Lab 5: Matlab Tutorial

Due Sunday, May 8 at midnight

For this final lab, you should work with a partner. You know how to do that at this point. Only one partner turns in the lab, but both of your names must be on the lab.

Matlab Introduction

To use Matlab:

You have 2 options on how to use Matlab:

1. Go to a campus lab where it is already installed and setup. This includes the Pearson lab and the lab in the basement of Smith Hall as well as Spencer Lab. This is the cheapest and probably the easiest option.

2. Buy a student / academic version of Matlab to install on your own computer (or better yet, get your partner to buy Matlab). Last check, the basic setup costs $99 and is good for your entire undergraduate years. http://www.mathworks.com/academia/student_version/faq/prodinfo.html for information.

Matlab Info:

1. Watch the matlab video, Getting Started with Matlab, at http://www.mathworks.com/help/matlab/getting-started-with-matlab.html

If you need more detail on how matlab works, you can look at www.mathworks.com/help/techdoc/learn_matlab/bqr_2pl.html

Specifically, you might want to look at:

- 1-7 to 1-9, Starting and Quitting the MATLAB Program
- 2-2 to 2-20, Matrices and Arrays
- 3-56 to 3-70, Plotting
- 4-2 to 4-5, Programming -- Flow Control
- 4-20 to 4-23, Programming -- Scripts and Functions

The linked PDF is also available in web form at www.mathworks.com/help/techdoc/learn_matlab/bqr_2pl.html

Instructions

Step 1: Start up both MATLAB and a web browser (Firefox)

Step 2**:** Type some expressions into MATLAB and see their effect

The following exercises are designed to get you comfortable with the MATLAB environment. For now, do all your work in the Command Window part of the MATLAB environment.

Try

```
>> 2^4
```

The python equivalent is 2**4. You should get the following result:

```
 ans = 16
```

Try

```
>> area = pi * 4^2
```

Python has pi built into is math library. In Matlab you don't need to include the math library to use pi. Try

```
>> x = sin(pi/2)
```

Again, Python has both sin and log built into a math library, whereas Matlab automatically includes these. Try

```
>> y = log(2)
```

Take a minute and play with the command window.

Step 3: Write a MATLAB function into a file (a so-called "M-file" file)

As in python, you can work in a separate editor window and save your python code in a file, ending with a .m for matlab:

- `myLab1.m`
- `calculateAnswer.m`
- etc...

To create a new .m file, choose **"File->New->Blank M-file".** This will open an Editor window where you can begin to type commands. Be patient—it may take several seconds for the new window to open!

Big Note Here: Matlab is annoying in that all functions must go in their own file. Unlike Python, in which we are able to place multiple functions in the same file, Matlab doesn't like that. Each function goes in its own separate file, named with the function's name. The idea is that every function should be usable by all other functions.

Comments:

Comments in Matlab are signified by putting % at the beginning of the line (as opposed to python, which uses #). The header comments in your file should follow the commenting style we used in class.

Example:

```
% Description: calculates the area of a circle given the radius by
%              multiplying the radius squared and pi
% Input: a number (int or double) (for the radius)
% output: a double  (representing the area of a circle)
% Examples: circleArea(5) = 78.5

function outputValue = circleArea(radius)
  outputValue = pi * radius^2
```

The equivalent in Python is:

```
# Description: calculates the area of a circle given the radius by
#              multiplying the radius squared and pi
# Input: a number (int or double) (for the radius)
# output: a double  (representing the area of a circle)
# Examples: circleArea(5) = 78.5

def circleArea(radius):
  return( pi * radius**2)
```

The big difference between how python defines functions and matlab defines functions (other than calling it function verus def) is how matlab returns values. What matlab does is creates something that is going to hold what is returned from the function. Right in the very top line of the function definition, it tells you what is going to hold the value returned from the function: we say outputValue = circleArea(radius). That means that the value in outputValue is the value being returned from the matlab function circleArea. Then inside the function, we give

outputValue a value (a number, a string, a Boolean, a list, or some other value) and that is what is returned from the function.

