# 7: QR Decomposition

*"One algorithmic idea in numerical linear algebra is more important than all the others: QR factorization."*

– Trefethen & Bau p.48

- linear systems of equations
- forward substitution
- QR factorization

# "Easy" solutions

we have encountered two cases for which we can easily compute solutions to

$$Ax = b, \quad \text{with } A \in \mathbb{F}^{n\times n} \quad \text{and} \quad b \in \mathbb{F}^n$$

- "easily" means we **do not** resort to a general purpose formula/algorithm

- diagonal systems

$$\begin{bmatrix} a_1 & & \\ & \ddots & \\ & & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \implies \begin{bmatrix} \frac{1}{a_1} & & \\ & \ddots & \\ & & \frac{1}{a_n} \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

- $A$ is **orthogonal** (p.6-25)

$$Ax = b \implies A^T Ax = A^T b \implies x = A^T b$$

# !Warning!

you will often encounter $x = A^{-1}b$ as a line in an algorithm

- mathematically, this is perfectly legitimate
- **numerically**, however, very rarely is this to be taken literally,

  ```
  >> x = inv(A)*b
  ```

  is almost never the way to go
- when you see $x = A^{-1}b$ it should be interpreted as "$x$ is a solution to $Ax = b$"
- often, for emphasis, the notation $x = A\backslash b$ is used

# Triangular matrices

let $A$ be a square matrix, $A$ is said to be **lower triangular** if $a_{ij} = 0$ when $i < j$

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

if $A^T$ is lower triangular, then $A$ is **upper triangular**

# Forward substitution

if $A$ is lower triangular, solving $Ax = b$ is easy:

$$
\begin{aligned}
x_1 &= b_1/a_{11} \\
x_2 &= (b_2 - a_{21}x_1)/a_{22} \\
x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\
&\vdots \\
x_n &= (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{nn}
\end{aligned}
$$

- requires $a_{ii} \neq 0$, i.e., $A$ must be invertible!
- solve for $x_1$, substitute into expression for $x_2$ and iterate
- for upper triangular matrices, use backward substitution, solve for $x_n$, then $x_{n-1}$

**Note that $A^{-1}$ was not computed**

# Floating point numbers

- digital computers can only store a finite number of bits to store a real number

# Floating point numbers

- digital computers can only store a finite number of bits to store a real number
- as a result, only a finite set of real numbers can be represented
  - the range of numbers expressed is limited (not really an issue)

    IEEE double precision arithmetic range: $2.23 \times 10^{-308} - 1.79 \times 10^{308}$
  - gaps between numbers is an issue

# Floating point numbers

- digital computers can only store a finite number of bits to store a real number
- as a result, only a finite set of real numbers can be represented
  - the range of numbers expressed is limited (not really an issue)

    IEEE double precision arithmetic range: $2.23 \times 10^{-308} - 1.79 \times 10^{308}$
  - gaps between numbers is an issue

**base 10 floating point numbers**

$$z = \textbf{sign bit}(.d_1 d_2 \ldots d_n) \times 10^e$$

- $e$: integer exponent in range $[e_{\min}, e_{\min}]$
- $.d_1 d_2 \ldots d_n$: mantissa

$$d_i \in \mathbb{Z} \quad \text{with} \quad 0 \leq d_i \leq 9 \quad z \neq 0 \iff d_1 \neq 0$$

- $n$: mantissa length (precision)

**Interpretation**

$$z = \pm(d_1 \times 10^{-1} + d_2 \times 10^{-2} + \ldots + d_n \times 10^{-n}) \times 10^{e}$$

**Example with $n = 5$**

$$\begin{aligned} -18.23 &= -(0.18230) \times 10^{2} \\ &= -(1 \cdot 10^{-1} + 8 \cdot 10^{-2} + 2 \cdot 10^{-3} + 3 \cdot 10^{-4} + 0 \cdot 10^{-5}) \cdot 10^{2} \end{aligned}$$

**Interpretation**

$$z = \pm(d_1 \times 10^{-1} + d_2 \times 10^{-2} + \ldots + d_n \times 10^{-n}) \times 10^{e}$$

