CMSE823 HW06

Anna Stephens, steph496@msu.edu
Due March 3

In [1]:
```
# LIBRARIES
import numpy as np
```

Question 1

$$A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 7 \\ 4 & 2 & 3 \\ 4 & 2 & 2 \end{bmatrix}$$

(a) Implement the following three algorithms to find the reduced QR decomposition of A_1:

- classical Gram-Schmidt
- modified Gram-Schmidt
- Householder transformation algorithm

(b) Use the built-in function to find the reduced QR decomposition of A_1. Compare this result with those in (a) and comment on the differences that you observe.

In [2]:
```
# Declare A as A_1
A=[[1, 2, 3],
   [4, 5, 6],
   [7, 8, 7],
   [4, 2, 3],
   [4, 2, 2]]
A=np.array(A)
```

Classical Gram-Schmidt

In [3]:
```
# implementation of classical Gram-Schmidt following Algorithm 7.1
def classicalGS(matrix):
    n=matrix.shape[1]                   # n = number of columns of matrix
    v=np.zeros(matrix.shape, float) # setup v as empty mxn
    q=np.zeros(matrix.shape, float) # setup q as empty mxn
    r=np.zeros((n,n), float)        # setup r as empty nxn
    for j in range(n):
        v[:,j]=matrix[:,j].copy()
        for i in range(j):
            r[i,j]=np.dot(q[:,i],matrix[:,j])
            v[:,j]=v[:,j]-(r[i,j]*q[:,i])
        r[j,j]=np.linalg.norm(v[:,j],2)
```

```
        q[:,j]=v[:,j]/r[j,j]
    return(q,r)
```

In [4]:

```
Q,R=classicalGS(A)
print(Q)
print(R)
```

```
[[ 0.10101525  0.31617307  0.5419969 ]
 [ 0.40406102  0.3533699   0.51618752]
 [ 0.70710678  0.39056673 -0.52479065]
 [ 0.40406102 -0.55795248  0.38714064]
 [ 0.40406102 -0.55795248 -0.12044376]]
[[9.89949494 9.49543392 9.69746443]
 [0.         3.29191961 3.01294337]
 [0.         0.         1.97011572]]
```

In [5]:

```
# Check that A=QR (with rounding)
QR=np.matmul(Q,R)
A==np.around(QR).astype(int)
```

Out[5]:

```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

Modified Gram-Schmidt

In [6]:

```
# implementation of modified Gram-Schmidt following Algorithm 8.1
def modifiedGS(matrix):
    n=matrix.shape[1]                # n = number of columns of matrix
    v=np.zeros(matrix.shape, float) # setup v as empty mxn
    q=np.zeros(matrix.shape, float) # setup q as empty mxn
    r=np.zeros((n,n), float)        # setup r as empty nxn

    for i in range(n):
        v[:,i]=matrix[:,i].copy()

    for i in range(n):
        r[i,i]=np.linalg.norm(v[:,i],2)
        q[:,i]=v[:,i]/r[i,i]
        for j in range(i,n):
            r[i,j]=np.dot(q[:,i],matrix[:,j])
            v[:,j]=v[:,j]-(r[i,j]*q[:,i])
    return(q,r)
```

In [7]:

```
Q,R=modifiedGS(A)
print(Q)
print(R)
```

```
[[ 0.10101525  0.31617307  0.5419969 ]
 [ 0.40406102  0.3533699   0.51618752]
 [ 0.70710678  0.39056673 -0.52479065]
 [ 0.40406102 -0.55795248  0.38714064]
 [ 0.40406102 -0.55795248 -0.12044376]]
```

```
[[9.89949494 9.49543392 9.69746443]
 [0.         3.29191961 3.01294337]
 [0.         0.         1.97011572]]
```

In [8]:

```
# Check that A=QR (with rounding)
QR=np.matmul(Q,R)
A==np.around(QR).astype(int)
```

Out[8]:

```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

Householder

In [9]:

```
def reducedHouseholder(matrix):
    Q,R=householder(matrix) # get full QR from householder algorithm
    m=matrix.shape[0]
    n=matrix.shape[1]
    Q=Q[:m,:n]
    R=R[:n,:n]
    return(Q,R)

# implementation of Householder using Algorithm 10.1
#  Notes:
#   - Alg 10.1 only returned R, made modifications derived from class notes
#     to get Q
#   - This returns full QR and question asked for reduced
def householder(matrix):
    A=matrix.copy().astype(float)
    n=matrix.shape[1]
    v=np.zeros(matrix.shape, float)
    q_star=np.identity(A.shape[0])
    for k in range(n):
        x=A[k:,k]
        v[k:,k]=getSign(x[0])*np.linalg.norm(x,2)*getE1(x.shape[0])+x
        v[k:,k]=v[k:,k]/np.linalg.norm(v[k:,k],2)
        q_k=np.identity(A.shape[0])
        q_k[k:,k:]=getHV(v[k:,k])
        q_star=np.matmul(q_k,q_star)
        # print(v[k:,k][np.newaxis].T.shape)
        # print(v[k:,k][np.newaxis].shape)
        A[k:,k:]=A[k:,k:]-2*np.matmul(v[k:,k][np.newaxis].T,np.matmul(v[k:,k][np

    Q=q_star.T
    R=A.round(8)
    return(Q,R)

# function that returns an e1 vector of length m
def getE1(m):
    e1=np.zeros((m))
    e1[0]=1
    return(e1)

# a sign function that checks and corrects for x=0 according to notes in book
def getSign(x):
    sign=np.sign(x)
```

```
    if sign==0: sign=1
    return(sign)

# calculate some H_v
def getHV(v):
    I=np.identity(v.shape[0])
    return(I-2*np.outer(v,v))
```

In [10]:

```
Q,R=reducedHouseholder(A)
print(Q)
print(R)
```

```
[[-0.10101525 -0.31617307  0.5419969 ]
 [-0.40406102 -0.3533699   0.51618752]
 [-0.70710678 -0.39056673 -0.52479065]
 [-0.40406102  0.55795248  0.38714064]
 [-0.40406102  0.55795248 -0.12044376]]
[[-9.89949494 -9.49543392 -9.69746443]
 [-0.         -3.29191961 -3.01294337]
 [-0.          0.          1.97011572]]
```

In [11]:

```
# Check that A=QR (with rounding)
QR=np.matmul(Q,R)
A==np.around(QR).astype(int)
```

Out[11]:

```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

Built-In Function: numpy.linalg.qr

In [12]:

```
Q,R=np.linalg.qr(A, mode='reduced')
print(Q)
print(R)
```

```
[[-0.10101525 -0.31617307  0.5419969 ]
 [-0.40406102 -0.3533699   0.51618752]
 [-0.70710678 -0.39056673 -0.52479065]
 [-0.40406102  0.55795248  0.38714064]
 [-0.40406102  0.55795248 -0.12044376]]
[[-9.89949494 -9.49543392 -9.69746443]
 [ 0.         -3.29191961 -3.01294337]
 [ 0.          0.          1.97011572]]
```

Comparison

The Q and R found by the built in function are identical to those found by my implementation of householder. They are, however, the negative versions of the ones found by the two gram-schmidt methods. All work as factorizations, however as $A = QR = (-Q)(-R)$

Question 2

$$A_2 = \begin{bmatrix} 0.70000 & 0.70711 \\ 0.70001 & 0.70711 \end{bmatrix}$$

(a) Implement the above three algorithms to find the reduced QR decomposition of A_2. Check the orthogonality of Q by computing the Frobenius norm of the matrix $Q^*Q - I$ where I is the $n \times n$ identity matrix.

(b) Use the built-in function to find the reduced QR decomposition of A_2. Again, check the orthogonality of Q by computing the Frobenius norm of the matrix $Q^*Q - I$. Compare this result with those in (a) and comment on the differences that you observe.

In [13]:

```
# Declare A as A_2
A=[[0.70000, 0.70711],
   [0.70001, 0.70711]]
A=np.array(A)
```

In [14]:

```
# if Q is orthogonal then the frobenius norm of (Q^* Q - I) should be zero
#  or very close to it
#
#  here we return both the calculated value and whether or not it is close to
#   zero with meaning the first 10 decimal places round to zero
def getOrthogonalCheck(Q):
    x=np.matmul(Q.T,Q)-np.identity(Q.shape[1]) # Q^* Q - I
    x=np.linalg.norm(x,ord='fro') # frobenius norm
    return(x,x.round(10)==0)
```

Classical Gram-Schmidt

Methods were already implemented as functions above. Can simply re-use here.

In [15]:

```
Q,R=classicalGS(A)
print(Q)
print(R)
```

```
[[ 0.70710173  0.70711183]
 [ 0.70711183 -0.70710173]]
[[9.89956565e-01 1.00000455e+00]
 [0.00000000e+00 7.14283864e-06]]
```

In [16]:

```
# Check that A=QR (with rounding)
QR=np.matmul(Q,R)
A==QR.round(5)
```

Out[16]:

```
array([[ True,  True],
       [ True,  True]])
```

In [17]:

```
getOrthogonalCheck(Q)
```

Out[17]:

```
(3.254726094493924e-11, True)
```

Modified Gram-Schmidt

Methods were already implemented as functions above. Can simply re-use here.

In [18]:
```
Q,R=modifiedGS(A)
print(Q)
print(R)
```

```
[[ 0.70710173  0.70711183]
 [ 0.70711183 -0.70710173]]
[[9.89956565e-01 1.00000455e+00]
 [0.00000000e+00 7.14286165e-06]]
```

In [19]:
```
# Check that A=QR (with rounding)
QR=np.matmul(Q,R)
A==QR.round(5)
```

Out[19]:
```
array([[ True,  True],
       [ True,  True]])
```

In [20]:
```
getOrthogonalCheck(Q)
```

Out[20]:
```
(3.254726094493924e-11, True)
```

Householder

Methods were already implemented as functions above. Can simply re-use here.

In [21]:
```
Q,R=reducedHouseholder(A)
print(Q)
print(R)
```

```
[[-0.70710173  0.70711183]
 [-0.70711183 -0.70710173]]
[[-9.89956560e-01 -1.00000455e+00]
 [ 0.00000000e+00  7.14000000e-06]]
```

In [22]:
```
# Check that A=QR (with rounding)
QR=np.matmul(Q,R)
A==QR.round(5)
```

Out[22]:
```
array([[ True,  True],
       [ True,  True]])
```

In [23]:
```
getOrthogonalCheck(Q)
```

Out[23]:
```
(1.111052298468932e-16, True)
```

Built-In Function: numpy.linalg.qr

In [24]:
```
Q,R=np.linalg.qr(A, mode='reduced')
print(Q)
print(R)
```

```
[[-0.70710173 -0.70711183]
 [-0.70711183  0.70710173]]
[[-9.89956565e-01 -1.00000455e+00]
 [ 0.00000000e+00 -7.14283864e-06]]
```

In [25]:

```
getOrthogonalCheck(Q)
```

Out[25]:

```
(2.3411870786352597e-16, True)
```

Comparison

Once again the magnitudes of the values for Q and R remained roughly the same accross the various methods. There were some small differences that weren't horribly significant like $7.14000000e-06$ vs $7.14283864e-06$. The major difference was once again in the sign of the values, although every QR pair still equaled A (with some rounding).

$$Q_{CGS} = \begin{bmatrix} 0.70710173 & 0.70711183 \\ 0.70711183 & -0.70710173 \end{bmatrix} R_{CGS} = \begin{bmatrix} 9.89956565e-01 & 1.00000455e+00 \\ 0.00000000e+00 & 7.14283864e-06 \end{bmatrix}$$

$$Q_{MGS} = \begin{bmatrix} 0.70710173 & 0.70711183 \\ 0.70711183 & -0.70710173 \end{bmatrix} R_{MGS} = \begin{bmatrix} 9.89956565e-01 & 1.00000455e+00 \\ 0.00000000e+00 & 7.14286165e-06 \end{bmatrix}$$

$$Q_{H} = \begin{bmatrix} -0.70710173 & 0.70711183 \\ -0.70711183 & -0.70710173 \end{bmatrix} R_{CGS} = \begin{bmatrix} -9.89956560e-01 & -1.00000455e+00 \\ 0.00000000e+00 & 7.14000000e-06 \end{bmatrix}$$

$$Q_{np} = \begin{bmatrix} -0.70710173 & -0.70711183 \\ -0.70711183 & 0.70710173 \end{bmatrix} R_{CGS} = \begin{bmatrix} -9.89956565e-01 & -1.00000455e+00 \\ 0.00000000e+00 & -7.14283864e-06 \end{bmatrix}$$

Every Q value was orthogonal by our test, which checked that the frobineous norm of $Q^*Q - I$ was equal to zero after rounding to the first 10 decimal places.

In [ ]: