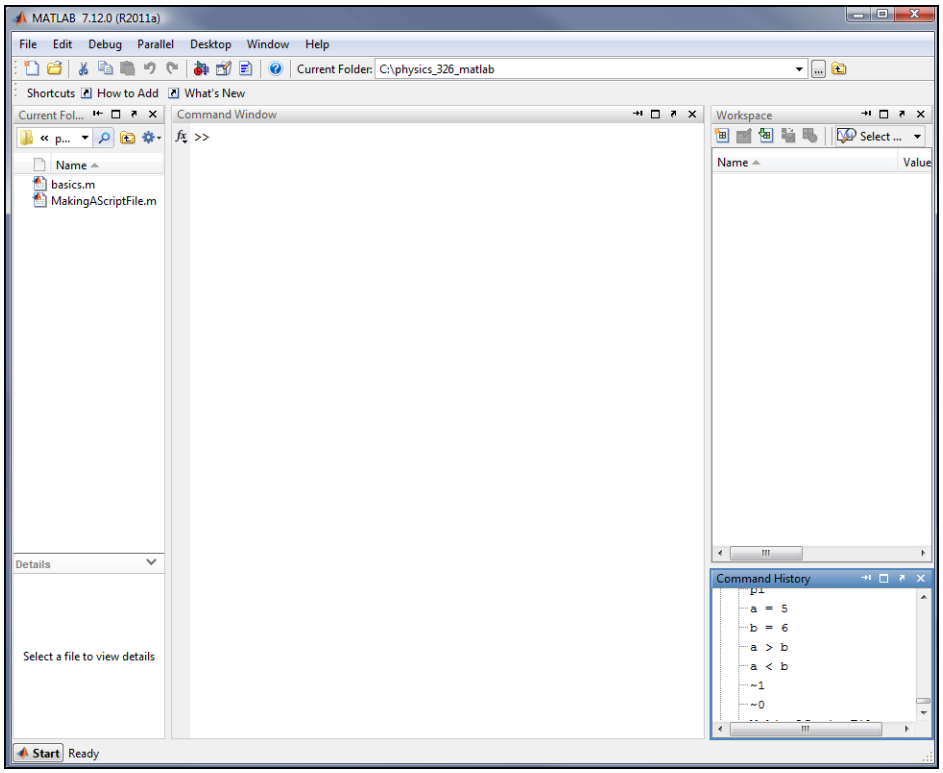
A Matlab Primer

Here is how the Matlab workspace looks on my laptop, which is running Windows Vista. Note the presence of the Command Window in the center of the display.



You'll want to create a folder in which you'll place files you create; mine is C:\physics_326_matlab. Notice the presence of a couple of script files in the "current folder" pane to the left: basics.m and MakingAScriptFile.m.

Keep in mind that Matlab is case sensitive: `pi` is not the same thing as `Pi`.

See the file basics.m, which contains much of the following.

Matlab processes scripts (as well as things you say directly to Matlab by typing something at the ">>" prompt) by reading what it is told, then deciding what answer to return. It is not a compiled language: the software does not generate an executable program which can be run outside the Matlab environment.

For example, try typing this to see what happens (don't type the >> prompt, of course):

```
>> a = 5
```

If you had put a semicolon after the 5, Matlab wouldn't have reported an answer after evaluating the line, e.g.

```
>> a = 5;
```

Matlab remembers the value of a: try typing

```
>> a
```

and it'll tell you that `a` is 5.

You can continue a line onto the next by using an ellipsis (three successive periods). For example, to define a 3 x 6 matrix you could tell Matlab this:

```
>> a = [10 20 30 50 60; 11 21 31 51 61; 12 22 32 52 62];
```

though your code would be easier to read when written like this:

```
>> a = [10 20 30 50 60; 11 21 31 51 61; …
12 22 32 52 62];
```

The above matrix is displayed by Matlab this way:

```
>> a
a =
      10    20    30    50    60
      11    21    31    51    61
      12    22    32    52    62
```

To access the 3rd row, 2nd column of this matrix, refer to it as, for example, `a(3,2)`:

```
>> a(3,2)
ans =
      22
```

Here are some basic operations you'll use in writing matlab code.

