MA-UY 3044B - QR Factorization

Ammaar Firozi

December 2021

Abstract

This project covers the processes to construct the **QR Factorization** of any **m x n** matrix **A** whose columns are linearly independent (i.e. $rank(A) = n$). In $A = QR$, the columns of **Q** will be an orthonormal basis for $Col(A)$, and **R** will be upper triangular. This project also mentions some breif applications of QR factorization, including the Linear Least Squares Problem and the QR method for finding eigenvalues. For inquiries, please email mmf8465@nyu.edu.

Contents

1 Theoretical Background

1.0.1 Theorem (QR Factorization)

Let $A \in \mathbb{R}^{m\times n}$ have rank **n** (i.e. linearly independent columns). Then **A** can be factored as $\mathbf{A = QR}$, where the columns of $Q \in \mathbb{R}^{m\times n}$ form an orthonormal basis for **Col(A)**, and $Q \in \mathbb{R}^{n\times n}$ is upper-triangular with positive diagonal entries. [6]

1.1 Proof

Key property of basis constructed by Gram-Schmidt

$$Span\{\vec{u}_1, \vec{u}_2, ..., \vec{u}_k\} = Span\{\vec{x}_1, \vec{x}_2, ..., \vec{x}_k\} \quad \forall k \in [n]$$

Hence,

$$\exists \quad r_{ik} \quad st \quad \vec{x}_k = r_{1k}\vec{u}_1 + ... + r_{kk}\vec{u}_k + \mathbf{0}\vec{u}_{k+1} + ... + \mathbf{0}\vec{u}_n$$

WLOG,

$$r_{kk} \geq 0(\text{ or change the sign of } \vec{u}_k).$$
$$\vec{r}_k = (r_{1k}, ..., r_{kk}, 0, ..., 0) \in \mathbb{R}^n.$$

Then, the above says

$$Q\vec{r}_k = \vec{x}_k$$

$$R = [\vec{r}_1 ... \vec{r}_n], getA = [\vec{x}_1 ... \vec{x}_n = [Q\vec{r}_1 ... Q\vec{r}_n] = QR.$$

Since

$$Q^TQ = I_n, \quad A = QR \rightsquigarrow Q^TA = Q^TQR = R$$

1.2 Why is R upper Triangular?

Computing $\vec{x}_1$ only requires computing $r_{11} \cdot \vec{u}_1$, and all other multiplications will be 0. This means that writing $\vec{x}_1$ in terms of the basis, the first column of R will only have one non-zero element, the 2^{nd} column will have two non-zero elements, and the n^{th} column will have all non-zero elements. Based on this construction, R must be upper-triangular.

$$R = \begin{bmatrix} r_{11} & r_{12} & ... & r_{1n} \\ 0 & r_{12} & \ddots & r_{2n} \\ 0 & 0 & ... & \vdots \\ \vdots & 0 & ... & \vdots \\ 0 & 0 & ... & r_{nn} \end{bmatrix}$$

1.2.1 Why is R invertible?

We can assume that R_{kk} was ≥ 0, as we know the basis constructed by Gram-Schmidt is constructed of linearly independent elements, as in R, a dimension is added for every $\vec{x}_k$, so each new column has an additional dimension every time a new x_k is added, so the coefficient has to be nonzero.

You can also think about this in another way. We know that

$$A = QR$$

Where A represents a transformation in the following form:

$$T(v) = Av \quad T : \mathbb{R}^n \to \mathbb{R}^m,$$

Q also has a similar transformation,

$$T(v) = Qv \quad T : \mathbb{R}^n \to \mathbb{R}^m,$$

and R has the following transformation,

$$T(v) = Rv \quad T : \mathbb{R}^n \to \mathbb{R}^n$$

From this setup, we can show R is invertable by contradiction.

Suppose A is decomposed into QR, where R is a mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$, and Q is a mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$. We know that A is a mapping $\mathbb{R}^n$ to $\mathbb{R}^m$. This relationship can be visualized as follows:

$$\left(\mathbb{R}^n\right) \to^R \left(\mathbb{R}^n\right) \to^Q \left(\mathbb{R}^m\right)$$

$$\left(\mathbb{R}^n\right) \to^A \left(\mathbb{R}^m\right),$$

Suppose R is not invertable. R is a square matrix, so by the invertable matrix theorem, R would have to have a nullspace that is not just 0.

$$Ker(R) \neq \{0\}$$

The transformation mapping of things going to 0 would include Ker(R). This implies that Ker(Q) would have a dimension greater than 0. However, A has linearly independent columns and therefore does not have any nullspace. Any things transformed in R to 0, will also be transformed to 0 in Q, so a non-trivial nullspace for R implies a non-trivial nullspace for Q, which cannot happen. Contradiction.

