MATH GR5430 Machine Learning for Finance
HomeWork 1 - QR Decomposition

Minze Li
ml5163@columbia.edu

Sep 8, 2024

1 Definition and Properties

1.1 Definition

The QR decomposition (also known as the QR factorization) is a fundamental matrix decomposition used in numerical linear algebra. It factorizes a matrix A into a product of an orthogonal matrix Q and an upper triangular matrix R. This decomposition is especially useful when A is a tall and skinny matrix (i.e., it has more rows than columns).

For an m × n matrix A with linearly independent columns ($m >= n$), the QR decomposition is given by:

$$A = QR$$

where:

- Q is an m × n matrix with orthonormal columns ($Q^T Q = I_n$)
- R is an n × n upper triangular matrix

1.2 Properties

1. Orthogonality of Q:
   - The columns of the matrix Q are orthonormal, meaning they are orthogonal to each other and have unit length.
   - Mathematically, this can be expressed as $Q^T Q = I_n$, where I_n is the n × n identity matrix.
   - This property is essential for many applications, such as solving least squares problems and orthogonalization.
2. Triangularity of R:
   - The matrix R is an upper triangular matrix, meaning that all elements below the main diagonal are zero.
   - This triangular structure is essential for simplifying computations and solving systems of linear equations efficiently.
   - The combination of the orthogonal matrix Q and the upper triangular matrix R allows for the development of numerically stable and efficient algorithms for various problems in linear algebra.

1.3 Common Uses

The QR decomposition has several important applications:

1. Solving linear least squares problems: The QR decomposition provides an efficient and numerically stable way to solve the linear least squares problem $argmin_x||Ax - b||_2$.

2. Orthogonalization: The QR decomposition can be used to orthogonalize a set of vectors, which is useful in many numerical algorithms, such as the Gram-Schmidt process.

3. Eigenvalue problems: The QR algorithm, which is based on the QR decomposition, is a powerful method for computing eigenvalues and eigenvectors of matrices.

4. Singular value decomposition (SVD): The QR decomposition can be used as a first step in computing the SVD of a matrix.

1.4 uniqueness

If the matrix A has full column rank (i.e., its columns are linearly independent), then the QR decomposition is unique, provided that the diagonal entries of R are required to be positive.
This uniqueness property is important for ensuring consistency and reproducibility in numerical computations.

2 Performing QR Decomposition in NumPy

NumPy provides the *numpy.linalg.qr()* function to compute the QR decomposition.

```
np.linalg.qr(a, mode=mode)
```

2.1 Parameters

1. *a*: An array-like object with the dimensionality of at least 2, with the shape of $(..., M, N)$.

2. *mode*: {'reduced', 'complete', 'r', 'raw'}, optional
   If $K = min(M, N)$, then:
   - 'reduced' (default): Returns Q, R with shapes (M, K) and (K, N).
   - 'complete': Returns Q, R with shapes (M, M) and (M, N).
   - 'r': Returns only R, of shape (K, N).
   - 'raw': Returns the raw QR factorization results h, τ with dimensions (N, M), K.

2.2 Default Values

The default value of mode is 'reduced', which is reasonable for most use cases because:

1. It provides both Q and R matrices.

2. It saves memory by returning a reduced form when $M > N$.

3. It's sufficient for many applications like solving least squares problems.

However, the choice of mode depends on the specific application:

- Use 'complete' if you need the full orthogonal matrix Q.

- Use 'r' if you only need the upper triangular factor R.

- Use 'raw' for more advanced applications or when you need to handle the factorization manually.

2.3 Returns

When the mode parameter is set to ‘reduced’ or ‘complete’, the *numpy.linalg.qr()* function returns a namedtuple with two attributes: Q and R.

- Q: An optional ndarray of float or complex numbers representing a matrix with orthonormal columns. If mode is set to ‘complete’, Q will be an orthogonal matrix (if a is real) or a unitary matrix (if a is complex). In this case, the determinant of Q may be either +1 or -1. If the input array a has more than two dimensions, a stack of matrices with the aforementioned properties will be returned.

