Math 281-3 **Due Thursday, April 28**
Homework 4 (*April 26, 2016*)

For this homework, you will be using *Matlab* (or the free and open-source version of *Matlab* called *Octave*). Include a clearly-labeled printout of the work you did in Matlab for each problem.

Before you start, read through
`http://web.gps.caltech.edu/classes/ge11d/doc/matlab_Resource_Seminar.pdf`
for a short Matlab introduction. For additional documentation, see the Matlab website
`http://www.mathworks.com/help/matlab/index.html`

Quick Tips:

- You can supress Matlab output by adding a '`;`' to the end of your Matlab command (this is useful for Matlab commands executed in loops).
- You can save plots/figures from the figure's dialog by selecting 'Save as' and then choosing `png` or `jpg` in the file-types dropdown.
- You can include Matlab code in LaTeX by copying and pasting it between `\begin{verbatim}` ... `\end{verbatim}`.

1. Find the general solution to the system of equations corresponding to each augmented matrix.

   (a)
$$\left[\begin{array}{ccccc|c} 1 & 11 & -22 & 0 & 3 & 1 \\ 5244 & 1542 & -3084 & 576 & 16308 & 2 \\ 108 & 1188 & -2376 & 0 & 324 & 3 \\ 264 & 465 & -930 & 25 & 817 & 4 \\ 2166 & 3729 & -7458 & 206 & 6704 & 5 \end{array}\right]$$

   (b)
$$\left[\begin{array}{ccccc|c} 7 & 3 & 2 & 3 & 7 & 1 \\ 9 & 3 & 2 & 1 & 10 & 2 \\ 7 & 5 & 2 & 2 & 8 & 3 \\ 4 & 10 & 3 & 1 & 10 & 4 \\ 4 & 6 & 1 & 3 & 9 & 5 \end{array}\right]$$

2. Although the `rref` command will row-reduce for you, Matlab makes it easy to row reduce "by hand." If `A` is a matrix, the row operation $R_2 \mapsto R_2 - 6R_1$ can be performed with the command `A(2,:)=A(2,:)-6*A(1,:)` and other row operations can be performed similarly.

   (a) Consider the 3×3 matrix $A = \begin{bmatrix} 1 & 4 & 0 \\ 1 & 5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. You only need to perform two row operations to fully reduce A. Call these two operations $R^{(1)}$ and $R^{(2)}$. Use Matlab to perform those row operations on A and also perform the same row operations on B (effectively, un-row reducing B).

   (b) Let $B_1 = R^{(1)}(B)$ and $B_2 = R^{(2)}(B)$ be the matrices obtained from B by applying the two row operations in isolation (in Matlab, you can name variables `B1` and `B2` if you like). What happens when you perform the matrix product B_1A? How about B_2A? And B_2B_1A? Finally compute the product BA, and explain why you get the results you do.

   (c) Let $X = \begin{bmatrix} 2 & 1 & 3 \\ 2 & 1 & 3 \\ 3 & 2 & 4 \end{bmatrix}$. Find a matrix Y so that $YX = \text{rref}(X)$.

3. A loop in Matlab/Octave is written with the

```
for i=1:n,
    [loop content]
end
```

syntax.

For example, if we want the variable x to be an accumulated sum of squares from 1^2 to 15^2 (that is $x = 1^2 + 2^2 + \cdots + 15^2$), we could write

```
x=0;
for i=1:15,
    x = x + i^2;
end
```

After executing this code x will be the value 1240. The code works as follows: first we assign the value 0 to x. Then we enter the loop. In the first iteration, when i is 1, the new value of x will be the current value, 0, plus 1^2. The next time through the loop, i is 2 and so we assign x to be the current value, 1^2, plus 2^2, etc.. Once we have looped through with the value of i being 15, we stop.

If statements are written with the

```
if [var]==[val],
    [if content]
end
```

syntax (note the double equals).

For example, if we wanted x to be the sum of the squares of only the even numbers between 1 and 15, we might write:

```
x=0;
for i=1:15,
    if round(i/2) == i/2,
        x = x + i^2;
    end
end
```

Here, `round(i/2)==i/2` is true when `i/2` has no decimal places and so it is even. Thus, when that happens (and only when that happens) we add the value i^2 to x.

Question:

(a) The command `rand(3,1)` will generate a random 3×1 column vector. Find out what percentage of random triples of vectors $\{\vec{a}, \vec{b}, \vec{c}\}$ are linearly independent. Sample at least 1000 random triples to gather a statistic.

Consider your result and come up with an explanation of why you got the percentage you did.

Tips: Build a loop to test this 1000 times, but start out with a much smaller loop; use '`;`' inside your loop so you don't get a bunch of unhelpful information printed to the screen; and use Matlab's built-in commands to help you like `rref` to row reduce, `rank` to find the rank, or `rand(3,3)` to make a 3×3 matrix of random numbers.

