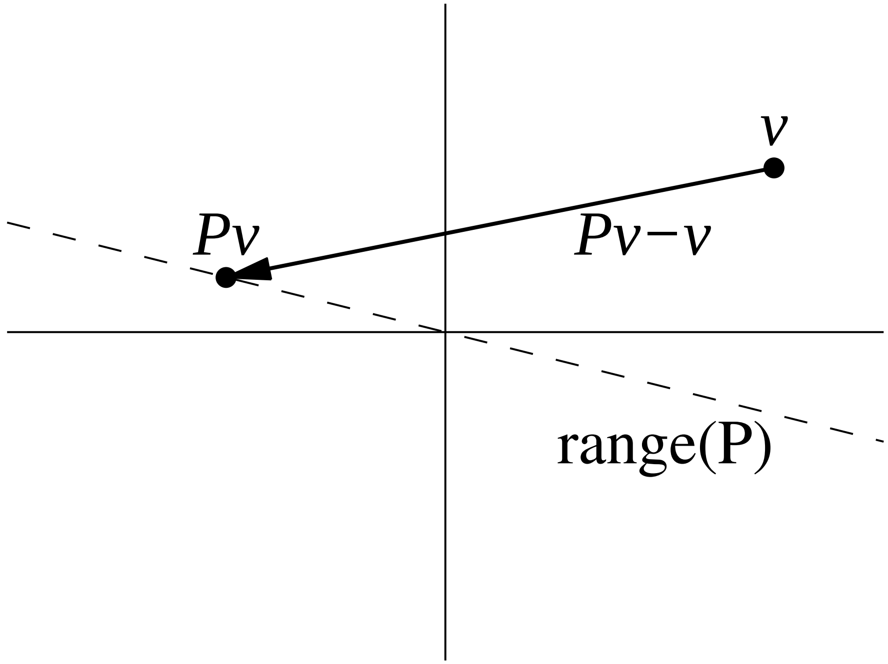
QR Factorization

Projectors

- A *projector* is a square matrix P that satisfies

$$P^2 = P$$

- Not necessarily an *orthogonal projector* (more later)
- If $v \in \mathrm{range}(P)$, then $Pv = v$
  - Since with $v = Px$,
    $Pv = P^2x = Px = v$
- Projection along the line
  $Pv - v \in \mathrm{null}(P)$
  - Since $P(Pv - v) =$
    $P^2v - Pv = 0$



Complementary Projectors

- The matrix $I - P$ is the *complementary projector* to P
- $I - P$ projects on the nullspace of P:
  - If $Pv = 0$, then $(I - P)v = v$, so $\text{null}(P) \subseteq \text{range}(I - P)$
  - But for any v, $(I - P)v = v - Pv \in \text{null}(P)$, so $\text{range}(I - P) \subseteq \text{null}(P)$
  - Therefore

$$\text{range}(I - P) = \text{null}(P)$$

and

$$\text{null}(I - P) = \text{range}(P)$$

Complementary Subspaces

- For a projector P,

$$\mathrm{null}(I - P) \cap \mathrm{null}(P) = \{0\}$$

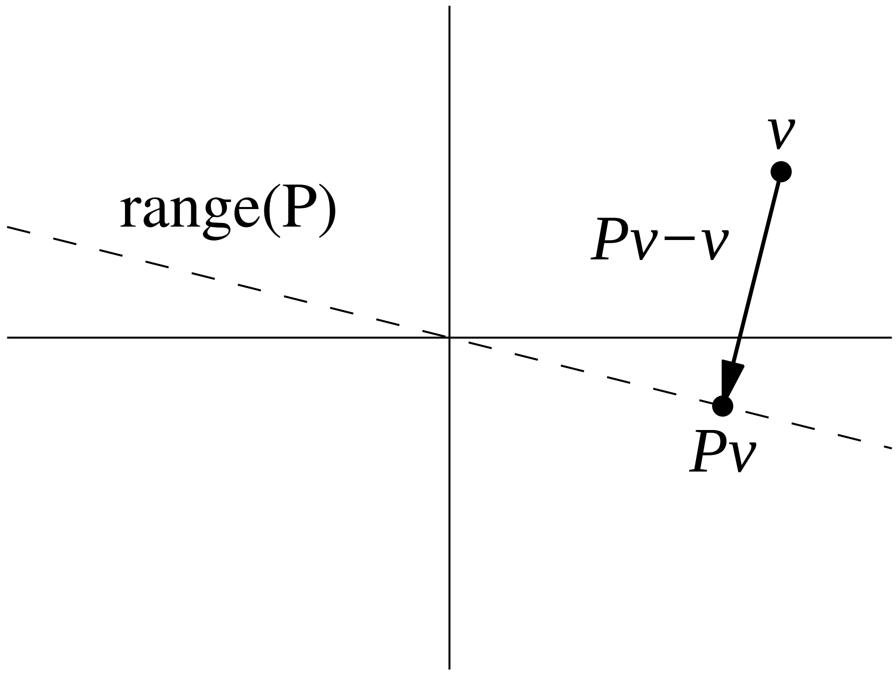
  or

$$\mathrm{range}(P) \cap \mathrm{null}(P) = \{0\}$$

- A projector separates $\mathbb{C}^m$ into two spaces S_1, S_2, with $\mathrm{range}(P) = S_1$ and $\mathrm{null}(P) = S_2$
- P is the projector *onto* S_1 *along* S_2

