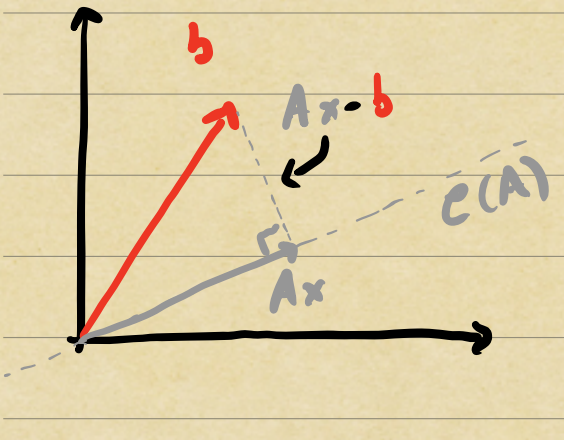
The QR factorization

$$x_* = \operatorname{argmin} \|Ax - b\|_2$$



$$x_* = (A^T A)^{-1} A^T b \qquad A$$

$$= V \tilde{\Sigma}^{-1} \tilde{U}^* b \qquad U \hat{\Sigma} \tilde{V}^*$$

$$= R^{-1} Q^* b \qquad QR$$

$$\underset{m \times n}{\hat{A}} = \underset{m \times n}{Q} \underset{n \times n}{R}$$

Q: orthogonal columns; R: upper triangular

Typically method-of-choice for least-squares:

e.g. $x_* = A \backslash b$ — Matlab, Julia

uses QR-based solvers when A is rectangular.

How to compute $A = QR$?

Idea 1 $\Rightarrow$ Orthogonalize columns of A (Gram-Schmidt)

Idea 2 $\Rightarrow$ Triangularize A w/ orthogonal row ops (Householder/Givens)

Gram-Schmidt (Classical | Modified)

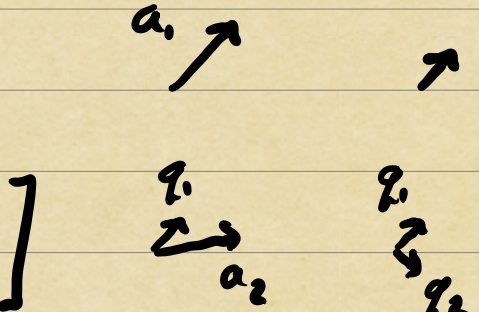
Idea 1: orthogonalize columns of A sequentially

$$q_1 = a_1 / r_{11}$$

$$q_2 = \frac{1}{r_{22}} \left[a_2 - \underbrace{(q_1^* a_2)}_{r_{12}} q_1 \right]$$

$$\vdots$$

$$q_n = \frac{1}{r_{nn}} \left[a_n - \sum_{i=1}^{n-1} \underbrace{(q_i^* a_n)}_{r_{in}} q_i \right]$$



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

In matrix notation:

$$\begin{bmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} r_{11}^{-1} & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} = \begin{bmatrix} | & | & & | \\ q_1 & a_2 & \cdots & a_n \\ | & | & & | \end{bmatrix}$$

$$\begin{bmatrix} | & | & & | \\ q_1 & a_2 & \cdots & a_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} 1 & r_{12}/r_{22} & & \\ & r_{22} & & \\ & & 1 & \\ & & & \ddots \end{bmatrix} = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & a_n \\ | & | & & | \end{bmatrix}$$

$$A R_1 R_2 \cdots R_n = \begin{bmatrix} | & | & & | \\ q_1 & \cdots & & q_n \\ | & | & & | \end{bmatrix}$$

Q

Sequence of upper triangular transformations from the <u>right</u> orthogonalizes col's of A.

Invert: $R = R_n^{-1} \cdots R_2^{-1} R_1^{-1}$

Unique when $r_{jj} > 0$

$$\begin{bmatrix} | & | & & | \\ a_1 & \cdots & & a_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ q_1 & \cdots & & q_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \cdot & | & | & | \\ & \cdot & | & | \\ & & \cdot & | \\ & & & \cdot \end{bmatrix}$$

A Q R

Classical Gram-Schmidt can be extremely unstable, in part, due to loss of orthogonality in columns of Q. Errors compound in rightmost columns of Q when A is ill-conditioned (cancellation).

Modified Gram-Schmidt rearranges order of operations to improve stability.

