Math 6610: Analysis of Numerical Methods, I
The QR decomposition

Department of Mathematics, University of Utah

Fall 2025

Resources: Trefethen and Bau 1997, Lectures 20, 21, 23
Atkinson 1989, Chapter 1
Salgado and Wise 2022, Chapter 3

Orthogonalization of vectors

The main goal of orthogonalization:

Given $\{\boldsymbol{a}_j\}_{j\in[n]} \subset \mathbb{C}^m$ with $n \leqslant m$, compute $\{\boldsymbol{q}_j\}_{j\in[n]}$ such that:

$$\left\langle \boldsymbol{q}_j, \boldsymbol{q}_k \right\rangle = \delta_{j,k}, \qquad \operatorname{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\}$$

Orthogonalization of vectors

The main goal of orthogonalization:

Given $\{\boldsymbol{a}_j\}_{j\in[n]} \subset \mathbb{C}^m$ with $n \leqslant m$, compute $\{\boldsymbol{q}_j\}_{j\in[n]}$ such that:

$$\left\langle \boldsymbol{q}_j, \boldsymbol{q}_k \right\rangle = \delta_{j,k}, \qquad \operatorname{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\}$$

Why?

One reason is that the $\mathbb{C}^m$-orthogonal projector onto $\operatorname{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ is given by,

$$\boldsymbol{P} = \boldsymbol{Q}\boldsymbol{Q}^*, \qquad \boldsymbol{Q} = \begin{pmatrix} | & | & & | \\ \boldsymbol{q}_1 & \boldsymbol{q}_2 & \cdots & \boldsymbol{q}_n \\ | & | & & | \end{pmatrix}$$

Gram-Schmidt orthogonalization

One essentially explicit algorithm to orthogonalize is Gram-Schmidt.

This algorithm is "triangular orthogonalization": I.e., it's an algorithm that orthogonalizes vectors by accessing them in a triangular pattern.

Input: n vectors $\{\boldsymbol{a}_j\}_{j\in[n]}$. (Assume they're linearly independent for now.)
Output: n vectors $\{\boldsymbol{q}_j\}_{j\in[n]}$.

Gram-Schmidt orthogonalization

One essentially explicit algorithm to orthogonalize is Gram-Schmidt.

This algorithm is "triangular orthogonalization": I.e., it's an algorithm that orthogonalizes vectors by accessing them in a triangular pattern.

Input: n vectors $\{\boldsymbol{a}_j\}_{j\in[n]}$. (Assume they're linearly independent for now.)
Output: n vectors $\{\boldsymbol{q}_j\}_{j\in[n]}$.

Basic idea is induction:

1. Set $\boldsymbol{q}_1 = \frac{\boldsymbol{a}_1}{\|\boldsymbol{a}_1\|_2}$, and $j = 1$.
2. Since $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}$ have been computed:
   - Define $\widetilde{\boldsymbol{q}}_{j+1} = (\boldsymbol{I} - \boldsymbol{P}_j)\boldsymbol{a}_{j+1}$, where $\boldsymbol{P}_j$ is the orthogonal projector onto $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}$.
   - Set $\boldsymbol{q}_{j+1} = \widetilde{\boldsymbol{q}}_{j+1}/\|\widetilde{\boldsymbol{q}}_{j+1}\|_2$.
3. If $j = n-1$, quit. Otherwise, $j \leftarrow j+1$ and go back to step 2.

If the input vectors are linearly independent, this procedure cannot fail.
(At least, not in exact arithmetic....)

$$\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} \quad \longrightarrow \quad \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\},$$
$$\operatorname{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j\} = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\} \quad j \in [n]$$

$$\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} \quad \longrightarrow \quad \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\},$$
$$\operatorname{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j\} = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\} \quad j \in [n]$$

We can rewrite this to explicitly express the original vectors $\boldsymbol{a}_j$ in terms of the orthogonalized vectors $\boldsymbol{q}_j$.

At each step:

$$\boldsymbol{q}_j = \frac{1}{r_{j,j}} \left(\boldsymbol{I} - \boldsymbol{P}_j\right) \boldsymbol{a}_j \quad \Longrightarrow \quad \boldsymbol{a}_j = r_{j,j}\boldsymbol{q}_j + \sum_{k \in [j-1]} r_{k,j}\boldsymbol{q}_k, \quad r_{k,j} := \boldsymbol{q}_k^* \boldsymbol{a}_j = \langle \boldsymbol{a}_j, \boldsymbol{q}_k \rangle,$$

where $r_{j,j} = \|(\boldsymbol{I} - \boldsymbol{P}_j)\boldsymbol{a}_j\|_2$.

$$\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} \quad \longrightarrow \quad \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\},$$
$$\operatorname{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j\} = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\} \quad j \in [n]$$

We can rewrite this to explicitly express the original vectors $\boldsymbol{a}_j$ in terms of the orthogonalized vectors $\boldsymbol{q}_j$.