> ***Note:*** ***DO NOT*** *copy and paste this text. Type it into the MATLAB editor. MS Word places special characters into its text that mess with MATLAB's brain.*

When you are finished, use the Save As command from the File menu to save the file with the name `circleArea.m`

Step 4: Test your function in the command window

In the command window (the shell, not the matlab file circleArea.m), type in the following:

```
>>circleArea(5)
```

Note: You should get 78.5398 as the output. If you didn't, check your code.

Step 5: Create a new file named **volume.m**. Inside volume.m, write a matlab function that takes 3 input parameters, the width, the height, and the depth. The function should return the volume of the rectangle. Make sure the function is commented appropriately. Save the file, then test it by running volume(3,7,2) (and other tests) in the shell.

Save this to turn in later

Step 6: Summing using loops:

Matlab has loops, just like python. It has recursion, while loops and for loops. The recursion and while loops are quite similar to what you've seen in python. However, the for loops use a bit of a different syntax (although they accomplish pretty much the same thing.)

An example of a for loop (in a function) would be:

```
% Description: This function calculates the sum of a series
%     of integers from 1 to finish and returns that sum.
% Input: last integer in series of ints to be summed
% Output: sum of integers from 1 to finish
% Examples:
%     sumInts(5) -> 15
%     sumInts(10) -> 55
%     sumInts(12) -> 78

function total = sumInts(finish)
     total = 0
     for x = 1:finish
           total = total + x
     end
```

In this case, the for loop starts by initializing the variable *x* to hold 1. x will then be set to hold every value up until the value finish . So if the value in the variable *finish* was 4, current would first hold 1, then 2, 3, and finally 4. It stops when it is equal to the value inside of *finish*. ***In other words, unlike Python, in matlab for loops the value starts at the first value and continues THROUGH the last value.***

We signify the end of the loop with the word ***end*****.**

for loops are flexible. If nothing is specified, the variable automatically increases by 1 each time through the loop. However, you can change the amount we increase the loop by. For instance,

```
function printbyfives(endnum)
        for count = 1:5:endnum
                disp (count)
```

```
end
```

This should count by fives. (Note that the increment value is in the **center**, unlike python, for which you'd say,

```
for count in range(1,endnum+1,5):
```

In the above function, disp is used instead of print. Disp will print out whatever is between the parentheses.

One other useful tidbit of information: you can use the mod function to get the remainder. So, for instance,

```
>>mod(13,5)

    ans = 3

>>mod(13,2)

    ans = 1
```

mod(13,2) is the same as 13%2 in python.

Step 7: Open a new file in matlab, and save it as sumOddInts.m Then, inside this file, write the functions sumOddInts.

sumOddInts will use a for loop to sum all the odd integers between two parameters called "start" and "finish" and return that value, e.g. sumOddInts(5,9) returns 21

Save this. Test it by running it from the shell.

Step 9: Creating an array (aka in Python as a list)

Create two arrays (vectors) of numbers (note the lack of commas between elements in your array, which is different from python):

```
>> ls=[8 4 2 9 0 7 12 3 22 11 1 5];
```

Note: you don't need the commas in lists (vectors) in Matlab.

Side not: the semicolon: Python doesn't care whether you put the ; at the end of each statement or not. It is a way to demarcate the end of a statement, but Python also considers a new line to indicate the end of a statement.

Matlab is a bit different. If you don't put a semicolon after a statement, Matlab with echo the variable in the shell (in other words, with the above list, if I didn't put the semicolon at the end of the line, Matlab would print out the list in the shell. Whereas if I do put the semicolon, Matlab doesn't echo the line in the shell.

Step 10: Create a new file in matlab, and save it as findsmallest.m

Write a matlab function that takes as input parameters an unsorted array (a list) and an integer. The function should then go through the array (list) and find the smallest value in the list. It should return the index of the smallest value in the list.

Note: the length command may be useful here. The command **length** returns the number of components of a vector

```
>>length (ls)
ans =
12
```

Save this and test it by running it in the shell

Step 11: Create a new file in matlab, and save it as selectionsort.m

Write a function that takes as input parameter an array (a list) of unsorted integers. The function should return the array in its sorted state, and it should use selectionsort to sort the array.

