> helpwin
```
There is another handy facility that is a word search function:
```
> lookfor elliptic
```
One always has access to the last output:
```
> exp(-3)
> x=ans;
> disp(x)
```
And one can always load and save variables:
```
> save 'junkfile' x y
> % Now clear all variables:
> clear
> who
> load 'junkfile' x
> who
```

Arithmetic

Now a few words about arithmetic, which conforms to double precision IEEE standard. Here eps is the distance from 1 to the next floating point number:
```
> eps
> 1+eps
> format long
> 1+eps
> format hex
> 1
> 1+eps
> 1+2*eps
> format
```
There are other kinds of numbers. Just for the record:
```
> realmax
> realmin
> realmin/2
> realmax/realmin
> 0/0 % indeterminate...see what Matlab reports
> cos(0)/sin(0)
```

Matrices

This is where Matlab really excels in a way that other computer packages don't. Building and manipulation matrices is quite easy with Matlab:
```
> A = [1 3 2; 2 1 1; 4 0 2] % create a matrix
> size(A)
```

```
> det(A)
> eye(3)
> inv(A)
> inv(A)*A,A*inv(A)
> A^(-1)
> A^2
> A + eye(3)
> A - 2*eye(3)
> (1:3) % create a vector using the colon operator
> b = (1:3)'
> x = A\b % solve system A*x = b
> x = A^(-1)*b % solve it using inverse of A
> (1:3) % create a vector using the colon operator
> v = (1:3)'
> eig(A) % calculate eigenvalues of A
> [V,lam]=eig(A) % eigenvectors and eigenvalues
> A*V(:,2)-lam(2,2)*V(:,2) % check eigen relation
> V^(-1)*A*V-lam % better check of eigen relation
> zeros(2) % create a 2x2 matrix of zeros
> zeros(2,1) % create a 2x1 matrix of zeros
> ones(3)
> ones(3,2)
> eye(3) % create a 3x3 identity matrix
> eye(2,3)
> rand(3)
> A=rand(3)-0.5 % create a 3x3 matrix of random numbers in [-0.5,0.5]
> size(A)
> x=ones(1,5)
> size(x)
> length(x)
```

Of course you can build and reshape matrices from the command line in many ways:

```
> a = [1 2 3
> 4, 5, 7
> 2,4, 6]
> a = [1 2 3; 4, 5, 7; 2,4, 6]
```

One accesses entries or changes them with a standard mathematical notation:

```
> a(1,3)
> a(1,3) = 10
> a(1,3)
```

One can access a vector (row or column) by a single coordinate:

```
> y = x' % the prime sign ' performs (Hermitian) transpose operation
> x(3)
> y(4) = 7
```

One can build matrices in blocks:

```
> b = [eye(3) a; a eye(3)]
```

```
> c = repmat(b,2,3)
```

The all-important colon notation gets used in two different ways. First as a separator in a type of vector constructor:

> x = 1:5
> y = 1:2:5
> z = (1:0.5:5)'

The other principle use if the colon notation is to work as a wild card of sorts. The colon in a position used for a row indicator means

> c = b(:, 3:5)

Matrix manipulations:

```
> triu(a)
> tril(a)
> diag(a)
> diag(diag(a))
> reshape(a,1,9)
```

Matrix constructions (there is a huge number of special matrices that can be constructed by a special command.) A sampling:

```
> toeplitz((1:5),(1:5).^2)
> hilb(6)
> vander((1:6))
```

Multidimensional arrays: Matlab can deal with arrays that are more than two dimensional. Try the following

```
> a = [1 2; 3 4]
> a(:,:,2) = [1 1;2 2]
> a
```

A final note on matrices: one can even use a convenient Array Editor to modify matrices. Do this: if the Workspace window is not already visible, click on the View button, then check Workspace. Now double-click on the variable 'a', a matrix we created earlier. The Array Editor will open up. Edit a few entries and close the Editor. Confirm your changes by typing at the command line:

```
> a
```

Objects

Objects are instances of classes. Matlab has some OOP capabilities, but for now we're going to confine our attention to some fairly simple types of objects. Matlab has five built-in classes of objects, one of which we've already seen:

- double: double-precision floating point numeric matrix or array
- sparse: two-dimensional real (or complex) sparse matrix
- char: character structure
- struct: structure array
- cell: cell array

Various Matlab toolboxes provide additional class definitions.

Of course, we've seen lots of doubles. Here's another very familiar sort of object, namely a string object:

```
s = 'Hello world'
size(s)
s
disp(s)
```

The struct object is what it sounds like, a way to create structures and access them. There are two ways to build a structure: command line assignments or the struct command. What actually gets constructed is a structure array. Try the following

```
> record.name = 'John Doe'
> record.hwkscore = 372
> record.ssn = [111 22 3333]
> record
> record(2).name = 'Mary Doe';
> record(2).hwkscore = 40;
> record(2).ssn = [222 33 4444];
> record
> record(1).hwkscore
```

Finally, a cell object is an array whose elements can be any other object. For example:

```
> A(1,1) = {[1 2; 3 4]};
> A(1,2) = {'John Smith'}
> A(2,1) = {249}
> A(2,2) = {[1;2;3;4]}
> A{1,2}
> A(1,2)
```

Basic Graphics

Start with some simplel plotting. Matlab's plotting, unlike a CAS like Maple, does not plot a function. Rather, it does a dot-to-dot on vectors of x- and y-coordinates. Most Matlab built-in scalar functions know to handle a vector argument by applying the function to each coordinate.