Initialize $v_k = a_k$ for $k = 1, \ldots, n$

CGS

for $j = 1$ to n
  for $i = 1$ to $j-1$ (skipped if $j-1 < 1$)
    $r_{ij} = q_i^* a_j$
    $v_j = v_j - r_{ij}$
  $r_{jj} = \|v_j\|$
  $q_j = v_j / r_{jj}$

MGS

for $i = 1$ to n
  $r_{ii} = \|v_i\|$
  $q_i = v_i / r_{ii}$
  for $j = i+1$ to n
    $r_{ij} = q_i^* v_j$
    $v_j = v_j - r_{ij} q_i$

For MGS, loss of orthogonality is controlled

$$\|Q^*Q - I\| \leq C_n \underbrace{\left[\frac{\sigma_1}{\sigma_n}\right]}_{\kappa_2(A)} \varepsilon_{\text{mach}}$$

Note: CGS is sometimes useful in "streaming" settings where each column of A is processed only once.

Note: Loss-of-orthogonality can often be mitigated by a 2nd pass over the columns, e.g. running GS ×2. See the famous "Twice-is-enough" principle due to Kahan & Parlett.

Householder Triangularization

Idea 2: Triangularize A

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{Q_1} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{Q_2} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{Q_3} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix}$$

$$\underbrace{Q_n \cdots Q_2 Q_1}_{\tilde{Q}^{-1} = \tilde{Q}^*} A = R$$

R ← upper triangular

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

$$m \begin{bmatrix} | & & | \\ a_1 & \cdots & a_n \\ | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_m \\ | & | & & | \end{bmatrix} \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}$$

n $\qquad$ m

Note: $\tilde{Q}$ is now $m \times m$ and $\tilde{R}$ is $m \times n$

"Full QR factorization"

To recover "thin" QR, drop columns $q_{n+1}, \ldots, q_m$ and $m-n$ "zero rows" of $\tilde{R}$.

How to construct $Q_1, \ldots, Q_n$?

Today $\Rightarrow$ Reflections (Householder)

$\Rightarrow$ Rotations (Givens)

Ex. 10.4 Trefethen & Bau

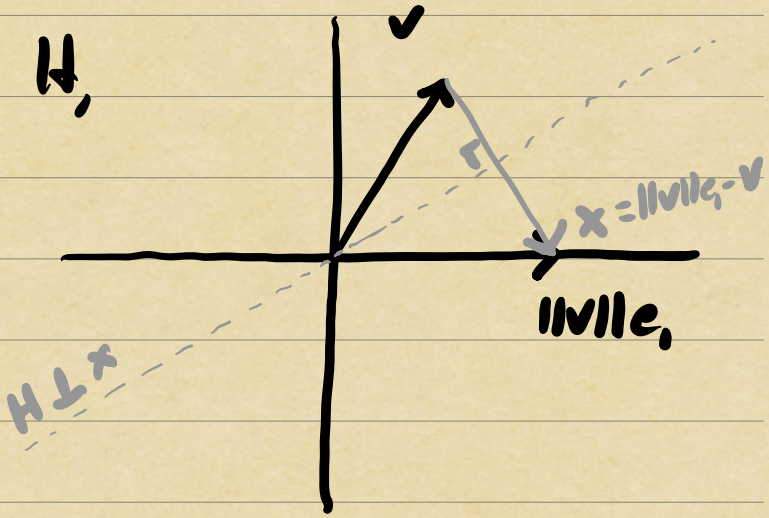
$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix}, \qquad Fv = \begin{bmatrix} \|v\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \|v\| e_1$$

F ↑ reflector

To project v onto H,

$$P_H = v - \frac{(x^* v)}{(x^* x)} x$$

$$= \underbrace{[I - x x^*]}_{\text{orthogonal projection onto orthogonal complement of span}(x)} v$$



To reflect v across H, we need to go twice as far:

$$Fv = v - 2\frac{(x^* v)}{(x^* x)} x = \underbrace{\left[I - 2\frac{x x^*}{x^* x}\right]}_{= F} v$$

Householder Reflector

In practice we take

$$x = -\text{sign}(v_1)\|v\| e_1 - v$$

Orientation of x isn't important, so can take

$$x = \text{sign}(v_1)\|v\|e_1 + v$$

too (as is done in textbook).