Alternatively, you can derive this solution by noting that R is upper triangular, and has a non-zero determinant, which implies invertability.

1.2.2 Why are all diagonal entries > 0 (not just ≥ 0)?

If an element on the diagonal of R contained 0, the determinant would be 0, which, as stated above, contradicts Q being an orthogonal matrix.

1.3 Example:

Suppose we want to find the Reduced QR factorization of the given matrix **A**

$$A = \begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 1 & 5 \end{bmatrix}$$

In general, we can decompose a full rank matrix, A, into the matrices Q and R in the following expression.

$$\begin{bmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \dots & u_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & \dots & r_{2n} \\ & & & r_{nn} \end{bmatrix}$$

$$A = QR$$

For the given matrix, We can write this expression as follows

$$\begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix} = \begin{bmatrix} | & | \\ u_1 & u_2 \\ | & | \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}, \text{where } r_{21} \text{ is 0.}$$

1.3.1 Computing Q

Finding Q is trivial, as you just compute the orthonormal basis of A. Using gram schmidt, we can see that Q is

$$Q = \begin{bmatrix} \frac{2}{3} & -\frac{1}{\sqrt{18}} \\ \frac{2}{3} & -\frac{1}{\sqrt{18}} \\ \frac{1}{3} & \frac{4}{\sqrt{18}} \end{bmatrix}$$

After finding Q, we need to calculate R. There are two methods to do so. One way is to use the relationships established by gram schmidt to find the column vectors of R, and another is more brute force. These two methods are explained in the following sections.

1.3.2 Computing R - Method 1

We can define the columns of A as the linear combination of two orhonormal vectors spanning the column space of A, hence

$$\begin{bmatrix} | & | \\ u_1 & u_2 \\ | & | \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix}$$

The relationship between the column vectors of Q and A is already determined by gram schmidt, hence

$$u_1 = \frac{1}{\|a_1\|} a_1 \Rightarrow 3u_1 = a_1 \Rightarrow r_{11} = 3, r_{21} = 0$$

and

$$u_2 = \frac{1}{\sqrt{18}}(a_2 - 3u_1) \Rightarrow a_2 = u_2\sqrt{18} + 3u_1 \Rightarrow r_{12} = 3, r_{22} = \sqrt{18}$$

hence,

$$R = \begin{bmatrix} 3 & 3 \\ 0 & \sqrt{18} \end{bmatrix}$$

1.3.3 Computing R - Method 2

We can compute R by using the given expression

$$A = QR, \text{multiply both sides by } Q^T$$

$$Q^T A = Q^T Q R,$$

Q is an orthogonal matrix, hence $Q^T Q = I_n$

$$Q^T A = R$$

The calculation is easy enough,

$$Q^T = \begin{bmatrix} \frac{2}{3} & \frac{2}{3} & \frac{1}{3} \\ -\frac{1}{\sqrt{18}} & -\frac{1}{\sqrt{18}} & \frac{4}{\sqrt{18}} \end{bmatrix}$$

$$Q^T A = \begin{bmatrix} \frac{2}{3} & \frac{2}{3} & \frac{1}{3} \\ -\frac{1}{\sqrt{18}} & -\frac{1}{\sqrt{18}} & \frac{4}{\sqrt{18}} \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 0 & \sqrt{18} \end{bmatrix}$$

In the end, the reduced QR factorization of A can be expressed as follows

$$\begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{\sqrt{18}} \\ \frac{2}{3} & -\frac{1}{\sqrt{18}} \\ \frac{1}{3} & \frac{4}{\sqrt{18}} \end{bmatrix} \begin{bmatrix} 3 & 3 \\ 0 & \sqrt{18} \end{bmatrix}$$

$$A = QR.$$

2 QR Factorization - Main Idea

Find orthonormal vectors that span the successive spaces spanned by the columns of **A**:

$$\vec{a}_1 \subseteq < \vec{a}_1, \vec{a}_2 > \subseteq ... \subseteq ...$$

This means that for a full rank matrix A,

$$< \vec{m}_1, \vec{m}_2, ..., \vec{m}_i > \subseteq < \vec{a}_1, \vec{a}_2, ..., \vec{a}_i > , for \quad i = 1, ...n$$

[3]

2.0.1 Matrix Form

In matrx form,

$$< \vec{m}_1, \vec{m}_2, ..., \vec{m}_i > \subseteq < \vec{a}_1, \vec{a}_2, ..., \vec{a}_i > becomes$$

$$\left[\begin{array}{c|c|c|c} a_1 & a_2 & ... & a_n \end{array} \right] = \left[\begin{array}{c|c|c|c} q_1 & q_2 & ... & q_n \end{array} \right] \begin{bmatrix} r_{11} & r_{12} & ... & r_{1n} \\ & r_{22} & & \vdots \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix} \text{ or}$$

$$A = \hat{Q}\hat{R}.$$

This is the reduced QR factorization. Add orthogonal extension to **Q** and add rows to **R** to obtain the *full* QR factorization