1. incrementing the value a variable:

```
>> k = 5;
>> k = k + 1
k =
      6
```

2. creating an equally-spaced sequence of numbers from 1 to 4, with each being 0.7 larger than the preceding:

```
>> 1 : 0.7 : 4
ans =
      1.0000    1.7000    2.4000    3.1000    3.8000
```

You can create a vector that is 1 row by 5 columns this way:

```
>> b = 1 : 0.7 : 4
b =
      1.0000    1.7000    2.4000    3.1000    3.8000
```

3. Asking for the dimensions of an object:

```
>> size(b)
ans =
      1     5
```

4. Taking the transpose of a matrix: use an apostrophe (a single quote) to take the transpose.

```
>> c = b'
c =
      1.0000
      1.7000
      2.4000
      3.1000
      3.8000
```

5. Multiplying matrices:

```
>> b*c
ans =
      33.7000

>> c*b
ans =
      1.0000    1.7000    2.4000    3.1000    3.8000
      1.7000    2.8900    4.0800    5.2700    6.4600
      2.4000    4.0800    5.7600    7.4400    9.1200
      3.1000    5.2700    7.4400    9.6100   11.7800
      3.8000    6.4600    9.1200   11.7800   14.4400
```

6. Displaying a text string in the command window: note the use of single quotes to delimit a text string.

```
>> disp('and that is that!')
```

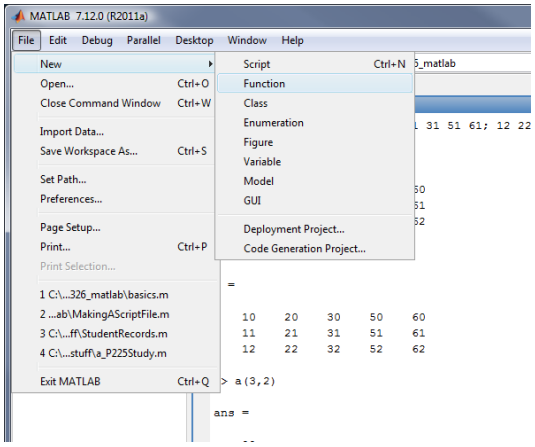
Use two successive apostrophes if you'd like to insert an apostrophe into a text string:

```
>> disp('and that''s that!')
```

No semicolon is required after the disp() command since it doesn't return a value.

See the file MakingAScriptFile.m, which contains much of the following.

To create a new script ("m") file, go to the File → New → Script menu. An untitled script file will open in the editor.



Here are some of the things you might want to do in a script file.

1. Conditional statements.
The format in the following should be clear to you, I think.

```
a = 5;
b = 6;

if a > b
    disp('a is greater than b')
end

if a >= b
    disp('a is greater than or equal to b')
end

if a < b
    disp('a is less than b')
end

if a <= b
    disp('a is less than or equal to b')
end

% note the use of a double equal sign here:
if a == b
    disp('a equals b')
end
```

We can construct more complicated conditions using the logical operators && ("and"), || ("or"), and ~ ("not"). Note that "true" is `1` and "false" is `0`. For example,

```
if (a < b) || (a > b)
    disp('a doesn''t equal b')
end

if ~(a == b)
    disp('a doesn''t equal b')
end

if a == 1
    disp('a is 1')
else
    disp('a is not 1')
end

if a == 1
    disp('a is 1')
elseif a == 2
    disp('a is 2')
elseif a == 3
    disp('a is 3')
else
    disp('I give up.')
end
```

2. Loops.
There are lots of ways to execute loops. Here's an example. I'm going to do something clever inside the loop to encode three numerical values into a string which I'll explain shortly.

```
% I will step in angle by 0.5 radians. Note that Matlab knows about pi.

dtheta = 0.5;

for theta = 0 : dtheta : 2*pi

      SinTheta = sin(theta);
      CosTheta = cos(theta);

      message = sprintf('th = %f, sin(th) = %f, and cos(th) = %f', ...
        theta, SinTheta, CosTheta);
      disp(message)

      % I would have used sind(theta) and cosd(theta) if theta had
      % been in degrees instead of radians.

end
```

The `sprintf` function creates a text string, inserting formatted representations of quantities as desired. The three `%f` that appear in the string are format specifiers representing a "floating point" number such as 3.14159. (You would use a `%d` for an integer when you don't want there to be a decimal point in the text representation of the number.)

