Numerical Matrix Analysis

Notes #15 — Conditioning and Stability
Least Squares Problems: Stability

Peter Blomgren
⟨blomgren@sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Spring 2024
(Revised: March 14, 2024)

Outline

Last Time

Theorem (Conditioning of Linear Least Squares Problems)

Let $\vec{b} \in \mathbb{C}^m$ and $A \in \mathbb{C}^{m\times n}$ of full rank be given. The least squares problem, $\min_{\vec{x}\in\mathbb{C}^n} \|\vec{b} - A\vec{x}\|$ *has the following 2-norm relative condition numbers describing the sensitivities of $\vec{y} = P\vec{b} \in \mathrm{range}(A)$ and $\vec{x}$ to perturbations in $\vec{b}$ and A:*

$\downarrow$ **Input**, **Output** $\rightarrow$	$\vec{y}$	$\vec{x}$
$\vec{b}$	$\dfrac{1}{\cos\theta}$	$\dfrac{\kappa(A)}{\eta\cos\theta}$
A	$\dfrac{\kappa(A)}{\cos\theta}$	$\kappa(A) + \dfrac{\kappa(A)^2\tan\theta}{\eta}$

$$\kappa(A) = \frac{\sigma_1}{\sigma_n} \in [1,\infty), \quad \cos(\theta) = \frac{\|\vec{y}\|}{\|\vec{b}\|} \in [0,1], \quad \eta = \frac{\|A\|\,\|\vec{x}\|}{\|A\vec{x}\|} \in [1,\kappa(A))$$

Deconstructing η...

$$\eta = \frac{\|A\|\,\|\vec{x}\|}{\|A\vec{x}\|} \in [1, \kappa(A))$$

Without loss of generality, rescale $\vec{x}$ so that $\|\vec{x}\| = 1$.

Now with $A = U\Sigma V^*$, the extreme cases correspond to

$$\vec{x} = \vec{v}_1 \quad \rightsquigarrow \quad \eta = \frac{\|A\|}{\|A\vec{v}_1\|} = \frac{\sigma_1}{\sigma_1} = 1,$$

$$\vec{x} = \vec{v}_n \quad \rightsquigarrow \quad \eta = \frac{\|A\|}{\|A\vec{v}_n\|} = \frac{\sigma_1}{\sigma_n} = \kappa(A).$$

So, we get the best conditioning of the Least Squares Problem when the formulation and model conspires such that the projection of the right-hand-side is parallel to the minor semi-axis of the ellipsoid $A\mathbb{S}^{n-1}$. — *"Obviously!"*

"But, why?!?" — It's a bit counter-intuitive: the problem is most sensitive to perturbations along that semi-axis (by the argument from the previous lecture), so if we maximize the "signal-to-noise-ratio" (minimizing the relative error along that semi-axis) by having significant model-action there, we get better behavior. It means that adding "irrelevant" parts to the model can significantly reduce the accurracy of the computation. — *"Careful Modeling Matters!"*

San Diego State University

Solving Least Squares Problems — 4 Approaches

Currently, we have four candidate methods for solving least squares problems:

- The **Normal Equations**

$$\vec{x} = (A^* A)^{-1} A^* \vec{b}$$

- **Gram-Schmidt Orthogonalization** (QR-factorization)

$$\vec{x} = R^{-1}(Q^* \vec{b})$$

- **Householder Triangularization** (QR-factorization)

$$\vec{x} = R^{-1}(Q^* \vec{b})$$

- The **Singular Value Decomposition**

$$\vec{x} = V(\Sigma^{-1}(U^* \vec{b}))$$

Our Test Problem

```
% The Dimensions of the Problem
m = 100;
n = 15;

% The Time-Vector --- Samples in [0,1]
t = (0:(m-1))' / (m-1);

% Build the Vandermonde Matrix A
A = [];
for p = 0:(n-1)
  A = [ A t.^p ];
end

% Build the Right-Hand-Side
b = exp(sin(4*t)) / 2006.787453080206;
```

2006.787453080206 ???

