Lecture 17: Stability of linear equations

Admin:

Recall: Singular-value decomposition

$$A = \sum_i \lambda_i \vec{u}_i \vec{v}_i^T$$

SVD applications...

* Solving linear equations $Ax = b$
  - → What is the sensitivity, e.g., to numerical errors?
  - → Find the shortest solution
  - → When there is no solution, find x to minimize $\|Ax - b\|$
    - → Least-squares regression analysis
* Rank minimization
  - → Principal Component Analysis (PCA)

Data mining, clustering, recommendation systems, ...

NUMERICAL STABILITY OF $Ax = b$

systems of linear equations

Reading:
Meyer 1.6
5.12.1-2
Strang 7.2

Example:

Let $A = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 1 \end{pmatrix}$, $b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

How sensitive is the solution $x = A^{-1}b$ to small errors (e.g., numerical errors) in b?

Let $B = \begin{pmatrix} 1+10^{-4} & 1 \\ 1 & 1 \end{pmatrix}$.

How sensitive is $B^{-1}\vec{b}$ to small errors in $\vec{b}$?

Why not experiment?

```
>> A = [10^-4 1; 1 1];
B = [1+10^-4 1; 1 1];
b = [5;5];
berror = 10^(-5) * randn(2,1);
>> (A\b) - (A\(b+berror))

ans =

   1.0304e-05
  -7.2699e-06

>> (B\b) - (B\(b+berror))

ans =

  -1.0303e-01
   1.0304e-01
```

← pretty small
solution not too sensitive — good!

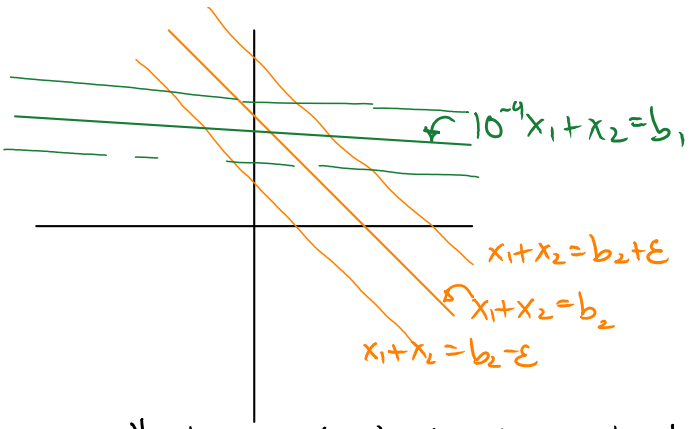
← huge error!

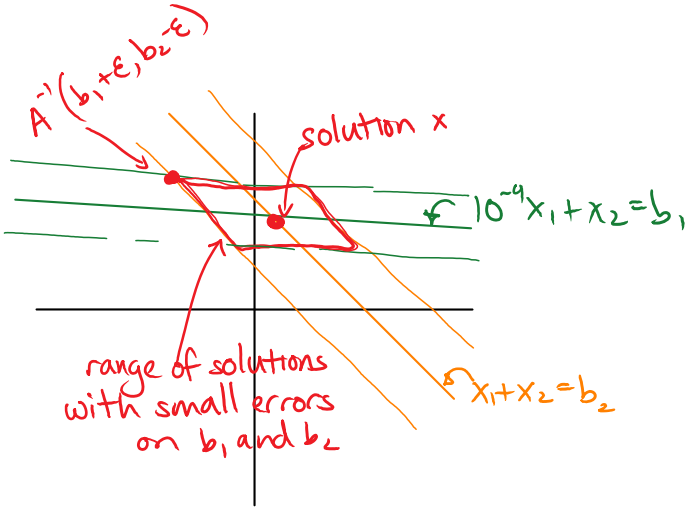
Why?