Use the functions `sind` and `cosd` when the argument is in degrees.

You can use `break` to jump out of a loop early:

```
for j = 1:20

      message = sprintf('j is %d', j);
      disp(message)

      if j > 5
            disp('time to leave this loop')
            break
      end

end
```

3. Drawing graphs.
One of many ways to plot a curve is to give Matlab a list of x and y values. Here's an example using a loop.

```
% create a vector of angles, and reserve space for a y array.
theta = 0 : 0.01 : 2*pi;
y1 = zeros(1,length(theta));

% fill the vector holding the sines of the angles using a loop
for index = 1 : length(theta)
    y1(index) = sin(theta(index));
end
```

```
% draw the graph
plot(theta, y1);
```

I could have done this more compactly, without a loop, by taking advantage of Matlab's all-at-once matrix manipulation features:

```
theta = 0 : 0.01 : 2*pi;
y2 = sin(theta);
plot(theta, y2);
```

or even just

```
plot(0 : 0.01 : 2*pi, sin(0 : 0.01 : 2*pi));
```

Here is a loop which plots a "Limaçon of Pascal," $r(\theta) = b + a\cos(\theta)$, for $a = 2$ and $b = 1$.

```
% set the values of a and b:
a = 2;
b = 1;

% generate a list of angles theta:
theta = 0 : 0.01 : 2*pi;

% ask how many elements are in theta:
thetalength = length(theta);

% create arrays of x and y points, initialized to zero. The function
% zeros(n,m) creates an n row by m column object. We want 1 row,
% lots of columns.
x = zeros(1, thetalength);
y = zeros(1, thetalength);

% Now loop over angles to generate r, then x and y.

for index = 1 : thetalength

    r = b + a * cos(theta(index));
    x(index) = r * cos(theta(index));
    y(index) = r * sin(theta(index));

end

% Now open a new figure window, plot the figure, then add a title.
figure;
plot(x,y);
title ('Limaçon of Pascal');

% now set the axes to be drawn with equal lengths per unit of distance.
axis equal
```

I could have done this much more compactly. The tricky thing in the following is the use of the "`.*`" operator to take a dot product (a term-by-term multiplication) of two vectors. Here it is:

```
plot( (b + a*cos(0 : 0.01 : 2*pi)) .* cos(0 : 0.01 : 2*pi), ...
    (b + a*cos(0 : 0.01 : 2*pi)) .* sin(0 : 0.01 : 2*pi) );
axis equal
title ('Limaçon of Pascal');
```

See the file DebuggingCode.m, which is listed below.

```
% this file is DebuggingCode.m From the editor windw, hit F5 to execute it.

% I will set some breakpoints using commands in this script, but you will
% (almost) always set them by clicking on the dash next to the line number.

% first clear any breakpoints left over from a previous run.
dbclear all

% now set new breakpoints.
dbstop in DebuggingCode at 22
dbstop in DebuggingCode at 33
dbstop in DebuggingCode at 47

% let's clear all the existing stored variables for the purposes of
% learning about the debugger.
clear all

% set the values of a and b. F10 will single-step through the program.
a = 2;      % try holding the mouse over a and b when the program pauses.
b = 1;

% generate a list of angles theta, going from 0 to 2pi in steps of 0.01
% radians. Note that theta is one row, lots of columns.
theta = 0 : 0.01 : 2*pi;

% ask how many elements are in theta:
thetalength = length(theta);

% create arrays of x and y points, initialized to zero. The function
% zeros(n,m) creates an n row by m column object. We want 1 row, lots of columns.
x = zeros(1, thetalength);
y = zeros(1, thetalength);

% Now loop over angles to generate r, then x and y.

for index = 1 : thetalength

    r = b + a * cos(theta(index));
    x(index) = r * cos(theta(index));
    y(index) = r * sin(theta(index));

end

% Now open a new figure window, plot the figure, then add a title.
figure;
plot(x,y);
title ('Limaçon of Pascal');

% now set the axes to be drawn with equal lengths per unit of distance.
axis equal

```