Chapter 2, Linear Systems

Outline

The Geometry of Linear Equations[1]

- Example, 2×2 system:

$$\begin{aligned} 2x - y &= 1 \\ x + y &= 5 \end{aligned}$$

- Can look at this system by *rows* or *columns*.
- We will do both.

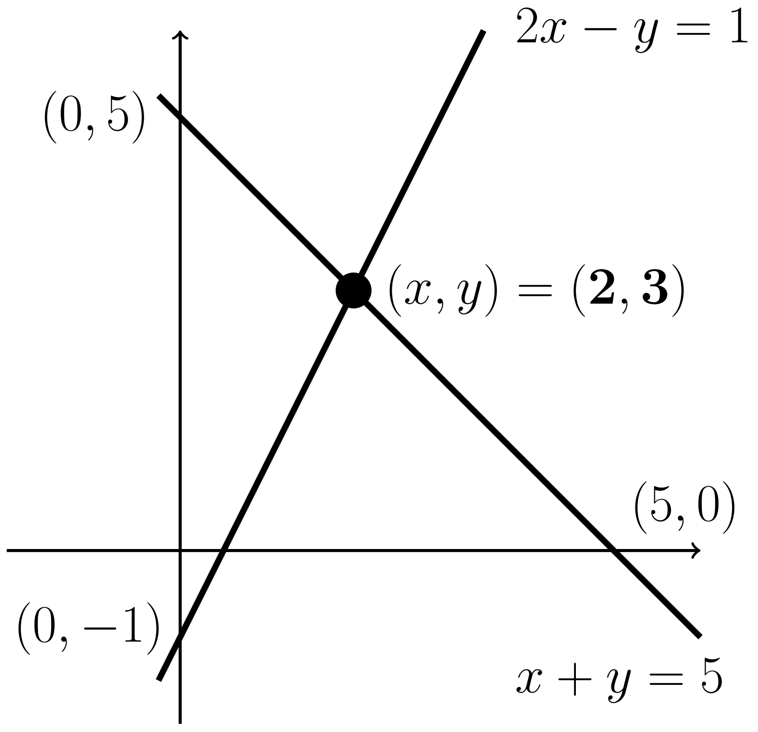
[1] Gilbert Strang: *Linear Algebra and Its Applications*

Row Form

- In the 2×2 system, each equation represents a line:

$$\begin{aligned} 2x - y &= 1 \qquad \text{line 1} \\ x + y &= 5 \qquad \text{line 2} \end{aligned}$$

- The intersection of the two lines gives the unique point $(x, y) = (2, 3)$, which is the solution.

