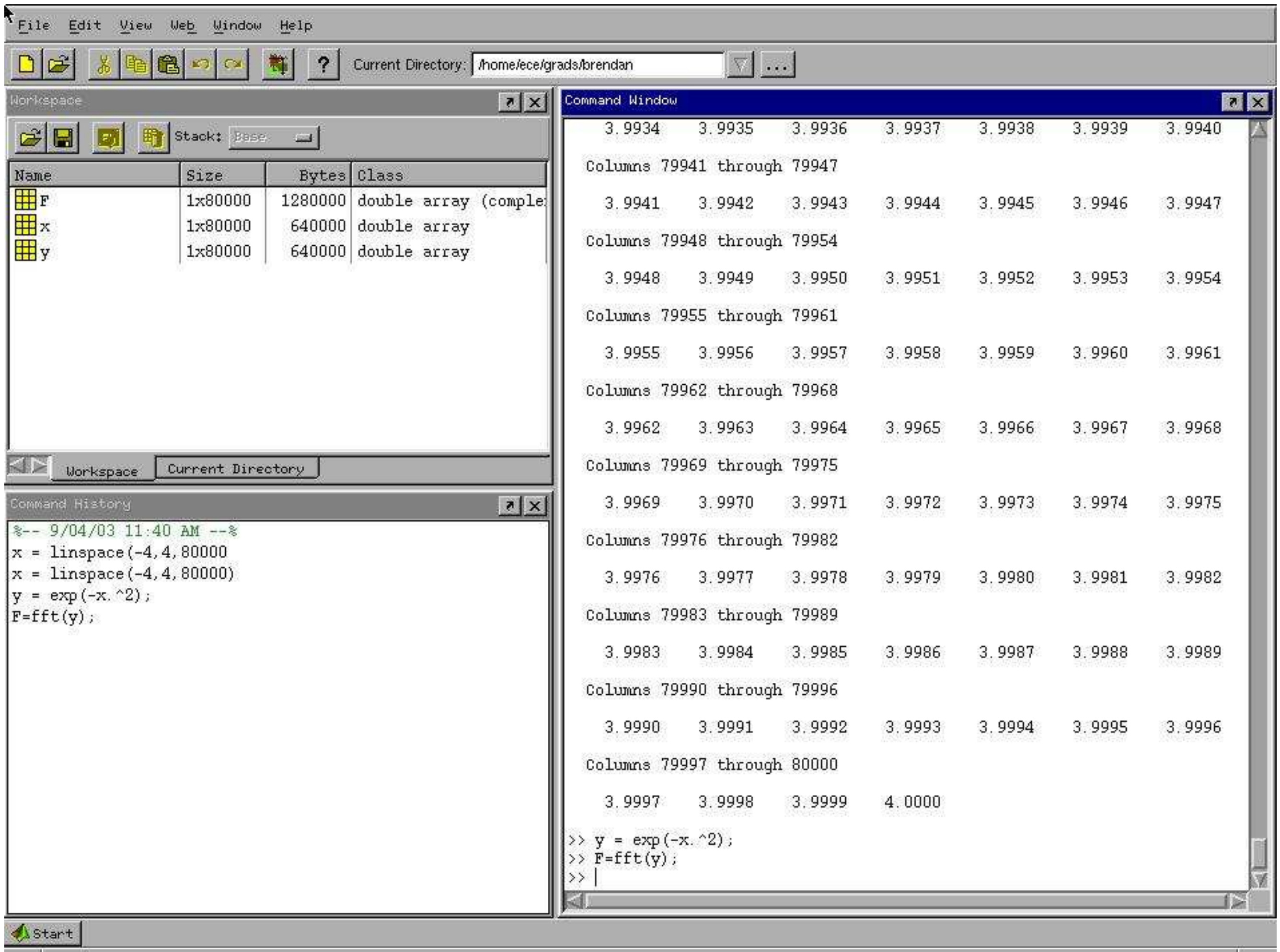
Matlab Tutorial

Basic Screen



You'll notice the three main windows are "Workspace", "Command History", and "Command Window".

The Workspace window is a list of everything used currently in your workspace. In this case it contains three matrices or arrays of size 1 X 80000. You'll learn more about what can appear here later.

The Command History window shows the list of commands that you've executed already.

The Command Window window is probably the one that you'll spend the most time looking at (which is why it's the biggest). It shows results, commands, errors, etc... Basically, everything you'd want to know about what's currently going on.