Orthogonal Projectors

- An *orthogonal projector* projects onto S_1 along S_2, with S_1, S_2 orthogonal
- A projector P is orthogonal $\Longleftrightarrow P = P^*$
- *Proof.* Textbook / Black board



Projection with Orthonormal Basis

- Reduced SVD gives projector for orthonormal columns $\hat{Q}$:

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

- Complement $I - \hat{Q}\hat{Q}^*$ also orthogonal, projects onto space orthogonal to $\mathrm{range}(\hat{Q})$
- Special case 1: Rank-1 Orthogonal Projector (gives component in direction q)

$$P_q = qq^*$$

- Special case 2: Rank $m-1$ Orthogonal Projector (eliminates component in direction q)

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

Projection with Arbitrary Basis

- Project v to $y \in \mathrm{range}(A)$. Then

$$y - v \perp \mathrm{range}(A), \text{ or } a_j^*(y - v) = 0, \forall j$$

- Set $y = Ax$:

$$a_j^*(Ax - v) = 0, \forall j \iff A^*(Ax - v) = 0 \iff A^*Ax = A^*v$$

- A^*A is nonsingular, so

$$x = (A^*A)^{-1}A^*v$$

- Finally, we are interested in the projection $y = Ax = A(A^*A)^{-1}A^*v$, giving the orthogonal projector

$$P = A(A^*A)^{-1}A^*$$

The QR Factorization - Main Idea

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

$$\langle a_1 \rangle \subseteq \langle a_1, a_2 \rangle \subseteq \langle a_1, a_2, a_3 \rangle \subseteq \ldots$$

- This means that (for full rank A),

$$\langle q_1, q_2, \ldots, q_j \rangle = \langle a_1, a_2, \ldots, a_j \rangle \,, \quad \text{for } j = 1, \ldots, n$$

The QR Factorization - Matrix Form

- In matrix form, $\langle q_1, q_2, \ldots, q_j \rangle = \langle a_1, a_2, \ldots, a_j \rangle$ becomes

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

or

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

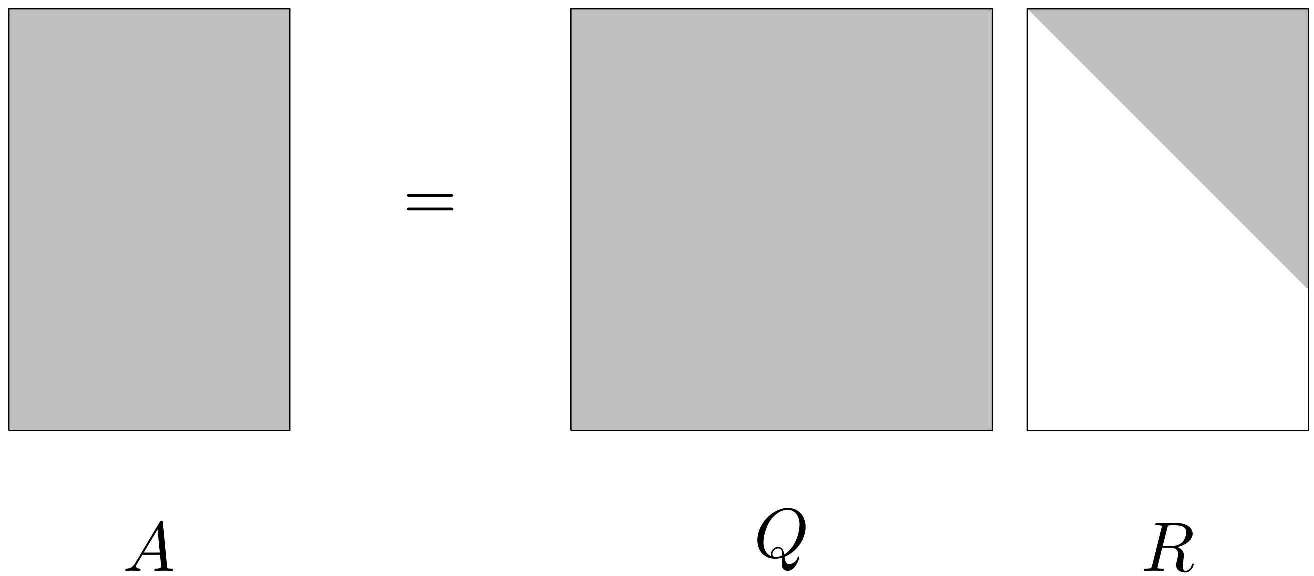
- This is the *reduced QR factorization*
- Add orthogonal extension to $\hat{Q}$ and add rows to $\hat{R}$ to obtain the *full QR factorization*