**Example with $n = 5$**

$$\begin{aligned} -18.23 &= -(0.18230) \times 10^{2} \\ &= -(1 \cdot 10^{-1} + 8 \cdot 10^{-2} + 2 \cdot 10^{-3} + 3 \cdot 10^{-4} + 0 \cdot 10^{-5}) \cdot 10^{2} \end{aligned}$$

**IEEE floating point arithmetic**

- uses base 2 (binary) instead of base 10

$$z = \textbf{sign bit}(.b_1 b_2 \ldots b_n) \times 2^{e} \quad b_i \in \{0, 1\} \quad b_1 = 1 \iff z \neq 0$$

- IEEE standard single precision: $n = 24,\ e_{\min} = -125,\ e_{\min} = 128$
- IEEE standard double precision: $n = 53,\ e_{\min} = -1021,\ e_{\min} = 1024$

# Flop counting

**approximate** cost of an algorithm is the number of floating-point operations required

- a single addition, subtraction, multiplication, division, or square root (of floating point numbers) is **one flop**
- no distinction between real and complex number is made
- time to move data around is ignored
- queuing time in multiprocessor machines also ignored
- most useful to specify flop counts in terms of data dimensions, e.g.,

$$m^3 + 2m^2n + n^2 + 3m + mn + 12$$

- typically we only keep leading terms, so the above is

$$m^3 + 2m^2n$$

# Examples

**vector-vector operations:** $x, y \in \mathbb{R}^n$

- $x + y$: $n$ flops
- $\alpha x$: $n$ flops
- $x^T y$: $2n - 1$ operations $\approx 2n$ flops

# Examples

**vector-vector operations:** $x, y \in \mathbb{R}^n$

- $x + y$: $n$ flops
- $\alpha x$: $n$ flops
- $x^T y$: $2n - 1$ operations $\approx 2n$ flops

**matrix-vector operations:** $x \mapsto Ax$ with $A \in \mathbb{R}^{m \times n}$

- $m$ inner products, $m(2n - 1) \approx 2mn$ flops
- if $A$ is sparse, $2N$ flops with $N = \mathbf{nnz}(A)$
- if $A = UV$ with $U \in \mathbb{R}^{m \times p}$ and $V \in \mathbb{R}^{p \times n}$, $2p(m + n)$ flops

# Examples

**vector-vector operations:** $x, y \in \mathbb{R}^n$

- $x + y$: $n$ flops
- $\alpha x$: $n$ flops
- $x^T y$: $2n - 1$ operations $\approx 2n$ flops

**matrix-vector operations:** $x \mapsto Ax$ with $A \in \mathbb{R}^{m \times n}$

- $m$ inner products, $m(2n - 1) \approx 2mn$ flops
- if $A$ is sparse, $2N$ flops with $N = \mathbf{nnz}(A)$
- if $A = UV$ with $U \in \mathbb{R}^{m \times p}$ and $V \in \mathbb{R}^{p \times n}$, $2p(m + n)$ flops

**matrix-matrix operations:** $AB$ with $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$

- $mp(2n - 1)$ operations ($mp$ inner products), $2mpn$ flops
- sparse $A$ and $B$ gives fewer flops
- structure in general decreases flop counts

# Forward Substitution flop count

**Forward substitution algorithm**

$$
\begin{aligned}
x_1 &= b_1/a_{11} && \texttt{// 1 flop} \\
x_2 &= (b_2 - a_{21}x_1)/a_{22} && \texttt{// 3 flops} \\
x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} && \texttt{// 5 flops} \\
&\vdots \\
x_n &= (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{nn} && \texttt{// } 2n-1 \texttt{ flops}
\end{aligned}
$$

- solving $Ax = b$ with triangular $A$ requires $1 + 3 + 5 + \ldots + (2n-1) = n^2$ flops

# Easy to solve linear systems

- **triangular** $A$**:** cost in $n^2$ flops
- **orthogonal** $A$**:** $x = A^T b$

  computing $A^T b$ is $n$ inner products

  at a cost of $2n^2$ flops
- $A$ **permutation matrices:** 0 flops! (highlights deficiencies in this metric)