The next part of the tutorial comes directly from the script file above. Please use that if available to follow along, this is simply a more printable version of the tutorial.

Assignments
Here are a few basic assignment statements think of an assignment as a variable.

```
a = 25;
b = 42*12;
c = 50
d = 36*12 + 120
```

Note that when the semicolon is present that Matlab doesn't display the results on the screen, otherwise is does display the results after the command is executed.

Important commands
clear command
The *clear* command is very important, it simply clears a variable from the workspace.

```
clear a;
clear b
clear c
clear all;
```

"clear all" clears EVERYTHING from the Matlab Workspace. It is recommended that you always put "clear all" as the first line of your M-file so that you don't have anything in your workspace that may cause problems with your script.

help command
The help command is another critical command to know. If you don't know what a particular command does you can always type "help <command>" and it will display what the command does and how to use it.

```
help clear;
help help;
```

lookfor command
The lookfor command is for that time when you don't know what the name of the command you're looking for is. Type "lookfor <keyword>" and it will search the first line of the comments for each command and return matches. Note, it sometime can take awhile.

```
lookfor fourier;
```

Vectors
Vectors form the basis of everything in Matlab. Becoming familiar with this concept is a must when working with Matlab. Think of a vector as a 1 X N matrix. Since Matlab is a numerical solver it requires that input to Matlab be completely numerical (not algebraic).

```
a = [1 2 3]
b = [4 5 6]

a
b

a*b
```

This command returned an error didn’t it? This is because Matlab tried to multiply a 1 X 3 vector times a 1 X 3 vector. Since vectors are really just 1 X N matrices this obviously creates a problem. You need to first decide what you desire from the result. Do you want the two to be multiplied like two matrices or do you want the corresponding elements of each matrix multiplied individually?

The tick mark ’ (same key as double quote) is used to imply the transpose of a matrix. Note, the transpose is different from the inverse which is the *inv*(<variable>) command. So if you desired to multiply a*b like two matrices then you need to transpose the second (or first) matrix.

```
a*b’
a’*b
b’*a
b*a’
```

remember matrix multiplication is noncommutative so var1*var2 is not necessarily equal to var2*var1.

If instead you wanted to have each corresponding element of a and b multiplied to form another 1 X 3 matrix (in this case) then you need to use Matlabs "dot notation". The dot notation assumes that you are running the command element by element.

```
a.*b
a./b
```

Notice how this command multiplied (or divided) 1*4, then 2*5, then 3*6? Often times you will forget to put the period in your equations, but it will usually give you an error (but not always!).

Polynomials
Polynomials in Matlab are represented in the same manner as vectors (a 1 X N matrix). For instance if we used a from above to form a polynomial meaning:

f = a(1)*x^2 + a(2)*x + a(3) Notice the use of the notation a(<num>), this allows you to return a value of vector with location <num>, also Matlab indices ALWAYS start with 1. All we need to do is use a as we've already defined it:

```
a
```

To find the value of a at a certain "x" we need to use the *polyval* command.

```
polyval(a,6)
```

To find the roots of the equation we can use the *roots* command.

```
roots(a)
```

In this case the roots are imaginary (1 +- sqrt(2)*j), again notice the use of the sqrt command, it takes the square-root of it's argument. Also note that both i and j are pre-defined as sqrt(-1).

Matrices
Matrices are handled in much the same way as a vector is handled, really they are the same thing if you think of a vector as a 1 X N matrix.

Matrix definition:
```
a = [5 9 2; 2 13 9; 12 8 20]
b = [10; 20; 30]
```

Notice how the semicolon separates rows of a matrix. The same theory as above regarding their multiplication follows here. This time a*b works (a is a 3 X 3 matrix and b is a 3 X 1 matrix).

```
a*b
```

In this case the commands:

```
a.*b
b.*a
```

yield errors. This is because there is not the same number of elements to do an element by element multiplication. You still can transpose a and multiply it by b.

```
a*b
```

But not the other way around, *why*?

```
a*b'    %error
```

Systems of equations can also be solved by Matlab using matrices. Remember back to using matrices to solve systems of equations, in this case we can solve this system by noting:

[a]*[x] = [b] => so [x] = inv([a])*[b] where:

$$
[a] = \begin{matrix} a11 & a12 & a13 & \cdots & a1n \\ a21 & a22 & a23 & \cdots & a2n \\ a31 & a32 & a33 & \cdots & a3n \\ . & . & . & & . \\ . & . & . & & . \\ an1 & an2 & an3 & \cdots & ann \end{matrix} \quad [x] = \begin{matrix} x1 \\ x2 \\ x3 \\ . \\ . \\ xn \end{matrix} \quad [b] = \begin{matrix} b1 \\ b2 \\ b3 \\ . \\ . \\ bn \end{matrix}
$$

```
x = inv(a)*b
```

This brings up the special functions available by Matlab for use with matrices. The *zeros* command generates a matrix of zeros (N X M) The *ones* command generates a matrix of ones (N X M) The *eye* command generates an identity matrix (which can be non-square)

Colon notation

The colon in Matlab has a special significance. It is used to specify a sequence of numbers.

```
a = 1:5

a
```

The above command returns *a* which is the vector [1 2 3 4 5]. This can be useful if you want to show the value of a function at many points.

```
clear a,b
```

```
a = [1 3 2 5]
b = 1:10
polyval(a,b)
```

In the above example b would increment by 1 each time, instead you may want it to increase by .01.

```
t = 0:pi/20:2*pi
```

Matlab returned a lot of information didn't it? Again, remember the semicolon will suppress that if you don't want to see it.

```
t = 0:pi/20:2*pi;
```

Plotting

This is useful if you want a pretty accurate continuous-time graph

```
x = sin(t);
y = cos(t);

figure(1);
plot(x);
figure(2);
plot(t,y);
figure(3);
plot(x,y);
```

Above two useful commands were used, *figure* and *plot*. The *plot* command opens a new windows (called a figure) and displays the data points you supplied. Note, it assumes you wish to connect the points together. Also, the *plot* command can take either one or two arguments. For one argument it assumes you want the data plotted against it's index (1,2,3....N). If you supply two arguments then the data is plotted one against the other. You can see this by looking at figures one and two and comparing their x-axis.

The *figure* command allows you to switch windows to plot to. In this example if you were to

```
plot(x);
```

you'll notice your window that has a circle now changes to display a sine wave, hence replacing the circle drawn previously.

Let’s say you wanted to plot sine and cosine on the same windows. There are two ways to accomplish this:

```
figure(1);
plot(t,x,t,y);

   or

figure(2);
plot(t,x);
hold on;
plot(t,y);
hold off;
```

The only difference is a color issue, but you can fix that as well by doing the following. Please type help plot for more information.

```
figure(2);
plot(t,x);
hold on;
plot(t,y,’g’);
hold off;
```

The hold command "holds" the contents of the current figure, while you plot more information on top of it. Simply typing "hold" changes the state of hold (from on to off) but I prefer to be explicit about it so there are no mistakes.

If you wish to use infomation in discrete-time using the **stem** command can be useful. It is very similar to plot except it doesn’t connect the dots.

```
figure(1);
stem(x);
```

3D Graphing

This is one of the most impressive features of Matlab (in my opinion). *meshgrid* is a cool command used to transform vectors into 3 dimensions. Don’t worry how this works now, it’s really a lot like making *t* above suitable for 3D.

```
[x,y] = meshgrid(-2:.1:2, -2:.1:2);
z = x.*exp(-x.^2 - y.^2);
mesh(z);
```

The [x,y] above is important as in this case *meshgrid* returns two vectors (notice the comma in the command). This means the first vector returned by *meshgrid* gets assigned to x and the second to y. If you simply put:

```
x = meshgrid(-2:.1:2, -2:.1:2);
```

Only the first item returned by *meshgrid* would be assigned, which isn't going to work for 3D. Notice that Matlab doesn't return an error it simply ignores the second vector returned. Sometimes you can ignore items returned from commands (or functions), but this often means you don't know how the command is supposed to work. To be safe type *help* <command> to ensure you are using it properly.

Next the above sequence used the *exp* command, which you might be familiar with from your calculator, it's simply the e^ function.

Finally we used the *mesh* command to display the information in a window (quite a fancy graph ehh?). It's much like the plot command for three dimensions.

Now for a little more advanced topics.

