QR Factorization

Let A be an $(m \times n)$ matrix and assume $m \geq n$. $A = QR$ is a *QR factorization* if and only if

(a) Q is $m \times n$ with $Q^T Q = I_n$ and

(b) R is upper triangular $n \times n$ matrix.

The columns of Q are orthogonal. If the diagonal components of R are not zero, then R is nonsingular and $AR^{-1} = Q$.

There are several methods for finding QR factors: modified Gram-Schmidt, classical Gram-Schmidt, Givens transform and Housholder transforms.

Example 1: Find a QR factorization of the following matrix: $A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

Let $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$, $\mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$, $\mathbf{x}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$.

Let us find an orthonormal basis for $Col(A) = Span\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$. We will use the Gram-Schmidt process:

Choose $\mathbf{v}_1 = \mathbf{x}_1$. Let $\mathbf{v}_2 = \mathbf{x}_2 - \frac{\mathbf{x}_2 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\mathbf{v}_1 = \begin{bmatrix} -\frac{3}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix}$.

Let $\mathbf{v}_3 = \mathbf{x}_3 - \frac{\mathbf{x}_3 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\mathbf{v}_1 - \frac{\mathbf{x}_3 \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2}\mathbf{v}_2 = \begin{bmatrix} 0 \\ -\frac{2}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}$.

$\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ is an orthogonal basis for $Col(A)$. We will normalize these three vectors to obtain the orthonormal set $\{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3\}$:

$\mathbf{u}_1 = \frac{\mathbf{v}_1}{||\mathbf{v}_1||} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$, $\mathbf{u}_2 = \frac{\mathbf{v}_2}{||\mathbf{v}_2||} = \begin{bmatrix} -\frac{3}{\sqrt{12}} \\ \frac{1}{\sqrt{12}} \\ \frac{1}{\sqrt{12}} \\ \frac{1}{\sqrt{12}} \end{bmatrix}$, $\mathbf{u}_3 = \frac{\mathbf{v}_3}{||\mathbf{v}_3||} = \begin{bmatrix} 0 \\ \frac{-2}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \end{bmatrix}$.

Then we use vectors $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ as the columns of Q:

$$Q = \begin{bmatrix} \frac{1}{2} & -\frac{3}{\sqrt{12}} & 0 \\ \frac{1}{2} & \frac{1}{\sqrt{12}} & -\frac{2}{\sqrt{6}} \\ \frac{1}{2} & \frac{1}{\sqrt{12}} & \frac{1}{\sqrt{6}} \\ \frac{1}{2} & \frac{1}{\sqrt{12}} & \frac{1}{\sqrt{6}} \end{bmatrix}.$$

Since $A = QR$, we can use the fact that Q is orthonormal matrix, and $Q^T Q = I$ to obtain matrix R:

$$Q^T A = Q^T(QR) = (Q^T Q)R = IR = R$$

So, $R = Q^T A = \begin{bmatrix} 2 & \frac{3}{2} & 1 \\ 0 & \frac{3}{\sqrt{12}} & \frac{2}{\sqrt{12}} \\ 0 & 0 & \frac{2}{\sqrt{6}} \end{bmatrix}$.

Example 2: Find a QR factorization of the matrix: $A = \begin{bmatrix} 1 & 3 & 5 \\ -1 & -3 & 1 \\ 0 & 2 & 3 \\ 1 & 5 & 2 \\ 1 & 5 & 8 \end{bmatrix}$.

Answer: $Q = \begin{bmatrix} \frac{1}{2} & -\frac{1}{\sqrt{8}} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{\sqrt{8}} & \frac{1}{2} \\ 0 & \frac{2}{\sqrt{8}} & 0 \\ \frac{1}{2} & \frac{1}{\sqrt{8}} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{\sqrt{8}} & \frac{1}{2} \end{bmatrix}$, $R = \begin{bmatrix} 2 & 8 & 7 \\ 0 & \sqrt{8} & \frac{12}{\sqrt{8}} \\ 0 & 0 & 6 \end{bmatrix}$.

Example 3: Use software to find a QR factorization of a matrix $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$.