# Factor-solve method

a method for solving $Ax = b$ by expressing $A$ as a product of nonsingular matrices

$$A = A_1 A_2 \dots A_k$$

where $A_k$ has a desirable structure

- express the solution in terms of the factorization

$$x \;=\; A^{-1}b \;=\; A_k^{-1} A_{k-1}^{-1} \dots A_1^{-1} b$$

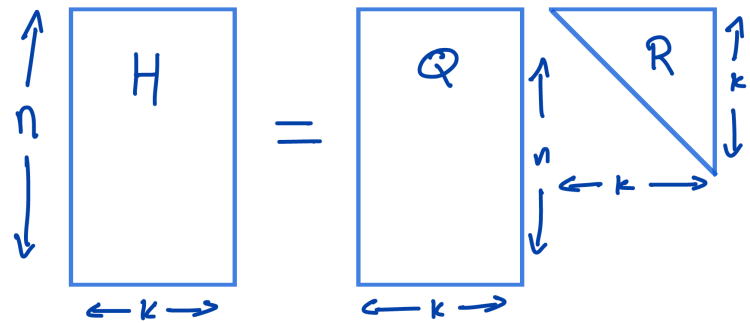
- proceed from right to left:

$$\begin{aligned} z_1 &= A_1^{-1} b \\ z_2 &= A_2^{-1} z_1 \\ &\vdots \\ z_{k-1} &= A_{k-1}^{-1} z_{k-2} \\ x &= A_k^{-1} z_{k-1} \end{aligned}$$

**Note**
although we write $x = A^{-1}b$, what we mean is $x$ is a solution to $Ax = b$

# QR decomposition

if $A \in \mathbb{R}^{n \times k}$ $(n \geq k)$ has linearly independent columns, it can be factored as $A = QR$




- dimensions: $Q \in \mathbb{R}^{n \times k}$, $R \in \mathbb{R}^{k \times k}$
- $R$ is upper-triangular
- $Q^T Q = I_k$

$$\|q_i\|_2 = 1, \quad q_i^T q_j = 0 \quad \text{for } i \neq j$$

- $R_{ii}$'s are non-zero
- most definitions require $R_{ii} > 0$, in this case, $Q$ and $R$ are unique
- this is often referred to as the **reduced QR factorization**

# Example

$$\underbrace{\begin{bmatrix} 1 & -2 & 2 \\ 1 & 5 & 2 \\ -1 & -5 & -4 \\ -1 & 2 & -2 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} 2 & 3 & 5 \\ 0 & 7 & 1 \\ 0 & 0 & \sqrt{2} \end{bmatrix}}_{R}$$

- `>> [Q,R]=QR(A,''econ'')`
- `>> Q, R = np.linalg.qr(a, mode='reduced')`
- in both cases, $R_{ii}$ may be negative – why isn't this a problem?

## Column interpretation

$A = QR$ expressed in terms of the columns (c.f. p.4-19)

$$\left[\begin{array}{c|c|c|c} & & & \\ a_1 & a_2 & \cdots & a_n \\ & & & \end{array}\right] = \left[\begin{array}{c|c|c|c} & & & \\ q_1 & q_2 & \cdots & q_n \\ & & & \end{array}\right] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{1n} \end{bmatrix}$$

