Lecture 19 - *Stability of Least-Squares Algorithms*

OBJECTIVE:
We have seen (Lecture 11) that there are various ways to solve least-squares problems, including the normal equations, Householder triangularization, Gram-Schmidt orthogonalization, and the SVD.
We now compare these methods and show that *the use of normal equations is generally unstable!*

◇ EXAMPLE

```
m = 100;  n = 15;          % problem size
t = (0:m-1)'/(m-1)';       % t discretizes [0,1]
A = fliplr(vander(t));     % construct Vandermonde matrix
b = exp(sin(4*t));         % right-hand side
b = b/2006.787453080206; % normalization for component x(15)
```

We want to find a least-squares fit of the function $e^{\sin(4t)}$ on $[0,1]$ by a polynomial of degree 14.

First, we discretize $[0,1]$ into $m = 100$ equally spaced points.

Then, we form the over-determined system $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is the appropriate Vandermonde matrix and $\mathbf{b}$ is (essentially) the function sampled at the appropriate points.

We scale $\mathbf{b}$ by a funny number so that the coefficient $x_{15} = 1$.
i.e., we scale by the true value of x_{15} in the non-normalized problem.
This is just to make the comparisons easy.

To analyze our results, we need the 3 quantities defined in Lecture 18.

```
x = A\b;  y = A*x;                  % find ‘‘exact’’ solution to
                                    %    least-squares problem
kappa = cond(A)                     % a highly ill-conditioned basis!
    kappa = 2.2718e+10
theta = asin(norm(b-y)/norm(b)) % to avoid cancellation errors
    theta = 3.7461e-06              %    because fit is so close
eta = norm(A)*norm(x)/norm(y)   % middling value between 1 and
    eta = 2.1036e+05                %    condition number of A
```

Theorem 18.1 now tells us the condition numbers of $\mathbf{y}$ and $\mathbf{x}$ with respect to perturbations $\mathbf{b}$ and $\mathbf{A}$ are approximately

	$\mathbf{y}$	$\mathbf{x}$
$\mathbf{b}$	1.0	1.1×10^5
$\mathbf{A}$	2.3×10^{10}	3.2×10^{10}

(verify!)

◇ HOUSEHOLDER TRIANGULARIZATION

As mentioned (Lecture 11) this is the standard way to solve least-squares problems.

(i.e., QR factorization via Householder triangularization)

```
[Q,R] = qr(A,0);
x = R\(Q'*b);
x(15)
    1.000 000 315 287 23
```

$\rightarrow$ relative error $\approx 3 \times 10^{-7}$

Since $\epsilon_{\text{machine}} \approx 10^{-16}$, rounding errors have been amplified by about 10^9.

This looks bad!

However, $\kappa_{\mathbf{A} \mapsto \mathbf{x}} \sim 10^{10}$.

So all of the inaccuracy in x_{15} can be explained by the poor conditioning of the problem, NOT an instability in the algorithm.

$\longrightarrow$ Algorithm 11.2 appears to be backward stable

Note 1. *In the above example, we formed $\hat{\mathbf{Q}}$ explicitly.*
Of course, this is not necessary (as emphasized in Lectures 10 and 16)
$\longrightarrow$ *it is enough to store the vectors $\mathbf{v}_k$ at the k^{th} step of Algorithm 10.1 and use them to compute $\hat{\mathbf{Q}}^T\mathbf{b}$.*

This effect can be achieved by finding the reduced QR factorization of the "augmented" matrix $\begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix}$.

$\longrightarrow$ during the factorization, the n Householder reflectors required to make $\mathbf{A}$ upper-triangular are also applied to $\mathbf{b}$.

$\longrightarrow$ this is equivalent to multiplication by $\hat{\mathbf{Q}}^T$.

So, $\hat{\mathbf{Q}}^T\mathbf{b}$ will be the first n positions of column $n+1$ of $\mathbf{R}$.

Note 2. *An $(n+1)st$ reflector is also applied to column $n+1$ to zero out elements below $n+1$.*
But, since $\hat{\mathbf{Q}}^T\mathbf{b}$ only has length n, we don't care about this.

```
[Q,R] = qr([A b],0);   % Householder triangularization of [A b]
Qb = R(1:n,n+1);       % extract Qhat' b
R = R(1:n,1:n);        % extract R
x = R\Qb;              % solve for x
x(15)                  % check accuracy
    ans = 1.000 000 315 294 65
```

$\longrightarrow$ almost the same as before.

i.e., the errors introduced in the QR factorization of $\mathbf{A}$ swamp those introduced in the computation of $\hat{\mathbf{Q}}^T\mathbf{b}$.

There is also a third way to solve least-squares problems in Matlab via Householder triangularization:

$\longrightarrow$ using the built-in $\backslash$ operator.

```
x = A\b;        % solve for x
x(15)
    ans = 0.999 999 943 110 87
```

$\longrightarrow$ an order of magnitude more accurate than the others!

This is because when $m > n$, the $\backslash$ operator is overloaded to perform *QR factorization with column pivoting* (which solves $\mathbf{AP} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$) where $\mathbf{P}$ is a permutation matrix.

$\longrightarrow$ we do not discuss such pivoting.

In terms of normwise stability analysis, all three variants of the QR factorization are equal.

Moreover, they are all backward stable.

Theorem 1. *The Householder triangularization method (Algorithm 11.2) for solving the full rank least-squares problems produces a computed solution* $\tilde{\mathbf{x}}$ *satisfying*

$$\|(\mathbf{A} + \delta\mathbf{A})\tilde{\mathbf{x}} - \mathbf{b}\| \text{ is minimized}$$

for some $\delta\mathbf{A}$ *satisfying* $\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} = \mathcal{O}(\epsilon_{\text{machine}})$

This result holds whether $\hat{\mathbf{Q}}^T\mathbf{b}$ is computed by explicitly forming $\hat{\mathbf{Q}}$ or through the use of the Householder reflector vectors $\mathbf{v}_k$ (Algorithm 10.2).

It also holds for Householder triangularization of an arbitrary column pivoting of $\mathbf{A}$.

◇ GRAM-SCHMIDT ORTHOGONALIZATION

Another way to solve a least-squares problem is by modified Gram-Schmidt orthogonalization (Algorithm 8.1).

For $m \approx n$, it is more expensive than Householder; but for $m \gg n$, both algorithms have complexity $\sim 2mn^2$.

```
[Q,R] = mgs(A);          % modified GS orthogonalization of A
x = R\(Q'*b);
x(15)
    ans = 1.029 265 945 326 72
```

$\rightarrow$ quite poor! Rounding errors have been magnified by 10^{14}, 1000 times worse than the inherent problem conditioning.

This algorithm is unstable!

The reason is that $\hat{\mathbf{Q}}$ has columns that are not accurately orthonormal (Lecture 9).

A way to stabilize Gram-Schmidt is to use it on the augmented system as before:

```
[Q,R] = mgs([A b]);
Qb = R(1:n,n+1);
R = R(1:n,1:n);
x = R\Qb;
x(15)
    ans = 1.000 000 056 533 99
```

$\longrightarrow$ just as good as Householder! (and it can be proved)

Theorem 2. *The modified Gram-Schmidt orthogonalization method for solving the full-rank least-squares problem is backward stable.*
i.e., it produces a solution $\tilde{\mathbf{x}}$ *satisfying*

$$\|(\mathbf{A}+\delta\mathbf{A})\tilde{\mathbf{x}}-\mathbf{b}\| \text{ is minimized}$$

for some $\delta\mathbf{A}$ *satisfying* $\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}=\mathcal{O}(\epsilon_{\text{machine}})$ *provided* $\hat{\mathbf{Q}}^T\mathbf{b}$ *is formed implicitly as part of the algorithm applied to the augmented system* $\begin{bmatrix}\mathbf{A} & \mathbf{b}\end{bmatrix}$.

◇ NORMAL EQUATIONS

This is a fundamentally different method than the previous ones.

In this case we form and solve the normal equations, typically accomplished by Cholesky factorization (Lecture 23).

For $m \gg n$, this method is twice as fast as the methods depending on explicit orthogonalization; it has complexity $\sim mn^2$.

```
x = (A'*A)\(A'*b);
x(15)
     ans = 0.393 390 698 702 83
```

$\rightarrow$ brutal! Not a single digit of accuracy!

$\rightarrow$ clearly an unstable algorithm.

The explanation turns out to be a perfect example of the interplay of conditioning and stability.

Suppose we have a backward stable algorithm for solving the full-rank least-squares problem
i.e., we have a computed solution $\tilde{\mathbf{x}}$ that minimizes

$$\|(\mathbf{A} + \delta\mathbf{A})\tilde{\mathbf{x}} - \mathbf{b}\|$$

for some $\delta\mathbf{A}$ satisfying $\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} = \mathcal{O}(\epsilon_{\text{machine}})$

Note 3. *This is a statement regarding perturbations of* $\mathbf{A}$ *only; what we say will still be true if perturbations in* $\mathbf{b}$ *are considered, or if we replace backward stability with stability.*

By Theorems 15.1 and 18.1, we have

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \mathcal{O}\left(\left[\kappa(\mathbf{A}) + \frac{\kappa^2(\mathbf{A})\tan\theta}{\eta}\right]\epsilon_{\text{machine}}\right)$$

Now suppose $\mathbf{A}$ is ill-conditioned; i.e., $\kappa(\mathbf{A}) \gg 1$ and let θ be bounded away from $\frac{\pi}{2}$ (why?)

We can have two very different scenarios, depending on the various parameter values.

1. Suppose $\tan\theta \approx 0$ (a close fit) or $\eta \approx \kappa(\mathbf{A})$ we have
$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \mathcal{O}(\kappa(\mathbf{A})\epsilon_{\text{machine}})$$

2. On the other hand, if $\tan\theta = \mathcal{O}(1)$ (the least-squares fit is not very close) then if $\eta \ll \kappa(\mathbf{A})$ we have
$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \mathcal{O}(\kappa^2(\mathbf{A})\epsilon_{\text{machine}})$$
i.e., *the condition number may effectively be* $\kappa^2(\mathbf{A})$!

Now suppose we use Cholesky factorization to solve the normal equations

$$(\mathbf{A}^T\mathbf{A})\mathbf{x} = \mathbf{A}^T\mathbf{b}.$$

Cholesky factorization is stable in the sense that it produces a solution $\tilde{\mathbf{x}}$ satisfying

$$(\mathbf{A}^T\mathbf{A} + \delta\mathbf{H})\tilde{\mathbf{x}} = \mathbf{A}^T\mathbf{b}$$

for some $\delta\mathbf{H}$ satisfying $\frac{\|\delta\mathbf{H}\|}{\|\mathbf{A}^T\mathbf{A}\|} = \mathcal{O}(\epsilon_{\text{machine}})$ (Theorem 23.3).

However, $\kappa(\mathbf{A}^T\mathbf{A}) = \kappa^2(\mathbf{A})$!

$\therefore$ the best we can expect from solving the normal equations is

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \mathcal{O}(\kappa^2(\mathbf{A})\epsilon_{\text{machine}})$$

THE CONDITIONING OF THE NORMAL EQUATIONS IS GOVERNED BY κ^2 (NOT κ)

So, if $\tan\theta = \mathcal{O}(1)$ and $\eta \ll \kappa$, or if $\kappa = \mathcal{O}(1)$, then

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \mathcal{O}(\kappa^2(\mathbf{A})\epsilon_{\text{machine}})$$ (as it should be)

$\rightarrow$ the normal equations are stable!

However, if $\kappa \gg 1$ and either $\tan\theta \approx 0$ or $\eta \approx \kappa$, then

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \mathcal{O}(\kappa^2(\mathbf{A})\epsilon_{\text{machine}}) \text{ (but it should be better!)}$$

$\longrightarrow$ the normal equations are unstable!

So, *the normal equations are typically unstable for ill-conditioned problems with close fits*.

In our example, $\kappa^2 = 10^{20}$, so we should not be surprised at the poor result.

$\diamond$ SVD

Like most computations based on the SVD, solving least-squares problems by SVD is stable.

```
[U,S,V] = svd(A,0);          % compute reduced SVD of A
x = V*(S\(U'*b));            % solve for x
x(15)
    ans = 0.999 999 982 304 71
```

$\longrightarrow$ this is the most accurate of all the methods (even Matlab's $\backslash \leftrightarrow$ QR with column pivoting).

The error is smaller by a factor of about 3.

Theorem 3. *The solution of the full-rank least-squares problem by the SVD (Algorithm 11.3) is backward stable.*

◇ RANK-DEFICIENT LEAST-SQUARES PROBLEMS

So far we have only analyzed the full-rank least-squares problem.

The differences between the stable algorithms we have seen are minor from the point of view of classical normwise stability analysis.

Householder, Householder with column pivoting, `mgs` with implicit $\hat{\mathbf{Q}}^T\mathbf{b}$, and SVD produce answers of roughly the same quality.

However some least-squares problems will not have full rank.
i.e., $\mathrm{rank}(\mathbf{A}) < m$, possibly with $m < n$, so the system of equations is *underdetermined*.

Such problems do not have a unique solution!

Typically the "solution" is specified as the one with *minimum norm*.

A further complication is that the correct solution depends on $\mathrm{rank}(\mathbf{A})$ and determining the rank of a matrix in the face of rounding errors is never a trivial task.

$\longrightarrow$ rank-deficient least-squares problems are *fundamentally different* than full-rank least-squares problems.

$\longrightarrow$ there is no reason to expect algorithms that are stable in the latter case be stable in the former!

In fact, the only fully stable method for solving rank-deficient least-squares problems is via SVD, with QR with partial pivoting being almost always stable in practice.