2.0.2 Full QR Factorization

Let

$$A \in \mathbb{R}^{m \times n}.$$

The full QR factorization of A is the factorization

$$A = QR, \text{ where}$$

$$Q \in \mathbb{R}^{m \times m}, \text{ and unitary. } R \in \mathbb{R}^{m \times n}, \text{ and upper-triangular}$$

2.0.3 Reduced QR Factorization

A more compact representation is the Reduced QR Factorization,

$$A = \hat{Q}\hat{R}, \text{ where (assuming } m \geq n)$$
$$Q \in \mathbb{R}^{m \times m}, \quad R \in \mathbb{R}^{n \times n}$$

2.1 Gram-Schmidt Orthogonalization

Find new q_i orthogonal to $q_1, .., q_{i-1}$ by subtracting components along previous vectors, i.e.

$$v_i = ai - (q_1 \cdot a_i)q_1 - (q_2 \cdot a_i)q_2 - ... - (q_{i-1} \cdot a_i)q_{i-1}$$

Normalize each to get

$$q_i = \frac{v_i}{\|\mathbf{v_i}\|}$$

We then obtain a reduced QR factorization, $A = \hat{Q}\hat{R}$, with $r_{ij} = q_i \cdot a_j$ for $i \neq j$

and

$$|r_{jj}| = \|a_j - \sum_{i=1}^{j-1} r_{ij}q_i\|$$

2.1.1 Classical Gram-Schmidt

Classical Gram-Schmidt is a very straight-forward application of Gram-Schmidt orthogonalization but is numerically unstable.

Algorithm 1 Classical Gram-Schmidt

```
for j ← 1, n do
    v_j = a_j
    for i ← 1, j − 1 do
        r_ij = q_i · a_j                [1]
        v_j = v_j − r_ij · q_i
    end for
    r_jj = ‖v_j = v_j‖_2
    q_j = v_j / r_jj
end for
```

2.1.2 Existence and Uniqueness

Every A $\in \mathbb{C}^{m\times n}$ $(m \geq n)$ has full QR factorization and reduced QR factorization.

Proof: for full rank A, Gram-Schmidt prove exisistance of $A = \hat{Q}\hat{R}$. Otherwise, when $v_j = 0$ choose an arbitrary vector orthogonal to the previous q_i. For full QR, add orthogonal extnsion to **Q** and zero rows to **R**.

Each A $\in \mathbb{C}^{m\times n}$ $(m \geq n)$ of full rank has a unique $A = \hat{Q}\hat{R}$ with $r_{jj} > 0$.

Proof: Again, Gram-Schmidt, $r_{jj} > 0$ determines the sign.

3 Gram-Schmidt Orthogonalization

3.1 Gram-Schmidt Projections

The orthogonal vectors produced by Gram-Schmidt can be written in terms of projectons.

$$q_1 = \frac{P_1 a_1}{\|P_1 a_1\|}, \quad q_2 = \frac{P_2 a_2}{\|P_2 a_2\|}, \ldots \quad q_n = \frac{P_n a_n}{\|P_n a_n\|}$$

where

$$P_j = I - \hat{Q}_{j-1}\hat{Q}_{j-1}^T \text{ with } \hat{Q}_{j-1} = \left[\ q_1 \mid qa_2 \mid \ldots \mid q_{j-1}\ \right]$$

In other words, P_j orthogonally projects onto the space orthogonal to

$$< q_1, \ldots, q_{j-1} > \text{, and } rank(P_j) = m - (j-1)$$

3.2 The Modified Gram-Schmidt Algorithm

THe projection P_j can equivalently be written as

$$P_j = P_{\perp q_{j-1}} \ldots P_{\perp q_2} P_{\perp q_1}$$

where

$$P_{\perp q} = I - qq^T$$

In other words, $P_{\perp q}$ projects orthogonally onto the space orthogonal to **q**, and $rank(P_{\perp q}) = m - 1$
The classical Gram-Schmidt algorithm computes an orthogonal vector via $v_j = P_j a_j$
while the Modified Gram-Schmidt algorithm uses

$$v_j = P_{\perp q_{j-1}} \ldots P_{\perp q_2} P_{\perp q_1} a_j$$

3.3 Classical vs. Modified Gram-Schmidt

A small modification of classical Gram-Schmidt gives modified Gram-Schmidt, which is numerically stable (less sensitive to floating-point rounding errors in large computations).