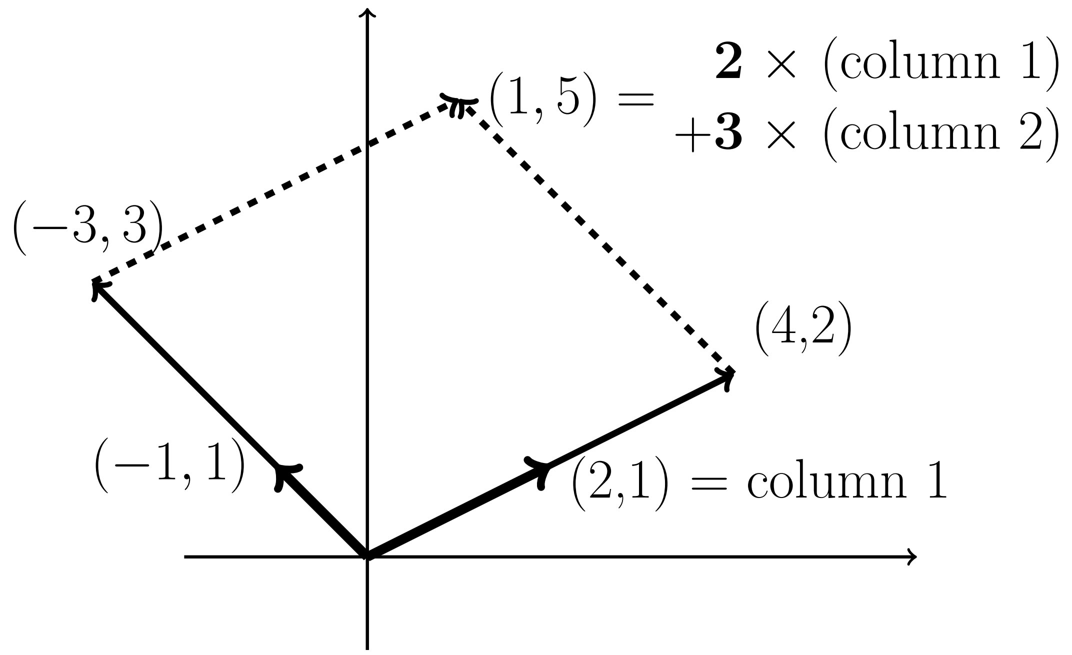


Column Form

- The second (and more important) geometry is column based.
- Here, we view the system of equations as *one vector equation*:

$$\textbf{Column form} \qquad x\begin{bmatrix} 2 \\ 1 \end{bmatrix} + y\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}.$$

- The problem is to find coefficients, x and y, such that the combination of vectors on the left equals the vector on the right.


$(1,5) =$ $\mathbf{2} \times$ (column 1)
$+\mathbf{3} \times$ (column 2)
$(-3,3)$
(4,2)
$(-1,1)$
(2,1) = column 1

Row Form: A Case with n=3.

Three planes

$$\begin{aligned} 2u + v + w &= 5 \\ 4u - 6v &= -2 \\ -2u + 7v + 2w &= 9 \end{aligned}$$

- Each equation (*row*) defines a plane in $\mathbb{R}^3$.
- The first plane is $2u + v + w = 5$ and it contains points ($\frac{5}{2}$,0,0) and (0,5,0) and (0,0,5).
- It is determined by three points, provided they do not lie on a line.
- Changing 5 to 10 would shift the plane to be parallel this one, with points (5,0,0) and (0,10,0) and (0,0,10).

Row Form: A Case with n=3, cont'd.