The normalization

```
% Build the right-hand side
b = exp(sin(4*t)) / 2006.787453080206;
```

Is chosen so that the correct (exact) value of the last component is $x_{15} = 1$.

We are trying to compute the 14^{th} degree polynomial $p_{14}(t)$ which fits $\exp(\sin(4t))$ on the interval $[0, 1]$.

Comment: Normalizing problems/results is crucial to make sure that you are indeed comparing solutions in a fair and unbiased manner, enabling accurate assessment and **meaningful insight**.

"The purpose of computation is insight, not numbers." — Richard Hamming

Our Test Problem: Visualized

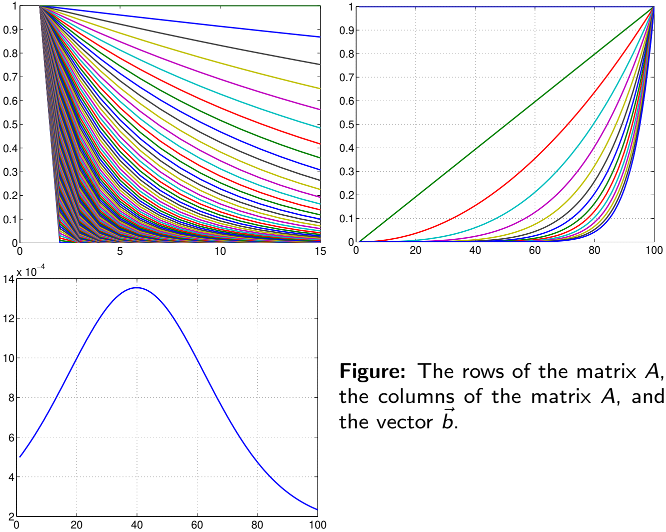

Figure: The rows of the matrix A, the columns of the matrix A, and the vector $\vec{b}$.

Finding 2006.787453080206 — Using Maple

Warning: Ancient Version of Maple Used

Some Maple Action...

```
with(linalg);
Digits := 512;
m      := 100;
n      := 15;
f      := (i,j) -> ((i-1)/(m-1))^(j-1);
A      := Matrix(m,n,f);
g      := (i) -> exp(sin(4*(i-1)/(m-1)));
b      := Vector(100,g);
x      := leastsqrs(A,b);

evalf( x[15] );
```

Gives

$$x_{15} = 2006.7874531048518338\ldots$$

Curious... However, using this value instead didn't change anything significantly in the following slides...

Approximation of Associated Condition Numbers

We use the best available Matlab solution (`x = A\b; y = A*x;`) to estimate the dimensionless parameters, and condition numbers

$\kappa(\mathbf{A})$	$\cos\theta$	η
`cond(A)`	`norm(y) / norm(b)`	`norm(A) * norm(x) / norm(y)`
2.27×10^{10}	0.99999999999426	2.10×10^{5}

$\downarrow$ **Input**, **Output** $\rightarrow$	$\vec{y}$	$\vec{x}$
$\vec{b}$	1.00	1.08×10^{5}
A	2.27×10^{10}	$\mathbf{3.10 \times 10^{10}}$

Bottom Line: If we get 6 correct digits (error $\sim 10^{-6}$) in matlab ($\varepsilon_{\text{mach}} \sim 10^{-16}$) then we are doing as well as we can.

Householder Triangularization

We have three ways of solving the least squares problem using the Matlab built-in Householder Triangularization

```
[Q,R] = qr(A,0);
x = R\(Q'*b);
e1 = abs(x(15)-1);
```

```
[~,R] = qr([A b],0);
QstarB = R(1:n,n+1);
R = R(1:n,1:n);
x = R\QstarB;
e2 = abs(x(15)-1);
```

```
x = A\b;
e3 = abs(x(15)-1);
```

- In the first approach, we explicitly form and use the matrix Q.
- In the second approach, we extract the "action" $Q^*\vec{b}$, by appending $\vec{b}$ as an additional column in A, and then identifying the appropriate components of the computed $\tilde{R}$ as R and $Q^*\vec{b}$.
- In the third approach, we rely on matlab's implementation... It uses Householder triangularization with column pivoting, for maximal accuracy.