```
> x = 0:  0.1:  3;
> y = sin(x);
> plot(x,y);
> z = x - x.^2/2 + x.^3/6;
> plot(x,z)
> % close the plot window, then
> plot(x,y)
> hold on
> plot(x,z)
```

Next, we'll get help to guide us along. The plot command has many options. We'll hit the highlights. We'll also keep a help window open so we can peruse the help files as we go. After we open it, we'll move it to a corner.

```
helpwin
```
Now click on plot in the Help Window and examine the possibilities for the 1-3 character string S.
```
x = 0:0.01:1; % create array of abscissas and suppress output
y1 = 4*x.*(1- x);
plot(x, y1, 'r+:')
```
A plot with multiple curves is possible without doing a "hold on":
```
y2 = sin(pi*x);
plot(x, y1, 'r+:', x, y2, 'g')
```
We can make it fancier. For example
```
xlabel('x')
ylabel('y')
title('Comparison of 4x(1-x) and sin(\pi x).')
```
As a matter of fact, you can massage your plot window quite a bit with the tools available in the tool bar. For example, click on the arrow icon and then draw an arrow on the inside of the curves pointing to the inner curve. Then click on the A icon (A for ascii) and click at the base of the arrow. Then type in "4*x*(1-x)". There are lots of other things one can do to a graph from the graph window itself.

You can even save your figures to many different formats. Click on the File button, then the menu choice Export. Select under "Save as type:" the choice Portable Document Format. Then browse to the directory you want, edit the name of the file, say to "junk.pdf." Then click on Save. Now use the Windows Explorer to find the file you've created and double click on it. Acrobat will now show you your picture.

There are a number of alternates to the plot command as well. Look in the Help Window. There's even an ezplot which is a Maple rip-off (or is it the other way around?).Try these:
```
ezplot('4*x*(1-x)')
polar(x,y2)
```
Interesting. Let's stretch it out.
```
x = 0:0.01:60;
y2 = sin(pi*x);
polar(x,y2)
```

Axes and other controls

There are other ways to massage your graph. For example, start with
```
ezplot('4*x*(1-x)')
hold on
ezplot('sin(pi*x)')
```
Not very good. Let's fiddle it a bit:
```
axis equal
axis off
axis([0 1 0 1.2])
axis on
```
Well, let's just do it again:
```
x = 0:.01:1;
y1 = 4*x*(1-x);
```

```
y2 = sin(pi*x);
plot(x,y1,'--', x, y2, 'r:')
legend(' 4x(1-x)', 'sin(\pi x)')
title('\it Comparison of 4x(1-x) and sin(\pi x)')
```
What do you notice about the graph?

Multiple plots

Here is the way to construct multiple plots, along with another variation on plot:
```
subplot(2,2,1)
plot(x,y1)
subplot(222)
fplot('sin(x)', [0 1])
subplot(223)
fplot('sin(round(2*pi*x))', [0 1], 'r--')
subplot(224), polar(x,y2)
```

A Bit of Approximation

OK, we're ready to do something serious. First recall the Weierstrauss theorem (or open up the file VectorSpaces.pdf and look at the last section. Let's start by viewing polynomial approximations to the function $f(x) = \sin x$, $-\pi \leq x \leq \pi$. In Matlab, we represent a polynomial by a vector of coefficients in descending order. Thus $x^3 - 2x + 3$ would be represented as $[1, 0, -2, 3]$. Here we go:
```
x = (-pi:0.1:pi)';
y = sin(x);
plot(x,y)
grid, hold on
tpoly = [1/120,0,-1/6,0,1,0] % start with a 5th degree Taylor about 0
ty = polyval(tpoly,x);
plot(x,ty);% not bad, sort of
n = 3;
nodes = linspace(-pi,pi,n+1)';
fnodes = sin(nodes);
ipoly = vander(nodes)\fnodes;
iy = polyval(ipoly,x);
plot(x,iy);% what do you think?
% play fair and do this again with n = 5
% now start over with the function f(x) = 1/(1+x^2) on the interval [-5,5]
% use y = 1./(1+x.^2); in place of y = sin(x);
```