Conditioning of LS problems

- $A$ is $m$ by $n$ and has full rank
- $x$ is LS solution with residual $r = b - Ax$
- $x + \delta x$ is solution to $\min \| (A + \delta A)(x + \delta x) - (b + \delta b) \|$

Conditioning of the LS problem

$$\kappa_{LS} \leq \frac{2\kappa(A)}{\cos \theta} + \tan \theta \cdot \kappa^2(A), \quad \sin \theta = \frac{\| r \|}{\| b \|}$$
QR (and SVD) are backward stable; e.g. they lead a solution $\tilde{x}$ minimizing $\|(A + \delta A)\tilde{x} - (b + \delta b)\|$ with
\[
\max \left( \frac{\|\delta A\|}{\|A\|}, \frac{\|\delta b\|}{\|b\|} \right) = O(\epsilon_m)
\]
It follows that the QR solution obeys
\[
\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_m) \cdot \kappa_{LS}
\]
Normal equations are not as accurate

\[(A^T A)x = A^T b\]

Accuracy depends on \(\kappa(A^T A) = \kappa^2(A)\).

Error always bounded by \(\kappa^2(A) \cdot O(\varepsilon_m)\), not by \(\kappa_{LS}(A) \cdot O(\varepsilon_m)\).

We expect that the normal equations can lose twice as many digits of accuracy as QR or SVD-based methods.
Solving normal equations is not necessarily backward stable: $\tilde{x}$ does not generally minimize $\| (A + \delta A)\tilde{x} - (b + \delta b) \|$ for small $\delta A$ and $\delta b$.

Still, when $\kappa(A)$ is small, we expect the normal equations to be as accurate as QR or SVD.

Since solving the normal equations is the fastest way, method of choice when $A$ is well-conditioned.
% Problem size

n = 34; m = 4*n;

% Make singular values

j = 0:n-1;
sigma = 2.^(-j);

% Make m by n matrix with prescribed singular values

X = randn(n);
[V,R] = qr(X);
X = randn(m);
[U,R] = qr(X,0);
A = U(:,1:n)*diag(sigma)*V';

% Check conditioning

cond(A)
ans =

    8.5899e+09

\sigma(1)/\sigma(n)

ans =

    8.5899e+09

% Make residuals and b

x = randn(n,1);
y = A*x;
theta = 1e-6;

r = U(:,n+1);
r = tan(theta)*norm(y)*U(:,n+1);
b = y+r;
% Solve via QR

\[ [Q, R] = \text{qr}(A, 0); \]
\[ x_{q\text{r}} = R \backslash (Q' \ast b); \]
\[ \text{norm}(x - x_{q\text{r}}) / \text{norm}(x) \]

\text{ans} =

\[ 2.1308e-05 \]

% Solve via normal equations

\[ x_{\text{chol}} = (A' \ast A) \backslash (A' \ast b); \]
\text{Warning: Matrix is close to singular or badly scaled.}
\text{Results may be inaccurate. RCOND = 1.693546e-18.}
\[ \text{norm}(x - x_{\text{chol}}) / \text{norm}(x) \]

\text{ans} =

\[ 2.1772 \]
% Solve via SVD

[U,S,V] = svd(A,0);
xsvd = V*(S\U' * b);
norm(x - xsvd)/norm(x)

ans =

2.1305e-05

% Matlab solve

xmat = A\b;
norm(x - xmat)/norm(x)

ans =

3.4615e-05

Matlab is not using the normal equations! Uses QR with additional pivoting.
Stability for well conditioned problems

% Problem size

n = 50; m = 200;

% Make matrix

A = randn(m,n);

% Check conditioning

cond(A)

ans =

    2.8575

% Make b and the residuals

x = randn(n,1);
y = A*x;
\[ [U, R] = qr(A); \]
\[ r = U(:, (n+1):m) \times \text{randn}(m-n, 1); \]
\[ r = r / \text{norm}(r); \]
\[ b = y + \text{norm}(y) \times r; \]
% Solve via QR

\[
[Q, R] = \text{qr}(A, 0);
\]
\[
x_{QR} = R \backslash (Q' \ast b);
\]
\[
\text{norm}(x - x_{QR})/\text{norm}(x)
\]

ans =

\[8.8049e-16\]

% Solve via normal equations

\[
x_{\text{chol}} = (A' \ast A) \backslash (A' \ast b);
\]
\[
\text{norm}(x - x_{\text{chol}})/\text{norm}(x)
\]

ans =

\[1.1709e-15\]

% Solve via SVD
\[
[U, S, V] = \text{svd}(A, 0);
\]
\[
xsvd = V \ast (S \backslash U' \ast b);
\]
\[
norm(x - xsvd) / norm(x)
\]

\[
\text{ans} = 2.3501e-15
\]

% Matlab solve

\[
xmat = A \backslash b;
\]
\[
norm(x - xmat) / norm(x)
\]

\[
\text{ans} = 9.5215e-16
\]
Stability of Householder triangularization

\begin{verbatim}
  m = 100; n = 50; % Problem size
  R = triu(randn(n)); % Make R
  [Q, Junk] = qr(randn(m,n),0); % Make Q
  A = Q*R; % Set A to be the product QR
  [Q2, R2] = qr(A,0); % Compute the QR decomposition of A
  A2 = Q2*R2;
  norm(A-A2)/norm(A) % Check backward stability
\end{verbatim}

\texttt{ans =}

9.8508e-16

Householder triangularization seems backward stable!
More on stability

```
m = 100; n = 50;  % Problem size
R = triu(randn(n));  % Make R
[Q, Junk] = qr(randn(m,n),0);  % Make Q
A = Q*R;  % Set A to be the product QR
[Q2, R2] = qr(A,0);

norm(Q-Q2)/norm(Q)

ans =

    2.0000

norm(R-R2)/norm(R)

ans =

    0.2358
```
A2 = Q2*R2;
norm(A-A2)/norm(A)

ans =

  6.9717e-16