① Geometrically

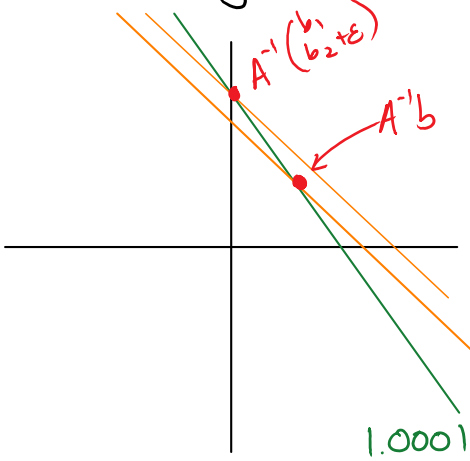
$Ax = b \iff$ $10^{-4}x_1 + x_2 = b_1$

$x_1 + x_2 = b_2$





small changes to b_1, b_2 shift the lines by small amounts



but the intersection of the lines can shift a lot, if they are nearly parallel

② Algebraic explanation for stability/instability

(*) $\|A^{-1}b - A^{-1}(b+\delta)\| = \|A^{-1}\delta\|$

$\leq \|A^{-1}\| \cdot \|\delta\|$

$\Rightarrow$ If $\|A^{-1}\|$ is large, then a small error in b can be amplified to a large error in $x = A^{-1}b$

Continuing the previous example:

```
octave:7> norm(A^-1)
ans =  1.6182
octave:8> norm(B^-1)
ans =  2.0001e+04
```

In the worst case, the upper bound in (*) is reached!

$A = \lambda_1 u_1 v_1^T + \lambda_2 u_2 v_2^T + \cdots + \lambda_n u_n v_n^T$

SVD, with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$

$A^{-1} = \frac{1}{\lambda_1} v_1 u_1^T + \frac{1}{\lambda_2} v_2 u_2^T + \cdots + \frac{1}{\lambda_n} v_n u_n^T$

$\Rightarrow \|A^{-1}\| = \lambda_n^{-1}$

and if error vector $\vec{\delta} = \varepsilon \vec{u}_n$,

then $A^{-1}\vec{\delta} = \frac{\varepsilon}{\lambda_n} v_n$

$\Rightarrow \|A^{-1}\delta\| = \|A^{-1}\| \cdot \|\delta\|$

Easy solution?

Instead of solving $Ax = b$,

solve $(1000A)y = b \Rightarrow y = \frac{1}{1000} A^{-1}b$

and then set $x = 1000y$

$\|(1000A)^{-1}\| = \frac{1}{1000}\|A^{-1}\|$ $\Rightarrow$ errors in b now lead to much smaller errors in y

$\rightsquigarrow$ But this is cheating; the error in x will be the same

$\Rightarrow$ **Definition**: The condition number of a matrix A is

$\|A\| \cdot \|A^{-1}\|$

equivalently, $= \frac{\text{largest singular value of } A}{\text{smallest singular value}}$.

smallest singular value

Observe: This is invariant under scaling:

$$\|(1000A)\| \cdot \|(1000A)^{-1}\| = \|A\| \cdot \|A^{-1}\|.$$

HIGH CONDITION NUMBER $\frac{\max}{\min}$ sing. value $\longleftrightarrow$ $Ax=b$ UNSTABLE

Example:

$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ↖ condition # = 1

$\begin{pmatrix} 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}$ ↙ (as small as it can get)

$\begin{pmatrix} 1 & 0 \\ 0 & 10^{-4} \end{pmatrix}$ ← condition # $= 10^4$ $\Rightarrow$ unstable!

What matters is not the scale of the matrix, but the **range of scales** in the matrix.

Formally: We have

$$\|A^{-1}f\| \le \|A^{-1}\| \cdot \|f\|$$

and $$\|b\| = \|A \cdot A^{-1} b\| \le \|A\| \cdot \|A^{-1} b\|$$

$$\Rightarrow \underbrace{\frac{\|A^{-1}f\|}{\|A^{-1}b\|}}_{\text{relative error in the solution } (\|A^{-1}f\|/\|x\|)} \le \underbrace{\|A\| \cdot \|A^{-1}\|}_{\text{condition \# of } A} \cdot \underbrace{\frac{\|f\|}{\|b\|}}_{\text{relative error in } b}$$

$\Rightarrow$ condition number bounds the relative errors

Three sources of error in solving $Ax=b$

1. Errors in the data $\vec{b}$
2. Errors in the linear solve routine

   Even if b is known exactly, errors such as

floating point errors might arise and accumulate while solving the equations.

Example:

$B = \begin{pmatrix} 1+10^{-4} & 1 \\ 1 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 5 \\ 5 \end{pmatrix}$ as above

Say you solve these and got $x = \begin{pmatrix} .5 \\ 4.5 \end{pmatrix}$

```
>> B * [.5;4.5]

ans =

   5.0000e+00
   5.0000e+00
```

← Looks good!

But is it?

