Damian Franco
dfranco24@unm.edu
101789677
CS-575

Homework 6

1) For the first problem we were asked to show that the Classical Gram-Schmidt (CGS) and Modified Gram-Schmidt (MGS) are mathematically equivalent. That is, prove that $q_i^{(CGS)} = q_j^{(MGS)}$ and $r_{i,j}^{(CGS)} = r_{i,j}^{(MGS)}$. To show this, we will prove by induction. Induction involves first proving a base case which I did with $n = 1$. Proving that $q_1^{(CGS)} = q_1^{(MGS)}$ and $r_{1,1}^{(CGS)} = r_{1,1}^{(MGS)}$ was fairly simple for the base case due to the q vectors being the same for both CGS and MGS. The r elements are different in CGS and MGS so there is some algebra done to get $r_{1,1}^{(CGS)}$ to equal to $r_{1,1}^{(MGS)}$ but the base case was able to be proved with some norm and Gram-Schmidt rules applied in certain steps. Next, we had to state an inductive hypothesis that states how we are going to prove that all elements of q *and* r elements are mathematically equal. Lastly, we need to prove the inductive case/step. This was done by proving that $q_{k+1}^{(CGS)} = q_{k+1}^{(MGS)}$ and $r_{k+1,h+1}^{(CGS)} = r_{k+1,h+1}^{(MGS)}$ where $k+1$ and $h+1$ are the inductive step to prove that all elements are mathematically equal. Once again, the proof for $q_{k+1}^{(CGS)} = q_{k+1}^{(MGS)}$ is simple for the inductive case due to the q vectors being the same for both CGS and MGS. Now, I need to prove $r_{k+1,h+1}^{(CGS)} = r_{k+1,h+1}^{(MGS)}$. To do this, I used some rules that were mentioned in class and some that were not. The one that stands out to me was the q vector is equal to a much larger equation according to the textbook and other online references. I used that rule, alongside some algebra to get $r_{k+1,h+1}^{(CGS)} = r_{k+1,h+1}^{(MGS)}$ which I found was true. This problem gave me a great refresher on proof by induction and finding that CGS and MGS are indeed mathematically equivalent was very fun. You can find my hand-written work on the next two pages.

1) Show that Classical Gram-Schmidt and Modified Gram-Schmidt (version 2) are mathematically equivalent in exact arithmetic

Base case:

Let $n = 1$ & $m = 1$

$$q_1^{(CGS)} = a_1 / r_{11}$$
$$= a_1 / \|a_1\|_2$$
$$= q_1^{(MGS)} \quad \checkmark \text{DONE}$$

$$r_{11}^{(CGS)} = q_1^T a_1$$
$$= q_1^T a_1$$
$$= q_1^T r_{11} q_1$$
$$= q_1^T \|a_1\|_2$$
$$= q_1^T q_1$$
$$= r_{11}^{(MGS)} \quad \checkmark \text{DONE}$$

Base case proves $q_1^{(CGS)} = q_1^{(MGS)}$
and $r_{11}^{(CGS)} = r_{11}^{(MGS)}$

Inductive hypothesis:

Assume that $q_i^{(CGS)} = q_i^{(MGS)}$ and $r_{ij}^{(CGS)} = r_{ij}^{(MGS)}$ for some integers $i \in \mathbb{N}$ and $j \in \mathbb{N}$ where $n = k$ and $m = h$ where $k \in \mathbb{N}$ and $h \in \mathbb{N}$ are the dimensions of $A_{k,h}$ (i.e. $i = 1 \ldots k$ and $j = 1 \ldots h$)

Inductive case:

Show the case holds for $k+1$ and $h+1$:

$$q_{k+1}^{(CGS)} = v_{k+1} / r_{k+1,k+1}$$

$$= \frac{a_{k+1} - \sum_{i=1}^{k} r_{i,k+1} \cdot q_i}{r_{k+1,k+1}}$$

$$= q_{k+1}^{(MGS)}$$

Therefore $q_{k+1}^{(CGS)} = q_{k+1}^{(MGS)}$ ✓DONE

$$r_{k+1,h+1}^{(CGS)} = q_{k+1} a_{h+1}$$

$$= \left(\frac{a_{k+1} - \sum_{i=1}^{k} r_{i,j} \cdot}{r_{k+1,k+1}} \right) a_{h+1}$$

$$= \left(\frac{a_{k+1} - \sum_{i=1}^{k} r_{i,j} q_i}{r_{k+1,k+1}} \right) a_{h+1}$$

$$= \left(\frac{a_{h+1} \cdot a_{k+1} - \sum_{i=1}^{k} r_{i,j} \cdot q_i}{r_{k+1,k+1}} \right)$$