- $a_1, \ldots, a_k$ expressed as **linear combinations** of $q_1, \ldots, q_k$

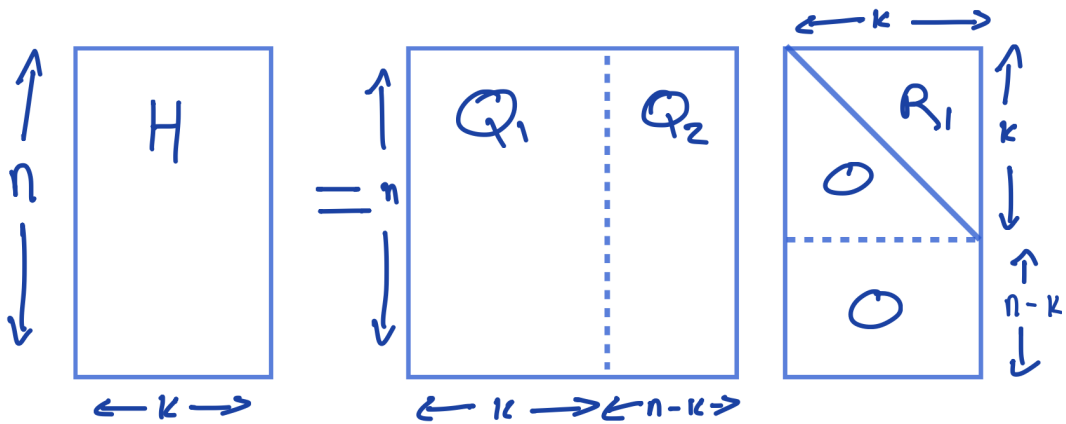
$$\begin{aligned} a_1 &= r_{11} q_1 \\ a_2 &= r_{12} q_1 + r_{22} q_2 \\ a_3 &= r_{13} q_1 + r_{23} q_2 + r_{33} q_3 \\ &\vdots \\ a_n &= r_{1n} q_1 + r_{2n} q_2 + \ldots + r_{nn} q_n \end{aligned}$$

# Full QR decomposition

the **full** QR decomposition of a full-rank matrix $A \in \mathbb{R}^{n \times k}$ is defined as

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \quad \text{where } [Q_1\ Q_2] \in \mathbb{R}^{n \times n} \text{ is orthogonal,}$$

and $R_1$ is upper-triangular and invertible




- $A = Q_1 R_1$ is the QR factorization defined previously
- the additional $n - k$ columns that form $Q_2$ are orthogonal to $\mathbf{range}(A)$
- `>>[Q,R]=qr(A)  %gives full qr factorization`

# Existence and uniqueness

**Theorem**

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

**Theorem**

Every $A \in \mathbb{C}^{m \times n}$ with $m \geq n$ and linearly independent columns, has a unique reduced QR factorization with $r_{jj} > 0$.

- the proof is simple and follows from the Gram-Schmidt algorithm (next week) - Bau & Trefethen, Ch 7

# Solution of $Ax = b$ via QR factorization

**Inverse formulation**

for nonsingular $A$ the inverse can be expressed in terms of the QR factorization

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

**Algorithm:** solve $Ax = b$

1. compute $A = QR$ `// QR factorization`
2. compute $y = Q^T b$ `//matrix-vector multiplication`
3. solve $Rx = y$ for $x$ `// use backward substitution`

**Complexity:** $2n^3$ flops: $\underbrace{2n^3}_{1} + \underbrace{2n^2}_{2} + \underbrace{n^2}_{3}$

# Extension to $AX = B$

may wish to solve $k$ sets of linear equations with the same l.h.s.

$$Ax_1 = b_1, \quad Ax_2 = b_2, \quad \ldots \quad Ax_k = b_k$$

- only need to factorize $A$ once (called factorization caching)
- interpret as solving $AX = B$ with $B \in \mathbb{R}^{n \times k}$ with columns $b_1, \ldots, b_k$
- for $k \ll n$, complexity is roughly $2n^3$

# Orthonormal basis for $\mathbf{range}(A)$

- it is often useful to have access to a basis for a subspace
- the $QR$ decomposition provides an orthonormal basis

# Orthonormal basis for $\mathbf{range}(A)$

- it is often useful to have access to a basis for a subspace
- the $QR$ decomposition provides an orthonormal basis

$$\mathbf{range}(A) = \{Ax \in \mathbb{R}^n \mid x \in \mathbb{R}^n\} \subseteq \mathbb{R}^m$$

# Orthonormal basis for $\mathbf{range}(A)$

- it is often useful to have access to a basis for a subspace
- the $QR$ decomposition provides an orthonormal basis

$$\mathbf{range}(A) = \{Ax \in \mathbb{R}^n \mid x \in \mathbb{R}^n\} \subseteq \mathbb{R}^m$$

if $A$ has linearly independent columns, then the columns of $Q$ are an orthonormal basis for $\mathbf{range}(A)$

$$\begin{aligned} b \in \mathbf{range}(A) &\iff b = Ax \\ &\iff b = QRx \\ &\iff b = Qz \\ &\iff b \in \mathbf{range}(Q) \end{aligned}$$

the result follows from $Q$ having orthonormal columns

# Example: Polynomial interpolation

polynomials are often used to approximate more complex functions as they have many desirable properties