Algorithm 2 Classical/Modified Gram-Schmidt Algorithm

for $j \leftarrow 1, n$ **do**
  $v_j = a_j$
  for $i \leftarrow 1, j-1$ **do**
    $r_{ij} = q_i^T \cdot a_j$ (CGS)
    $r_{ij} = q_i^T \cdot v_j$ (MGS)
    $v_j = v_j - r_{ij} \cdot q_i$
  end for
  $r_{jj} = \|v_j = v_j\|_2$
  $q_j = v_j / r_{jj}$
end for

[1]

3.3.1 Implementation of Modified Gram-Schmidt

In modified Gram-Schmidt, $P_{\perp q_i}$ can be applied to all v_j as soon as q_i is known. This makes the inner loop iterations independent (like in classical Gram-Schmidt)

Algorithm 3 Classical Gram-Schmidt

for $j \leftarrow 1, n$ **do**
  $v_j = a_j$
  for $i \leftarrow 1, j-1$ **do**
    $r_{ij} = q_i^T \cdot a_j$
    $v_j = v_j - r_{ij} \cdot q_i$
  end for
  $r_{jj} = \|v_j = v_j\|_2$
  $q_j = v_j / r_{jj}$
end for

Algorithm 4 Modified Gram-Schmidt

for $i \leftarrow 1, n$ **do**
  $v_i = a_i$
end for
for $i \leftarrow 1, n$ **do**
  $r_{ii} = \|v_i\|$
  $q_i = v_i / r_{ii}$
  for $j \leftarrow i+1, n$ **do**
    $r_{ij} = q_i^T v_j$
    $v_j = v_j - r_{ij} \cdot q_i$
  end for
end for

[1]

4 Householder Reflectors

Note: Much work in this section was derived thanks to the book, Matrix Computations [1].

4.1 Gram-Schmidt as Triangular Orthogonalization

Gram-Schmidt multiplies with triangular matrices to make columns orthogonal, for example at the first step:

$$\begin{bmatrix} v_1 \mid v_2 \mid ... \mid v_n \end{bmatrix} \begin{bmatrix} \frac{1}{r_{11}} & \frac{-r_{12}}{r_{11}} & \frac{-r_{13}}{r_{11}} & ... \\ & 1 & & \\ & & 1 & \\ & & & \ddots \end{bmatrix} = \begin{bmatrix} q_1 \mid v_2^{(2)} \mid ... \mid v_n^{(2)} \end{bmatrix}$$

After all these steps, we get a product of triangular matrices:

$$AR_1R_2...R_n = \hat{Q}$$

where

$$AR_1R_2...R_n = \hat{R}^{-1}$$

This is call "Triangular orthogonalization"

4.2 Householder Triangularization

Gram-Schmidt method for computing QR factorization is often unstable when using floating point arithmetic. One alternative, which is somewhat similar to LU factorization just with orthogonal matrices, is to introduce zeroes into R one column at a time. This statement is the essence of Householder QR and is summarized below.

The Householder method multiplies by unitary matrices to make columns triangular, for example, at the first step:

$$Q_1 A = \begin{bmatrix} r_{11} & \mathbf{x} & \dots & \mathbf{x} \\ 0 & \mathbf{x} & & \mathbf{x} \\ 0 & \mathbf{x} & & \mathbf{x} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \mathbf{x} & \dots & \ddots \end{bmatrix}$$

After all these steps, we get a product of orthogonal matrices,

$$Q_n ... Q_2 Q_1 A = R$$

where

$$Q_n ... Q_2 Q_1 = Q^T \quad \text{i.e. } Q^T A = R$$

This is called "Orthogonal triangularization"

Q_k introduces zeroes below the diagonal in column **k**, preserving all the zeros previously introduced

$$\begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \rightarrow^{Q_1} \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & \mathbf{x} \end{bmatrix} \rightarrow^{Q_2} \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} \\ x & x & x \\ 0 & \mathbf{x} & \mathbf{x} \\ & 0 & \mathbf{x} \\ & 0 & \mathbf{x} \end{bmatrix} \rightarrow^{Q_3} \begin{bmatrix} x & x & x \\ & x & x \\ & & \mathbf{x} \\ & & 0 \\ & & 0 \end{bmatrix}$$

$$A \rightarrow Q_1 A \rightarrow Q_2 Q_1 A \rightarrow Q_3 Q_2 Q_1 A$$

4.3 Householder Reflectors

The basic operation of a Householder reflector is the following: Given a column vector **z**, find an orthogonal P, so that **Pz** is zero. Let Q_k be of the form

$$Q_k = \begin{bmatrix} I & 0 \\ 0 & F \end{bmatrix}$$

where **I** is $(k-1)\times(k-1)$, and **F** is $(m-k+1)\times(m-k+1)$

We can create a Householder reflector, **F**, that introduces zeros:

$$x = \begin{bmatrix} x \\ x \\ \vdots \\ x \end{bmatrix} \qquad Fx = \begin{bmatrix} \|x\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \|x\| e_1$$