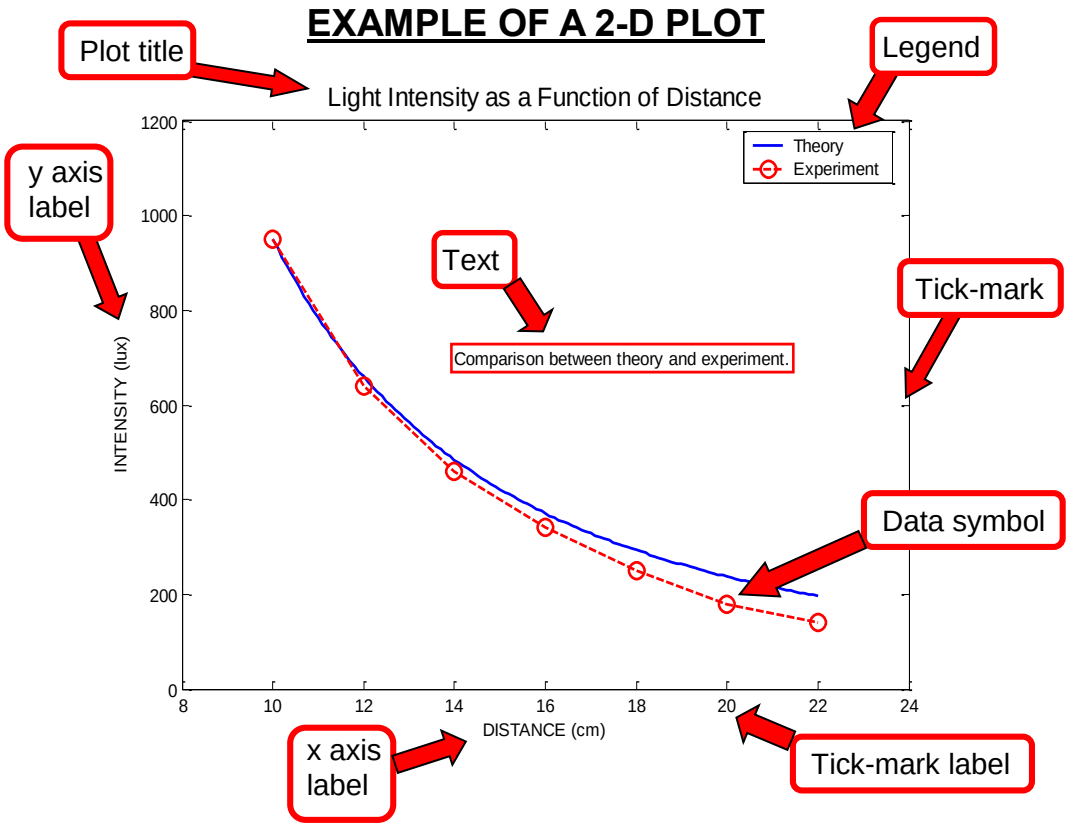
Note: you can copy and paste the code from the findsmallest function above, and then modify that code for your selectionsort.

Note 2: If you don't know the equivalent matlab syntax for something, look at the matlab cheet sheat.

Save this and test it by running it in the shell

Matlab Part 2: 2-Dimensional Plotting:

Introduction to 2-D Plotting:



Plotting Introduction:

The basic 2-D plot command is: **plot (x,y)** where **x** is a vector (an array, or a list), and **y** is a vector (an array, or list). Both vectors **must** have the same number of elements.

- The plot command creates a single curve with the **x** values on the abscissa (horizontal axis) and the **y** values on the ordinate (vertical axis).
- The curve is made from segments of lines that connect the points that are defined by the **x** and **y** coordinates of the elements in the two vectors.

Try it: Create two arrays (vectors) of numbers (note the lack of commas between elements in your array, which is different from python):

```
>> x=[1 2 3 5 7 7.5 8 10];

>>  y=[2 6.5 7 7 5.5 4 6 8];
```

Note: you don't need the commas in lists (vectors) in Matlab.

Then plot the two vectors:

```
>> plot(x,y)
```

Well, that was exciting. Now you've got a line plotted. I think we need to add a few things though to make it more interesting. We can start with line specifiers. Line specifiers specify information about how the line will appear. So, for instance: plot(x, y, '- r +') will plot the above line with a solid line (-), the line will be red (r) and the markers on the line will be + (+). Below is a partial chart of possible line specifiers (awfully similar to python's…).