- R: An optional ndarray of float or complex numbers representing the upper-triangular matrix. If the input array a has more than two dimensions, a stack of upper-triangular matrices will be returned.

When the mode parameter is set to ‘raw’, the function returns a namedtuple with two attributes: h and τ.

- (h, τ): Optional ndarrays of np.double or np.cdouble. The array h contains the Householder reflectors that generate the matrix Q along with the matrix R. The array τ contains the scaling factors for the reflectors.

3 Example

We can create a working example using a randomly-generated matrix:

```
# This is the Python Code of Homework 1
# Author: Minze Li
# Date: Sep 8, 2024
# MATH GR5430 Machine Learning for Finance
import numpy as np

rng = np.random.default_rng() # Create a random number generator
a = rng.normal(size=(10, 5)) # Create a random 10x5 matrix
print(a)

q,r = np.linalg.qr(a)
print(q, r)
print(np.allclose(a, np.dot(q, r))) # Test if QR = A

q2, r2 = np.linalg.qr(a, mode='complete')
print(q2, r2)
print(np.allclose(a, np.dot(q2, r2)))

r3 = np.linalg.qr(a, mode='r')
print(np.allclose(r, r3)) # Test if R = R3

h, tau = np.linalg.qr(a, mode='raw')
print(h, tau)
```

This example demonstrates:

1. Creating a random matrix a

2. Performing reduced QR decomposition, displaying Q and R matrices, and verifying the decomposition by reconstructing a from Q and R

3. Performing complete QR decomposition, displaying Q2 and R2 matrices, and verifying the decomposition by reconstructing a from Q2 and R2

4. Performing QR decomposition only returning R, proving R and R3 are equal

5. Performing raw QR decomposition, displaying h and τ

And here is the outputs of the example:

$$a = \begin{bmatrix} -2.04381298 & -0.77736929 & 0.24218269 & -0.05532419 & -0.72578899 \\ -1.42802939 & -1.87795501 & -0.44131166 & 0.95490125 & 0.03982269 \\ 1.15434388 & -0.90420823 & -1.70505873 & -2.07776297 & -1.44958343 \\ -0.75041777 & 0.24641222 & -0.1725188 & -1.94143691 & 1.97027348 \\ -0.37826909 & -1.04522331 & -0.47599108 & -0.79015921 & 1.096496 \\ -0.37483228 & -1.1541198 & 0.70195118 & 1.7145625 & -2.08461522 \\ 0.16277105 & 0.55547518 & 0.28344157 & -0.2816453 & -0.25438268 \\ 0.90376429 & 0.95353656 & -0.443515 & 0.00395152 & 1.21629109 \\ 1.31475571 & 0.70182954 & 0.49919128 & -0.94774312 & -0.17986485 \\ -0.16454612 & 0.73960016 & 1.35017875 & 1.02123698 & -0.51166492 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.61638397 & 0.10262763 & -0.11347702 & -0.20388276 & 0.47762403 \\ -0.43067269 & -0.43959133 & -0.01313831 & 0.21111782 & -0.28233864 \\ 0.34813316 & -0.57250475 & -0.35277205 & -0.25617169 & 0.46768665 \\ -0.22631498 & 0.24140201 & -0.32380601 & -0.55450635 & -0.26187541 \\ -0.1140804 & -0.3262688 & -0.0313457 & -0.30311053 & -0.5399699 \\ -0.11304391 & -0.36865513 & 0.59705055 & 0.0553907 & 0.11075129 \\ 0.04908936 & 0.18088768 & 0.02359017 & -0.12094724 & 0.20298337 \\ 0.27256203 & 0.18819897 & -0.29189739 & 0.39634493 & -0.1921766 \\ 0.39651101 & 0.01126531 & 0.34818128 & -0.51778888 & -0.13179801 \\ -0.04962469 & 0.31554138 & 0.43955569 & -0.07741368 & 0.07424617 \end{bmatrix}$$