Solution:

Listing 1: Python QR factorization example

```
import numpy as np
import matplotlib.pyplot as plt
import math

A = [  [1,0],
       [1,0],
       [0,1]  ]

# "full" QR decomposition
Q,R = np.linalg.qr(A, 'complete')
print(Q), print(' ')

# "economy" QR decomposition
Q,R = np.linalg.qr(A) # is the default option in Python
print(Q)

[[-0.70710678  0.          0.70710678]
 [-0.70710678  0.         -0.70710678]
 [-0.         -1.          0.        ]]

[[-0.70710678  0.        ]
 [-0.70710678  0.        ]
 [-0.         -1.        ]]
```

You can see that there are three columns in matrix Q. However, the default in Python is what is called the economy decompositions. That means that it gives you a Q matrix that is (3×2) which does make sense since there is no third column in matrix A. Of cause, we can create a third column because it is possible to have a third vector that is orthogonal to two other vectors that are orthogonal to each other. So there is no problem with adding the third column but the economy QR decomposition leaves Q to have the same number of columns as A.

Example 4: Implement the Gram-Schmidt process. Check your answer against Q from *np.linalg.qr*.

Listing 2: Python Gram-Schmidt example

```
 import numpy as np
import matplotlib.pyplot as plt
import math
# create the matrix
m = 4
n = 4
A = np.random.randn(m,n)

# initialize
Q = np.zeros((m,n))

# the GS algorithm
for i in range(n):

    # initialize
    Q[:,i] = A[:,i]

    # orthogonalize
    a = A[:,i] # convenience
    for j in range(i): # only to earlier cols
        q = Q[:,j] # convenience
        Q[:,i]=Q[:,i]-np.dot(a,q)/np.dot(q,q)*q

    # normalize
    Q[:,i] = Q[:,i] / np.linalg.norm(Q[:,i])

# "real" QR decomposition for comparison
Q2,R = np.linalg.qr(A)

print('Q:'),print(Q), print('')

print('Q2:'),print(Q2), print('')
print('R:'), print(R), print('')

# note the possible sign differences.
# seemingly non-zero columns will be 0 when adding
```

```
print('Q-Q2:'),print( np.round( Q-Q2 ,10) ), print('␣')
print('Q+Q2:'), print( np.round( Q+Q2 ,10) ), print('␣')

# one more check:
print(np.round(Q.T@Q,3))
plt.imshow(Q.T@Q)
plt.show()

Q:
[[ 0.25599717  0.41284207  0.42708449 -0.76264389]
 [-0.03747572 -0.82393519 -0.1530823  -0.54432727]
 [ 0.96359932 -0.11582249 -0.18042206  0.15971654]
 [ 0.06735992 -0.37050961  0.87270414  0.31076156]]

Q2:
[[-0.25599717  0.41284207  0.42708449 -0.76264389]
 [ 0.03747572 -0.82393519 -0.1530823  -0.54432727]
 [-0.96359932 -0.11582249 -0.18042206  0.15971654]
 [-0.06735992 -0.37050961  0.87270414  0.31076156]]

R:
[[-1.70760727 -0.75922249 -0.39284339 -0.87137631]
 [ 0.          0.66446252  1.20618106 -2.00392234]
 [ 0.          0.          1.60524455 -1.07688701]
 [ 0.          0.          0.          1.79080087]]

Q-Q2:
[[ 0.51199435 -0.          0.          0.        ]
 [-0.07495145  0.         -0.          0.        ]
 [ 1.92719865  0.         -0.          0.        ]
 [ 0.13471984  0.         -0.          0.        ]]

Q+Q2:
[[-0.          0.82568414  0.85416899 -1.52528779]
 [ 0.         -1.64787039 -0.3061646  -1.08865454]
 [-0.         -0.23164498 -0.36084412  0.31943308]
 [-0.         -0.74101923  1.74540829  0.62152311]]

[[ 1. -0.  0. -0.]
 [-0.  1. -0. -0.]
 [ 0. -0.  1. -0.]
 [-0. -0. -0.  1.]]
```

QR and Inverses

QR decomposition provides a more numerically stable way to compute the matrix inverse.

Let us start with the QR decomposition formula and inverting both sides of the equation:

$$A = QR$$
$$A^{-1} = (QR)^{-1}$$
$$A^{-1} = R^{-1}Q^{-1}$$
$$A^{-1} = R^{-1}Q^{T}$$

Thus, we can obtain the inverse of A as the inverse of R times the transpose of Q.

An illustration: We consider the matrix:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Step 1: Compute QR Decomposition Since A is already composed of orthogonal columns (after normalization), we obtain:

$$Q = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad R = \begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix}$$

Thus, $A = QR$.