Line Style	Specifier	Line Color	Specifier	Marker Type	Specifier
Solid	-	Red	r	Plus sign	+
Dotted	:	Green	g	Circle	o
Dashed	--	Blue	b	Asterisk	*
Dash-dot	-.	Cyan	c	Point	.
		Magenta	m	Square	s
		Yellow	y	diamond	d
		Black	k		
		white	w		

Try the following (and feel free to try any other combination you'd like:

```
>> plot(x,y)
>> plot(x,y,'r')
>> plot(x,y,'--y')
>> plot(x,y,'*')
>> plot(x,y,'g:d')
```

Okay, that's a nice start. But what if we have something like this that we want to plot:

Year	1988	1989	1990	1991	1992	1993	1994
Sales (M)	127	130	136	145	158	178	211

First, let's create our x coordinate:

```
>>year = [1988:1:1994]
```

This creates an array with the first value being 1988, and including as a separate entry in our array every value between 1988 and 1994 in increments of 1. In other words, our new array called year will hold [1988 1989 1990 1991 1992 1993 1994] Type >>year at the prompt to confirm.

Then create our y-coordinate:

```
>> sales=[127,130,136,145,158,178,211]
```

Now let's plot it:

>>plot(year,sales,'- - r*')

You can also set the minimum and maximum limits for the x and y axis using axis([xmin xmax ymin ymax]):

>>axis([1988 1994 120 220])

Okay, so far so good. However, this clearly represents the house sales in one neighborhood. What if we wanted more than one line on our plot? Let's say we had a chart that looked something like this:

Year	1988	1989	1990	1991	1992	1993	1994
Sales(B)	137	130	172	204	178	158	141

We can do this two ways: One is kind of moot at this point considering we've already created our plot, but in the future if we know we want to plot more than one line simultaneously, we can do it using:

plot(x,y,u,v,t,h)

This plots 3 graphs simultaneously: **y** versus **x**, **v** versus **u**, and **h** versus **t.** By default, MATLAB makes the curves in different colors. The curves can have a specific style by adding specifiers after each pair, for example:

plot(x,y,'-b',u,v,'—r',t,h,'g:')

In our case, however, we've already created our plot. We don't want to start over. Instead, we can add another line to our plot using the **hold on/hold off** commands. What **hold on** does is hold the current plot and all axis properties so that subsequent plot commands add to the existing plot. **Hold off** returns to the default mode whereby plot commands erase the previous plots and reset all axis properties before drawing new plots.

So let's try this method. Notice that in this case, the y array is the same as it was previously for the first line on the graph. So we don't need to create a new array for the years. However, we do need to create a new array for the new sales amounts. Let's do that first:

>> sales2 = [137,130,172,204,178,158,141]

Now let's place the new plot line on our existing plot:

>>hold on

>>plot(year,sales2,': cs')

>>hold off

Cool! We added another line to our plot!

Okay, our plot is looking pretty colorful. But what now? We probably want to save this, and maybe we want to save it in a way we can use it in, say, a MS Word document. We can do this using the print option: print <options> <filename>

Print options include:

- -dtiff – prints current figure to .tiff format (into filename.tiff)
- -deps – eps file
- -depsc – color eps file
- -djpeg – jpeg image
- -dpng – png image

So, for instance, if we wanted to save the current plot as a png image, we'd do: print –dpng Ourplot.png.

We'll save ours as a jpeg file. (If this doesn't work, save it as a png file)

```
>> print –djpeg Houseplot.jpg
```

Now open up **Houseplot.jpg** (outside of Matlab) to make sure it looks like your Matlab plot.

Save this to turn in later.

Plotting equations:

We'll plot the equation, $y=3x^3 – 26x + 10$ and its first and second derivatives, for -2 <=x <=4, all in the same plot.