The Full QR Factorization

- Let A be an $m \times n$ matrix. The full QR factorization of A is the factorization $A = QR$, where

$$Q \text{ is } m \times m \text{ unitary}$$

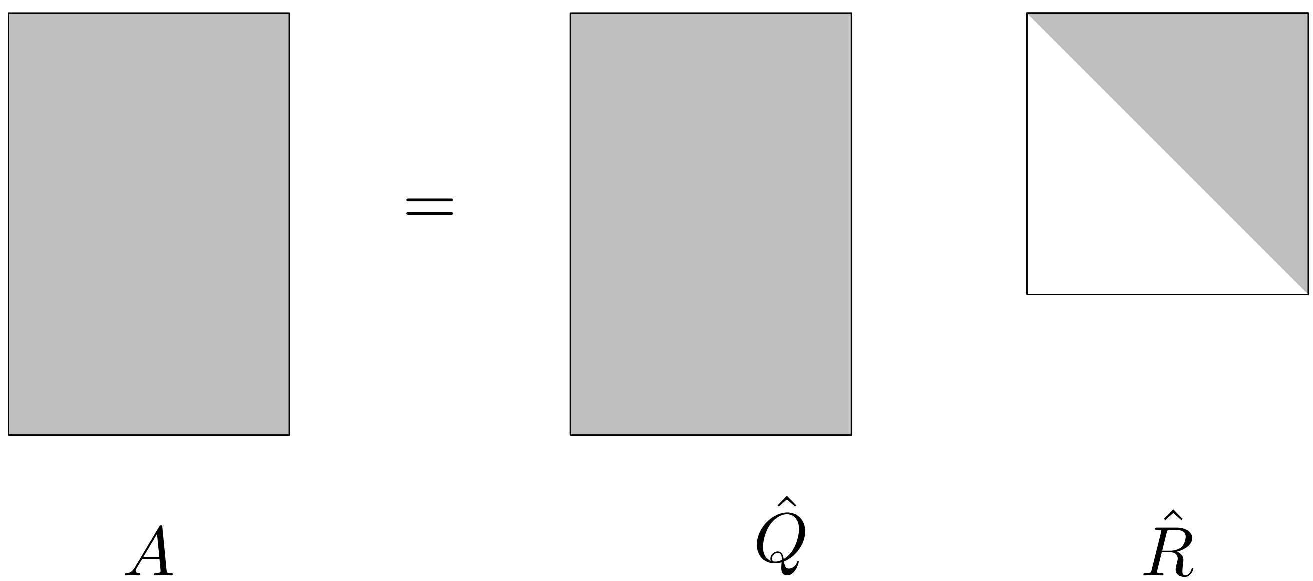
$$R \text{ is } m \times n \text{ upper-triangular}$$



The Reduced QR Factorization

- A more compact representation is the *Reduced QR Factorization* $A = \hat{Q}\hat{R}$, where (for $m \geq n$)

$\hat{Q}$ is $m \times n$ and $\hat{R}$ is $m \times n$



Gram-Schmidt Orthogonalization

- Find new q_j orthogonal to $q_1, \ldots, q_{j-1}$ by subtracting components along previous vectors

$$v_j = a_j - (q_1^* a_j) q_1 - (q_2^* a_j) q_2 - \cdots - (q_{j-1}^* a_j) q_{j-1}$$

- Normalize to get $q_j = v_j / \|v_j\|$

- We then obtain a reduced QR factorization $A = \hat{Q}\hat{R}$, with

$$r_{ij} = q_i^* a_j, \quad (i \neq j)$$

and

$$|r_{jj}| = \|a_j - \sum_{i=1}^{j-1} r_{ij} q_i\|_2$$

Classical Gram-Schmidt

- Straight-forward application of Gram-Schmidt orthogonalization
- Numerically unstable

Algorithm: Classical Gram-Schmidt

for $j = 1$ **to** n

$\quad v_j = a_j$

$\quad$ **for** $i = 1$ **to** $j - 1$

$\quad\quad r_{ij} = q_i^* a_j$

$\quad\quad v_j = v_j - r_{ij} q_i$

$\quad r_{jj} = \|v_j\|_2$

$\quad q_j = v_j / r_{jj}$

Existence and Uniqueness

- Every $A \in \mathbb{C}^{m \times n}$ ($m \geq n$) has a full QR factorization and a reduced QR factorization
- *Proof.* For full rank A, Gram-Schmidt proves existence of $A = \hat{Q}\hat{R}$. Otherwise, when $v_j = 0$ choose arbitrary vector orthogonal to previous q_i. For full QR, add orthogonal extension to Q and zero rows to R.
- Each $A \in \mathbb{C}^{m \times n}$ ($m \geq n$) of full rank has unique $A = \hat{Q}\hat{R}$ with $r_{jj} > 0$
- *Proof.* Again Gram-Schmidt, $r_{jj} > 0$ determines the sign