No! The right answer is $\begin{pmatrix} 0 \\ 5 \end{pmatrix}$!

```
>> format long e
B * [.5;4.5]

ans =

   5.000050000000000e+00
   5.000000000000000e+00
```

when you check the answer, it looks great — but it is very far from the correct answer!

This is actually **the same** problem we already considered. Even though the error in b is small, the corresponding error in x is huge. This is only a good check for well-conditioned systems.

Moral: With an ill-conditioned system of linear equations
- use high-precision arithmetic
- find another way to check the answer?
- be careful! possibly try to reformulate the problem
  - → use a **preconditioner**

Solve $P^{-1}Ax = P^{-1}b$, where P is such that $P^{-1}A$ is better conditioned, but P^{-1} can be applied efficiently. This is an important, but more advanced, topic, that we might get to later.

(3) Errors in the matrix A

Comparing

$$Ax = b \quad \text{and} \quad (A + \delta A)(x + \delta x) = b$$

(δA: some error in the matrix; δx: the resulting error in the solution)

$$\Rightarrow A \cdot \delta x + (\delta A) \cdot (x + \delta x) = 0$$

$$\Rightarrow \delta x = -A^{-1} \cdot (\delta A) \cdot (x + \delta x)$$

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leq \|A^{-1}\| \cdot \|A\| \cdot \frac{\|\delta A\|}{\|A\|}$$

- $\frac{\|\delta x\|}{\|x + \delta x\|}$: relative error in the solution
- $\|A^{-1}\| \cdot \|A\|$: condition # of A
- $\frac{\|\delta A\|}{\|A\|}$: relative error in A

Example:

```
octave:11> Aerror = 10^(-5) * randn(2,2);
octave:12> (A\b) - ((A+Aerror)\b)
ans =

  -1.1004e-04
   7.6036e-05

octave:13> (B\b) - ((B+Aerror)\b)
ans =

  -210.22
   210.26
```

Moral: In all three cases, the (relative) effect of the error is bounded by the condition number

$$= \|A\| \cdot \|A^{-1}\| = \frac{\text{largest singular value}}{\text{smallest singular value}}$$

Note: In practice, computing the condition number can be a lot of work.

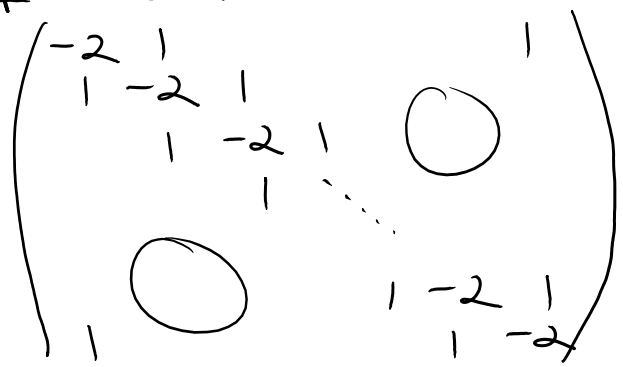
It is enough usually to approximate it, or give upper bounds.

Certain classes of matrices reliably have large condition numbers; with experience you learn when to be careful.

Example: What is the condition number of

$$\begin{pmatrix} -2 & 1 \\ & 1 \end{pmatrix}$$

Example: What is the condition number of



?

Answer: Infinite! The matrix is singular, since $(1, 1, 1, \ldots, 1)$ is in its nullspace.

$$\Rightarrow \frac{\text{largest s.v.}}{\text{smallest s.v.}} \approx \frac{4}{0} = \infty$$

Example: What is the condition number of

$A =$ 

?

Answer: Largest s.v. = norm ≈ 4 still.

Smallest singular value?

Observe:

$x = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)$ has norm 1

$Ax = (-1/\sqrt{n}, 0, 0, 0, \ldots, 0, -1/\sqrt{n})$

has norm $\sqrt{2/n}$

$\Rightarrow$ smallest singular value $\leq \sqrt{\frac{2}{n}}$

$\Rightarrow$ condition # $\gtrsim \sqrt{n}$

So be careful!

