Matrices and three-dimensional graphing

Module Summary

Technical Tasks:

1. Create matrices in Matlab by assigning values, using for loops, or importing data
2. Do simple arithmetic with matrices in Matlab
3. Portray relationships between multiple variables
4. Refine graphical techniques

Matlab tools:

1. For loops
2. Accessing parts of matrices for graphing or computations
3. Surface plots
4. Contour plots: Colorbar, RBG colors

Reference:

Key learning experience:

i. *Authentic applications:* Understand how to relate multiple variables simultaneously.

ii. *Relate to past experience:* Recognize how spreadsheets are used to relate data and how this can be manipulated or represented. Recognize that our graphing is usually performed with vectors. Extend to two-dimensional matrices and three-dimensional graphs.

iii. *Visualize:* Relate complicated relationships to physical objects (curves or surfaces in three-dimensional space or contour plots).

iv. *Observe and synthesize* the methods of linking large amounts of data to each other and choosing the best way to display the data. Interpret the intersection curves and relate them to the intersections of curves of one dimension.

Expected final product:

Students will create and work with matrices and vectors.

Students will be able to create three-dimensional plots for given ranges of the independent variabels.

Students will interpret the three-dimensional plots by finding points of surface intersections and general trends of data as either independent variable is changed.

Resulting graphs should be refined and include axis labels and legends.

In most scientific research, significant amounts of data are collected. A good way to store this data is with a matrix. The most fundamental description of a matrix is an organized collection of numbers or expressions. You are likely familiar with using spreadsheets (Excel or others), which offer a visual representation of a matrix. Matlab, too, is particularly adept at dealing with matrices (in fact, the program name is the abbreviation of `matrix labratory'). We will approach matrices from the standpoint of data representation.

IN-CLASS ACTIVITY: CREATING MATRICES

We begin by creating matrices in Matlab. There are several methods to do this.

1. Assign all entry values

The first way to create a matrix is to simply assign values to all of the items in the matrix. We want our matrix to look like this

$$A = \begin{bmatrix} 3 & 5 & 2 \\ 6 & 3 & 9 \end{bmatrix}$$

and we use the following matlab command to do this:

```
A=[3,5,2;6,3,9]
```

We have created a matrix with 6 entries. For some nomenclature, this matrix has 2 *rows* and 3 *columns*. The size of the matrix is 2×3 (always the number of rows followed by the number of columns) and the matrix is always named using a capital letter. For the Matlab command, we assigned the values as if we were reading (left to right then top to bottom) with a comma or a space between two elements in the same row and a semicolon separating rows.

Use Matlab to create this matrix $B = \begin{bmatrix} 3 & 2 \\ 4 & 27 \\ 23 & 9 \\ 93 & 3 \end{bmatrix}$

```
B=[3 2;4 27;23 9;93 3]
```

2. Use a *for* loop

Most of our matrices will be too large to assign each entry, and this method is not very efficient, so we turn to another method: a *for* loop. We have already created *while* loops. The loop structure for a *for* loop is well suited when we know beforehand exactly how many times we want the loop to run. The format follows this structure:

```
for iteration=1:1:10  % start at number 1 and go up by 1 each time until 100 is reached
    % perform some action
    iteration
end
```

The first line tells the loop how many times to run. It says `Start by assigning *iteration* the value of 1. Count up by ones until *iteration* equals 100 and run through the actions in the loop each time. It is implicit in the for loop that *iteration* will update itself, so we need not include a line that says *iteration=iteration+1.*

We will give an example of using a for loop to create a matrix. Suppose we wish to create this matrix

$$C = \begin{bmatrix} 2 & 15 \\ 4 & 30 \\ 6 & 45 \\ 8 & 60 \\ \vdots & \vdots \\ 100 & 750 \end{bmatrix}$$

Notice that the columns of the matrix are following specific patterns. The first column only contains even numbers, and they are counting up by two. The second column contains multiples of 15. Another way to describe this is to say that in each row of the matrix, the first entry is twice the row number, the second entry is 15 times the row number. In total, we will have 50 rows.