$$R = \begin{bmatrix} 3.31581138 & 1.69584246 & -0.51484935 & -1.20400794 & -0.13667183 \\ 0. & 2.61061102 & 1.44934523 & 0.18250374 & 1.64369367 \\ 0. & 0. & 1.97313133 & 2.51491306 & -1.97233174 \\ 0. & 0. & 0. & 2.60345939 & -0.36705198 \\ 0. & 0. & 0. & 0. & -2.67442489 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} -0.61638397 & 0.10262763 & -0.11347702 & -0.20388276 & 0.47762403 & 0.16644402 \\ -0.0566147 & 0.1250255 & 0.52952741 & -0.00538089 & & \\ -0.43067269 & -0.43959133 & -0.01313831 & 0.21111782 & -0.28233864 & -0.56108621 \\ 0.1641044 & 0.37032598 & 0.02482711 & 0.13159154 & & \\ 0.34813316 & -0.57250475 & -0.35277205 & -0.25617169 & 0.46768665 & 0.00444555 \\ -0.02887996 & 0.12205036 & -0.10974773 & 0.33829586 & & \\ -0.22631498 & 0.24140201 & -0.32380601 & -0.55450635 & -0.26187541 & 0.09164478 \\ -0.12615031 & 0.37750165 & -0.48895092 & -0.0608859 & & \\ -0.1140804 & -0.3262688 & -0.0313457 & -0.30311053 & -0.5399699 & 0.42332763 \\ 0.23574937 & -0.33817158 & 0.26882587 & 0.2733073 & & \\ -0.11304391 & -0.36865513 & 0.59705055 & 0.0553907 & 0.11075129 & 0.49567148 \\ -0.00948452 & 0.35274673 & -0.25302347 & -0.21279056 & & \\ 0.04908936 & 0.18088768 & 0.02359017 & -0.12094724 & 0.20298337 & -0.02973203 \\ 0.94145145 & 0.0664947 & -0.1089322 & -0.07057137 & & \\ 0.27256203 & 0.18819897 & -0.29189739 & 0.39634493 & -0.1921766 & 0.36324425 \\ 0.0672596 & 0.59283453 & 0.2990914 & 0.18354063 & & \\ 0.39651101 & 0.01126531 & 0.34818128 & -0.51778888 & -0.13179801 & -0.29396311 \\ -0.06081227 & 0.30091365 & 0.46637224 & -0.19437424 & & \\ -0.04962469 & 0.31554138 & 0.43955569 & -0.07741368 & 0.07424617 & -0.07013516 \\ -0.05381484 & 0.06789609 & -0.11293178 & 0.81736039 & & \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 3.31581138 & 1.69584246 & -0.51484935 & -1.20400794 & -0.13667183 \\ 0. & 2.61061102 & 1.44934523 & 0.18250374 & 1.64369367 \\ 0. & 0. & 1.97313133 & 2.51491306 & -1.97233174 \\ 0. & 0. & 0. & 2.60345939 & -0.36705198 \\ 0. & 0. & 0. & 0. & -2.67442489 \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix}$$

$$h = \begin{bmatrix} 3.31581138 & 0.26644207 & -0.21537776 & 0.14001313 & 0.07057754 & 0.0699363 \\ -0.03036986 & -0.16862456 & -0.24530744 & 0.03070105 & & \\ 1.69584246 & 2.61061102 & 0.37520463 & -0.15476671 & 0.22735285 & 0.25620246 \\ -0.12543459 & -0.14009101 & -0.02484133 & -0.21295454 & & \\ -0.51484935 & 1.44934523 & 1.97313133 & 0.22063148 & 0.01967565 & -0.43408114 \\ -0.01610868 & 0.22306391 & -0.23183211 & -0.32283289 & & \\ -1.20400794 & 0.18250374 & 2.51491306 & 2.60345939 & 0.24429475 & 0.12304013 \\ 0.07179937 & -0.34961403 & 0.46815243 & 0.10285458 & & \\ -0.13667183 & 1.64369367 & -1.97233174 & -0.36705198 & -2.67442489 & -0.41513808 \\ -0.15986052 & 0.38495855 & -0.29521307 & -0.19333762 & & \end{bmatrix}$$

$$\tau = \begin{bmatrix} 1.61638397 & 1.46693565 & 1.38362725 & 1.39669712 & 1.35997221 \end{bmatrix}$$

And all the np.allclose() return to be True.