Idea: Reflect across hyper-plane **H** orthogonal to $v = \|x\| e_1 - x$ by the unitary matrix. Geometry implies that **P** could be an operator of reflection. Suppose we want to reflect a vector x to Pz. We can think of an additional vector v that maps x to Px. The length of the projection of x onto the "mirror" between x and Px can be computed as follows: $\|x\| \cos\theta = \frac{-x^T v}{\|v\|}$, whose direction is $\frac{v}{\|v\|}$. Multiplying by two yields $-2v\frac{x^T v}{v^T v}$ These properties are summarized below:

Suppose we have a vector w going from x to the "mirror," (hyper-plane) and from the "mirror" to Px.

For any vector x onto a "mirror": $\|w\| \cos\theta = \frac{-x^T v}{\sqrt{v^T v}} \cdot \frac{v}{\sqrt{v^T v}} = -v\frac{x^T v}{v^T v} = w$

For any x to its reflection: $-2v\frac{x^T v}{v^T v}$, (i.e. $x + 2w = Px$).

$$Px = x - -2\frac{v(x^T v)}{v^T v} \Rightarrow P = I - 2\frac{vv^T}{v^T v},$$

for all x, this is the Orthogonal matrix / Householder reflector.

$$F = I - 2\frac{vv^T}{v^T v}$$

compare with the projection

$$P_{\perp v} = I - \frac{vv^T}{v^T v}$$

$$Q_nQ_{n-1}...Q_2Q_1A = R, \quad Q^T = Q_nQ_{n-1}...Q_2Q_1$$

4.4 Choice of Reflector

We can choose to reflect any multiple **z** of $\|x\|e_1$ with $|z| = 1$.

However, there are better numerical properties with large $\|v\|$, for example

$$v = sign(x_1)\|x\|e_1 + x$$

4.5 The Householder Algorithm

The Householder Algorithm essentially computes the factor R of a QR factorization from a matrix $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$. We use the 2-norm as $\|Pz\|_2 = \|z\|_2$ by orthogonality. In the end, the result is left in place of A, and the reflectors v_k are stored for later use.

Algorithm 5 Householder QR Factorization

1: **for** $k \leftarrow 1, n$ **do**
2: $x = A_{k:m,k}$
3: $v_k = sign(x_1)\|x\|_2 e_1 + x$
4: $v_k = v_k/\|v_k\|_2$
5: $A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^T A_{k:m,k:n})$
6: **end for**

4.6 Applying or Forming Q

This process is relatively easy. Compute $Q^Tb = Q_n...Q_2Q_1b$ and $Qx = Q_1Q_2...Q_nx$ implicitly.

To create **Q** explicitly, apply to $x = I$

Algorithm 6 Implicit Calculation of Q^Tb

1: **for** $k \leftarrow 1, n$ **do**
2: $b_{k:m} = b_{k:m} - 2v_k(v_k^T b_{k:m})$
3: **end for**

Algorithm 7 Implicit Calculation of Qx

for $k \leftarrow n, 1$ **do**
$x_{k:m} = x_{k:m} - 2v_k(v_k^T x_{k:m})$
end for

4.7 Givens Rotations

An alternative to Householder reflectors are Givens Rotations. These are elementary rotations in two dimensions of a matrix problem. Givens rotations can be used in sequence to from the QR decomposition of a matrix. This method is often less efficient than other methods, but provides additional flexibility in some cases, which are listed later on.

The givens rotation matrix for a 2 by 2 matrix can be defined as follows:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

rotates $x \in \mathbb{R}^2$ by θ

More generally a givens rotation matrix can be defined as follows,

$$G(i,j,\theta) = \begin{bmatrix} 1 & ... & 0 & ... & 0 & ... & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & ... & \cos\theta & ... & \sin\theta & ... & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & ... & -\sin\theta & ... & \cos\theta & ... & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & ... & 0 & ... & 0 & ... & 1 \end{bmatrix},$$

which applies a rotation within the space spanned by the i^{th} and j^{th} coordinates.