First we'll make the vector x, holding the domain of the equation:

```
>>x=[-2:0.01:4]
```

(x now holds a vector from -2 to 4 by increments of .01)

Then we'll make the vector y with the equation value at each x value (Note: the **.^** means to the power of…)

```
>>y=3*x^3-26*x+6
```

(3 multiplied by x to the 3rd power, minus 26 multiplied by x, plus 6)

(Note, when you multiply by x, you're creating an array in which you multiply by each value in the x array. So really what we're saying above is:

y[0] = 3*x[0].^3 – 26*x[0] + 6,

y[1] = 3*x[1].^3 – 26*x[1] + 6,

y[2] = 3*x[2].^3 – 26*x[2] + 6,

…

for each value in x)

Now we'll create a vector yd with the values of the first derivative:

```
>>yd = 9*x.^2-26
```

(9 multiplied by x squared, minus 26)

Now we'll create a vector ydd with the values of the second derivative:

```
>>ydd = 18*x
```

(In case you couldn't figure this one out, it's just 18 multiplied by each x value)

And now we'll plot 3 lines, the function, the first derivative, and the second derivative:

```
>>plot(x,y,'-b',x,yd,'—r',x,ydd,':k')
```

You should have a plot with a blue, a red, and a black line. Add a title, a legend, and an x and y label. Then save this to a jpg file as **derivativeplot.jpg**. Note: If .jpg doesn't work, try saving it as a png file.

Save this to turn in later.

Part 3: Plotting a function

What if you wanted to plot a function? You can! Matlab wouldn't be matlab if it didn't let you plot a function! Say we wanted to plot:

$y = 3.5^{-0.5x}\cos(6x)$ for -2<=x<=4

First we'd create a vector, running from -2 to 4, with spacing of 0.01:

```
>>x = [-2:0.01:4]
```

Then we' calculate a value of y for each value in x:

```
>>y=3.5.^(-0.5*x).*cos(6*x)
```

And then we'd plot it (adding the line specifiers of your choice):

```
>>plot(x,y)
```

Add a title, a legend, and an x and y label. Then save this to a jpg file as **plottingcos.jpg**. (or png)

Save this to turn in.

A simple line plot (with labels)

Here are the MATLAB commands to create a simple plot of y = sin(3*pi*x) from 0 to 2*pi.

```
>> x = 0:pi/30:2*pi;
```
x is a vector (list) holding values from 0 to (and including) 2*pi, in increments of pi/30

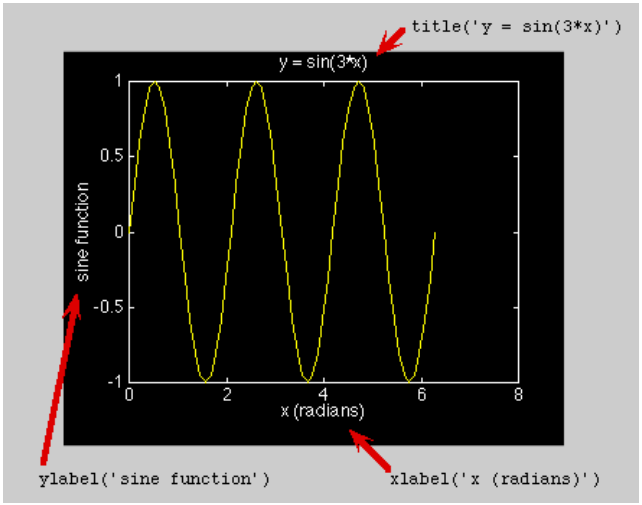
```
>> y = sin(3*x);
```
y is a vector (list) holding the sin of 3* each value in x

```
>> plot(x,y)
>> xlabel('x (radians)');
```
label the x-axis

```
>> ylabel('sine function');
```
label the y-axis

```
>> title('sin(3*x)');
```
put a title on the plot

The effect of the labeling commands, xlabel, ylabel, and title are indicated by the text and red arrows in the figure.



To turn in: Save the file using the File-> Save As->**plot1.jpg** (or png)

Problem to solve:

Suppose the distance traveled by an object fired from an airplane towards the earth (ignoring air resistance) is given by the equation:

$$y = v_0t + (1/2)at^2$$

where v_0 is the object's initial velocity, a is the acceleration due to earth's gravity, and t is time. Use MATLAB to calculate and plot how far an object travels for times 0-7 at intervals .2 seconds if v_0 = 15m/s and a = 9.81m/second^2

Save your *well labeled* plot as **distanceTraveled.jpg** (or png).