At each step:

$$\boldsymbol{q}_j = \frac{1}{r_{j,j}} (\boldsymbol{I} - \boldsymbol{P}_j)\, \boldsymbol{a}_j \quad \Longrightarrow \quad \boldsymbol{a}_j = r_{j,j}\boldsymbol{q}_j + \sum_{k \in [j-1]} r_{k,j}\boldsymbol{q}_k, \quad r_{k,j} := \boldsymbol{q}_k^* \boldsymbol{a}_j = \langle \boldsymbol{a}_j, \boldsymbol{q}_k \rangle,$$

where $r_{j,j} = \|(\boldsymbol{I} - \boldsymbol{P}_j)\boldsymbol{a}_j\|_2$.

If $\boldsymbol{A} \in \mathbb{C}^{m \times n}$ has $\boldsymbol{a}_j$ as columns, and $\boldsymbol{Q} \in \mathbb{C}^{m \times n}$ has $\boldsymbol{q}_j$ as columns, then this is equivalent to,

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}, \qquad (R)_{j,k} = r_{j,k}.$$

Columnwise: this expression is a record of how to reconstruct columns of $\boldsymbol{A}$ from the orthonormal columns of $\boldsymbol{Q}$.

In fact, these computations implies the following result:

Theorem

Let $\boldsymbol{A} \in \mathbb{C}^{m \times n}$ be any matrix. Then there exists a unitary matrix $\boldsymbol{Q} \in \mathbb{C}^{m \times m}$, and an upper-triangular matrix $\boldsymbol{R} \in \mathbb{C}^{m \times n}$ such that

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

If $\boldsymbol{A}$ has full rank, then the diagonal entries of $\boldsymbol{R}$ can be chosen to be positive.

In fact, these computations implies the following result:

Theorem

Let $\boldsymbol{A} \in \mathbb{C}^{m\times n}$ be any matrix. Then there exists a unitary matrix $\boldsymbol{Q} \in \mathbb{C}^{m\times m}$, and an upper-triangular matrix $\boldsymbol{R} \in \mathbb{C}^{m\times n}$ such that

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

If $\boldsymbol{A}$ has full rank, then the diagonal entries of $\boldsymbol{R}$ can be chosen to be positive.

We previously considered $n = \operatorname{rank}(\boldsymbol{A}) \leqslant m$. The remaining $m - n$ columns of $\boldsymbol{Q}$ are an(y) orthonormal completion of $\{\boldsymbol{q}_1, \ldots \boldsymbol{q}_n\}$.

Other cases:

- $m = \operatorname{rank}(\boldsymbol{A}) < n$: Only m vectors are linearly independent.
  $\boldsymbol{Q} \in \mathbb{C}^{m\times m}$, and $\boldsymbol{R}$ is short+fat.
- $m \geqslant n > r = \operatorname{rank}(\boldsymbol{A})$: Columns of $\boldsymbol{A}$ are dependent.
  The first r columns of $\boldsymbol{Q}$ span the range of $\boldsymbol{A}$.
  $\boldsymbol{R}$ is upper triangular, but the main diagonal may have zeros, and the last $m - r$ rows of $\boldsymbol{R}$ vanish.
- "Thin" QR: If $\operatorname{rank}(\boldsymbol{A}) < n$, then $\boldsymbol{Q}$ has only r columns.

Orthogonalization, summary

In summary:

Given $\{\boldsymbol{a}_j\}_{j\in[n]} \subset \mathbb{C}^m$, compute $\{\boldsymbol{q}_j\}_{j\in[n]}$ such that:

$$\left\langle \boldsymbol{q}_j, \boldsymbol{q}_k \right\rangle = \delta_{j,k}, \qquad\qquad \operatorname{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\}$$

Any algorithm to accomplish this (e.g., Gram-Schmidt) implies:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}, \qquad \boldsymbol{A} = \begin{pmatrix} | & | & & | \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_n \\ | & | & & | \end{pmatrix}, \qquad \boldsymbol{Q} = \begin{pmatrix} | & | & & | \\ \boldsymbol{q}_1 & \boldsymbol{q}_2 & \cdots & \boldsymbol{q}_n \\ | & | & & | \end{pmatrix},$$

with $\boldsymbol{R}$ upper triangular.