Gram-Schmidt Orthogonalization

Gram-Schmidt Projections

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

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

where

$$P_j = I - \hat{Q}_{j-1}\hat{Q}_{j-1}^* \text{ with } \hat{Q}_{j-1} = \left[\begin{array}{c|c|c|c} q_1 & q_2 & \cdots & q_{j-1} \end{array}\right]$$

- P_j projects orthogonally onto the space orthogonal to $\langle q_1, \ldots, q_{j-1}\rangle$, and $\operatorname{rank}(P_j) = m - (j-1)$

The Modified Gram-Schmidt Algorithm

- The projection P_j can equivalently be written as

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

where (last lecture)

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

- $P_{\perp q}$ projects orthogonally onto the space orthogonal to q, and $\mathrm{rank}(P_{\perp q}) = m - 1$

- The *Classical Gram-Schmidt* algorithm computes an orthogonal vector by

$$v_j = P_j a_j$$

while the *Modified Gram-Schmidt* algorithm uses

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

Classical vs. Modified Gram-Schmidt

- Small modification of classical G-S gives modified G-S (but see next slide)
- Modified G-S is numerically stable (less sensitive to rounding errors)

Classical/Modified Gram-Schmidt

for $j = 1$ **to** n

$\quad v_j = a_j$

$\quad$ **for** $i = 1$ **to** $j - 1$

$$\left\{ \begin{array}{ll} r_{ij} = q_i^* a_j & \text{(CGS)} \\ r_{ij} = q_i^* v_j & \text{(MGS)} \end{array} \right.$$

$\qquad v_j = v_j - r_{ij} q_i$

$\quad r_{jj} = \|v_j\|_2$

$\quad q_j = v_j / r_{jj}$

Implementation of Modified Gram-Schmidt

- In modified G-S, $P_{\perp q_i}$ can be applied to all v_j as soon as q_i is known
- Makes the inner loop iterations independent (like in classical G-S)

Classical Gram-Schmidt	Modified Gram-Schmidt
for $j = 1$ **to** n	**for** $i = 1$ **to** n
$v_j = a_j$	$v_i = a_i$
for $i = 1$ **to** $j - 1$	**for** $i = 1$ **to** n
$r_{ij} = q_i^* a_j$	$r_{ii} = \|v_i\|$
$v_j = v_j - r_{ij} q_i$	$q_i = v_i / r_{ii}$
$r_{jj} = \|v_j\|_2$	**for** $j = i + 1$ **to** n
$q_j = v_j / r_{jj}$	$r_{ij} = q_i^* v_j$
	$v_j = v_j - r_{ij} q_i$

Example: Classical vs. Modified Gram-Schmidt

- Compare classical and modified G-S for the vectors

$$a_1 = (1, \epsilon, 0, 0)^T, \quad a_2 = (1, 0, \epsilon, 0)^T, \quad a_3 = (1, 0, 0, \epsilon)^T$$

making the approximation $1 + \epsilon^2 \approx 1$

- Classical:

$$v_1 \leftarrow (1, \epsilon, 0, 0)^T, \quad r_{11} = \sqrt{1 + \epsilon^2} \approx 1, \quad q_1 = v_1/1 = (1, \epsilon, 0, 0)^T$$

$$v_2 \leftarrow (1, 0, \epsilon, 0)^T, \quad r_{12} = q_1^T a_2 = 1, \quad v_2 \leftarrow v_2 - 1q_1 = (0, -\epsilon, \epsilon, 0)^T$$

$$r_{22} = \sqrt{2}\epsilon, \quad q_2 = v_2/r_{22} = (0, -1, 1, 0)^T/\sqrt{2}$$

$$v_3 \leftarrow (1, 0, 0, \epsilon)^T, \quad r_{13} = q_1^T a_3 = 1, \quad v_3 \leftarrow v_3 - 1q_1 = (0, -\epsilon, 0, \epsilon)^T$$

$$r_{23} = q_2^T a_3 = 0, \quad v_3 \leftarrow v_3 - 0q_2 = (0, -\epsilon, 0, \epsilon)^T$$

$$r_{33} = \sqrt{2}\epsilon, \quad q_3 = v_3/r_{33} = (0, -1, 0, 1)^T/\sqrt{2}$$

Example: Classical vs. Modified Gram-Schmidt

- Modified:

$$v_1 \leftarrow (1, \epsilon, 0, 0)^T, \quad r_{11} = \sqrt{1+\epsilon^2} \approx 1, \quad q_1 = v_1/1 = (1, \epsilon, 0, 0)^T$$

$$v_2 \leftarrow (1, 0, \epsilon, 0)^T, \quad r_{12} = q_1^T v_2 = 1, \quad v_2 \leftarrow v_2 - 1q_1 = (0, -\epsilon, \epsilon, 0)^T$$

$$r_{22} = \sqrt{2}\epsilon, \quad q_2 = v_2/r_{22} = (0, -1, 1, 0)^T/\sqrt{2}$$

$$v_3 \leftarrow (1, 0, 0, \epsilon)^T, \quad r_{13} = q_1^T v_3 = 1, \quad v_3 \leftarrow v_3 - 1q_1 = (0, -\epsilon, 0, \epsilon)^T$$

$$r_{23} = q_2^T v_3 = \epsilon/\sqrt{2}, \quad v_3 \leftarrow v_3 - r_{23} q_2 = (0, -\epsilon/2, -\epsilon/2, \epsilon)^T$$

$$r_{33} = \sqrt{6}\epsilon/2, \quad q_3 = v_3/r_{33} = (0, -1, -1, 2)^T/\sqrt{6}$$

- Check Orthogonality:
  - Classical: $q_2^T q_3 = (0, -1, 1, 0)(0, -1, 0, 1)^T/2 = 1/2$
  - Modified: $q_2^T q_3 = (0, -1, 1, 0)(0, -1, -1, 2)^T/\sqrt{12} = 0$

Operation Count

- Count number of floating points operations – “flops” – in an algorithm
- Each $+$, $-$, $*$, $/$, or $\sqrt{\ }$ counts as one flop
- No distinction between real and complex
- No consideration of memory accesses or other performance aspects

Operation Count - Modified G-S

- Example: Count all $+$, $-$, $*$, $/$ in the Modified Gram-Schmidt algorithm (not just the leading term)

(1) **for** $i = 1$ **to** n

(2) $\quad v_i = a_i$

(3) **for** $i = 1$ **to** n

(4) $\quad r_{ii} = \|v_i\|$ — m multiplications, $m-1$ additions

(5) $\quad q_i = v_i / r_{ii}$ — m divisions

(6) $\quad$ **for** $j = i + 1$ **to** n

(7) $\qquad r_{ij} = q_i^* v_j$ — m multiplications, $m-1$ additions

(8) $\qquad v_j = v_j - r_{ij} q_i$ — m multiplications, m subtractions

Operation Count - Modified G-S

- The total for each operation is

$$\#A = \sum_{i=1}^{n} \left(m - 1 + \sum_{j=i+1}^{n} m - 1 \right) = n(m-1) + \sum_{i=1}^{n} (m-1)(n-i) =$$

$$= n(m-1) + \frac{n(n-1)(m-1)}{2} = \frac{1}{2} n(n+1)(m-1)$$

$$\#S = \sum_{i=1}^{n} \sum_{j=i+1}^{n} m = \sum_{i=1}^{n} m(n-i) = \frac{1}{2} mn(n-1)$$

$$\#M = \sum_{i=1}^{n} \left(m + \sum_{j=i+1}^{n} 2m \right) = mn + \sum_{i=1}^{n} 2m(n-i) =$$

$$= mn + \frac{2mn(n-1)}{2} = mn^2$$

$$\#D = \sum_{i=1}^{n} m = mn$$

Operation Count - Modified G-S

and the total flop count is

$$\frac{1}{2}n(n+1)(m-1) + \frac{1}{2}mn(n-1) + mn^2 + mn =$$
$$2mn^2 + mn - \frac{1}{2}n^2 - \frac{1}{2}n \sim 2mn^2$$

- The symbol $\sim$ indicates asymptotic value as $m, n \longrightarrow \infty$ (leading term)
- Easier to find just the leading term:
  - Most work done in lines (7) and (8), with $4m$ flops per iteration
  - Including the loops, the total becomes

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} 4m = 4m \sum_{i=1}^{n} (n-i) \sim 4m \sum_{i=1}^{n} i = 2mn^2$$

Householder Reflectors

Gram-Schmidt as Triangular Orthogonalization

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

$$\left[\begin{array}{c|c|c|c} v_1 & v_2 & \cdots & v_n \end{array}\right] \begin{bmatrix} \frac{1}{r_{11}} & \frac{-r_{12}}{r_{11}} & \frac{-r_{13}}{r_{11}} & \cdots \\ & 1 & & \\ & & 1 & \\ & & & \ddots \end{bmatrix} = \left[\begin{array}{c|c|c|c} q_1 & v_2^{(2)} & \cdots & v_n^{(2)} \end{array}\right]$$