PART 3: Basic Linear Algebra: Matrices

This part will introduce you to some basic linear Algebra functions available to you in MATLAB. MATLAB allows for very easy computations with vectors and matrices. First perform the following matrix and vector operations in MATLAB and make sure you understand the output. m' represents the transpose of m. The ':' is a wildcard character that matches all values.

```
>> m = [1 2 5; 4 7 8; 2 4 6]
>> sum(m)
>> diag(m)
>> sum(m')
>> diag(m')
>> m(:,1)
>> m(1,:)
>> m(:,1) == 1
>> m(1,:) < 5
>> any(m(1,:)) < 5
>> all(m(:,1) == 1)
>> all(m(:,1) > 0)
```

Turn on the diary function which will record everything you are about to do in this MATLAB session in a file named **MATLABsession.txt**

```
>> diary   MATLABsession.txt
```

Basic Vector Manipulation. Start by creating the following matrices:

```
>> a = [1 2 3]
a =
1 2 3

>> b = [2 ; 4 ; 6]
b =
2
4
6
```

```
>> c = [4 3 5]
c =
4 3 5
```

Create a few other matrices of your own.

The command **length** returns the number of components of a vector

```
>>length (a)
ans =
3
```

The *dot operator* **.** in MATLAB is used for the component-wise application of the operator that follows the dot. In the following example, a .* c goes through the vector a systematically and multiplies the first value in a by the first value in c, the second by the second, etc. Try it:

```
>>a .* c
ans =  ??
```

The same result is obtained by applying the dot operator to the *power operator* **^** to the vector **a**

```
>>a .^ 2
ans =  ??
```

Consider multiplying two vectors **a** times **b**: 1X3 * 3X1 results in a 1X1

a*b is 1*2 + 2*4 + 3*6. Try it:

```
>> a * b
ans =
28
```

Consider multiplying **b** times **a**: 3X1 * 1X3 results in a 3X3

```
>>b*a
ans =
2  4   6
4  8   12
6  12  18
```

Matrices:

We will now deal with operations on matrices.

Create a 3-by-3 matrix:

```
>>A = [1 2 3;  4 5 6;  7 8 10]
A =
1   2   3
4   5   6
7   8  10
```

NOTE: Python indices start with 0. MATLAB indices start with 1. To extract a submatrix **B** consisting of rows 2 and 3 and columns 1 and 2 of the matrix **A** do the following

```
>>B  =  A([2 3],  [1 2])
B =
4 5
7 8
```

Dot Operator on Matrices:

The dot operator **.** works for matrices too. Let now create a 2x3 matrix:

```
>>A = [2 5 7;  3 4 5]
A =
2  5  7
3  4  5

>>B = [1 4 2;  2 3 5]
B=
1  4  2
2  3  5
```

The following command:

```
>>A   .*  B
ans =
2  20  14
6  12  25
```

computes the entry-by-entry product of **A** with **B**. This is called a "componentwise" operation. However, the following command generates an error message:

```
>>A*B
¨??? Error using ==> *
Inner matrix dimensions must agree.
```

Why? Because A*B is attempting to do matrix multiplication**.** ***(Make sure you understand this!)***

Matrix Transpose

```
>>A = [1 2 3;  4 5 6;  7 8 10]
A =
1   2   3
4   5   6
7   8  10

>>B = A'
B =
1   4    7
2   5    8
3   6  10
```

Matrix Inverse

MATLAB function **inv** is used to compute the inverse matrix.

Let the matrix **A** be defined as follows:

```
>>A = [1 2 3;  4 5 6;  7 8 10]
A =
1 2 3
4 5 6
7 8 10
```

Then

```
>>B = inv (A)
B =
-0.6667   -1.3333   1.0000
-0.6667    3.6667  -2.0000
1.0000   -2.0000    1.0000
```

To verify that B is the inverse matrix of A, one must show that **A*B = I** and **B*A = I**, where **I** is the 3-by-3 identity matrix. We have

```
>>A*B
ans =
1.0000   0   -0.0000
0        1.0000      0
0         0      1.0000
```

In a similar way, check that **B*A = I**.

Leaving the diary file on, solve the following exercise:

Problem:

Alice buys three apples, a dozen bananas, and one cantaloupe for $2.36. Bob buys a dozen apples and two cantaloupes for $5.26. Carol buys two bananas and three cantaloupes for $2.77. Use MATLAB to figure out how much single pieces of each fruit cost. Hint: Set up a system of 3 linear equations. Each equation has the form: ax + by + cz = cost, where a is the number of apples, b is the number of bananas, and c is the number of cantaloupes. That is, encode these 3 linear equations in one matrix for the quantities of fruit purchased, and one column vector for the cost of the purchase.

