QRDecomposition

February 19, 2020

1 QR Decomposition

Author : Satrya Budi Pratama

How to make **A** which is not orthogonal to orthogonal using Gram-Schmidt to get **Q**. Using that method we can also calculate the **R** for QR Decomposition.

$A = QR$

where Q is $(n \times n)$ orthogonal $(Q^T Q = I)$ and R is $(n \times n)$ upper triangular.

For example given $A = \begin{bmatrix} 1 & 2 & 4 \\ 0 & 0 & 5 \\ 0 & 3 & 6 \end{bmatrix}$, where $a_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a_2 = \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix}, a_3 = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$ our goals it to calculate q_1, q_2 and q_3

using this formula $q_n = \frac{w_n}{||w_n||}$ where $w_n = a_n - \sum_{i=1}^{n-1} (a_n^\intercal q_i) q_i$ and $||w_n|| = \sqrt{w_n^\intercal w_n}$

The pseudocode are :

1. Generate A
2. Call QR Decomposition
   - generate zeros matrix as Q, and R
   - iterate over column
     - for first iterate update first Q = current cols / euclidean of first cols and update R for its norm
     - and next iterate we do iteration as many of q then calculate w using the formula, update Q and R.
   - return Q and R
3. Prove $A = QR$

[1]:
```
import numpy as np
A = np.array([[1. , 2. , 4.],
              [0., 0., 5.],
              [0., 3., 6.]])
A
```

[1]:
```
array([[1., 2., 4.],
       [0., 0., 5.],
       [0., 3., 6.]])
```

[2]:
```
def QRDecomposition(A):
    rows,cols = A.shape
    Q = np.zeros((rows,cols))
    R = np.zeros((rows,cols))

    # iterate over cols
    for i in range(cols):
        print('\n --- Step {} : ---- \n'.format(i+1))
        if i == 0:
            # first cols
            w_norm =  np.sqrt(np.transpose(A[:,i]).dot(A[:,i])) # norm␣
↪euclidean of first cols
            Q[:,i] = A[:,i] / w_norm # calculate Q
            R[i,i] = w_norm # put norm to R
        else:
            idx_rows = 0
            print('-- inner loop --')
            w = np.zeros(rows)
            #print(w)
            for j in range(i):
                # inner loop for calculate sigma
                w_product = np.transpose(A[:,i]).dot(Q[:,idx_rows]) # calculate␣
↪dot product An^T with Qn.
                R[idx_rows,i] = w_product # put w_product to R
                w_sigma = w_product * Q[:,idx_rows] # dot q, (An^T*Qi)Qi
                w =  w + w_sigma # summation the result
                idx_rows += 1
                print('w_product : {} \t w_sigma: {} \n w : {} '.
↪format(w_product,w_sigma, w))

            w = A[:,i].transpose() - w # An - summation
            w = w.transpose() # change the shape
            w_norm = np.sqrt(np.transpose(w).dot(w)) # calculate norm of w
            R[i,i] = w_norm # put norm to R
            Q[:,i] = w / w_norm # calculate Q
            print('-- outer -- ')
            print('w : {} , w_norm : {}'.format(w,w_norm))

        print('-- Result --')
        print(' Q: {} \n\n R: {}'.format(Q, R))

    return Q, R

Q,R = QRDecomposition(A)
```

```
--- Step 1 : ----
```

```
-- Result --
 Q: [[1. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

 R: [[1. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

 --- Step 2 : ----

-- inner loop --
w_product : 2.0          w_sigma: [2. 0. 0.]
 w : [2. 0. 0.]
-- outer --
w : [0. 0. 3.] , w_norm : 3.0
-- Result --
 Q: [[1. 0. 0.]
 [0. 0. 0.]
 [0. 1. 0.]]

 R: [[1. 2. 0.]
 [0. 3. 0.]
 [0. 0. 0.]]

 --- Step 3 : ----

-- inner loop --
w_product : 4.0          w_sigma: [4. 0. 0.]
 w : [4. 0. 0.]
w_product : 6.0          w_sigma: [0. 0. 6.]
 w : [4. 0. 6.]
-- outer --
w : [0. 5. 0.] , w_norm : 5.0
-- Result --
 Q: [[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]

 R: [[1. 2. 4.]
 [0. 3. 6.]
 [0. 0. 5.]]
```

