MATLAB Project 5

Orthogonality, Least Squares & Graphing

1 Introduction

Once again, the first portion of this project will parallel what we have been learning (or will soon be learning) in class, namely, Orthogonality (Sections 6.1-6.3), and Least Squares Problems (Section 6.5).

The second portion will provide a basic introduction to graphing in MATLAB. We will barely scratch the surface with all of the graphing features MATLAB has to offer, but hopefully you will master the basics we cover.

2 Obligatory Tasks

As with your last projects, you will need to open your favorite word processor (like Microsoft Word) so that you can complete the Exercises listed in this project. Be sure to include your full name at the top of your document!

If you choose to submit your project electronically, you must using the following naming convention (points WILL BE DEDUCTED if you fail to do so!) – THIS SHOULD BE THE LEAST DIFFICULT PART OF THIS PROJECT, THERE IS NO REASON NOT TO DO IT!!!

`Lastname_Firstname_MATLAB_Project5.docx`

(or whatever file extension you chose to use).

3 Exercise Key

The several icons will appear next to each Exercise you are asked to complete. Use the below table to determine what information should be included in your submitted document to fully complete the exercise.

Icon:	Item:	Comments:
	Commands entered in MATLAB & resulting output	You should copy relevant input and output from MATLAB and paste it into your document. You need only include commands that worked.
	Plots & Graphs	Include all graphs generated in an exercise unless the problem specifically tells you which/how many to include.
	Full sentence response	Each exercise contains a question that you should use at least one or two complete sentences to answer. Even if you're stuck, write down any reasoning or ideas you've had.

4 Orthogonality

Recall that we defined two vectors to be orthogonal if their dot product equals zero, i.e., $\mathbf{u}$ and $\mathbf{v}$ are orthogonal if and only if $\mathbf{u} \cdot \mathbf{v} = 0$.

There are two ways to compute the dot product in MATLAB:

```
>> u'*v
```

or alternatively,

```
>> dot(u,v)
```

Enter the following vectors into MATLAB – they will be used in the following exercise:

$$\mathbf{v} = \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix}, \ \mathbf{w} = \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} 6 \\ 1 \\ -3 \end{pmatrix}, \ \mathbf{y} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}, \ \mathbf{z} = \begin{pmatrix} 2 \\ -15 \\ -1 \end{pmatrix}.$$

Exercise 1. **(a)** List all maximal orthogonal subsets of the set $X = \{\mathbf{v}, \mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}\}$.

Note, a subset B is called a *maximal orthogonal subset* of X if there are no larger orthogonal subsets of X containing B. In other words, this question is asking you to group the vectors $\mathbf{v}, \mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}$ in as many ways as possible so that all the vectors in each group are orthogonal to each other, and so that you cannot add any other vector to each group that are orthogonal to the ones already in that group. For example, the subset $\{\mathbf{w}, \mathbf{x}\}$ contains two vectors that are orthogonal to each other, and no other vector is orthogonal to both of these. (This is only one example, there are others!)

What is the maximum number of non-zero orthogonal vectors that you can find in $\mathbb{R}^3$? What about in $\mathbb{R}^n$? Explain.

(b) Pick the largest subset from part (a) and normalize all of the vectors in it using the following command:

```
>> u = u/norm(u)
```

Store the resulting vectors as columns of a matrix W. Enter them in alphabetical order from left to right.

Let us take a closer look at the resulting matrix W – it is an orthogonal matrix! (Recall that orthogonal matrices are square matrices with orthonormal columns.)

Exercise 2. **(a)** Calculate $W^T W$. What do you obtain? Why do you think this happens?

(b) Enter the following vectors into MATLAB:

$$\mathbf{a} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix}.$$

Compute the norm of $\mathbf{b}$ and the norm of $W\mathbf{b}$. What do you notice? Compute the dot product $\mathbf{a} \cdot \mathbf{b}$ and compare it to $W\mathbf{a} \cdot W\mathbf{b}$, what do you notice?

(c) It should be clear that W is invertible (the columns are linearly independent as the vectors are orthogonal). Hence we can compute W^{-1}. Use `inv(W)` to compute the inverse of W and compare it to W^T, what do you notice?