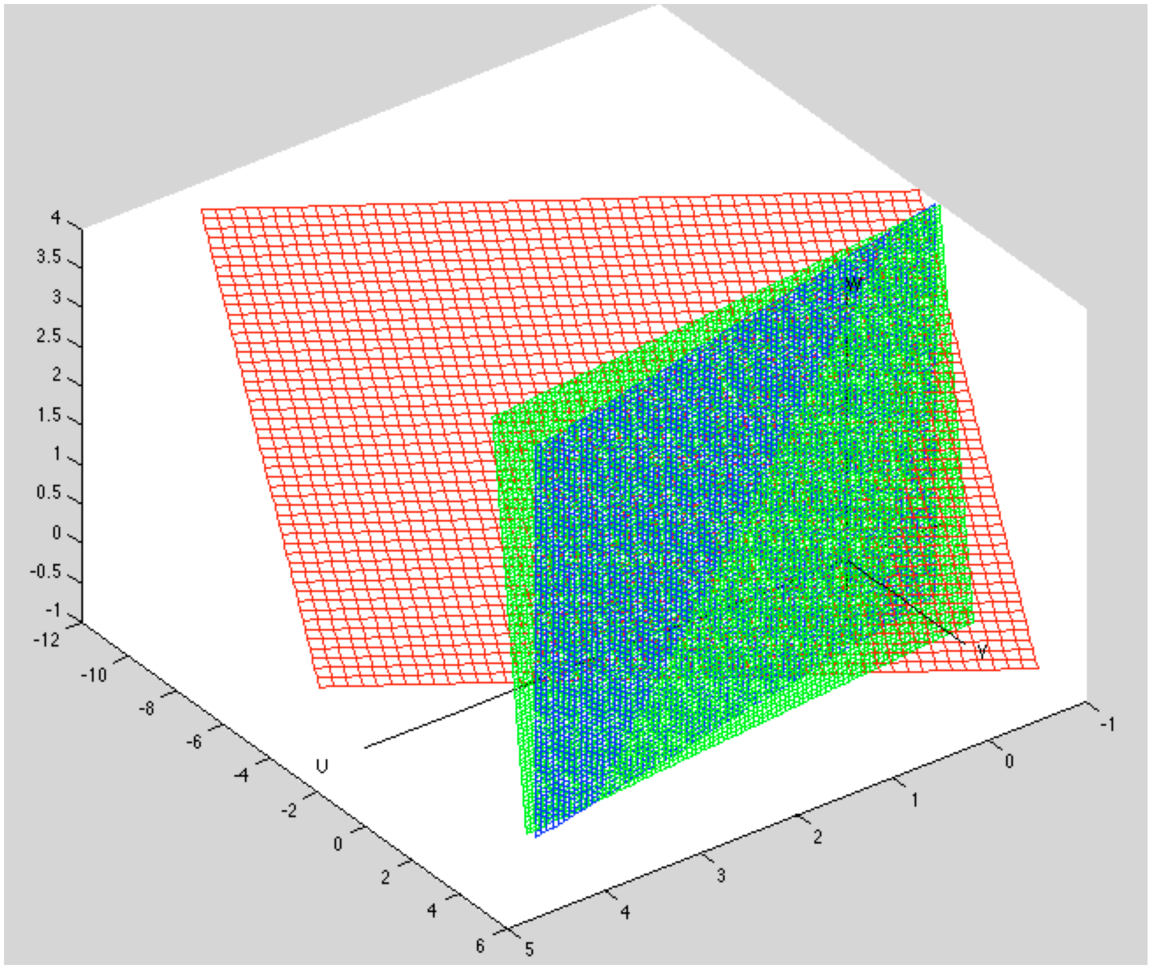
- The second plane is $4u - 6v = -2$.
- It is vertical because it can take on any w value.
- The intersection of this plane with the first is a *line.*
- The third plane, $-2u + 7v + 2w = 9$ intersects this line at a point, $(u, v, w) = (1, 1, 2)$, which is the solution.
- In n dimensions, the solution is the intersection point of n hyperplanes, each of dimension $n - 1$. A bit confusing.

Note that u=5 is also a plane….

Row Form

The green & blue planes (rows 2 and 3) intersect in a line.
Equation 1 (red) intersects this line.

$$\begin{aligned} 2u + v + w &= 5 \\ 4u - 6v &= -2 \\ -2u + 7v + 2w &= 9 \end{aligned}$$