Orthogonalization, summary

In summary:

Given $\{\boldsymbol{a}_j\}_{j\in[n]} \subset \mathbb{C}^m$, compute $\{\boldsymbol{q}_j\}_{j\in[n]}$ such that:

$$\left\langle \boldsymbol{q}_j, \boldsymbol{q}_k \right\rangle = \delta_{j,k}, \qquad\qquad \operatorname{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n\}$$

Any algorithm to accomplish this (e.g., Gram-Schmidt) implies:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}, \qquad \boldsymbol{A} = \begin{pmatrix} | & | & & | \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_n \\ | & | & & | \end{pmatrix}, \qquad \boldsymbol{Q} = \begin{pmatrix} | & | & & | \\ \boldsymbol{q}_1 & \boldsymbol{q}_2 & \cdots & \boldsymbol{q}_n \\ | & | & & | \end{pmatrix},$$

with $\boldsymbol{R}$ upper triangular. "Classical" Gram-Schmidt numerically does:

$$\boldsymbol{u}_j = \boldsymbol{a}_j - \boldsymbol{P}_{j-1}\boldsymbol{a}_j, \qquad \boldsymbol{q}_j = \frac{\boldsymbol{u}_j}{\|\boldsymbol{u}_j\|_2}, \qquad \operatorname{range}(\boldsymbol{P}_j) = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}.$$

It turns out this is unstable ☹

$$\boldsymbol{u}_j = \boldsymbol{a}_j - \boldsymbol{P}_{j-1}\boldsymbol{a}_j, \qquad \boldsymbol{q}_j = \frac{\boldsymbol{u}_j}{\|\boldsymbol{u}_j\|_2}, \qquad \operatorname{range}(\boldsymbol{P}_j) = \operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}.$$

The cause of numerical instability is that, if $\boldsymbol{a}_j$ is nearly parallel to $\operatorname{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}\}$, this projection step can produce numerically incorrect results. (More precisely, the inner products $r_{k,j}$ for $k < j$ don't accurately represent the coordinates of $\boldsymbol{a}_j$.)

$$\boldsymbol{u}_j = \boldsymbol{a}_j - \boldsymbol{P}_{j-1}\boldsymbol{a}_j, \qquad \boldsymbol{q}_j = \frac{\boldsymbol{u}_j}{\|\boldsymbol{u}_j\|_2}, \qquad \text{range}(\boldsymbol{P}_j) = \text{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}.$$

The cause of numerical instability is that, if $\boldsymbol{a}_j$ is nearly parallel to $\text{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}\}$, this projection step can produce numerically incorrect results. (More precisely, the inner products $r_{k,j}$ for $k < j$ don't accurately represent the coordinates of $\boldsymbol{a}_j$.)

This problem can be fixed with a "modified" version of Gram-Schmidt, which computes $r_{k+1,j}$ by first orthogonalizing $\boldsymbol{a}_j$ against $\boldsymbol{q}_k$:

1. Set $\boldsymbol{u}_j = \boldsymbol{a}_j$, $j \in [n]$
2. Set $\boldsymbol{q}_1 = \frac{\boldsymbol{a}_1}{\|\boldsymbol{a}_1\|_2}$, and $j = 1$.
3. For $\ell > 1$: Set $r_{1,\ell} = \boldsymbol{q}_1^* \boldsymbol{u}_\ell$, and $\boldsymbol{u}_\ell = \boldsymbol{u}_\ell - r_{1,\ell}\boldsymbol{q}_1$.
4. Since $\boldsymbol{u}_{j+1}$ is orthogonal to $\boldsymbol{q}_k$, $k \in [j]$:
   - Set $\boldsymbol{q}_{j+1} = \boldsymbol{u}_{j+1}/\|\boldsymbol{u}_{j+1}\|_2$
   - For $\ell > j+1$: Set $r_{j+1,\ell} = \boldsymbol{q}_{j+1}^* \boldsymbol{u}_\ell$.
   - For $\ell > j+1$: Set $\boldsymbol{u}_\ell = \boldsymbol{u}_\ell - r_{j+1,\ell}\boldsymbol{q}_{j+1}$.
5. If $j = n-1$, quit. Otherwise, $j \leftarrow j+1$ and go back to step 5.

Thus, the projections are computed "one at a time".