To set an element to zero, choose $\cos(\theta)$ and $\sin(\theta)$ so that

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_i \\ x_k \end{bmatrix} = \begin{bmatrix} \sqrt{x_i^2 + x_j^2} \\ 0 \end{bmatrix}$$

or

$$\cos(\theta) = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \qquad \sin(\theta) = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}}$$

However, this method is susceptible to overflow/underflow if $\sqrt{x_i^2 + x_j^2}$ is very small. A better computation is as follows:

$if |X_i| > |x_j|$, set $t = \tan\theta = \frac{x_j}{x_i} \Rightarrow$

$$\cos\theta = \frac{1}{\sqrt{1+t^2}}, \quad \sin\theta = \cos(\theta)t$$

$if |X_j| \geq |x_i|$, set $\tau = \cot\theta = \frac{x_i}{x_j} \Rightarrow$

$$\sin\theta = \frac{1}{\sqrt{1+\tau^2}}, \quad \cos\theta = \sin(\theta)\tau$$

4.7.1 Givens QR

To perform a Givens QR on an $m \times n$ matrix A where $m \geq n$, the following algorithm can be used,

Algorithm 8 Givens rotation algorithm [4]

```
R ← A, Q ← I
for k ← 1, n do
    for k ← m, k + 1 do
        G = G(j − 1, j, θ) (to eliminate a_jk)
        R = GR
        Q = QG^T
    end for
end for
```

The following visualization may be helpfull to visualize Givens QR,

$$\begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \rightarrow^{(3,4)} \begin{bmatrix} x & x & x \\ x & x & x \\ \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & \mathbf{x} \end{bmatrix} \rightarrow^{(2,3)} \begin{bmatrix} x & x & x \\ \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} \\ & x & x \end{bmatrix} \rightarrow^{(1,2)}$$

$$\begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} \\ & x & x \\ & x & x \end{bmatrix} \rightarrow^{(3,4)} \begin{bmatrix} x & x & x \\ & x & x \\ & \mathbf{x} & \mathbf{x} \\ & \mathbf{0} & x \end{bmatrix} \rightarrow^{(2,3)} \begin{bmatrix} x & x & x \\ & \mathbf{x} & \mathbf{x} \\ & \mathbf{0} & \mathbf{x} \\ & & x \end{bmatrix} \rightarrow^{(2,3)} \mathbf{R}$$

Note: This computation is roughly 50 percent more computationally expensive than Householder QR. For a more detailed explanation of this method, I recommend the following video.

4.8 Givens Rotations vs. Householder Reflections

This work shows how to rotate two elements of a column vector, such that one element would be zero, using Givens rotations. Because each rotation modifies at most two rows of a given matrix, these rotations can be done in parallel, which is why Givens rotations are preferred over Householder reflections. They are also easier to use when the QR factorization of a given matrix needs to be updated after adding a new row or deleting a column. Givens rotations are also more efficient compared to Householder rotations when a given matrix is sparse.

5 Applications

5.1 Linear Least Squares Problem

Note: most of the following material are inspired by the following sources: [5][2]

In general,

$$Ax = b \text{ with } m > n \text{ has no solution.}$$

Instead, we can try to minimize the residual

$$r = b - Ax.$$

Using the Matirx 2-norm, we obtain the following linear least squares problem (LSP):

$$\text{Given } A \in \mathbb{C}^{m\times n}, m \geq n, b \in \mathbb{C}^m, \text{ find } x \in \mathbb{C}^n \text{ such that } \|b - Ax\|_2 \text{ is minimized.}$$

The minimizer, $\mathbf{x}$, is the solution of the normal equations

$$A^T Ax = A^T b,$$

or in terms of the pseudo-inverse A^+:

$$x = A^+ b, \quad \text{where } A^+ = (A^T A)^{-1} A^T \in \mathbb{C}^{n\times m}$$

5.1.1 Geometric Interpretation

Find the point Ax in $range(A)$ closest to $\mathbf{b}$. This $\mathbf{x}$ will minimize the 2-norm of

$$r = b - Ax.$$

$Ax = Pb$, where P is an orthogonal projection onto $range(A)$, so the residual must be orthogonal to $range(A)$

5.1.2 Solving LS - 1. Normal Equations

Assuming $\mathbf{A}$ is of full rank, $A^T A$ is a square, hermatian positive definite system that can be solved via Gaussian elimination / Cholesky factorization

Idea:

1. Form the matrix $A^T A$ and the vector $A^T b$
2. Compute the Cholesky factorization $A^T A = R^T R$
3. Solve the lower-triangular system $R^T w = A^T b$ for w
4. Solve the upper triangular system $Rx = w$ for x

Note: Allegorically, this is fast, but is still semi-unstable / sensitive to rounding errors.

