Deconstructing \( \eta \)...

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

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

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

\[
\vec{x} = \vec{v}_1 \quad \Rightarrow \quad \eta = \frac{\|A\|}{\|A\vec{v}_1\|} = \frac{\sigma_1}{\sigma_1} = 1,
\]

\[
\vec{x} = \vec{v}_n \quad \Rightarrow \quad \eta = \frac{\|A\|}{\|A\vec{v}_n\|} = \frac{\sigma_1}{\sigma_n} = \kappa(A).
\]

So, we get the best conditioning of the Least Squares Problem when the formulation and model conspire such that the projection of the right-hand-side is parallel to the minor semi-axis of the ellipsoid \( \mathcal{A}^{\kappa(A)-1} \).

---

"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" (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 accuracy of the computation.

---

"Careful Modeling Matters!"
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

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

% The time-vector --- samples in [0,1]
t = (0:(m-1))' / (m-1);

% Build the 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;
```

The normalization

```matlab
% 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 \( \chi_{15} = 1 \).

We are trying to compute the 14\textsuperscript{th} degree polynomial \( p_{14}(t) \) which fits \( \exp(\sin(4t)) \) on the interval \([0,1]\).
Finding 2006.787453080206 — Using Maple

Some Maple Action...

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

Gives

\[
\begin{align*}
x_{15} & = 2006.7874530802068338 \ldots
\end{align*}
\]

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 solution \((x = A\backslash b; y = A\times x)\) to estimate the dimensionless parameters, and condition numbers

<table>
<thead>
<tr>
<th>(k(A))</th>
<th>(\cos \theta)</th>
<th>(\eta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond(A) norm(y) / norm(b)</td>
<td>norm(A) * norm(x) / norm(y)</td>
<td></td>
</tr>
<tr>
<td>(2.27 \times 10^{10})</td>
<td>(0.9999999999426)</td>
<td>(2.10 \times 10^{5})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\downarrow) Input, Output (\rightarrow)</th>
<th>(\bar{y})</th>
<th>(\bar{x})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>(1.00)</td>
<td>(1.08 \times 10^{5})</td>
</tr>
<tr>
<td>(A)</td>
<td>(2.27 \times 10^{10})</td>
<td>(3.10 \times 10^{10})</td>
</tr>
</tbody>
</table>

**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);
K = R([1:n,1:n]);
x = K*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” \(Q^*\bar{b}\), by appending \(\bar{b}\) as an additional column in \(A\), and then identifying the appropriate components of the computed \(\bar{R}\) as \(R\) and \(Q^*\bar{b}\).
- In the third approach, we rely on matlab’s implementation... It uses Householder triangularization with column pivoting, for maximal accuracy.

The approaches described above gives us the following errors

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

Implicitly forming \(Q^*\bar{b}\) improves the result marginally, which means that the errors introduced in the explicit formation of \(Q^*\bar{b}\) 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 \( \hat{x} \) has the property

\[
\| (A + \delta A) \hat{x} - \hat{b} \| = \min_{x \in \mathbb{C}^n} \| \hat{b} - A x \|, \quad \frac{\| \delta A \|}{\| A \|} = \mathcal{O}(\varepsilon_{\text{mach}})
\]

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

Modified Gram-Schmidt Orthogonalization

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

\[
\begin{align*}
Q, R &= \text{qr} \_\text{ags}(A); \\
x &= R \setminus (Q^* \_b); \\
e_4 &= \text{abs}(x(15)-1);
\end{align*}
\]

- 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 \( Q^* b \) implicitly (the second approach), the result is much better
  
  \( e_5 = 2.4854 \times 10^{-8} \)

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 \( Q^* b \) is formed implicitly, as indicated on the previous slide.
Normal Equations

Even though the condition number for the least squares problem

\[ \kappa_{LS} = \kappa(A) + \frac{\kappa(A)^2 \tan \theta}{\eta} \]

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

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

\[ \kappa(A^*A) \sim \kappa(A)\kappa(A^*) \sim \kappa(A)^2. \]

Matlab “barks” at us, if we try — \( x = (A^*A) \backslash (A^*b) \);

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

and \( |\bar{x}_{15} - x_{15}| = 1.678 \).

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 \( \frac{\tan \theta}{\eta} \) is uniformly bounded below.

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

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

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

But for \( A^*A \) there are no mitigating factors, and

\[ \kappa_{est}(A^*A) = 2.0 \times 10^{18} \] underestimate?

so

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

The Singular Value Decomposition

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.
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 $Q^*b$ 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 $A\backslash b$ — Householder triangularization with column pivoting.

When $\text{rank}(A) < n$, 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 $\hat{x}$; i.e. among the infinitely many solutions we select the one with smallest norm.

The solution depends (strongly) on $\text{rank}(A)$, 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...