$$= \frac{a_{h+1}}{r_{k+1,k+1}} \cdot \frac{a_{k+1} - \sum_{i=1}^{k} r_{i,j} q_i}{r_{k+1,k+1}}$$

$$= \frac{a_{k+1} - \sum_{i=1}^{k} r_{i,j} q_i}{r_{k+1,k+1}} \cdot \frac{a_{h+1}}{r_{h+1,h+1}}$$

$$= q_{k+1}^{T} q_{h+1}$$

$$= r_{k+1,h+1}^{(MGS)}$$

Therefore $r_{k+1,h+1}^{(CGS)} = r_{k+1,h+1}^{(MGS)}$

2) The next question asks us to show/prove two characteristics of the reduced QR decompositions. In lecture, we spoke about $\widehat{Q}\,\widehat{R}$ and how it relates to the linear system $Ax = b$ where $A = \widehat{Q}\,\widehat{R}$. We know that the rank of the matrix A is n. I will be using the Q and R hat letters to represent my reduced QR decomposition in both of these proofs.
    a) This first proof specifically asks us to first show that we have a unique solution y and that the solution y minimizes to the euclidean norm of $Qy-b$. To show this, I started with the 2-norm of the system $Qy-b$. Next, I squared the norm and used matrix norm rules and algebra to get all the terms out of the squared 2-norm. This allowed me to find where I can eliminate variables with the use of rules like $\widehat{Q}^T\widehat{Q} = I$, but not the opposite. I then took the derivative of the entire function in respect to y and found more terms I can eliminate. With some simple algebra, I was able to cancel all terms and find that it is indeed bound by the 2-norm of our system (= 0). I then showed for an arbitrary variable z, that we indeed do have a unique solution y by showing that the solution for z, as well as $y+z$ will always produce the same unique solution. My work is shown on the next page.

2) Let $A \in \mathbb{R}^{m \times n}$, $m > n$, have rank $= n$.
Let $A = \hat{Q}\hat{R}$ be the reduced QR decomposition of A (so $\hat{Q} \in \mathbb{R}^{m \times n}$ and $\hat{R} \in \mathbb{R}^{n \times n}$

a) Show the unique $y \in \mathbb{R}^n$ that minimizes $\|\hat{Q}y - b\|_2$ is equal to $\hat{Q}^T b$ $(y = \hat{Q}^T b)$

$$\|\hat{Q}y - b\|_2 = \|\hat{Q}y - b\|_2^2$$
$$= (\hat{Q}y - b)^T(\hat{Q}y - b)$$
$$= y^T\hat{Q}^T\hat{Q}y - 2b^T\hat{Q}^Ty + b^Tb$$
$$= y^Ty - 2b^T\hat{Q}^Ty + b^Tb$$
$$df/dy = df/dy(y^Ty) - df/dy(2b^T\hat{Q}^Ty) + df/dy(b^Tb)$$
$$= ((Q^TQ) + (Q^TQ)^T)y - (2b^T\hat{Q})^T + 0$$
$$= 2\hat{Q}^T\hat{Q}y - 2\hat{Q}^Tb$$
$$= 2Iy - 2\hat{Q}^Tb$$
$$= 2y - 2\hat{Q}^Tb$$
$$= 2y - 2y$$
$$= 0 \quad \checkmark \text{DONE}$$

Let z be a solution for the system $\hat{Q}z = b$ which means $\hat{Q}(y+z) = b$

$$\|\hat{Q}(y+z) - b\|_2 = \|\hat{Q}(y+z) - b\|_2^2$$

so we can say

$$\|\hat{Q}y - b\|_2 = \|\hat{Q}z - b\|_2$$

and

$$\|\hat{Q}y - b\|_2 = 0 \text{ and } \|\hat{Q}z - b\|_2 = 0$$

so we can say

$$\|\hat{Q}(y+z) - b\|_2 = 0$$

Since $y = z$ then we can say y is a unique solution.

b) The next part of the question, we must show the least-squares solution *x* of *Ax* = *b* is the solution of $\widehat{R}x = \widehat{Q}^{T}b$. To show this, I first started with *Ax* = *b* and used *A* = $\widehat{Q}\widehat{R}$ to change it to our reduced QR form. Next, I did the same steps that I did for the first question by first taking the euclidean norm of the system then squaring it. I used the same technique on eliminating and expanding terms to reduce my entire term. The next step was to take the derivative which set me up for success because I could see that I had the terms $\widehat{R}x$ and $\widehat{Q}^{T}b$ alongside two terms that could cancel each other out when put on either side of the equation. I canceled those terms out and found that indeed the least-squares solution of *Ax* = *b* is the same solution of $\widehat{R}x = \widehat{Q}^{T}b$. My work is shown on the next page.