Orthogonalization

We've seen "classical" (unstable) Gram-Schmidt and "modified" Gram-Schmidt.

In terms of stability, modified Gram-Schmidt effectively fixes the problem.

Both of these operations are "triangular orthogonalization", i.e., they perform the operation,

$$\boldsymbol{A} \mapsto \boldsymbol{A}\boldsymbol{R}^{-1} = \boldsymbol{Q}.$$

In terms of (ℓ^2-type) conditioning, this operation suffers a condition number of $\kappa(\boldsymbol{R})$.

Orthogonalization

We've seen "classical" (unstable) Gram-Schmidt and "modified" Gram-Schmidt.

In terms of stability, modified Gram-Schmidt effectively fixes the problem.

Both of these operations are "triangular orthogonalization", i.e., they perform the operation,

$$\boldsymbol{A} \mapsto \boldsymbol{A}\boldsymbol{R}^{-1} = \boldsymbol{Q}.$$

In terms of (ℓ^2-type) conditioning, this operation suffers a condition number of $\kappa(\boldsymbol{R})$.

Instead of attempting to compute $\boldsymbol{Q}$, we could attempt to compute $\boldsymbol{R}$.
This amounts to "orthogonal triangularization". The reason to cnosider this is that the operation,

$$\boldsymbol{A} \mapsto \boldsymbol{Q}^* \boldsymbol{A} = \boldsymbol{R},$$

is a much more well-conditioned operation.

There are two high-level strategies for orthogonal triangularization:

- Givens rotations: performs 2×2 unitary operations
- Householder reflectors: performs dimension-n unitary operations

Let $\boldsymbol{P}$ be an orthogonal projection matrix. Then $I - 2\boldsymbol{P}$ is Hermitian, unitary, and involutory (is its own inverse).

Householder reflectors

Let $\boldsymbol{P}$ be an orthogonal projection matrix. Then $I - 2\boldsymbol{P}$ is Hermitian, unitary, and involutory (is its own inverse).

Thus, application of this matrix, $\boldsymbol{x} \mapsto (\boldsymbol{I} - 2\boldsymbol{P})\boldsymbol{x}$, is well-conditioned.

In particular, if $\boldsymbol{P}$ is a rank-1 projector, then there is a unit vector $\boldsymbol{v}$ such that

$$P = \boldsymbol{v}\boldsymbol{v}^*.$$

(And in particular, $\boldsymbol{x} \mapsto (\boldsymbol{I} - 2\boldsymbol{P})\boldsymbol{x}$ does not require (expensive) matrix-vector multiplications.)

Use of Householder reflectors

Our main use of these reflectors is the following:

Given $\boldsymbol{x} \in \mathbb{C}^m$, we want to achieve:

$$\boldsymbol{x} \quad \overset{\text{Householder reflector}}{\longrightarrow} \quad \|\boldsymbol{x}\| e^{i\theta} \boldsymbol{e}_1,$$

for some $\theta \in [0, 2, \pi)$.

Use of Householder reflectors

Our main use of these reflectors is the following:

Given $\boldsymbol{x} \in \mathbb{C}^m$, we want to achieve:

$$\boldsymbol{x} \quad \overset{\text{Householder reflector}}{\longrightarrow} \quad \|\boldsymbol{x}\| e^{i\theta} \boldsymbol{e}_1,$$

for some $\theta \in [0, 2, \pi)$.

This is achieved by the reflector $\boldsymbol{I} - 2\boldsymbol{v}\boldsymbol{v}^*$, with $\boldsymbol{v}$ given by

$$\boldsymbol{v} = \frac{\boldsymbol{x} - \|\boldsymbol{x}\| e^{i\theta} \boldsymbol{e}_1}{\|\boldsymbol{x} - \|\boldsymbol{x}\| e^{i\theta} \boldsymbol{e}_1\|},$$

for arbitrary θ.

Use of Householder reflectors

Our main use of these reflectors is the following:

Given $\boldsymbol{x} \in \mathbb{C}^m$, we want to achieve:

$$\boldsymbol{x} \xrightarrow{\text{Householder reflector}} \|\boldsymbol{x}\| e^{i\theta} \boldsymbol{e}_1,$$

for some $\theta \in [0, 2, \pi)$.

This is achieved by the reflector $\boldsymbol{I} - 2\boldsymbol{v}\boldsymbol{v}^*$, with $\boldsymbol{v}$ given by

$$\boldsymbol{v} = \frac{\boldsymbol{x} - \|\boldsymbol{x}\| e^{i\theta} \boldsymbol{e}_1}{\|\boldsymbol{x} - \|\boldsymbol{x}\| e^{i\theta} \boldsymbol{e}_1\|},$$

for arbitrary θ.