Step 2: Compute A^{-1} Using the property:

$$A^{-1} = R^{-1}Q^{T}$$

we compute:

$$R^{-1} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

Since $Q^T = Q$, we obtain:

$$A^{-1} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$
$$= \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Thus, the inverse of A is:

$$A^{-1} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Example 5:

Listing 3: Python Inverse via QR factorization example

```
import numpy as np
import matplotlib.pyplot as plt
import math

# generate a large matrix and invert using QR and inv()
x=100
A=np.random.randn(x,x)
# explicit inverse
Ai=np.linalg.inv(A)
# inverse via QR
Q,R=np.linalg.qr(A)
plt.imshow(R)
plt.show()

Ai1=np.linalg.inv(R)@Q.T
# solves RA^{-1}=Q^{T} to find A^{-1}
Ai2=np.linalg.solve(R,Q.T)

plt.subplot(1,3,1)
plt.imshow(Ai)
plt.title('A$^{-1}$ from linalg.inv')


plt.subplot(1,3,2)
plt.imshow(Ai1)
plt.title('A$^{-1}$ from $A^{-1}=R^{-1} Q^T$')

plt.subplot(1,3,3)
plt.imshow(Ai2)
plt.title('A$^{-1}$ from solve')

plt.show()
```

Example 6:

Listing 4: Python Inverse via QR factorization example

```
 import numpy as np

# Define a square matrix A
A = np.array([[2, 1], [3, 4]])

# Compute QR decomposition
Q, R = np.linalg.qr(A)
```

```
# Compute A inverse using QR decomposition: A^{-1} = R^{-1} Q^T
A_inv = np.linalg.solve(R, Q.T)

# Verify by multiplying with A (should give identity matrix)
I_check = np.dot(A, A_inv)

# Print results
print("Matrix A:\n", A)
print("Orthogonal Matrix Q:\n", Q)
print("Upper Triangular Matrix R:\n", R)
print("Computed A inverse using QR:\n", A_inv)
print("A * A_inv (should be identity):\n", I_check)

Matrix A:
 [[2 1]
 [3 4]]
Orthogonal Matrix Q:
 [[-0.5547002  -0.83205029]
 [-0.83205029  0.5547002 ]]
Upper Triangular Matrix R:
 [[-3.60555128 -3.88290137]
 [ 0.          1.38675049]]
Computed A inverse using QR:
 [[ 0.8 -0.2]
 [-0.6  0.4]]
A * A_inv (should be identity):
 [[ 1.00000000e+00  5.55111512e-17]
 [-4.44089210e-16  1.00000000e+00]]
```

QR and Computing the Determinant

$$|\det(A)| = |\det(QR)| = |\det(Q)||\det(R)| = |r_{11}r_{22}\dots r_{nn}|$$

since $|\det(Q)| = 1$ (Q has orthonormal columns, $Q^TQ = I$) and the determinant of an upper-triangular matrix is the product of its diagonal elements.

Example 7 Find the absolute value of the determinant of the matrix

$$A = \begin{bmatrix} 8 & 2.6 & 4.0 & 9.8 \\ 4.2 & 6.3 & -1.2 & 5.0 \\ -2.0 & 0.0 & 9.1 & 8.5 \\ 18.7 & 25.0 & -1.0 & 23.5 \end{bmatrix}$$

if we know that in factorization $A = QR$, the upper triangular matrix R is the following

$$R = \begin{bmatrix} 20.8646 & 24.6715 & -0.4764 & 25.0113 \\ 0 & 7.9226 & -1.3135 & 3.4602 \\ 0 & 0 & 9.9648 & 10.3876 \\ 0 & 0 & -0.0000 & -0.3156 \end{bmatrix}$$

$|\det(A)| = |\det(R)| = |r_{11}r_{22}r_{33}r_{44}| = |20.8646{\cdot}7.9226{\cdot}9.9648{\cdot}(-0.3156)| = 519.8238$

Example 8 Use the QR decomposition to compute the absolute value of the determinant of the matrix

$$A = \begin{bmatrix} 1 & 9 & 0 & 5 & 3 & 2 \\ -6 & 3 & 8 & 2 & -8 & 0 \\ 3 & 15 & 23 & 2 & 1 & 7 \\ 3 & 57 & 35 & 1 & 7 & 9 \\ 3 & 5 & 6 & 15 & 55 & 2 \\ 33 & 7 & 5 & 3 & 5 & 7 \end{bmatrix}.$$

Compare the computed value to that obtained from the build in MATLAB (or Python) function for the determinant.