b) Show that the least-squares solution x of $Ax = b$ is a solution of $\hat{R}x = \hat{Q}^T b$

$$Ax = \hat{Q}\hat{R}x$$

$$\Rightarrow \hat{Q}\hat{R}x = b$$

$$= \hat{Q}\hat{R}x - b$$

$$= \| \hat{Q}\hat{R}x - b \|_2$$

$$= \| \hat{Q}\hat{R}x - b \|_2^2$$

$$= (\hat{Q}\hat{R}x - b)^T (\hat{Q}\hat{R} - b)$$

$$= x^T \hat{Q}^T \hat{R}^T x - x^T \hat{Q}^T \hat{R}^T b - b^T \hat{Q}^T \hat{R}^T$$

$$= x^T \hat{Q}^T \hat{R}^T \hat{Q}\hat{R}x - 2b^T \hat{Q}^T \hat{R}x + b^T b$$

$$= x^T Q^T \hat{Q} \hat{R}^T \hat{R}x - 2\hat{R}x\hat{Q}^T b + b^T b$$

$$= x^T I \hat{R}^T \hat{R}x - 2\hat{R}x\hat{Q}^T b + b^T b$$

$$df/dx = df/dx(x^T \hat{R}^T \hat{R}x) - df/dx(2\hat{R}x\hat{Q}^T b) + df/dx(b^T b)$$

$$= 2\hat{R}^T(\hat{R}x) - 2\hat{R}x(\hat{Q}^T b)$$

$$2\hat{R}^T(\hat{Q}^T b) = 2\hat{R}^T(\hat{R}x)$$

$$\hat{Q}^T b = \hat{R}x \quad \checkmark \text{DONE}$$

Therefore the least squares solution x of $Ax = b$ is a solution of $\hat{R}x = \hat{Q}^T b$

3) This problem asks us to implement code for classical Gram-Schmidt, modified Gram-Schmidt Version 1, and modified Gram-Schmidt Version 2. All three approaches were provided for us in pseudocode form by Professor Zeb. To test these algorithms, a simple A matrix and a large matrix with almost linearly dependent columns were used to compare their efficiency and accuracy. After writing the code, the algorithms were tested and compared to determine which method was the most effective in terms of speed and accuracy when dealing with matrices with almost linearly dependent columns. Implementation of Classical Gram-Schmidt to Modified Gram-Schmidt Version 1 to Modified Gram-Schmidt Version 2 is very similar in all three senses. However, they do differ in their implementation of the Gram-Schmidt orthogonalization process so it is worth noting. CGS and MGS Version 2 seem to be very similar besides the dot operation on CGS is using both Q and the A matrix while MGS Version 2 uses only the Q matrix for computation. The big difference between MGS Version 1 and the other two is in the order of updating the columns of the Q matrix and computing the elements of the R matrix.
    a) Verifying all three algorithms and what QR decomposition they produced was very straightforward and I found that all three implementations produced that same exact QR decomposition with the seemingly little to no errors. I would assume that this is because our A matrix is linearly independent and Gram-Schmidt has no issues with linearly independent matrices. On the next pages, you can see my code and output.

Classical Gram-Schmidt function and output:

```
[ ]  # Classical GS
     def qr_cgs(A):
         (m,n) = A.shape
         Q = A.copy()
         R = np.zeros((n,n))
         for j in range(n):
             for i in range(j):
                 R[i,j] = np.dot(Q[:,i],A[:,j])
                 Q[:,j] = Q[:,j] - R[i,j]*Q[:,i]
             R[j,j] = np.linalg.norm(Q[:,j])
             Q[:,j] = Q[:,j]/R[j,j]

         return Q, R
```

```
[ ]  # Lets test the algorithms on a simple 2x2 system
     A = np.array([ [1.,2.], [3.,4.], [5.,6.] ])
```

```
[ ]  Q_cgs, R_cgs = qr_cgs(A)
     print('Q =', Q_cgs)
     print('R =', R_cgs)
```

```
Q = [[ 0.16903085  0.89708523]
 [ 0.50709255  0.27602622]
 [ 0.84515425 -0.34503278]]
R = [[5.91607978 7.43735744]
 [0.         0.82807867]]
```

Modified Gram-Schmidt Version 1 function and output:

```
[ ]  # Modified GS - Version 1.
     def qr_mgs_ver_1(A):
         (m,n) = A.shape
         Q = A.copy()
         R = np.zeros((n,n))
         for j in range(n):
             R[j,j] = np.linalg.norm(Q[:,j])
             Q[:,j] = Q[:,j]/R[j,j]
             for i in range(j+1, n):
                 R[j,i] = np.dot(Q[:,j], Q[:,i])
                 Q[:,i] = Q[:,i] - R[j,i]*Q[:,j]
         return Q, R
```

```
[ ]  Q_ver1, R_ver1 = qr_mgs_ver_1(A)
     print('Q =', Q_ver1)
     print('R =', R_ver1)
```

```
Q = [[ 0.16903085  0.89708523]
 [ 0.50709255  0.27602622]
 [ 0.84515425 -0.34503278]]
R = [[5.91607978 7.43735744]
 [0.         0.82807867]]
```