Solve for the 3 variables (x, y, z) which represent the price of each fruit. Hint: Use Gaussian elimination manually manipulating the rows as shown in class, or learn about the matrix left-division operator in MATLAB .

Your solution should show how much 1 of each fruit costs.

Turn off the diary function which will stop the recording of your MATLAB session in the diary file you named **MATLABsession.txt**

```
>>diary off
```

Other Cool Plottings:

Matlab does a lot of plotting. Here is just a small subset of some of the plotting you can do with matlab:

Polar Plots:

Matlab provides plotting capabilities with polor coordinates. polar(theta, r) generates a polar lot of angle theta (in radians) and a radial distance r.

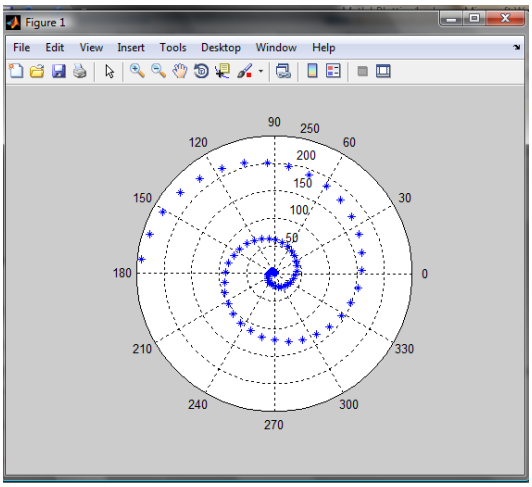
Try:

```
>> theta = 0:0.2:5*pi

>> rho = theta.^2;

>> polar(theta, rho, '*')

>> subplot(1,1,1)
```

This sets the figure window back to being a 1x1 grid, with you plotting in the first square in the grid.

```
>> polar(theta, rho, '*')
```

Your plot should look like this:



To turn in: Save the file using the File-> Save As->plot6.jpg

Bar Graphs and Pie Charts

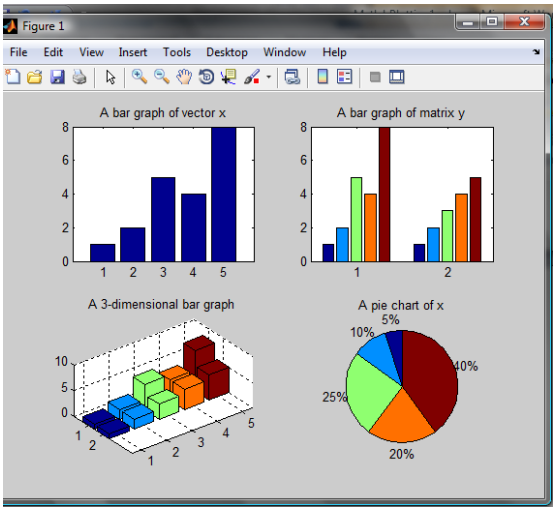
Bar graphs, histograms, and pie charts are popular for reporting data. Here is a table of some of the commonly used Matlab functions for creating bar graphs and pie charts:

bar(x)	When x is a vector (list), bar generates a vertical bar graph. When x is a 2-dimensional matrix, bar groups the data by row
barh(x)	When x is a vector (list), bar generates a horizontal bar graph. When x is a 2-dimensional matrix, bar groups the data by row
bar3(x)	Generates a 3-dimensional bar chart
bar3h(x)	Generates a 3-dimensional horizontal bar chart
pie(x)	Generates a pie chart. Each element in the matrix is represented as a slice of the pie.
pie3(x)	Generates a 3-dimensional pie chart. Each element in the matrix is represented as a slice of the pie.
hist(x)	Generates a histogram.