- After all the steps we get a product of triangular matrices

$$A \underbrace{R_1 R_2 \cdots R_n}_{\hat{R}^{-1}} = \hat{Q}$$

- "Triangular orthogonalization"

Householder Triangularization

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

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

- After all the steps we get a product of orthogonal matrices

$$\underbrace{Q_n \cdots Q_2 Q_1}_{Q^*} A = R$$

- “Orthogonal triangularization”

Introducing Zeros

- Q_k introduces zeros below the diagonal in column k
- Preserves all the zeros previously introduced

$$
\underset{A}{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}
\xrightarrow{Q_1}
\underset{Q_1 A}{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}
\xrightarrow{Q_2}
\underset{Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} \end{bmatrix}}
\xrightarrow{Q_3}
\underset{Q_3 Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & & \boldsymbol{\times} \\ & & \mathbf{0} \\ & & \mathbf{0} \end{bmatrix}}
$$

Householder Reflectors

- 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 $(m-k+1) \times (m-k+1)$

- Create *Householder reflector* F that introduces zeros:

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

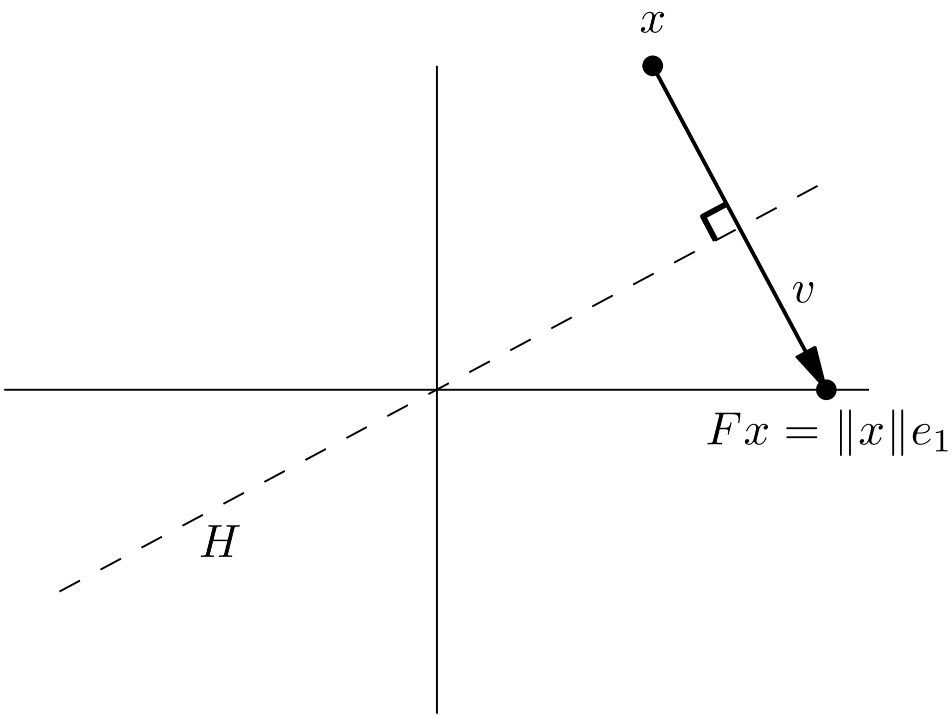
Householder Reflectors

- Idea: Reflect across hyperplane H orthogonal to $v = \|x\|e_1 - x$, by the unitary matrix