- infinitely times differentiable
- convenient basis
- clear hierarchy

# Example: Polynomial interpolation

polynomials are often used to approximate more complex functions as they have many desirable properties

- infinitely times differentiable
- convenient basis
- clear hierarchy

**Polynomial interpolation**

we consider the problem of finding a polynomial that **exactly** hits a a finite number number of data points

recall the Vandermonde matrix (p.4-28)

$$\begin{bmatrix} 1 & t_1 & t_1^2 & \dots & t_1^n \\ 1 & t_2 & t_2^2 & \dots & t_2^n \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & t_m & t_m^2 & \dots & t_m^n \end{bmatrix}$$

the objective is to find the coefficients of the fixed-degree polynomial

$$p(x) = c_1 + c_2 x + c_3 x^2 + c_4 x^3$$

such that

$$p(-1.1) = b_1, \quad p(-0.4) = b_2, \quad p(0.2) = b_3, \quad p(0.8) = b_4$$

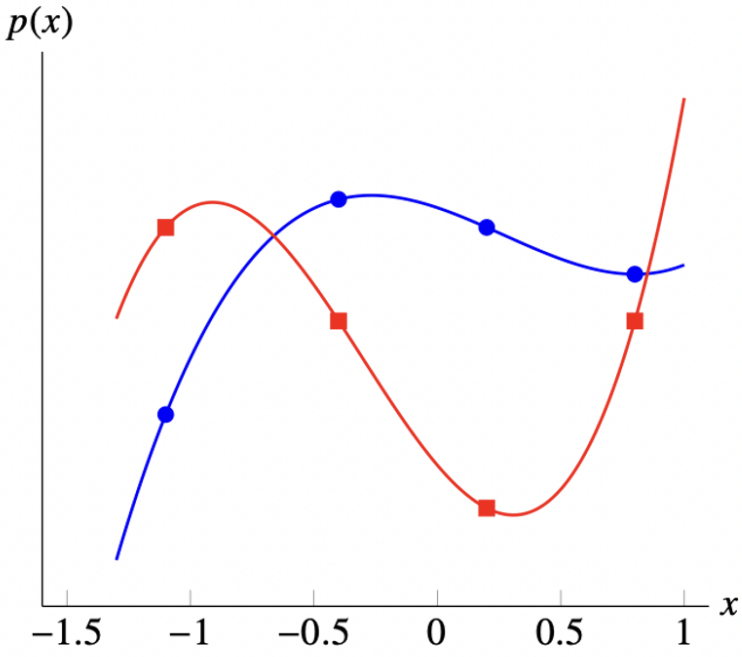
we can express this as a system of linear equations:

$$\underbrace{\begin{bmatrix} 1 & -1.1 & (-1.1)^2 & (-1.1)^3 \\ 1 & -0.4 & (-0.4)^2 & (-0.4)^3 \\ 1 & 0.2 & 0.2^2 & 0.2^3 \\ 1 & 0.8 & 0.8^2 & 0.8^3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}}_{c} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}}_{b}$$

to determine the coefficients, we solve $Ac = b$ for $c$

[Example from Boyd & Vandenberghe]

two cubic polynomials interpreting two sets of data points




$$A^{-1} = \begin{bmatrix} -0.0370 & 0.3492 & 0.7521 & -0.0643 \\ 0.1388 & -1.8651 & 1.6239 & 0.1023 \\ 0.3470 & 0.1984 & -1.4957 & 0.9503 \\ -0.5784 & 1.9841 & -2.1368 & 0.7310 \end{bmatrix}$$

[Image from Boyd & Vandenberghe]

# Further topics

- Gram-Schmidt algorithm (next week)
- Lagrange polynomials
- $LDL$ factorization