In fact, the condition # is $\sim \frac{4}{\pi^2}(n+1)^2 = \Theta(n^2)$

A better test vector is $x = \sqrt{\frac{2}{n+1}}\left(\sin\frac{\pi}{n+1}, \sin\frac{2\pi}{n+1}, \ldots, \sin\frac{n\pi}{n+1}\right)$

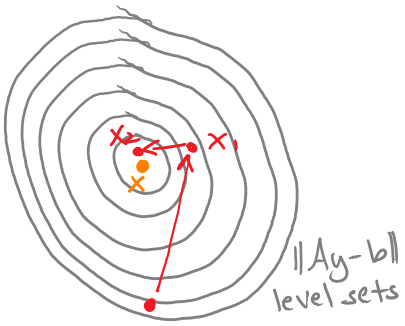
ANOTHER REASON THE CONDITION NUMBER IS IMPORTANT:

Poorly Conditioned Systems of Equations Take Longer to Solve

Rough intuition: Iterative solvers for $Ax=b$ start with a guess, and try to improve it.

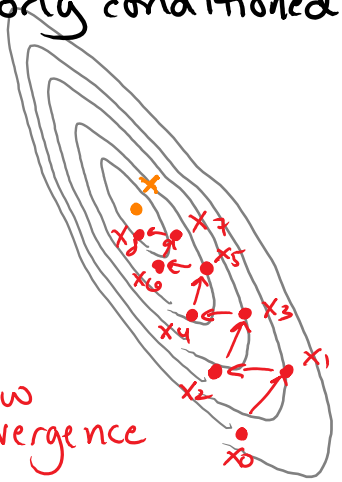
$x_0 \longrightarrow x_1 \longrightarrow x_2 \longrightarrow x_3 \longrightarrow \cdots$

Well-conditioned A



Fast convergence

Poorly conditioned A



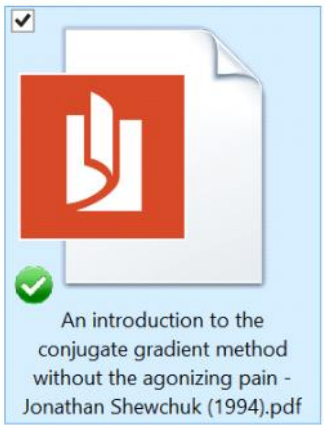
Slow convergence

Theorem:
Conjugate-gradient method on a positive semi-definite matrix A, with condition number κ, takes

$$O\left(\sqrt{\kappa}\cdot\log\frac{1}{\varepsilon}\right)$$

steps to get within ε of the true solution x.
(Or $O(\kappa\log\frac{1}{\varepsilon})$ steps for a general matrix A.)
Each step uses a matrix-vector multiplication
$= O(\#\text{ of nonzero entries in } A)$ steps.

Time permitting, we'll discuss this later.
Or, see →



Example: Gradient Descent

Goal: Solve $A\vec{x}=\vec{b}$.

Define the cost function

$$C(\vec{x}) = \|Ax-b\|^2$$
$$= \|A(x-x^*)\|^2 \quad \text{where } Ax^* = b$$
$$= (x-x^*)^T \underbrace{A^TA}_{\text{positive semi-definite}}(x-x^*)$$

If A has SVD

$$A = \sum_i \lambda_i u_i v_i^T$$
$$A^TA = \sum_i \lambda_i^2 v_i v_i^T$$

$$\Rightarrow C(\vec{x}) = \sum_i \lambda_i^2 |v_i \cdot (x-x^*)|^2$$

$\Rightarrow$ level set $\{\vec{x} \mid C(\vec{x}) = b\}$ is an ellipse!
(like $\lambda_1^2 x_1^2 + \lambda_2^2 x_2^2 + \cdots + \lambda_n^2 x_n^2 = b$)

Algorithm: Repeat until convergence:

$$x^t = x^{t-1} - \alpha\left(\frac{\partial C}{\partial x_1}(x^{t-1}), \ldots, \frac{\partial C}{\partial x_n}(x^{t-1})\right)$$

i.e., step in the direction opposite the largest increase in C.

$$\frac{\partial C}{\partial x_j} = \frac{\partial}{\partial x_j}\left[(Ax-b)^T(Ax-b)\right]$$
$$= \frac{\partial}{\partial x_j}\sum_i (Ax-b)_i^2$$
$$= 2\sum_i (Ax-b)_i \cdot \frac{\partial}{\partial x_j}\left[(Ax-b)_i\right]$$
$$= 2\sum_i (Ax-b)_i \cdot a_{ij}$$

$$\Rightarrow \left(\frac{\partial C}{\partial x_1}, \ldots, \frac{\partial C}{\partial x_n}\right) = 2\sum_i \underbrace{(Ax-b)_i}_{(Ax-b)^Te_i} \cdot \underbrace{(a_{i1}, \ldots, a_{in})}_{e_i^TA}$$
$$= 2(Ax-b)^T\left(\sum_i e_i e_i^T\right)A$$
$$= 2(Ax-b)^TA$$

(As you might guess!)