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

- Compare with projector

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

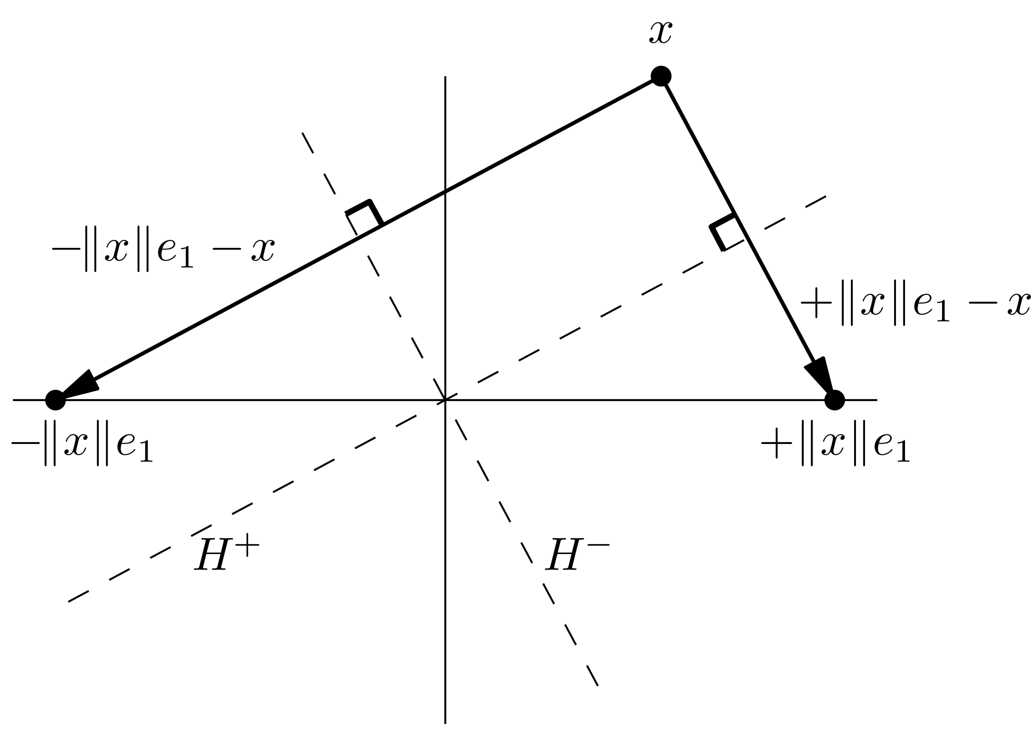


Choice of Reflector

- We can choose to reflect to any multiple z of $\|x\|e_1$ with $|z| = 1$
- Better numerical properties with large $\|v\|$, for example

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

- Note: $\text{sign}(0) = 1$, but in MATLAB, `sign(0)==0`



The Householder Algorithm

- Compute the factor R of a QR factorization of $m \times n$ matrix A ($m \geq n$)
- Leave result in place of A, store reflection vectors v_k for later use

Algorithm: Householder QR Factorization

for $k = 1$ **to** n

$$x = A_{k:m,k}$$

$$v_k = \operatorname{sign}(x_1)\|x\|_2 e_1 + x$$

$$v_k = v_k / \|v_k\|_2$$

$$A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^* A_{k:m,k:n})$$

Applying or Forming Q

- Compute $Q^*b = Q_n \cdots Q_2 Q_1 b$ and $Qx = Q_1 Q_2 \cdots Q_n x$ implicitly
- To create Q explicitly, apply to $x = I$

Algorithm: Implicit Calculation of Q^*b

for $k = 1$ **to** n

$$b_{k:m} = b_{k:m} - 2v_k(v_k^* b_{k:m})$$

Algorithm: Implicit Calculation of Qx

for $k = n$ **downto** 1

$$x_{k:m} = x_{k:m} - 2v_k(v_k^* x_{k:m})$$

Operation Count - Householder QR

- Most work done by

$$A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^* A_{k:m,k:n})$$

- Operations per iteration:
  - $2(m-k)(n-k)$ for the dot products $v_k^* A_{k:m,k:n}$
  - $(m-k)(n-k)$ for the outer product $2v_k(\cdots)$
  - $(m-k)(n-k)$ for the subtraction $A_{k:m,k:n} - \cdots$
  - $4(m-k)(n-k)$ total

- Including the outer loop, the total becomes

$$\sum_{k=1}^{n} 4(m-k)(n-k) = 4\sum_{k=1}^{n}(mn - k(m+n) + k^2)$$

$$\sim 4mn^2 - 4(m+n)n^2/2 + 4n^3/3 = 2mn^2 - 2n^3/3$$

Givens Rotations

- Alternative to Householder reflectors
- A Givens rotation $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ rotates $x \in \mathbb{R}^2$ by θ
- To set an element to zero, choose $\cos\theta$ and $\sin\theta$ so that

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

or

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

Givens QR

- Introduce zeros in column from bottom and up

$$
\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}
\xrightarrow{(3,4)}
\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}
\xrightarrow{(2,3)}
\begin{bmatrix} \times & \times & \times \\ \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times \end{bmatrix}
\xrightarrow{(1,2)}
$$

$$
\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times \\ & \times & \times \end{bmatrix}
\xrightarrow{(3,4)}
\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} \end{bmatrix}
\xrightarrow{(2,3)}
\begin{bmatrix} \times & \times & \times \\ & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} \\ & & \times \end{bmatrix}
\xrightarrow{(3,4)} R
$$

- Flop count $3mn^2 - n^3$ (or 50% more than Householder QR)

Least Square Problems

The Linear Least Squares Problem

- In general, $Ax = b$ with $m > n$ has no solution
- Instead, try to minimize the *residual* $r = b - Ax$
- With the 2-norm we obtain the linear *least squares problem* (LSP):