Let's try creating some of these plot types. We'll be using the subplot function to put them all in one figure window:

```
>> x = [1,2,5,4,8];
>> y = [x;1:5]
```
This creates a matrix (list of lists) of the x list and a list with values from 1 through 5.

```
>> subplot(2,2,1)
>> bar(x),title('A bar graph of vector x')
>> subplot(2,2,2)
>> bar(y),title('A bar graph of matrix y')
>> subplot(2,2,3)
>> bar3(y),title('A 3-dimensional bar graph')
>> subplot(2,2,4)
>> pie(x),title('A pie chart of x')
```
This stuff is just too cool! Here's what your results should look like:



To turn in: Save the file using the File-> Save As->plot7.jpg

Mesh and Surface Plot:

These plots allow you to plot the data as a surface.

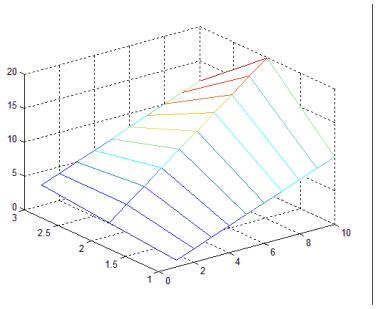
You can use the mesh plot to plot a 2-dimensional matrix. If you do this, the x and y values are the dimensions of the matrix, and the z value is the value at x,y in the matrix. So, for example, if I made the following matrix:

>> z = [1:10;2:2:20;3:12]

I just created a matrix with 3 rows (separated by ;) and 10 columns, 1-10 in the first row, 2-20 by 2s in the second row, and 3-12 in the third row. Now if I do:

>>mesh(z)

the x axis is 3, the y axis is 10, and z is the value at each point, or the surface being plotted. The resulting plot will look like this:



The graph is created by connecting the points defined in z (the matrix) into a rectilinear grid. Notice that x goes from 1 – 3 (the number of rows) and y goes from 1 – 10 (the number of columns in the matrix).

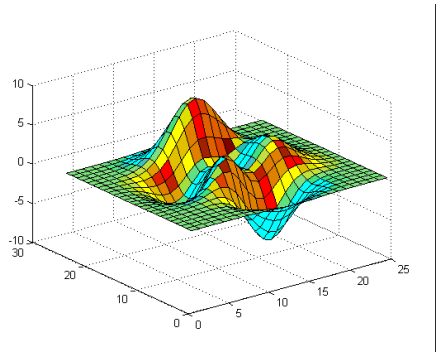
To turn in: ***a mesh plot using mesh, saved as a .jpg file (Mesh.jpg) (or png)***

Surface Plots:

Surface plots are mesh plots colored in. You can get cooler surface maps using shading interp. Try the following:

```
>>z=peaks(25);

>>surfl(z);
```

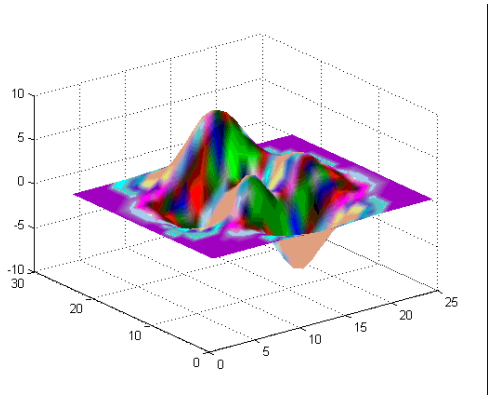
You should get something like this:



Then try:

```
>>shading interp;

>>colormap(colorcube);
```

You should get something like:



Oh, c'mon. That's really cool. I have no idea when you'd need to use the colorcube for shading of a surface map, but it looks really neat. Other colormap options are:

autumn	bone	hot
spring	colorcube	hsv
summer	cool	pink
winter	copper	prism
jet(default)	flag	white

To turn in: ***a surface plot with interpolated shading, using the colormap of your choice, saved as a .jpg (Interp.jpg) (or png)***

There are many, many more cool plots Matlab can do. Feel free to explore on your own.

Files To Turn In:

- Matlab files:
  - CircleArea.m
  - Volume.m
  - SumOddInts.m
  - Findsmallest.m
  - Selectionsort.m

Plots(2-D)

1. Houseplot.jpg (png)
2. derivativeplot.jpg (png)
3. plottingcos.jpg (png)
4. plot1.jpg (png)
5. distanceTraveled.jpg (png)
6. MatlabSession.txt
7. Plot6.jpg (png)
8. Plot7.jpg (png)
9. MeshPlot.jpg (png)
10. Interp.jpg (png)