(b) The command `round(rand(3,1))` will generate a random 3×1 column vector of just zeros and ones. Find out what percentage of random triples $\{\vec{a}, \vec{b}, \vec{c}\}$ of zero-one vectors are linearly independent. Sample at least 1000 random triples to gather a statistic.

Explain your result. Why is it different or the same as the previous part? (Hint: it may be useful to think of random zero-one vectors as lying on the vertices of the unit cube.)

4. The power of the right definition. Our definition of projection was slightly complicated and hard to compute with, especially compared with the textbook's. However, the benefit of

our definition is that it works, almost unchanged, in many more contexts. In this problem we will be projecting onto subspaces.

Definition If $\vec{v}$ is a vector and U is a subspace, *the projection of $\vec{v}$ onto U* is the vector $\vec{w} \in U$ such that $\vec{v} - \vec{w}$ is orthogonal to every vector in U.

For the remainder of this problem, let

$$\vec{a}' = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \qquad \vec{b}' = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \vec{c}' = \begin{bmatrix} -1 \\ 6 \\ 1 \end{bmatrix}$$

and let $\vec{a} = \vec{a}'/\|\vec{a}'\|$, $\vec{b} = \vec{b}'/\|\vec{b}'\|$, and $\vec{c} = \vec{c}'/\|\vec{c}'\|$.

(a) Let's make life a little easier first. Show that if $\mathcal{B}$ is a basis for a subspace U, then $\vec{w}$ is orthogonal to every vector in U if and only if $\vec{w}$ is orthogonal to every vector in $\mathcal{B}$.

(b) Let $U = \text{span}\{\vec{a}, \vec{b}\}$. Analytically find the exact vector $\vec{d} = \text{proj}_U \vec{c}$. (Hint, $U = \text{span}\{\vec{a}, \vec{b}\} = \text{span}\{\vec{a}', \vec{b}'\}$, so you can avoid some square roots!)

(c) Imagine you're an engineer who forgot how to do math but still wanted to find this projection. "Ah ha," you think, "I could use Matlab to search for this vector!"

In Matlab, '`'`' transposes a matrix. So, if you want to compute $\vec{a} \cdot \vec{b}$, and $\vec{a}$ and $\vec{b}$ are column vectors, you could do `a'*b` in Matlab.

You decide to try an brute force the problem. Using the command `r = randn(1)*a + randn(1)*b` you can generate a random vector in U (`randn(1)` produces a normally distributed random number, so it is equally likely to be positive or negative, unlike `rand` which always produces a number in $[0, 1]$). You decide, since computers are fast, you will generate 1000 random vectors in U and see which one most closely fits what the projection should fit.

Since it is unlikely any of your random vectors will be *exactly* the projection, you will need a *score* function. That is, you will need a function $s : \mathbb{R}^3 \to [0, \infty)$ such that $s(\vec{v}) = 0$ if $\vec{v} = \vec{d} = \text{proj}_U \vec{c}$ and $s(\vec{v}) < s(\vec{w})$ if $\vec{v}$ is "closer to" the correct projection than $\vec{w}$. Your score function doesn't need to be complicated, but it shouldn't involve $\vec{d}$. (If it could involve $\vec{d}$, a wonderful score function would be $s(\vec{v}) = \|\vec{d} - \vec{v}\|$.)

With score function in hand, your guess for $\text{proj}_U \vec{c}$ is the random vector with the lowest score.

Write Matlab code to execute this procedure. If your estimate for $\text{proj}_U \vec{c}$ is $\vec{p}$, call $e = \|\vec{p} - \vec{d}\|$ the *error* in your estimate. Repeat the process of picking the best estimate from 1000 random vectors at least 100 times and report the **average** error.

(d) In the procedure outlined above, we could use any basis for U to generate random vectors. Find a unit vector $\vec{B}$ that is orthogonal to $\vec{a}$ such that $U = \text{span}\{\vec{a}, \vec{B}\}$ (Hint, think geometrically and use projections). Repeat the guessing procedure above using `r = randn(1)*a + randn(1)*B` to generate your random vectors. What is the average error using this new basis? Why is it smaller? Explain.

(e) (Optional) We can dramatically improve the our guessing from above by using an *adaptive algorithm*. That is, instead of using `r = randn(1)*a + randn(1)*b` to generate a new random vector each time, we could use our previous best guess and see if we improve by perturbing it in a random direction. Thus, our random vectors might look like

```
r = bestGuessSoFar + (randn(1)*a + randn(1)*b)/10
```

where we divide by 10 so that we don't guess something too far from our previous good guess (a more advanced adaptive algorithm would use the score function and adjust the factor of 10 based on that). How much improvement do you see with an adaptive algorithm?