Solution:

Listing 5: Python Determinant via QR factorization example

```
import numpy as np

# Define the matrix A
A = np.array([[1, 9, 0, 5, 3, 2],
              [-6, 3, 8, 2, -8, 0],
              [3, 15, 23, 2, 1, 7],
              [3, 57, 35, 1, 7, 9],
              [3, 5, 6, 15, 55, 2],
              [33, 7, 5, 3, 5, 7]])

# Perform QR decomposition
Q, R = np.linalg.qr(A)

print(np.round(Q,2)), print(' ')
print(np.round(R,2))

# Compute the determinant from QR decomposition (det(A) = product of diagonal el
det_from_qr = np.prod(np.diagonal(R))

# Compute the determinant using the built-in np.linalg.det() function
det_from_np = np.linalg.det(A)

# Display the results
print(f"Determinant from QR decomposition: {np.abs(det_from_qr)}")
print(f"Determinant from np.linalg.det: {np.abs(det_from_np)}")
```

```
# Check if both methods give the same result
if np.isclose(np.abs(det_from_qr), np.abs(det_from_np)):
    print("Both methods give the same determinant.")
else:
    print("The methods yield different results.")

[[-0.03  -0.15   0.37   0.36   0.69  -0.48]
 [ 0.18  -0.09  -0.38   0.09   0.61   0.66]
 [-0.09  -0.23  -0.81  -0.09   0.07  -0.52]
 [-0.09  -0.95   0.19  -0.11  -0.15   0.15]
 [-0.09  -0.06  -0.17   0.92  -0.34   0.1 ]
 [-0.97   0.1   -0.01  -0.06   0.13   0.16]]

[[-33.96  -13.34   -9.1    -4.3   -11.93   -8.45]
 [  0.    -58.82  -39.23   -3.     -9.65   -9.9 ]
 [  0.      0.    -16.04   -2.81   -4.47   -3.6 ]
 [  0.      0.      0.     15.25   49.51    0.51]
 [  0.      0.      0.      0.    -21.58    0.78]
 [  0.      0.      0.      0.      0.     -1.93]]
Determinant from QR decomposition: 20377807.99999999
Determinant from np.linalg.det: 20377807.999999978
Both methods give the same determinant.
```

Numerical stability

Stability of the Gram-Schmidt process

During the execution of the Gram-Schmidt process, the vectors u_i are often not quite orthogonal, due to rounding errors. For the classical Gram-Schmidt process this loss of orthogonality is particularly bad. The computation also yields poor results when some of the vectors are almost linearly dependent. For these reasons, it is said that the classical Gram-Schmidt process is numerically unstable.

Givens QR decomposition

Assume A is an $m \times n$ matrix. If c and s are constants, an $m \times m$ Givens matrix $J(i, j, c, s), i < j$ places c at indicies (i, i) and (j, j), $-s$ at (j, i), and s at (i, j) in the identity matrix. $J(i, j, c, s)$ is orthogonal, and by a careful choice of constants $J(i, j, c, s)A$ affects only rows i and j of A and zeros out a_{ji}.The product is performed implicitly by changing just rows i and j , so it is rarely necessary to build a Givens matrix. By zeroing out all the elements below the main diagonal, the Givens QR algorithm produces the upper triangular matrix R in the decomposition. By maintaining a product of Givens matrices, Q can also be found. The algorithm is stable, and its perturbation analysis does not involve the condition number of A. While the Givens QR decompositionis efficient and stable. Householder reflections are normally used for the QR decomposition. However, because premultiplication by a Givens matrix can zero out a particular element, these matrices are very useful when A has a structure that lends itself to zeroing out one element at a time.

Householder QR decomposition

The use of Householder matrices the most commonly used method for performing the QR decomposition. If $\mathbf{u}$ is an $m \times 1$ vector, the Householder matrix defined by

$$H_u = I - \left(\frac{2}{\mathbf{u}^T\mathbf{u}}\right)\mathbf{u}\mathbf{u}^T$$

is orthogonal and symmetric. Products $H_u\mathbf{v}, H_uA$, and AH_u, where A is an $m \times n$ matrix and $\mathbf{v}$ is an $m \times 1$ vector can be computed implicitly without the need to build H_u. By a proper choice of $\mathbf{u}$, H_uA zeros out all the elements below a diagonal element a_{ii}, and so it is an ideal tool for the QR decomposition. The Householder QR decomposition is stable and, like Givens QR process, its perturbation analysis does not depend on the condition number of A. It is this "all at once" feature of Householder matrices that makes them so useful for matrix decompositions.