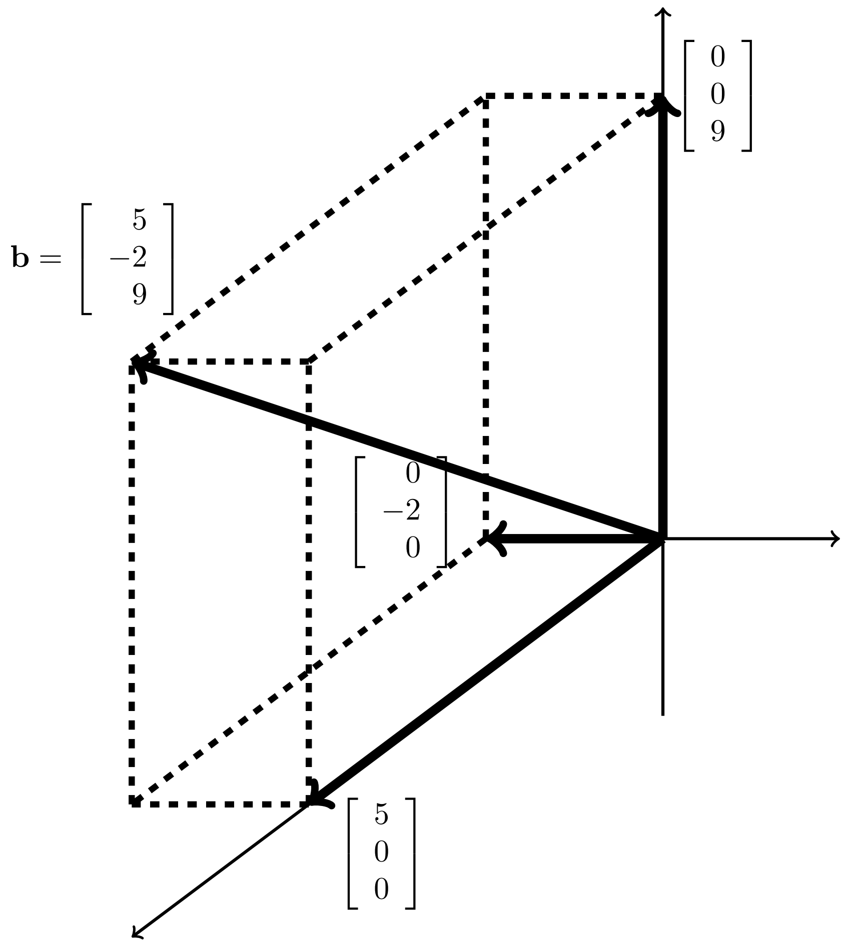
Column Vectors and Linear Combinations

- The preceding system is viewed as the vector equation

$$u \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} + v \begin{bmatrix} 1 \\ -6 \\ 7 \end{bmatrix} + w \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix} = \mathbf{b}.$$

- Our task is to find the multipliers, u, v, and w.
- The vector **b** is identified with the point (5,-2,9).
- We can view **b** as a list of numbers, a point, or an arrow.
- For $n > 3$, it's probably best to view it as a list of numbers.

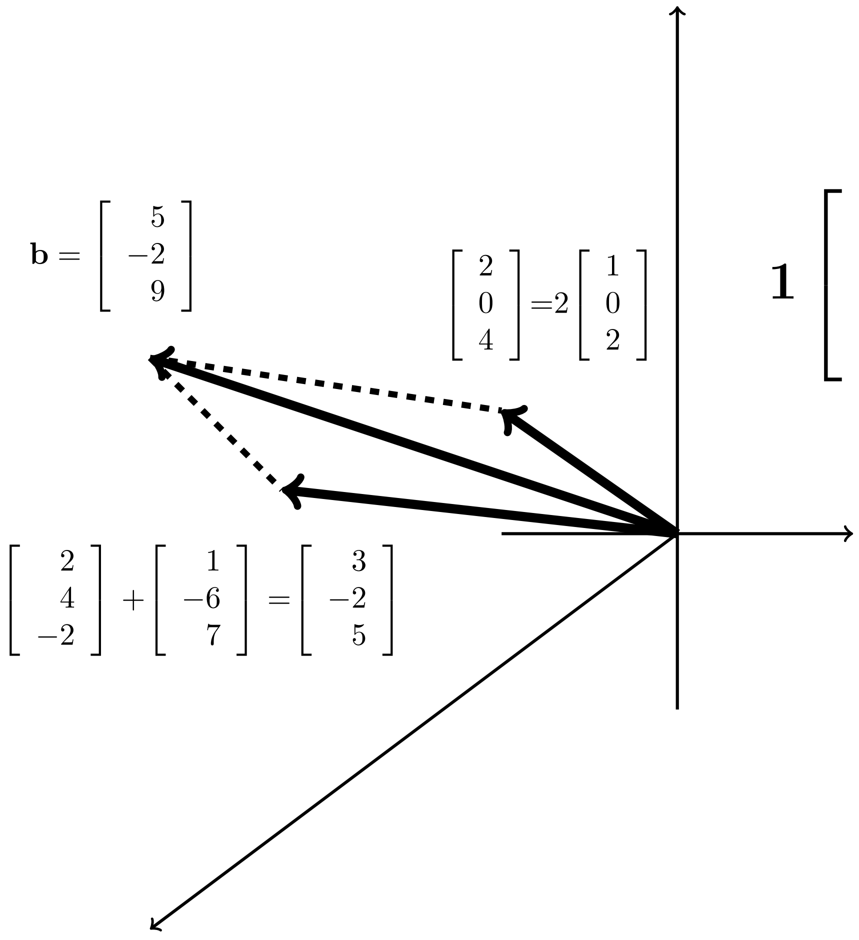
Vector Addition Example



$\begin{bmatrix} 0 \\ 0 \\ 9 \end{bmatrix}$
$\mathbf{b} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$
$\begin{bmatrix} 0 \\ -2 \\ 0 \end{bmatrix}$
$\begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}$

$$\begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 9 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$$