Testing $A = Q * R$

[3]:
```
# prove
# A = Q*R
```

```
A_qr = Q.dot(R)
A_qr
```

[3]:
```
array([[1., 2., 4.],
       [0., 0., 5.],
       [0., 3., 6.]])
```

[4]:
```
A_qr == A
```

[4]:
```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

[5]:
```
# try with A random
n = 5
A = np.random.rand(n,n)
A
```

[5]:
```
array([[0.33018593, 0.33826849, 0.16869569, 0.0850701 , 0.2712306 ],
       [0.0911052 , 0.90746082, 0.31563168, 0.28123423, 0.84223027],
       [0.83341689, 0.04291006, 0.33920302, 0.22461525, 0.85921653],
       [0.20239956, 0.74883359, 0.31364049, 0.06406597, 0.75625559],
       [0.63979366, 0.15124491, 0.89162676, 0.86078162, 0.75762148]])
```

[6]:
```
Q, R = QRDecomposition(A)
```

```
 --- Step 1 : ----

-- Result --
 Q: [[0.29389559 0.         0.         0.         0.        ]
 [0.08109194 0.         0.         0.         0.        ]
 [0.74181705 0.         0.         0.         0.        ]
 [0.18015407 0.         0.         0.         0.        ]
 [0.56947472 0.         0.         0.         0.        ]]

 R: [[1.12348036 0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]]

 --- Step 2 : ----

-- inner loop --
w_product : 0.4258703533405222   w_sigma: [0.12516142 0.03453465 0.31591789
0.07672228 0.2425224 ]
```

```
 w : [0.12516142 0.03453465 0.31591789 0.07672228 0.2425224 ]
-- outer --
w : [ 0.21310707  0.87292617 -0.27300783  0.67211131 -0.09127749] , w_norm :
1.158452931138443
-- Result --
 Q: [[ 0.29389559  0.18395833  0.          0.          0.        ]
 [ 0.08109194  0.75352752  0.          0.          0.        ]
 [ 0.74181705 -0.23566588  0.          0.          0.        ]
 [ 0.18015407  0.58018008  0.          0.          0.        ]
 [ 0.56947472 -0.07879258  0.          0.          0.        ]]

 R: [[1.12348036 0.42587035 0.         0.         0.        ]
 [0.         1.15845293 0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]]

 --- Step 3 : ----

-- inner loop --
w_product : 0.8910631941587853   w_sigma: [0.26187955 0.07225804 0.66100587
0.16052866 0.50743796]
 w : [0.26187955 0.07225804 0.66100587 0.16052866 0.50743796]
w_product : 0.3006459511995648   w_sigma: [ 0.05530633  0.226545   -0.07085199
0.17442879 -0.02368867]
 w : [0.31718587 0.29880304 0.59015388 0.33495745 0.48374929]
-- outer --
w : [-0.14849018  0.01682864 -0.25095086 -0.02131696  0.40787747] , w_norm :
0.5021228045471823
-- Result --
 Q: [[ 0.29389559  0.18395833 -0.29572484  0.          0.        ]
 [ 0.08109194  0.75352752  0.03351498  0.          0.        ]
 [ 0.74181705 -0.23566588 -0.49977984  0.          0.        ]
 [ 0.18015407  0.58018008 -0.04245367  0.          0.        ]
 [ 0.56947472 -0.07879258  0.8123062   0.          0.        ]]

 R: [[1.12348036 0.42587035 0.89106319 0.         0.        ]
 [0.         1.15845293 0.30064595 0.         0.        ]
 [0.         0.         0.5021228  0.         0.        ]
 [0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]]

 --- Step 4 : ----

-- inner loop --
w_product : 0.7161660918528925   w_sigma: [0.21047806 0.0580753  0.53126422
0.12902023 0.40783848]
 w : [0.21047806 0.0580753  0.53126422 0.12902023 0.40783848]
```