5.1.3 Solving LS - 2. QR Factorization

Using

$$A = \hat{Q}\hat{R}, b$$

can be projected onto (A) via

$$P = \hat{Q}\hat{Q}^T$$

Insert into

$$Ax = b \text{ to get } \hat{Q}\hat{R}x = \hat{Q}\hat{Q}^T b, \text{ or } \hat{R}x = \hat{Q}^T b$$

Idea:

1. Compute the reduced QR factorization $A = \hat{Q}\hat{R}$
2. Compute the vector $\hat{Q}^T b$
3. Solve the upper-triangular system $\hat{R}x = \hat{Q}^T b$ for x

Note: This is relatively fast and has good stability. This is actually used in MATLAB's "backslash"

5.1.4 Solving LS - 3. SVD

Using

$$A = \hat{U}\hat{\Sigma}V^T,$$

b can be projected onto $range(A)$ by

$$P = \hat{U}\hat{U}^T.$$

Insert into

$$Ax = b$$

to get

$$\hat{U}\hat{\Sigma}V^T x = \hat{U}\hat{U}^T b, \text{ or } \hat{\Sigma}V^T x = \hat{U}^T b$$

Idea:

1. Compute the reduced SVD $A = \hat{U}\hat{\Sigma}V^T$
2. Compute the vector $\hat{U}^T b$
3. Solve the diagonal system $\hat{\Sigma}\omega = \hat{U}^{Tb,}$ for ω
4. Set $x = V\omega$

Note: This method is the most stable of the three and is typically used if a matrix is close to rank-deficient.

5.2 Finding Eigenvalues

QR Factorization can be used to find Eigenvalues of a given matrix.
Suppose $A = Q_0R_0$ is a QR factorization of A;
Create $A_1 = R_0Q_0$
This step can be repeated for A_2, i.e. let $A_1 = Q_1R_1$ ne a QR factorization of A_1;
create $A_2 = R_1Q_1$.
Repeat this process.
Once A_m is created, continue this process, i.e. create $A_{m+1} = R_mQ_m$
This process stops when the entries below the main diagonal of A_m are sufficiently small, or just end if the process fails to converge after a set amount of iterations.

5.2.1 Optimization: Shifting

The QR method to compute Eigenvalues can fail to converge, and is typically very slow on its own. One slight modification to the algorithm, which makes it converge more quickly is **shifting:** at each step, choose a scalar **c** such that we do not perform QR factorization on A_i, but rather $A_i - cI$, then add cI back when defining A_{i+1}. If the scalars are chosen such that they are closer and closer to an eigenvalue of a matrix, this dramatically speeds up convergence of this algorithm

5.2.2 Optimization: Deflating

Another improvement can be made to compute Eigenvalues of a matrix more efficiently through a process called **deflating**, which removes the last row and column of a matrix when the last row has zero entries on all columns except the last (i.e., reduce dimensionality of matrix for less expensive computations).

5.2.3 Modified QR Algorithm to compute Eigenvalues

Idea:

1. Choose **c** to be the last diagonal entry in the matrix A. Let $A - cI = Q_0R_0$ be a QR factorization of $A - cI$; then create $A_1 = R_0Q_0 + cI$.
2. Choose **c** to be the last diagonal entry in A_1. Let $A_1 - cI = Q_1R_1$ be a QR factorization of $A_1 - cI$; then create $A_2 = R_1Q_1 + cI$.
3. Continue this process; Once A_m is created, choose c to be the last diagonal entry in A_m, Let $A_m - cI = Q_mR_m$ be a QR factorization of $A_m - cI$; then create $A_{m+1} = R_mQ_m + cI$.
4. Repeat the above steps until the eigenvalues have been found, or the triangular limit matrix does not converge after a set amount of iterations.

Remark 1: This method only applies to matrices that have **real** eigenvalues. If a matrix A has complex eigenvalues, this algorithm will still generate a converging sequence of matrices, but the limit matrix of this algorithm won't be upper triangular.

Remark 2: If a matrix A has different magnitude eigenvalues, this method will converge to an upper triangular matrix.

Remark 3: Most applications perform this process in two steps. First by finding a matrix similar to A and is an **upper Hessenberg matrix** that has all zeroes below its first sub-diagonal. Then this QR method is applied to the matrix. Given an upper Hessenberg matrix, this method also produces an upper Hessenberg matrix, so the sequence still converges to a block upper triangular matrix.