Householder QR

for $k = 1$ to n

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

$x_k = \text{sign}(v_1)\|v\|_2 e_1 + v$

$x_k = x_k / \|x_k\|_2$

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

Notice, $Q_1, Q_2, \ldots, Q_n$ are not stored explicitly! Only Householder reflectors are stored. We can apply Q and Q^* to vectors, by applying the reflections:

Implicit Q^*b

for $k = 1$ to n

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

Implicit Qv

for $k = n$ to 1

$v_{k:m} = v_{k:m} - 2x_k(x_k^* v_{k:m})$

Counting FLOPS

* flops QR $\sim 2mn^2 - \frac{2}{3}n^2$ flops

* flops MGS $\sim 2mn^2$

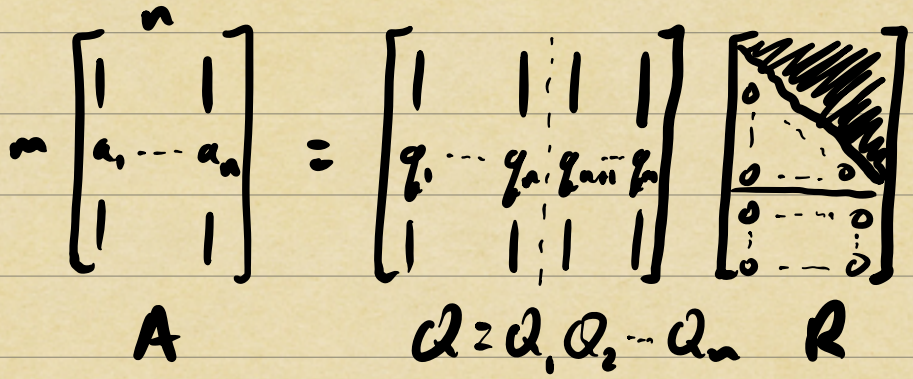
Householder QR is arithmetically cheaper than MGS b/c it works on successively smaller $(m-k+1) \times (n-k+1)$ submatrices.

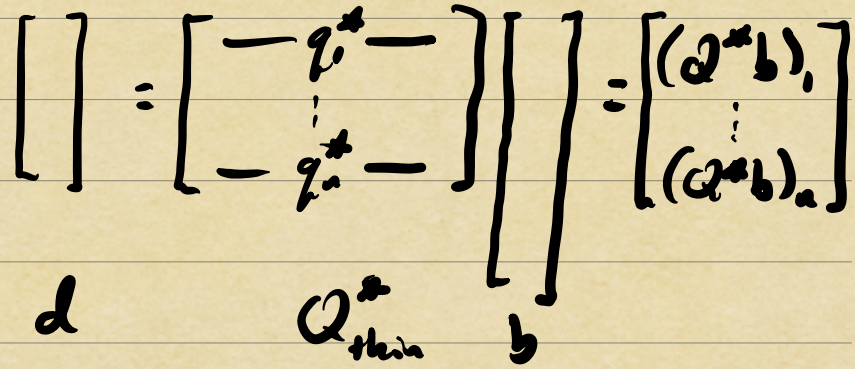
Stability

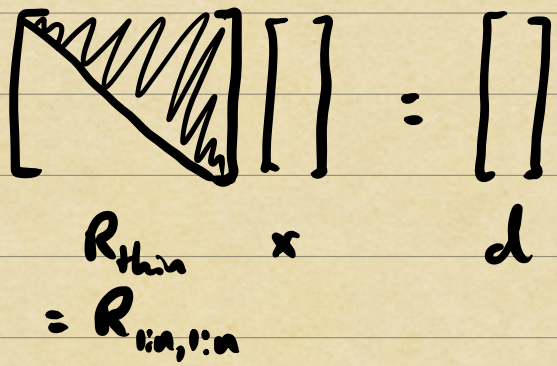
Householder QR is backward stable if we store Q implicitly as product of Householder transformations.

Thm $A = QR$ computed by Householder triangularization in F.P.A., with computed factors $\tilde{R}$ and $\tilde{Q} = \tilde{Q}_1 \tilde{Q}_2 \cdots \tilde{Q}_n$. Then, $\tilde{Q}\tilde{R} = A + \delta A$, $\|\delta A\| \leq C_{m,n} \|A\| \varepsilon_{mach}$

Householder : Least-Squares

1) 

2) Construct 

3) Solve 

$= R_{1:n,1:n}$

Householder Least-Squares is back-stable

↓ computed soln. solves "nearby" least-squares solve

$$\tilde{x} = \operatorname{argmin}_x \|(A + \Delta A)x - b\|, \quad \|\Delta A\| \le C_{m,n} \|A\| \varepsilon_{mach}$$