```
w_product : 0.14397952920791038          w_sigma: [ 0.02648623  0.10849254
-0.03393106  0.08353405 -0.01134452]
 w : [0.23696429 0.16656783 0.49733315 0.21255429 0.39649397]
w_product : 0.568508457767369    w_sigma: [-0.16812207  0.01905355 -0.28412907
-0.02413527  0.46180294]
 w : [0.06884222 0.18562138 0.21320409 0.18841902 0.85829691]
-- outer --
w : [ 0.01622788  0.09561284  0.01141116 -0.12435305  0.00248471] , w_norm :
0.15813041773139175
-- Result --
 Q: [[ 0.29389559  0.18395833 -0.29572484  0.10262339  0.        ]
 [ 0.08109194  0.75352752  0.03351498  0.60464547  0.        ]
 [ 0.74181705 -0.23566588 -0.49977984  0.07216296  0.        ]
 [ 0.18015407  0.58018008 -0.04245367 -0.78639551  0.        ]
 [ 0.56947472 -0.07879258  0.8123062   0.01571307  0.        ]]

 R: [[1.12348036 0.42587035 0.89106319 0.71616609 0.        ]
 [0.         1.15845293 0.30064595 0.14397953 0.        ]
 [0.         0.         0.5021228  0.56850846 0.        ]
 [0.         0.         0.         0.15813042 0.        ]
 [0.         0.         0.         0.         0.        ]]

 --- Step 5 : ----

-- inner loop --
w_product : 1.3530818301963936   w_sigma: [0.39766479 0.10972403 1.00373917
0.2437632  0.77054589]
 w : [0.39766479 0.10972403 1.00373917 0.2437632  0.77054589]
w_product : 0.8611202754345596   w_sigma: [ 0.15841025  0.64887783 -0.20293667
0.49960483 -0.06784989]
 w : [0.55607504 0.75860186 0.8008025  0.74336803 0.70269601]
w_product : 0.1019134031438641   w_sigma: [-0.03013832  0.00341563 -0.05093426
-0.0043266   0.08278489]
 w : [0.52593671 0.76201748 0.74986824 0.73904143 0.7854809 ]
w_product : 0.016277495350849665         w_sigma: [ 0.00167045  0.00984211
0.00117463 -0.01280055  0.00025577]
 w : [0.52760716 0.7718596  0.75104287 0.72624088 0.78573667]
-- outer --
w : [-0.25637657  0.07037068  0.10817366  0.03001471 -0.02811519] , w_norm :
0.28995493373175624
-- Result --
 Q: [[ 0.29389559  0.18395833 -0.29572484  0.10262339 -0.88419453]
 [ 0.08109194  0.75352752  0.03351498  0.60464547  0.24269522]
 [ 0.74181705 -0.23566588 -0.49977984  0.07216296  0.3730706 ]
 [ 0.18015407  0.58018008 -0.04245367 -0.78639551  0.10351509]
 [ 0.56947472 -0.07879258  0.8123062   0.01571307 -0.09696399]]

 R: [[1.12348036 0.42587035 0.89106319 0.71616609 1.35308183]
```

```
[0.         1.15845293 0.30064595 0.14397953 0.86112028]
[0.         0.         0.5021228  0.56850846 0.1019134 ]
[0.         0.         0.         0.15813042 0.0162775 ]
[0.         0.         0.         0.         0.28995493]]
```

[7]:
```
A_qr = Q.dot(R)
A_qr
```

[7]:
```
array([[0.33018593, 0.33826849, 0.16869569, 0.0850701 , 0.2712306 ],
       [0.0911052 , 0.90746082, 0.31563168, 0.28123423, 0.84223027],
       [0.83341689, 0.04291006, 0.33920302, 0.22461525, 0.85921653],
       [0.20239956, 0.74883359, 0.31364049, 0.06406597, 0.75625559],
       [0.63979366, 0.15124491, 0.89162676, 0.86078162, 0.75762148]])
```

[8]:
```
np.round(A) == np.round(A_qr)
```

[8]:
```
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

2 References

http://ee263.stanford.edu/lectures/qr.pdf

http://people.inf.ethz.ch/gander/papers/qrneu.pdf