Householder Triangularization: Errors

The approaches described above gives us the following errors

$$e_1 = 3.16387 \times 10^{-7},\ e_2 = 3.16371 \times 10^{-7},\ e_3 = 2.18674 \times 10^{-7}$$

Implicitly forming $Q^* \vec{b}$ improves the result marginally, which means that the errors introduced in the explicit formation of $Q^* \vec{b}$ are small compared to the errors introduced by the QR-factorization itself.

The Matlab solver, which includes all the bells and whistles, improves the result a little more;

All three variants are backward stable.

Householder Triangularization: Theorem

Theorem (Finding the Least Squares Solution Using Householder QR-Factorization is Backward Stable)

Let the full-rank least squares problem be solved by Householder triangularization in a floating-point environment satisfying the floating point axioms. This algorithm is backward stable in the sense that the computed solution $\tilde{x}$ has the property

$$\|(A + \delta A)\tilde{x} - \vec{b}\| = \min_{\vec{x} \in \mathbb{C}^n} \|\vec{b} - A\vec{x}\|, \quad \frac{\|\delta A\|}{\|A\|} = \mathcal{O}(\varepsilon_{mach})$$

for some $\delta A \in \mathbb{C}^{m \times n}$. This is true whether $\widehat{Q}^* \vec{b}$ is formed explicitly or implicitly. Further, the theorem is true for Householder triangularization with arbitrary column pivoting.

Householder Triangularization: Relative Error

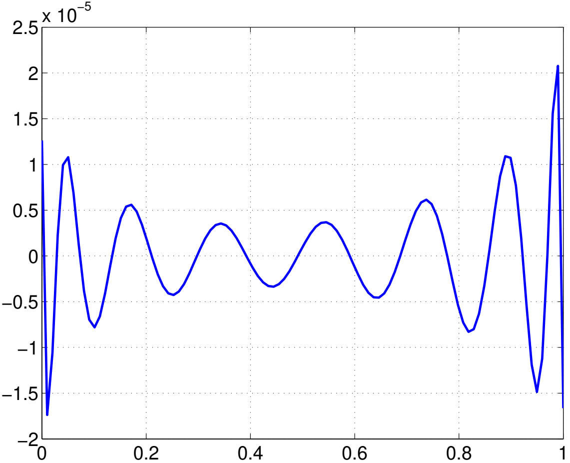

Figure: The relative error $(p(x) - b(x))/b(x)$ on the interval $[0, 1]$.

Modified Gram-Schmidt Orthogonalization

From homework, we have two ways of solving the least squares problem using modified Gram-Schmidt orthogonalization

```
[Q,R] = qr_mgs(A);
x = R\(Q'*b);
e4 = abs(x(15)-1);
```

```
[~,R] = qr_mgs([A b]);
QstarB = R(1:n,n+1);
R = R(1:n,1:n);
x = R\QstarB;
e5 = abs(x(15)-1);
```

- The explicit formation of Q in the first approach suffers from forward errors, and the result is quite disastrous

$$e_4 = 0.03024$$

- If instead we form $Q^*\vec{b}$ implicitly (the second approach), the result is much better

$$e_5 = 2.4854 \times 10^{-8}$$

Modified Gram-Schmidt Orthogonalization: Comments and Theorem

The fact that $e_5 < e_{1,2,3}$ in this example is not an indication of anything in particular — it is just luck.

The following is a provable result:

Theorem

The solution of the full-rank least squares problem by modified Gram-Schmidt orthogonalization is also backward stable, **provided** *that $Q^* \vec{b}$ is formed implicitly, as indicated on the previous slide.*

For "Fun" Only: Classical Gram-Schmidt Orthogonalization

We have two ways of solving the least squares problem using classical Gram-Schmidt orthogonalization

```
[Q,R] = qr_cgs(A);
x = R\(Q'*b);
e4 = abs(x(15)-1);
```

```
[~,R] = qr_cgs([A b]);
QstarB = R(1:n,n+1);
R = R(1:n,1:n);
x = R\QstarB;
e5 = abs(x(15)-1);
```

- Bad Things[TM] Happen

$$e_4 = 0.999385013507972$$

$$e_5 = 0.999385013507972$$

Normal Equations

Even though the condition number for the least squares problem

$$\kappa_{\mathrm{LS}} = \kappa(A) + \frac{\kappa(\mathbf{A})^{\mathbf{2}} \tan\theta}{\eta}$$

contains $\kappa(A)^2$, we have successfully found the solution with ~ 6 correct digits.

Using the **normal equations** $\tilde{x} = (A^* A)^{-1}(A^* \vec{b})$, we are subject to the full "force" of $\kappa(A)^2$, since

$$\kappa(A^* A) \sim \kappa(A)\kappa(A^*) \sim \kappa(\mathbf{A})^{\mathbf{2}}.$$

Matlab "barks" at us, if we try — `x = (A'*A)\(A'*b);`

```
Warning:  Matrix is close to singular or badly scaled.
          Results may be inaccurate.  RCOND = 1.512821e-19.
```

and $|\tilde{\mathbf{x}}_{\mathbf{15}} - \mathbf{x}_{\mathbf{15}}| = \mathbf{1.678}$.

Normal Equations: What Happened?!?

Even though the worst-case conditioning for the least squares problem is $\kappa(A)^2$, that is almost never realized.

In our test problem

$$\tan\theta \sim 3 \times 10^{-6}, \quad \eta \sim 2 \times 10^5$$

so, whereas

$$\kappa(A)^2 = 5.16 \times 10^{20}, \quad \frac{\kappa(A)^2 \tan\theta}{\eta} = 3.10 \times 10^{10}.$$

For A^*A there are no mitigating factors, and

$\kappa_{\text{est}}(A^*A) = 2.0 \times 10^{18}$ underestimate using the `cond()` command

so

$$\kappa_{\text{est}}(A^*A) \cdot \varepsilon_{\text{mach}} = 4.4 \times 10^2$$

Normal Equations: Theorem

Theorem

The solution of the full-rank least squares problem via the normal equations is **unstable**. *Stability can be achieved, however, by restriction to a class of problems in which $\kappa(A)$ is uniformly bounded above or $\dfrac{\tan\theta}{\eta}$ is uniformly bounded below.*

Bottom Line: The normal equations only work for "easy" least squares problems, *a.k.a. "Friendly Homework problems."*

The Singular Value Decomposition

```
[U,S,V] = svd(A,0);
x = V*(S\(U'*b));
e6 = abs(x(15)-1)
```

Solving the least squares problem using the SVD is the most expensive, but also the most stable method; here we get our error to be of the same order of magnitude as the other backward stable methods

$$e_6 = 3.16383 \times 10^{-7}$$

Theorem

The solution of the full-rank least squares problem by the SVD is backward stable.

Comments

At this point we have four working backward stable approaches to solving the full rank least squares problem

- Householder triangularization
- Householder triangularization with column pivoting
- Modified Gram-Schmidt with implicit $Q^*\vec{b}$ calculation
- The SVD

The differences, in terms of classical norm-wise stability, among these algorithms are minor.

For everyday use, select the simplest one — Householder triangularization — as your default algorithm. If you are working in matlab use $A\backslash\vec{b}$ — Householder triangularization with column pivoting.

Rank-Deficient Least Squares Problems

When $\mathrm{rank}(A) < n$, quite possibly with $m < n$, the least squares problem is **under-determined**.

No unique solution exists, unless we add additional constraints. Usually, we look for the **minimum norm** solution $\vec{x}$; *i.e.* among the infinitely many solutions we select the one with smallest norm.

The solution depends (strongly) on $\mathrm{rank}(A)$, and determining numerical rank is non-trivial. Is $10^{-14} = 0$???

For this class of problems, the only fully stable algorithms are based on the SVD.

Householder triangularization with column pivoting is stable for "almost all" such problems.

Rank-deficient least squares problems are a completely different class of problems, and we sweep all the details under the rug...