For numerical stability, this reflector should make large changes to $\boldsymbol{x}$, rather than small changes. The largest change is achieved by selecting

$$e^{i\theta} = -\frac{x_1}{|x_1|}.$$

We now have the following procedure:
Given $\boldsymbol{x} \in \mathbb{C}^m$, we compute $\boldsymbol{v} \in \mathbb{C}^m$ such that

$$(\boldsymbol{I} - 2\boldsymbol{P})\boldsymbol{x} = (\boldsymbol{I} - \boldsymbol{v}\boldsymbol{v}^*)\boldsymbol{x} = c\,\boldsymbol{e}_1,$$

for some scalar $c \in \mathbb{C}$.

Use of Householder reflectors, II

We now have the following procedure:
Given $\boldsymbol{x} \in \mathbb{C}^m$, we compute $\boldsymbol{v} \in \mathbb{C}^m$ such that

$$(\boldsymbol{I} - 2\boldsymbol{P})\boldsymbol{x} = (\boldsymbol{I} - \boldsymbol{v}\boldsymbol{v}^*)\boldsymbol{x} = c\,\boldsymbol{e}_1,$$

for some scalar $c \in \mathbb{C}$.

Put another way: we can, via an efficiently-applicable unitary transform, map $\boldsymbol{x}$ to $\boldsymbol{e}_1$.
The idea now is that one

- first reflects the first column to $\boldsymbol{e}_1$
- then reflects rows 2 through n of column 2 to $\boldsymbol{e}_2$
- then reflects rows 3 through n of column 3 to $\boldsymbol{e}_3$
- $\vdots$

Use of Householder reflectors, II

We now have the following procedure:
Given $\boldsymbol{x} \in \mathbb{C}^m$, we compute $\boldsymbol{v} \in \mathbb{C}^m$ such that

$$(\boldsymbol{I} - 2\boldsymbol{P})\boldsymbol{x} = (\boldsymbol{I} - \boldsymbol{v}\boldsymbol{v}^*)\boldsymbol{x} = c\,\boldsymbol{e}_1,$$

for some scalar $c \in \mathbb{C}$.

Put another way: we can, via an efficiently-applicable unitary transform, map $\boldsymbol{x}$ to $\boldsymbol{e}_1$.
The idea now is that one

- first reflects the first column to $\boldsymbol{e}_1$
- then reflects rows 2 through n of column 2 to $\boldsymbol{e}_2$
- then reflects rows 3 through n of column 3 to $\boldsymbol{e}_3$
- $\vdots$

We expect Householder reflectors to be stable since we are simply applying unitary (well-conditioned) matrices to $\boldsymbol{A}$.

(This is in fact what a typical standard implementations of QR decomposition uses.)

Givens Rotations

In real arithmetic, a Givens rotation is a 2×2 matrix defined by a 3-tuple $(i, j, \theta) \in [m] \times [n] \times [0, 2\pi)$:

$$\boldsymbol{G}_{\{i,j\},\{i,j\}} = \boldsymbol{R}(\theta) \qquad \boldsymbol{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

For real matrices, this accomplishes a rotation of θ radians in the two-dimensional (i, j) plane.

Givens Rotations

In real arithmetic, a Givens rotation is a 2×2 matrix defined by a 3-tuple $(i, j, \theta) \in [m] \times [n] \times [0, 2\pi)$:

$$\boldsymbol{G}_{\{i,j\},\{i,j\}} = \boldsymbol{R}(\theta) \qquad \boldsymbol{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

For real matrices, this accomplishes a rotation of θ radians in the two-dimensional (i, j) plane.

An alternative to Householder reflectors:

- In column 1, use row 1 to eliminate row j via a Givens rotation, for $j = 2, \ldots, n$.
- In column 2, use row 2 to eliminate row j via a Givens rotation, for $j = 3, \ldots, n$.
- $\vdots$

In real arithmetic, a Givens rotation is a 2×2 matrix defined by a 3-tuple $(i, j, \theta) \in [m] \times [n] \times [0, 2\pi)$:

$$\boldsymbol{G}_{\{i,j\},\{i,j\}} = \boldsymbol{R}(\theta) \qquad \boldsymbol{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

For real matrices, this accomplishes a rotation of θ radians in the two-dimensional (i, j) plane.