Remark: Orthogonal matrices are incredibly important in geometry and physics. One of the reasons why, is that orthogonal 2×2 matrices with positive determinant represent rigid motions of the plane that keep the origin fixed. Similarly, 3×3 orthogonal matrices with positive determinant represent rigid motions of space keeping

the origin fixed. If you allow the determinant to be negative, you also allow reflections through a line in the case of the plane, or through a plane in the case of 3-space. In any case, $n \times n$ orthogonal matrices are precisely the distance preserving transformations of n-space that keep the origin fixed.

Given a basis for a vector space MATLAB can also perform the Gram-Schmidt process to generate an orthogonal basis for that same space; however, the command isn't quite what you may think. To use the Gram-Schmidt process in MATLAB one must actually compute the QR decomposition of the matrix whose columns form a basis for the vector space in question. Given a matrix A, the command `qr(A)` performs the QR-factorization algorithm and returns an orthogonal matrix Q who columns are obtained from the columns of A by using the Gram-Schmidt process (and then normalized), and an upper triangular matrix R such that $A = QR$.

One thing to note here is that if A is not square the QR-factorization method in MATLAB will actually return a square matrix! This is because MATLAB performs the so called "full" QR-factorization, which is helpful in certain circumstances. To obtain the QR-factorization that we learned about in class we must pass an optional argument into the `qr()` command; namely, we must execute `qr(A,0)`, e.g.,

```
>> A = [1,1,0;2,0,3]

    A =
            1     2
            1     0
            0     3

>> [Q,R]=qr(A,0)

    Q =
        -0.7071     0.3015
        -0.7071    -0.3015
         0          0.9045

    R =
        -1.4142     -1.4142
         0           3.3166
```

From this we now know that the columns of Q give us our orthonormal basis found by the Gram-Schmidt process on the columns of A.

Exercise 3 The following is a basis for $\mathbb{R}^3$, use the Gram-Schmidt process (via the QR-factorization) to obtain an orthonormal basis:

$$\left\{ \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} \right\}.$$

5 Least Squares

Let us now turn our attention to something a bit more practical. Suppose we are interested in studying the relationship between temperature and pressure. The first step is to obtain some experimental data. In order to do that, we attach a temperature and pressure sensor to a metal container. Starting at room temperature (75°F) we slowly heat the air inside our container to 195°F and record temperature and pressure readings. We get a total of five data points:

Temperature (°F)	Pressure (lb/in^2)
75	15
100	23
128	26
159	34
195	38

Since we suspect the relationship to be linear, we will be looking for a straight line with the equation $y = mx+b$, where y is the pressure, x the temperature, m the slope, and b the y-intercept. We would like this line to pass through the data points we collected from the experiment. Plugging in these data points into the equation of the line, we obtain the following system of equations:

$$\begin{cases} b + 75m = 15 \\ b + 100m = 23 \\ b + 128m = 26 \\ b + 159m = 34 \\ b + 195m = 38 \end{cases}$$

We can attempt to solve this system of equations, by first rewriting it in the matrix form:

$$\begin{pmatrix} 1 & 75 \\ 1 & 100 \\ 1 & 128 \\ 1 & 159 \\ 1 & 195 \end{pmatrix} \begin{pmatrix} b \\ m \end{pmatrix} = \begin{pmatrix} 15 \\ 23 \\ 26 \\ 34 \\ 38 \end{pmatrix}.$$

Unfortunately, this system does not have a solution, even though the physical law is of linear order. This is due to the fact that it is not possible to get perfect accuracy when recording data.

Since it is not possible to solve the above system, we use the least squares method to find the closest solution. That is, we want to get the best fit line, i.e., a line that minimizes the sum of the squares of the distances from each point to the line.

To simplify notation let us assign names to the matrices and vectors of the above system:

$$B = \begin{pmatrix} 1 & 75 \\ 1 & 100 \\ 1 & 128 \\ 1 & 159 \\ 1 & 195 \end{pmatrix}, \qquad \mathbf{c} = \begin{pmatrix} b \\ m \end{pmatrix}, \qquad \mathbf{d} = \begin{pmatrix} 15 \\ 23 \\ 26 \\ 34 \\ 38 \end{pmatrix}.$$

To remind you of the idea behind least squares, recall that we are trying to find a vector $\mathbf{c}$ that minimizes the distance $\|B\mathbf{c} - \mathbf{d}\|$. Of course, we can never hope for this norm to be zero since $B\mathbf{c} = \mathbf{d}$ has no solutions. However, the equation $B\mathbf{c} = \text{proj}_V \mathbf{d}$, where V is the column space of B, does have a solution. This is because $\text{proj}_V \mathbf{d}$ lies in the column space of B, and B hits every vector in its column space. The solution to this equation turns out to minimize the norm $\|B\mathbf{c} - \mathbf{d}\|$. In the following exercise we will solve the equation $B\mathbf{c} = \text{proj}_V \mathbf{d}$ directly.