A for loop to create the matrix C looks like this

```
for iteration=1:50  % if we only include two numbers in the for loop, it is assumed that we wi
    C(iteration,1)=2*iteration;
    C(iteration,2)=15*iteration;
    jjjj
end
fprintf('The first 10 rows of the matrix are')
C(1:10,1:2) % show rows 1 thorugh 10 and both column 1 and 2.
```

We can access specific entries in a matrix. For example, if we would like to know the entry in row 5, column 2, we write

```
C(5,2)
```

We can access the two columns separately (each is called a *column vector*) in this way

```
Evens=C(:,1);  % A(r,c) calls for
Fifteens=C(:,2);
```

Check the sizes of the matrices *Evens* and *Fifteens.* In the future, we will call these *column vectors* since they only consist of a single column (if they were a single row of entries, they would be called *row vectors*). We can also acces partial pieces of the matrix or the vectors we have just created.

```
First4RowsOfC=C(1:4,:)
First6RowsOfEvens=Evens(1:6)
```

Practice accessing matrix entries by finding rows 5:10 and only column 2 of the matrix

```
C(5:10,2)
```

The majority of data collected in science is time-related, and testing or observations are performed at regular intervals--a scenario that is appropriate to use a *for* loop

Try This:

1. Create a vector $x = [3\,1\,2\,5\,4]$, then use a *for* loop and x to create y where y contains the same elements of x in reverse order. You will have to examine the indices.

```
x=[3 1 2 5 4]
```

```
for iteration=1:5
    y(iteration)=x(6-iteration);
end
fprintf('xmatrix in reverse')
y(1:5)
```

1. Now create a matrix z by using a for loop to scroll through the values of x: keep the same value if it is larger than 2, but turn if to zero if the valueis less than or equal to 2.

```
x=[3 1 2 5 4]
for iteration=1:5
    if x(iteration)>2
        z(iteration)=x(iteration)
    else
        z(iteration)=0
    end
end
```

3. Use built-in functions

Matlab has a few built-in functions to create particular matrices. We can create a matrix of any size that consists of all zeros. As an example, let's create a matrix called MyZeroMat which is a 4×3 matrix of zeros.

```
MyZeroMat=zeros(4,3)
```

We can create matrices whose values are all one.

```
MyOnesMat=2*ones(10,10)
```

Create a matrix which is 5×2 whose elements are all equal to 0.37.

```

```

We have already used the function linspace to create row vectors of values between two endpoints that are linearly (evenly) spaced. To address this again, we create a vector of 5 values between 6 and 10. Alternatively, we could find values between 1 and 100 that are spaced logarithmically.

```
MyLogVector=logspace(
```

Linspace is useful if we know exactly how many values we want to have. If, on the other hand, we want the values to be equally spaced, we will use a semicolon as we did in the for loop.

4. Combine existing matrices

If we already have matrices in our workspace, we can combine them in various ways to create new matrices. We could use vertcat (stands for vertical concatonation) to stack the matrices, one atop the other. Alternatively, we could use the semicolon. In both of these cases, the number of columns in the matrices A and B must be the same.

```
A=[4,6,3;2,3,6]
B=[8,6,2;10,4,3]
```

```
Vertical1=vertcat(A,B) %
Vertical2=[A;B]
```

Similarly, we could stack the matrices one to the left of the other. The built-in command is horzcat (for horizontal concatonation). Alternatively, we could use a comma or space to separate the matrices. In both cases, the number of rows in columns A and B must be the same.

```
Horizontal1=
Horizontal2=[A B]
```

We have mentioned several times the sizes of matrices and the number of rows or columns. If we wish to change the rows into columns and columns into rows, we can use an apostrophe (this is called the *transpose* of the matrix)

```
Horizontal2'
```

5. Read from a file

If data is already well organize in a spreadsheet or text file, we can create the corresponding matrix in Matlab using one of several commands. The most common are these:

load, import, fread, fscanf--all appropriate for reading text files

xlsread--appropriate for reading excel spreadsheet data where some rows or columns might be label information (words or date formats)

csvread--appropriate for reading csv files where some rows or columns might be label information (words or date formats)

Note the commands differ slightly in the information they require.

For practice, we have created a file named `WaterLevelCalifornia.csv' on Google drive. This was pulled from NOAA's website and represents the water depth in Monterey Bay, California. Save this file into your working folder. Feel free to open it and take a look at the format of the data. To access the data in Matlab, use one of the above commands.

```
DataRetrieve=
```

Create the data matrix that has two columns. You can see the dimensions of this matrix in the workspace (7440×2). Plot the first two days of second vector as a function of the first using the following commands

```
plot(DataRetrieve(1:480,1),DataRetrieve(1:480,2))
set(gca,'FontSize',14)
xlabel('Time [hr]')
ylabel('Water level [ft]')
```

What patterns do you see? Knowing what this data represents, does the graph make sense? Could you have expected the graph to look like this?

Plot all of the values for 31 days (time units have been converted to days).

```
plot(DataRetrieve(:,1)/24,DataRetrieve(:,2))
```

```
set(gca,'FontSize',14)
xlabel('Time [day]')
ylabel('Water level [ft]')
```

What patterns do you see? Does this graph make sense? Did you expect the graph to look like this?