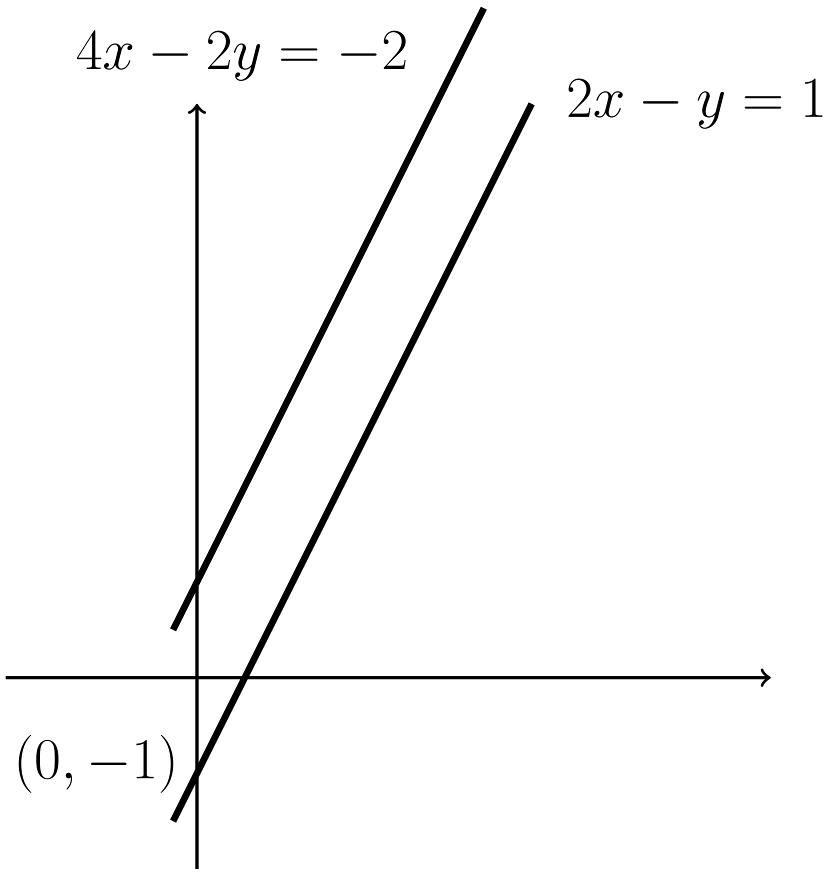
Linear Combination



$\mathbf{b} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$
$\begin{bmatrix} 2 \\ 0 \\ 4 \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$
$\begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} + \begin{bmatrix} 1 \\ -6 \\ 7 \end{bmatrix} = \begin{bmatrix} 3 \\ -2 \\ 5 \end{bmatrix}$

$$\mathbf{1} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} + \mathbf{1} \begin{bmatrix} 1 \\ -6 \\ 7 \end{bmatrix} + \mathbf{2} \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$$