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

- The minimizer x is the solution to the *normal equations*

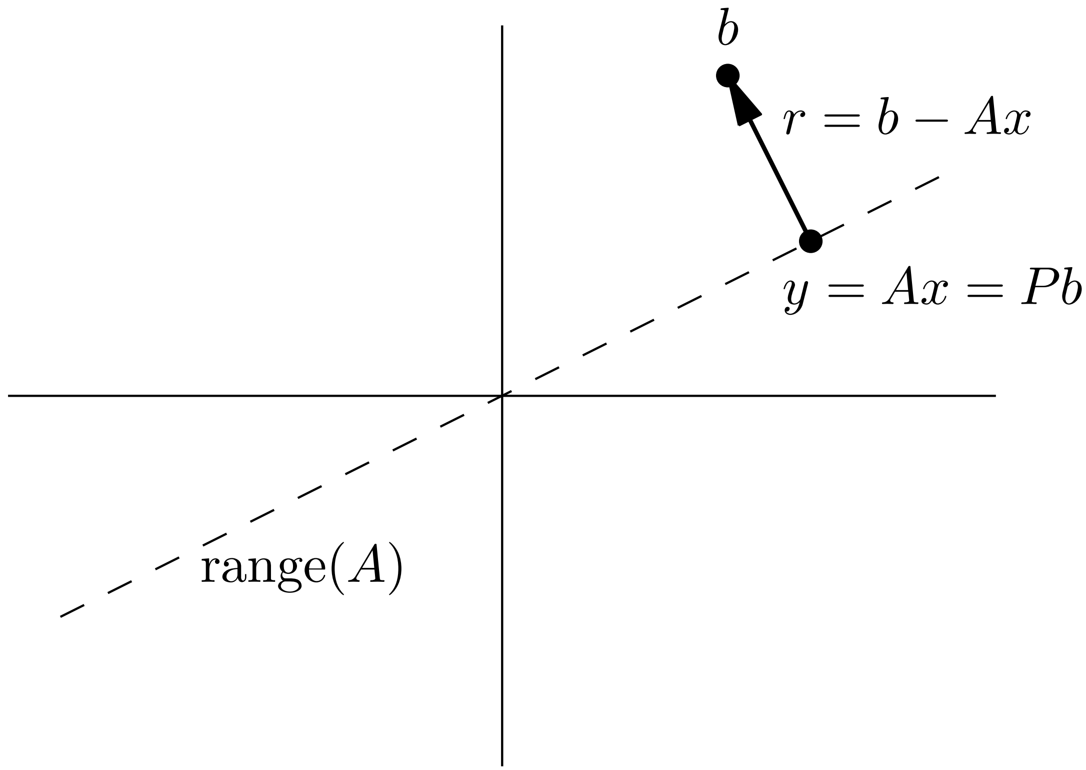
$$A^* A x = A^* b$$

or, in terms of the *pseudoinverse* A^+:

$$x = A^+ b, \qquad \text{where } A^+ = (A^* A)^{-1} A^* \; \in \; \mathbb{C}^{n,m}$$

Geometric Interpretation

- Find the point Ax in $\mathrm{range}(A)$ closest to b
- This x will minimize the 2-norm of $r = b - Ax$
- $Ax = Pb$ where P is an orthogonal projector onto $\mathrm{range}(A)$, so the residual must be orthogonal to $\mathrm{range}(A)$



Solving the LSP – 1. Normal Equations

- If A has full rank, A^*A is square, hermitian positive definite system
- Solve by *Cholesky factorization* (Gaussian elimination)

Algorithm: Least Squares via Normal Equations

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

- Work $\sim$ Forming A^*A + Cholesky $\sim mn^2 + n^3/3$ flops
- Fast, but sensitive to rounding errors

Solving the LSP – 2. QR Factorization

- Using $A = \hat{Q}\hat{R}$, b can be projected onto $\operatorname{range}(A)$ by $P = \hat{Q}\hat{Q}^*$
- Insert into $Ax = b$ to get $\hat{Q}\hat{R}x = \hat{Q}\hat{Q}^*b$, or $\hat{R}x = \hat{Q}^*b$

Algorithm: Least Squares via QR Factorization

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

- Work $\sim$ QR Factorization $\sim 2mn^2 - 2n^3/3$ flops
- Good stability, relatively fast, used in MATLAB's "backslash" $\backslash$

Solving the LSP – 3. SVD

- Using $A = \hat{U}\hat{\Sigma}V^*$, b can be projected onto $\mathrm{range}(A)$ by $P = \hat{U}\hat{U}^*$
- Insert into $Ax = b$ to get $\hat{U}\hat{\Sigma}V^*x = \hat{U}\hat{U}^*b$, or $\hat{\Sigma}V^*x = \hat{U}^*b$

Algorithm: Least Squares via SVD

1. Compute the reduced SVD $A = \hat{U}\hat{\Sigma}V^*$
2. Compute the vector $\hat{U}^*b$
3. Solve the diagonal system $\hat{\Sigma}w = \hat{U}^*b$ for w
4. Set $x = Vw$

- Work $\sim$ SVD $\sim 2mn^2 + 11n^3$ flops
- Very good stability properties, use if A is close to rank-deficient