Example:

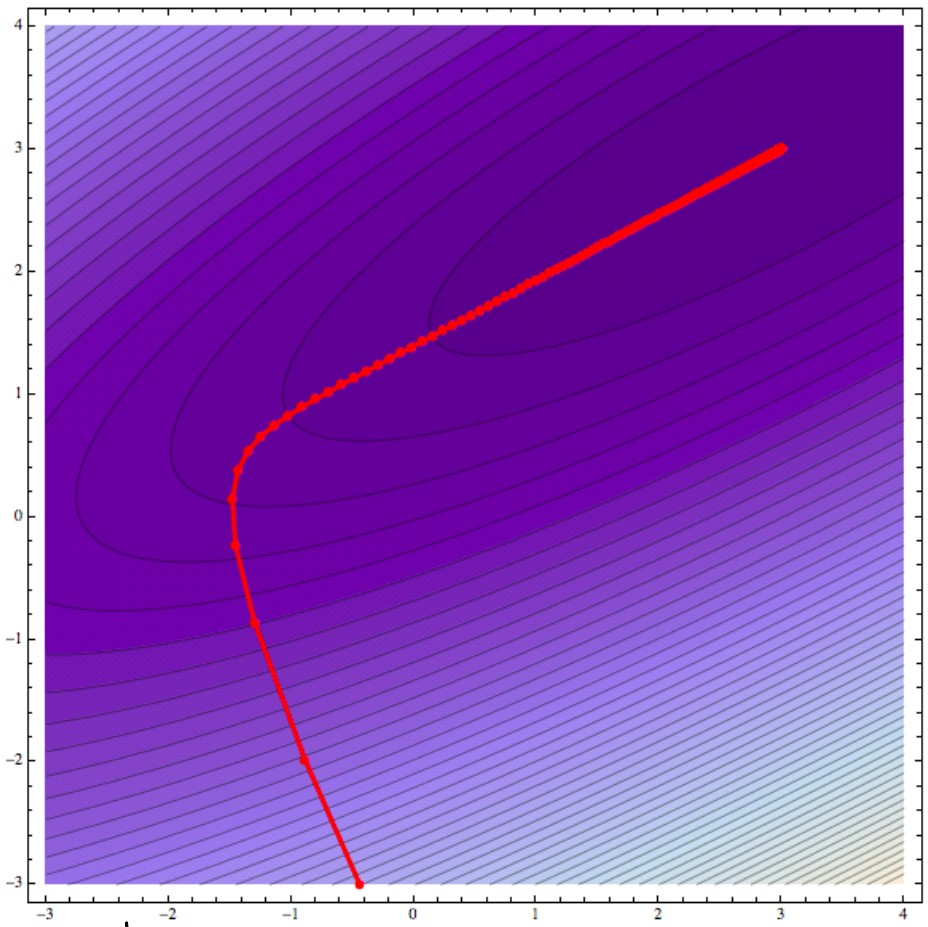
```
A = {{2, -1.5}, {-1.5, 4}};
b = A.{3, 3};
Eigenvalues[A]

α = .01;
x = {0, -4};
list = {x};
For[k = 1, k ≤ 1000, k++,
   x = x - 2 α (A.x - b).A;
   AppendTo[list, x];
 ];
"The solution is:"
x
```

```
{4.80278, 1.19722}
```

```
The solution is:
```

```
{3., 3.}
```

```
"Display the results:";
iter = ListPlot[list, PlotRange → 3 {{-1, 1}, {-1, 1}}, Joined → True,
   PlotStyle → {Red, Thickness[.005]}, PlotMarkers → Automatic];
contour = ContourPlot[Norm[A.{x, y} - b]^2, {x, -3, 4}, {y, -3, 4}, Contours → 50];
Show[contour, iter]
```



For small enough α, this has to converge, since there is a unique local and global minimum.

Observe: Smaller condition number
$\Rightarrow$ Rounder ellipse $\Rightarrow$ Faster convergence

Remark: Although not generally the fastest way of solving a set of linear equations, the gradient descent approach is very robust, and **many** variants are used in **many** applications.