Modified Gram-Schmidt Version 2 function and output:

```
[10] # Modified GS - Version 2. Notice the similarity to Classical GS.
     def qr_mgs_ver_2(A):
         (m,n) = A.shape
         Q = A.copy()
         R = np.zeros((n,n))
         for j in range(n):
             for i in range(j-1):
                 R[i,j] = np.dot(Q[:,i], Q[:,j])
                 Q[:,j] = Q[:,j] - R[i,j]*Q[:,i]
             R[j,j] = np.linalg.norm(Q[:,j])
             Q[:,j] = Q[:,j]/R[j,j]
         return Q, R
```

```
[11] Q_ver2, R_ver2 = qr_mgs_ver_2(A)
     print('Q =', Q_ver2)
     print('R =', R_ver2)
```

```
Q = [[ 0.16903085  0.89708523]
 [ 0.50709255  0.27602622]
 [ 0.84515425 -0.34503278]]
R = [[5.91607978 7.43735744]
 [0.         0.82807867]]
```

b) Lastly, we were told to test our three algorithms on a matrix with nearly linearly dependent columns. We are given the python code for this part where it computes the euclidean norm of A-QR and Q^TQ-I where Q and R are the outputs of our functions. We are computing both norms of these functions because it is a way to check the accuracy of the function (||A-QR||) and orthogonality of the columns of the Q matrix ($||Q^TQ-I||$). Both are important to view how reliable each approach can be, even in an extreme circumstance such as nearly dependent columns. I found that the error itself was very low for every algorithm, but the orthogonality of the columns was very high for CGS and MGS Version 2. This tells me that those algorithms have computed a Q matrix that may not be very close to an orthogonal matrix. This can then lead to numerical errors in the algorithm which implies to me that the best algorithm that we can use is MGS Version 1 which is very surprising and not surprising at the same time. Below and on the next page you can see my results and code.

Results of error and orthogonality calculations:

Version	||A-QR||	||QTQ-I||
CGS	7.459892324685454e-14	0.00043386787336402015
MGS Ver.1	7.451390748880851e-14	4.5864016513518555e-10
MGS Ver.2	7.394211914570254e-14	3.3716815787946195

Code for calculating error and orthogonality calculations:

```
#Lets test the algorithms on a matrix with nearly linearly dependent columns
m = 200; n = 150
a = np.random.normal(loc=0, scale=1, size=(m, 1))
epsi = 1e-5
b=a@np.ones((1,n))

#The matrix A below has nearly linearly dependent columns
A = np.multiply ((a@np.ones((1,n))), np.ones((m,n))) + epsi*np.random.normal(loc=0, scale=1, size=(m, n))

# Empty lists for norms calculated
normAs = []
normQs = []

# Classical Gram-Schmidt
Q,R = qr_cgs(A)
norm_of_A_minus_QR = np.linalg.norm(A-Q@R)
norm_of_QTQ_minus_I = np.linalg.norm(Q.T@Q - np.identity(n))
normAs.append(norm_of_A_minus_QR)
normQs.append(norm_of_QTQ_minus_I)
print('CGS: ||A-QR|| = ', norm_of_A_minus_QR, '&& ||QTQ-I|| = ',norm_of_QTQ_minus_I)

# Modified Gram-Schmidt Version 1
Q,R = qr_mgs_ver_1(A)
norm_of_A_minus_QR = np.linalg.norm(A-Q@R)
norm_of_QTQ_minus_I = np.linalg.norm(Q.T@Q - np.identity(n))
normAs.append(norm_of_A_minus_QR)
normQs.append(norm_of_QTQ_minus_I)
print('MGS-ver1: ||A-QR||  = ', norm_of_A_minus_QR, '&& ||QTQ-I|| = ',norm_of_QTQ_minus_I)

# Modified Gram-Schmidt Version 2
Q,R = qr_mgs_ver_2(A)
norm_of_A_minus_QR = np.linalg.norm(A-Q@R)
norm_of_QTQ_minus_I = np.linalg.norm(Q.T@Q - np.identity(n))
normAs.append(norm_of_A_minus_QR)
normQs.append(norm_of_QTQ_minus_I)
print('MGS-ver2: ||A-QR||  = ', norm_of_A_minus_QR, '&& ||QTQ-I|| = ',norm_of_QTQ_minus_I)
```