Exercise 4. **(a)** Enter the matrix B and vector $\mathbf{d}$ that are defined above into MATLAB. Follow the steps below to compute $\text{proj}_V \mathbf{d}$:

1. We must first find an orthonormal basis for V. For this we use the `qr( , 0)` command. Execute the following

   ```
   >> [Q , R] = qr(B,0)
   ```

2. The columns of Q form an orthonormal basis for V. Let us give them their own names:

   ```
   >> x = Q(:,1)
   >> y = Q(:,2)
   ```

3. Now we are ready to compute $\text{proj}_V \mathbf{d}$. Recall that to do this, we must add the projections of $\mathbf{d}$ onto each element in the orthonormal basis of V. Executing the following command will accomplish just that:

   ```
   >> v = dot(x,d)*x  + dot(y,d)*y
   ```

   (Note, we did not need to divide by the dot products `x`·`x` or `y`·`y` since the QR-factorization (and hence MATLAB's Gram-Schmidt implementation) provides *orthonormal* vectors, not just orthogonal as we did in class!)

(b) Now solve the equation $B\mathbf{c} = \text{proj}_V \mathbf{d} = \mathbf{v}$ by executing

```
>> c = B\v
```

Check that your answer is correct using

```
>> B*c - v
```

and making sure that the answer is zero. (Remember that MATLAB may return a very small number instead of zero due to a rounding error.)

(c) Now let's compare the answer in the previous part to what MATLAB gives us using the built-in least squares routine called `lscov()`. To use it, we must specify three parameters: the matrix B, the vector d, and a covariance matrix X, which you don't have to worry about in this course. Type in the following command:

```
>> c1 = lscov (B, d, eye(5))
```

(Note that for the matrix X we just took a 5×5 identity matrix, but don't worry about why.) How does your answer to this part compare to your answer in part (b)?

Now, as a lead-in to the next topic, we will plot our data points, along with the best-fit line to illustrate the effectiveness of the least squares method.

Exercise 5. **(a)** What is the equation of the best fit line? (Remember that the equation of the line is $y = mx + b$, and the vector **c** you obtained in the previous exercise is equal to $\begin{pmatrix} b \\ m \end{pmatrix}$

(b) Use the equation of the line you obtained in part (a) to calculate the pressure at the following temperatures: 35°F, 170°F, and 290°F.

(c) We can now plot the data points and our line to see visually if the approximation is good or not. Execute the following:

```
>> x=B(:,2);
>> y=d;
>> t=0:1:300;
>> z=polyval([c(2);c(1)],t);  % polyval()returns a polynomial of
                              % specific degree with coefficient
                              % specified by a vector.
>> plot(x,y,'x',t,z)
```

6 Graphing

MATLAB provides a variety of functions for displaying data as 2- or 3-dimensional graphics. We will cover a few here, but for a comprehensive list with examples see

`http://www.mathworks.com/help/matlab/creating_plots/types-of-matlab-plots.html`

For 2-dimensional graphics, the basic command is

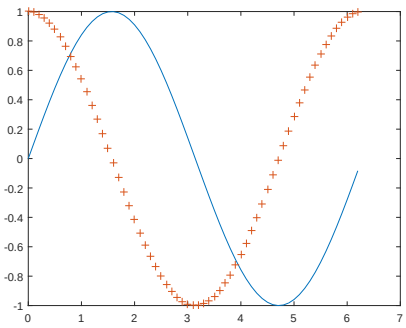
```
plot(x1, y1, 'line style', x2, y2, 'line style' ...)
```

This command plots vector `x1` versus vector `y1`, vector `x2` versus vector `y2`, etc., on the same coordinate axes. Other common commands for 2-dimensional graphics are: `polar`, `bar`, `stairs`, `loglog`, `semilogx`, and `semilogy`.