IN-CLASS ACTIVITY: ARITHMETIC WITH MATRICES

We can perform standard arithmetic with matrices in the same way that we do with numbers.

```
FirstNumber=5;
SecondNumber=FirstNumber+3
% Adding three to a matrix is the same as adding three to each entry in the matrix.
A=[4,6,3;2,3,6]
A+3
A(1,:)+1
A+[1;1]
```

We can also add two matrices together, which means adding the entries in corresponding positions. As a result, we can only add matrices which are the same size (same number of rows and columns). If we try to combine matrices whose sizes don't match, Matlab cannot perform the calculation.

```
B=[8,6,2;10,4,3]
A+B
C=[5,2;7,3;3,5]
A+C    % Matrices must be same size.
```

Since A is a 2×3 matrix and C is a 3×2 matrix, they cannot be added. On the other hand, we could exchange the rows and columns of C, making it a 2×3 matrix, so that it can be added to A.

```
C'     % Transpose, flips rows and columns (ex: 3x2 becomes 2x3).
A+C'
```

Subtracting numbers from matrices or finding the difference between two matrices works in the same way.

```
A-5
A-B
```

Multiplication by a constant number is the same as multiplying each entry separately by the constant

```
2*A
```

Multiplying two matrices is an unusual operation and requires the matrices be particularly sized relative to each other. On the other hand, Matlab has a built-in feature to multiply two matrices the corresponding entries of the matrices (as with addition and subtraction). It requires adding a period before the operation. You may have noticed these errant periods in previous codes.

```
A*B
A.*B     % The decimal goes element by element.
```

Finally, we can compute values that relate to the entries of a matrix, like the sum of elements, maximum value, minimum value, etc. using the commands *sum*, *max*, and *min*. What are the results of these commands when we apply them to *A*? Can you describe how these functions apply to a matrix? How might we find the max value of the whole matrix?

```
sum(A) %finding the sum of each column
max(A) %find the max in each column

max(max(A))
```

IN-CLASS EXERCISE

Task 1: Create a matrix with the following properties.

1. The size of the matrix will be 5×1400. The first column will represent time--70 seconds in total. You should represent time in seconds. What do we expect the first time value to be? What do we expect the last time value to be? Use a *for* loop to create this column.

```
for i=1:1400
    myTime(i)=i*70/1400;
end
```

1. The second column will be read from the file: ExerciseData.dat (found on Google drive, use the command load). The data is acceleration, with units of m/sec^2.

```
MySecondCol=load('ExerciseData.dat');
```

1. The third column will be the absolute value of column 2. For help using the absolute value, either type 'help abs' in the command line, or see documentation. Units are still m/sec^2.

```
MyThirdCol=abs(MySecondCol);
```

1. The fourth column will be acceleration from column 3, but we convert the units to ft/min^2. How do we convert units?
2. The fifth column will be 2 times column two plus 3 times column three.

Create a labeled plot that shows the graphs of both columns 2 and 5 as dependent variables on the y-axis versus time (the independent variable) on the x-axis.

```

```

Task 2: Create a matrix that is read from the file: MatrixExample.dat (on Google Drive). This is hypothetical data for a study on blood pressure. The first column represents the fiber intake for a collection of 100 subjects. It ranges from 10.1-39.7 grams. The second column represents the weight of the subjects ranging from 50.3-122.9 kg. The last column is the systolic blood pressure of the subject with units mmHg.

1. Suppose the researchers in charge of the experiment believed that blood pressure depended on fiber intake alone. What sort of graph would they generate? Because this is not a 'function' in the traditional sense but is instead 'observed data', we will use the command 'scatter' rather than 'plot'. The commands have the same general format, but the resulting data points will not be connected. How would the researcher interpret their graph?

```

```

2. Suppose the researchers believed that weight was the only factor affecting blood pressure. Again, use the scatter command to create a graph that the researchers would generate. What would they interpret from their graph?

```

```

Blood pressure depends on both fiber intake and weight, but we wouldn't know it if we plot them separately. We will now learn how to create a plot that includes ALL this information.

IN-CLASS ACTIVITY: Three-dimensional graphs

Task 1: We will practice graphing in three dimensions by plotting an analytic function t in terms of both p and v. The equation looks like

$$t(p,v) = 0.7 - \left(\frac{p}{2}\right)^2 \cdot \sqrt{v}$$

We allow p to vary between 0.05 and 0.7 and v to vary between 0 and 1. Notice on the left hand side that our function includes two inputs. To make a plot of this, we will need to use several commands we have not seen previously.

1.First, we define a range for p and v.

```
P=linspace(-2*pi,2*pi);
V=linspace(-2*pi,2*pi);
```

2. Then we use the command 'meshgrid' to create a grid of values for p and v.

```
[first,second]=meshgrid(V,P);
```

Go to the workspace and look at each of v and p. How would you describe them? What did the *meshgrid* command do?