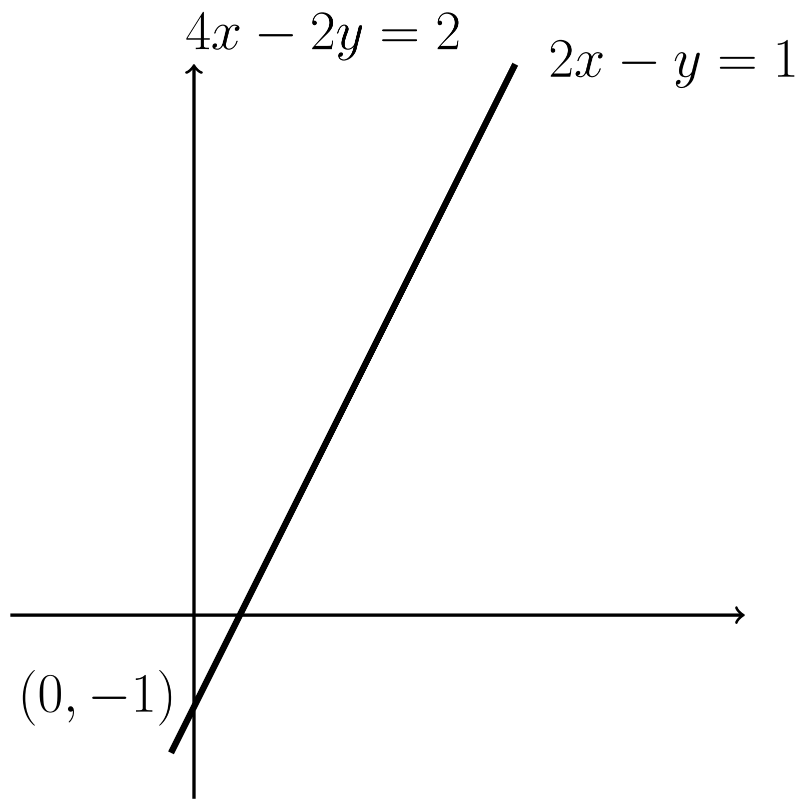
The Singular Case: Row Picture



$$\begin{aligned} 2x - y &= 1 \\ 4x - 2y &= -2 \end{aligned}$$

- No solution.

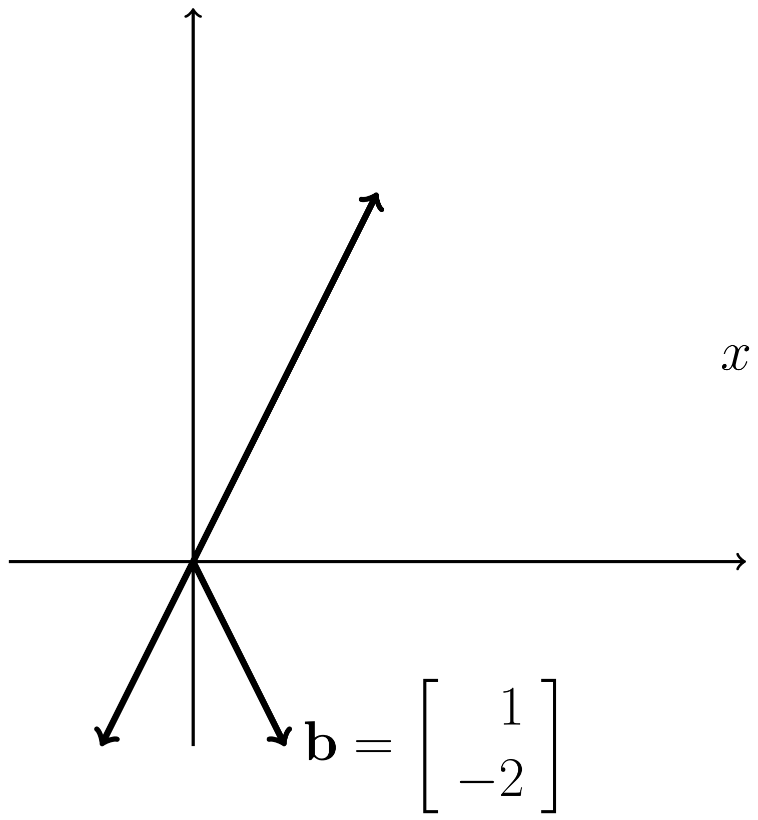
The Singular Case: Row Picture



$$\begin{array}{rcrcl} 2x & - & y & = & 1 \\ 4x & - & 2y & = & 2 \end{array}$$

- Infinite number of solutions.