We saw one example above of how the plot command can be used; here is another example:

Suppose we want to plot the two functions $f(x) = \sin(x)$ and $g(x) = \cos(x)$ with $x \in [0, 2\pi]$ on the same axes. We will use a solid line for the graph of $\sin(x)$ and the symbol + for $\cos(x)$:

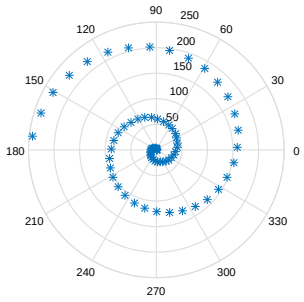
```
>> x = 0 : 0.1 : 2*pi;
>> y1 = sin(x);
>> y2 = cos(x);
>> plot(x, y1, '-', x, y2, '+')
```



We can also use MATLAB to plot using the polar coordinate system:

Suppose we want to plot $\rho = \theta^2$ with $0 \leq \theta \leq 5\pi$ in polar coordinates. We need to execute the following commands:

```
>> theta = 0 : 0.2 : 5*pi;
>> rho = theta.^2;
>> polar(theta, rho, '*')
```



Note that we used the operation `.^2` rather than just `^2`. When working with vectors/matrices, if you want an operation to be applied to each element of the vector you must use a dot `.` before the operation, without it, you would be trying to multiply the vector by itself which is impossible due to its dimensions.

For 3-dimensional graphs, the most commonly used commands are:

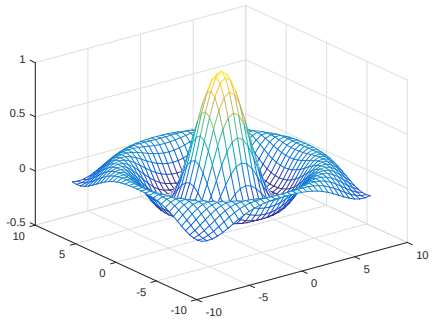
```
>> plot3(x1, y1, z1, 'line style', x2, y2, z2, 'line style'...)
>> contour(x,y, Z)
>> mesh(x,y,Z)
>> surf(x,y,Z)
```

The first statement is a 3-dimensional analogue of `plot()` and plots lines and points in 3-D. The second statement produces contour plots of the matrix `Z` using vectors `x` and `y` to control the scaling on the `x`- and `y`- axes. For surface or mesh plots, you use the third statement where `x`, `y` are vectors or matrices and `Z` is a matrix. Other commands available for 3-D graphics are: `pcolor`, `image`, `contour3`, `fill3`, `cylinder`, and `sphere`.

For example, we can generate a mesh surface for $z = \sin(r)/r$ with $r = \sqrt{x^2 + y^2}$ in MATLAB in the following way:

The first step in displaying a function of two variables, $z = f(x, y)$ is to use the `meshgrid` function to generate `X` and `Y` matrices consisting of repeated rows and columns, respectively, over the domain of the function. THe function can then be evaluated over this mesh and graphed:

```
>> x = -8 : 0.5 : 8;
>> y = -8 : 0.5 : 8;
>> [X,Y] = meshgrid(x,y);
>> R = sqrt(X.^2 + Y.^2) + eps;  %we add this "eps" to avoid the R=0 case
>> Z = sin(R)./R;
>> mesh(x, y, Z)   % mesh(X, Y, Z) also would work here
```



To obtain a surface plot rather than a mesh one simply execute `surf(x, y, Z)` – try it!

	**Exercise 6. (a)** Let $f(x) = e^{-x^2}$ be defined on the interval $-2.5 \leq x \leq 2.5$, and $g(x) = \sqrt{4 - x^2}$ be defined on $-2 \leq x \leq 2$. Plot both $f(x)$ and $g(x)$ on the same coordinate axis, incrementing the x-values by 0.1 for both functions. (Be careful to use the correct operations, i.e., you don't just want `^`!)
	**(b)** Notice that $g(x)$ is the equation for a semicircle of radius 2; however its graph from part (a) does not look like a semicircle. This is because the axes are not scaled equally. Plot both functions again, but this time use the command: `>> plot(...); axis equal`
	**(c)** Create a 3-dimensional surface graph of $Z = xe^{-x^2-y^2}$ on a mesh of x and y values between -2 and 2, incremented by 0.2.