Fourier Transform

fft command

The fft command actually returns the Discrete Fourier Transform of the given vector.

```
clear all;
```

Start by creating the Gaussian function in the interval from -1 to 1 sampled every 0.0001 units.

```
x=-4:0.0001:4;
```

or alternatively

```
x=linspace(-4,4, 80000);
```

Define a function

```
y=exp(-x.^2);
```

Check your results by plotting y(x)

```
figure(1);
plot(x,y);
axis([-4,4,0.0,1.0]);
xlabel('x-coordinate');
ylabel('y(x)');
title('y=exp(-x^2)');

F=fft(y);
F=F/max(F);
```

Now let's try to plot this to see what we get:

```
figure(2);
plot(F);
```

That plot doesn't look right, does it? The problem here is that the Fourier Transform produces a complex function of the spatial frequency. Thus the vector F contains complex numbers instead of just real numbers (to see the contents of the vector F, just type F). When you plot a complex vector in MATLAB, it tries to plot the real part against the imaginary part, which is not what we want to do. Instead, what we should do is use the abs function (or real function to get the real part of F) to plot the magnitude of the Fourier Transform |F(y)| as follows:

```
figure(3);
plot(abs(F));
```

That looks better, but it’s still not quite what we would expect. We see a function that’s been split down the middle and then the left and right halves of the plot swapped. You see, when MATLAB computes the FFT, it actually outputs the positive frequency components first and then outputs the negative frequency components. We can correct this by using the *fftshift* function, which will put the negative frequencies before the positive frequencies. To do this type:

```
figure(4);
plot(fftshift(abs(F)));
axis([39980,40020,0.0,1.1]);
```

Voila! We just showed that the Fourier transform of a gaussian is a gaussian function. Let’s not forget to label the axes and give the plot a title

```
xlabel( ’Spatial frequency’);
ylabel(’|F|’);
title(’Fourier Transform of a Gaussian’);
```

You can also use the *ifft* command in much the same way to perform the Inverse Discrete Fourier Transform.

Statistical example

Let’s try using another command provided by Matlab called the *randn* command. It returns gaussian random numbers (*rand* returns uniformly distributed random numbers).

```
clear all;
x = randn(1,1000);
hist(x,100);

figure(2);
t=-4:.1:4;
y = exp(-t.^2);
plot(t,y);
```

Both graphs are pretty similar. Try increasing the number of random numbers inside of *randn* to 10000, 100000, 1000000. This *hist* command is similar to plot except it prints a histogram using N bins. In this case the number of bins is 100. You can try changing the number of bins to something smaller or larger if you like.

Programming

Possibly one of the most useful parts of Matlab is that it allows for many C-like functions. This means that in fact you can write a "program" in Matlab.

if, elseif, else structure

The if statement works much like you'd expect. It follows standard logical operations. Valid logical operations are: == (equality), < (less than), > (greater than), <= (less than or equal), >= (greater than or equal), or ~= (not equal).

```
clear all;
x = randn(1,1);
y = randn(1,1);

   if x > 0
      z = 0;
   elseif y > 0
      z = 1;
   else
      z = 2;
   end
```

The above example uses two random numbers x and y to determine the outcome of the if block. z will equal zero if x > 0, z will equal 1 if x < 0 and y > 0, otherwise z will equal 2.

for structure

Again, the for structure works in much the same way that you'd expect.

```
clear all;
num_iter = 20;
   for i = 1:num_iter
      if i == 1;
         x(i) = 0;
      elseif i == 2;
         x(i) = 1;
      else
         x(i) = x(i-1) + x(i-2);
      end
   end

x
```

The above section uses the for loop to return a Fibonacci sequence.

while structure
The while loop also works the same way that you'd expect it to.

```
clear all;
count = 0;
   while count < 10
      count = count + 1;
   end

count
```

Remember, many things can be done using Matlabs assignments instead of using explicit if,for,while loops. If this is an option in your case your code will run much quicker with the built-in Matlab functions.

Example:

```
a = 1:10;
b = 1:10;

   for i = 1:10
      c(i) = a(i) * b(i);
   end

c
```

Which can obviously be computed in the following ways:

```
clear c;
c = (1:10).^2;

clear c;
c = a.*b;
```

Always be on the lookout for ways to do things like the above example when you find yourself using loops.

Other References

http://www.glue.umd.edu/~nsw/ench250/matlab.htm (one of my favorites)

http://www-ccs.ucsd.edu/matlab/ (Matlab Help Desk)

http://www.math.utah.edu/lab/ms/matlab/matlab.html

http://www.math.ufl.edu/help/matlab-tutorial/matlab-tutorial.html

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

http://www.mines.utah.edu/gg_computer_seminar/matlab/matlab.html

http://users.ece.gatech.edu/~bonnie/book/TUTORIAL/tutorial.html

http://www.owlnet.rice.edu/~ceng303/Matlab/MatCont.html