An alternative to Householder reflectors:

- In column 1, use row 1 to eliminate row j via a Givens rotation, for $j = 2, \ldots, n$.
- In column 2, use row 2 to eliminate row j via a Givens rotation, for $j = 3, \ldots, n$.
- $\vdots$

While both Householder reflectors and Givens rotations are both effective (well-conditioned), Householder reflectors are generally employed for generic dense QR decomposition operations since they require fewer conceptual and computational steps.
(To zero out column 1, we need just one Householder reflector, but $n - 1$ Givens rotations.)

Least-squares solutions

If $\boldsymbol{A} \in \mathbb{C}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{C}^n$, we are interested in computing the least-squares solution to

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

This arises in several situations, e.g., data fitting.

If $\boldsymbol{A} \in \mathbb{C}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{C}^n$, we are interested in computing the least-squares solution to

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

This arises in several situations, e.g., data fitting.

The following is a result we have essentially already proven:

Theorem

Suppose $\boldsymbol{A} \in \mathbb{C}^{m \times n}$ has full column rank (n). Then, for any $\boldsymbol{b} \in \mathbb{C}^n$, there is a unique solution $\boldsymbol{x}$ that solves

$$\arg\min_{\boldsymbol{x} \in \mathbb{C}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2.$$

Furthermore, this solution $\boldsymbol{x}$ is the unique solution to $\boldsymbol{A}^*\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^*\boldsymbol{b}$, and the residual $\boldsymbol{r} := \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$ is orthogonal to $\operatorname{range}(\boldsymbol{A})$.

The system $\boldsymbol{A}^*\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^*\boldsymbol{b}$ is called the *normal equations*.

Computational solutions

While the normal equations are typically useful for analysis, they are typically not used for computation.

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \qquad \Longrightarrow \qquad \boldsymbol{x} = \boldsymbol{R}^{-1}\boldsymbol{Q}^{*}\boldsymbol{b}.$$

In most cases, the QR decomposition is used through the above procedure, largely for stability reasons.

While the normal equations are typically useful for analysis, they are typically not used for computation.

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \qquad \Longrightarrow \qquad \boldsymbol{x} = \boldsymbol{R}^{-1}\boldsymbol{Q}^*\boldsymbol{b}.$$

In most cases, the QR decomposition is used through the above procedure, largely for stability reasons.

One can still use this same idea if $\boldsymbol{A}$ doesn't have full column rank, but has to be more careful.

E.g., $\boldsymbol{Q}$ should only span the range of $\boldsymbol{A}$.

Finally, we note that QR can fail when $\boldsymbol{A}$ does not have full column rank, $\operatorname{rank}(\boldsymbol{A}) = r < n$.

One standard way to address this is by column pivoting.

If doing triangular orthogonalization (Gram-Schmidt-type): At orthogonalization step j, identify column index k satisfying:

$$\|\boldsymbol{a}_k - \boldsymbol{P}_{j-1}\boldsymbol{a}_k\|_2 \geqslant \|\boldsymbol{a}_\ell - \boldsymbol{P}_{j-1}\boldsymbol{a}_\ell\|_2\,, \qquad \ell \geqslant j$$

Then interchange (permute) columns k and j. Repeat at every orthogonalization step. This results in the factorization:

$$\boldsymbol{AP} = \boldsymbol{QR}.$$

Finally, we note that QR can fail when $\boldsymbol{A}$ does not have full column rank, $\operatorname{rank}(\boldsymbol{A}) = r < n$.

One standard way to address this is by column pivoting.

If doing orthogonal triangularization (Householder/Givens): At step j, with $S = \{j, j+1, \ldots, n\}$, identify column index k satisfying:

$$\|\boldsymbol{A}_{S,k}\|_2 \geqslant \|\boldsymbol{A}_{S,\ell}\|_2, \qquad \ell \geqslant j.$$

Then interchange (permute) columns k and j. Repeat at every step. This (again) results in the factorization:

$$\boldsymbol{AP} = \boldsymbol{QR}.$$


Atkinson, Kendall (1989). *An Introduction to Numerical Analysis*. New York: Wiley. ISBN: 978-0-471-62489-9.


Salgado, Abner J. and Steven M. Wise (2022). *Classical Numerical Analysis: A Comprehensive Course*. Cambridge: Cambridge University Press. ISBN: 978-1-108-83770-5. DOI: 10.1017/9781108942607.


Trefethen, Lloyd N. and David Bau (1997). *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics. ISBN: 0-89871-361-7.