6 MATLAB Code

6.1 QR - Gram Schmidt

[h]

```
%% QR Factorization Script
%  Ma-UY 3044 B | Linear Algebra - Ammaar Firozi
%
%  Information
%  ------------
%
%  This file contains code that computes the QR factorization of a given
%  matrix. The matrix is read from the file inputMatrix.txt
%

%% Initialization
clear ; close all; clc




fprintf('Loading and Data ...\n')
% Load from inputMatrix.mat
A = load('inputMatrix.txt');
disp('A = ');
disp(A);

Q = get_Q(A);
R = get_R(A,Q);

disp('A = QR = ');
disp(A);
disp('Q= ');
disp(Q);
disp('R= ');
disp(R);
disp('Q*R =');
disp(Q*R);

%% ============== Part 1: Computing Q  ================================
%  Q is computed easaly via gram schmidt.
%

function [Q] = get_Q(A)
    [m,n]=size(A);
    Q=A;
    for i=1:n
        for j=1:i-1
            Q(:,i)=Q(:,i)- (Q(:,j)'*Q(:,i))*Q(:,j);
        end
        Q(:,i)=Q(:,i)/norm(Q(:,i));
    end
end


%% ================== Part 2: Computing R ====================
%  The following code compute R in the QR factorization of a
%  given matrix A. This process relies on subtracting the dot product of
%  each element across the diagonal of A with Q from R. Alternatively
%  (commented out), you could just use R = transpose(Q)*A.
%
```

```
function [R] = get_R(A,Q)
    [m,n] = size(A);
    Q_tmp = A;
    R=zeros(n);
    %or
    %R = Q'*A;
    for i=1:n
        for j=1:i-1
            R(j,i)=Q_tmp(:,j)'*Q_tmp(:,i);
            Q_tmp(:,i)=Q_tmp(:,i)- R(j,i)*Q_tmp(:,j);
        end
        R(i,i)=norm(Q_tmp(:,i));
        Q_tmp(:,i)=Q_tmp(:,i)/R(i,i);
    end
end
```

QR Factorization using Gram Schmidt

6.2 QR - Householder

[h]

```
%% QR Factorization Script
%  Ma-UY 3044 B | Linear Algebra - Ammaar Firozi
%
%  Information
%  ------------
%
%  This file contains code that computes the Householder QR factorization of a given
%  matrix. The matrix is read from the file inputMatrix.txt
%

%% Initialization
clear ; close all; clc

fprintf('Loading and Data ...\n')
% Load from inputMatrix.mat
A = load('inputMatrix.txt');
disp('A = ');
disp(A);

[Q, R] = qr_factorization_householder(A);

disp('A = QR = ');
disp(A);
disp('Q= ');
disp(Q);
disp('R= ');
disp(R);
disp('Q*R =');
disp(Q*R);

function [Q, R] = qr_factorization_householder(A)
    [m, n]= size(A);
    Q=eye(m);
    R=A;
    for i=1:n
        %Compute v vector
        x = R(i:end, i);
        e1 = zeros(size(x));
        e1(1)=1;
        v=x-norm(x)*e1;

        H=eye(m);

        %Compute Householder matrix
        I=eye(length(v));
        H(i:end, i:end)=I-2*(v*v')/(v'*v);

        %Update Q and R
        R=H*R;
        Q=Q*H;
    end
end
```

QR Factorization using Householder Reflectors

7 References

References

[1] Cleve B Moler. *Numerical computing with MATLAB*. In collab. with Society for Industrial and Applied Mathematics. Rev. reprint. Book Title: Numerical computing with MATLAB. Philadelphia, Pa.: Society for Industrial and Applied Mathematics SIAM Market Street, Floor 6, Philadelphia, PA 19104, 2008. ISBN: 978-0-89871-795-2. URL: http://proxy.library.nyu.edu/login?url=https://epubs.siam.org/doi/book/10.1137/1.9780898717952 (visited on 12/19/2021).

[2] Gene H Golub (Gene Howard). *Matrix computations*. In collab. with Charles F Van Loan author. Fourth edition. Johns Hopkins studies in the mathematical sciences. Book Title: Matrix computations. Baltimore: The Johns Hopkins University Press, 2013. ISBN: 978-1-4214-0794-4.

[3] Anne Greenbaum. *Numerical methods: design, analysis, and computer implementation of algorithms*. In collab. with Timothy P. Chartier. Book Title: Numerical methods : design, analysis, and computer implementation of algorithms. Princeton, N.J.: Princeton University Press, 2012. ISBN: 978-0-691-15122-9.

[4] William H. Press. *Numerical recipes: the art of scientific computing*. 3rd ed. Book Title: Numerical recipes : the art of scientific computing. Cambridge, UK %3B New York: Cambridge University Press, 2007. xxi+1235. ISBN: 978-0-521-88068-8.

[5] Gilbert Strang. *Linear algebra and its applications*. 2d ed.. Book Title: Linear algebra and its applications. New York: Academic Press, 1980. xi+414. ISBN: 978-0-12-673660-1. URL: http://proxy.library.nyu.edu/login?url=https://ebookcentral.proquest.com/lib/nyulibrary-ebooks/detail.action?docID=1901785 (visited on 12/19/2021).

[6] Lloyd N. Trefethen. *Numerical linear algebra*. In collab. with David Bau. Book Title: Numerical linear algebra. Philadelphia: Society for Industrial and Applied Mathematics, 1997. xii+361. ISBN: 978-0-89871-361-9.