3. We evaluate the function at each of these points and create a matrix that represents those function values.

```
T=first.*sin(second)-second.*cos(first);
```

Notice in the command above that there is a period before the '*' symbol. This ensures that the values in our meshgrid align properly. Take a look at T in the workspace.

A more interesting T is given by $T(v, p) = p\sin(v) - v\cos(p)$ which has the following code,
T=p.*sin(v)-v.*cos(p);

For the rest of our plotting practice, I would recommend using this function (it is a little bit more interesting). To do so, we will wish to redefine P and V to range between -2π and 2π.

Task 2: There are several different ways to represent three dimensional objects. They each have advantages and disadvantages so that we can choose the most appropriate for our given situation.

1. The first will create a collection of lines. We will add some labels.

```
plot3(first,second,T) % a line for each v value
xlabel('1st Indep Var')
ylabel('2nd Indep Var')
zlabel('Function T')
```

This viewpoint is not ideal. Luckily, we can move it around. Hover the mouse over the plot. Several buttons appear at the top of the figure. The first is for finding specific data points by clicking somewhere on the plot. The second looks like a cube with an arrow circling it. Click this one, then click somewhere on the plot and drag your mouse around. Experiment to get the viewpoint that is most interesting to you.

This plot contains 50 lines. Why 50?

2. This is not necessarily the ideal way of inspecting 3-D plots. We consider a few others and examine their outputs

```
mesh(first,second,T) % 3d plots needs 3 argumetns, each of them there own matrix
xlabel('1st Indep Var v')
ylabel('2nd Indep Var p')
zlabel('Function T')
```

What is good about this graph? What does the color seem to represent?

3. We can use the color scale to describe height of the function even in the two dimensional sense. These are called contour plots and look a little bit like topographical maps of landscape. You can read more about creating contour plots here.

```
contour(first,second,T,20)
xlabel('1st Indep Var first')
ylabel('2nd Indep Var second')
```

Notice we only use two axes where each represents one of the independent variables. The dependent variable is represented by the color of the line. In general, darker color means a lower value. To clarify and get an idea of scale, colorbar to show what the color values mean.

```
contour(first,second,T,20)
xlabel('1st Indep Var first')
ylabel('2nd Indep Var p')
colorbar
```

Compare this to the mesh plot above to understand how color describes the 3-dimensional function. Examine the colors here.

4. Alternatively, there is a 3-dimensional style of plot that includes the contours below the curve.

```
surfc
xlabel('1st Indep Var v')
ylabel('2nd Indep Var p')
zlabel('Function T')
```

5. The plot above helps us to understand how the contour plots, but is not used very often. The most common three-dimensional plot used is a surface plot. It includes the curve itself and the grid that we saw above using the mesh command. If it would benefit us, we can remove the grid, or even make the surface transparent.

```
surf(first,second,T,'EdgeColor','None')
xlabel('1st Indep Var first')
ylabel('2nd Indep Var second')
zlabel('Function T')
colormap lines
```

What are the benefits of a three dimensional graph? What are the disadvantages? If we *must* represent a function of two variables, how can we overcome the disadvantages?

IN-CLASS ACTIVITY: Interpretting multiple surfaces

1. In this section, we will create a plot with multiple curves and interpret the results. We have already created (many) plots of $t(v, p) = 0.7 - pv$. This can represent the capillary flow in our field capacity example. We now include the surface for the gravitational flow, $f(v, p) = 9.81 \cdot 0.06 \cdot v$. Where do the these curves intersect? Is it at a point as in the two-dimensional plot? What is represented by the intersection of the curves?

In order to see both curves, we make one of them slightly transparent.

```
f=9.81.*0.06.*first
t=0.7-first.*second
first=linspace(0,1,100)
second=linspace(.06,.7,100)
[first,second]=meshgrid(first,second)
k=0.06;
%
h=surf(first,second,t);
alpha(h,0.7)
hold on
colormap Winter
surf(first,second,f)
hold off
surf(first,second,t)
hold on
surf(first,second,f)
hold off
%
```

Notice that, for some values of p, the curves do not intersect. Which values? What does this mean as it applies to our field capacity example? For $p = 0.6$, where do the curves intersect (find v and flow).

Because we are given the function for a specific p, we can create a polygon (in our case a rectrangle) to highlight that p value. This is called a patch and has this format

$$patch([x_1, x_2, ..., x_n], [y_1, y_2, ...y_n], [z_1, z_2, ...z_n], [Color])$$

```
%
surf()
patch([0,1,1,0],[0.6,0.6,0.6,0.6],[0,0,1,1],[1,0,0])
```

Experiment with the color vector. It has three numbers, each between 0 and 1. They represent the red, green, and blue scale $[R, G, B]$. As an example, $[1, 0, 0]$ represents red.