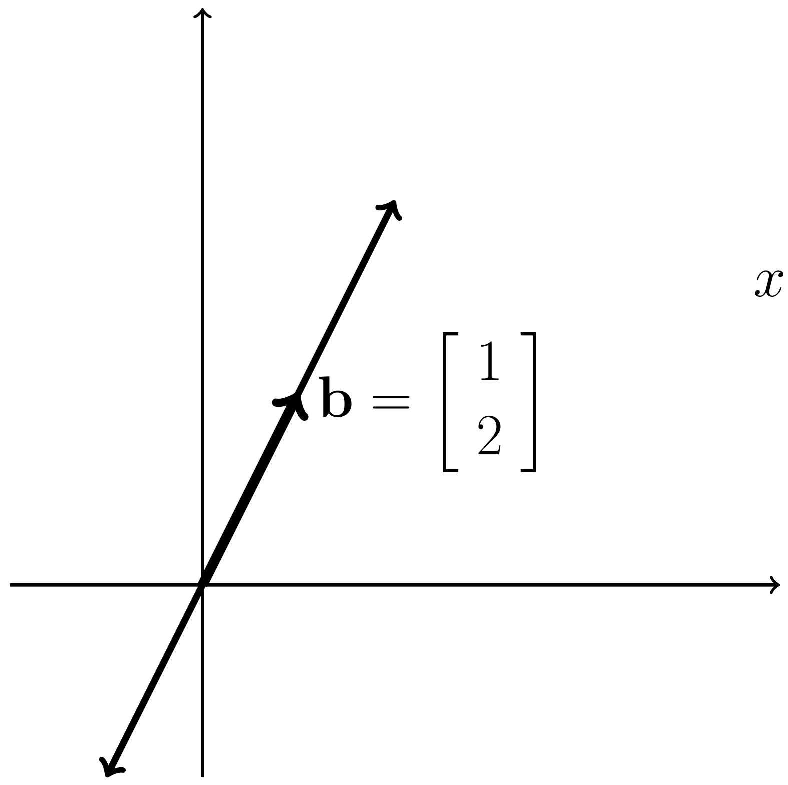
The Singular Case: Column Picture



$$x \begin{bmatrix} 2 \\ 4 \end{bmatrix} + y \begin{bmatrix} -1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

- No solution.

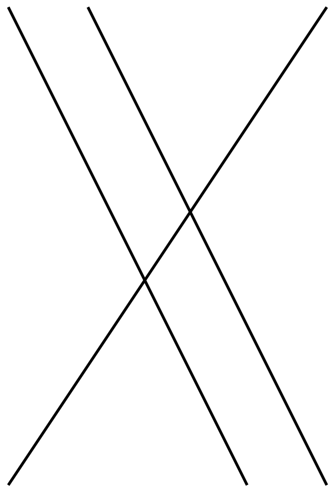
The Singular Case: Column Picture

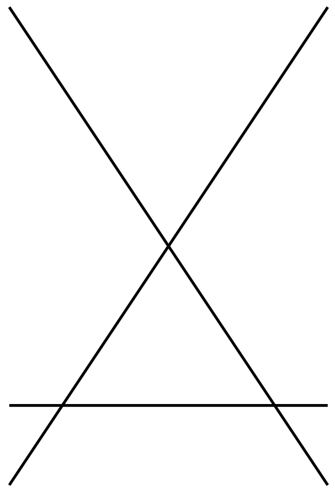


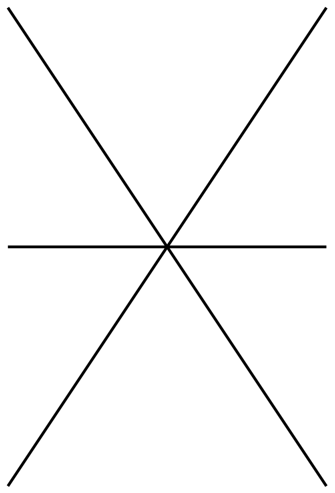
$$x \begin{bmatrix} 2 \\ 4 \end{bmatrix} + y \begin{bmatrix} -1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

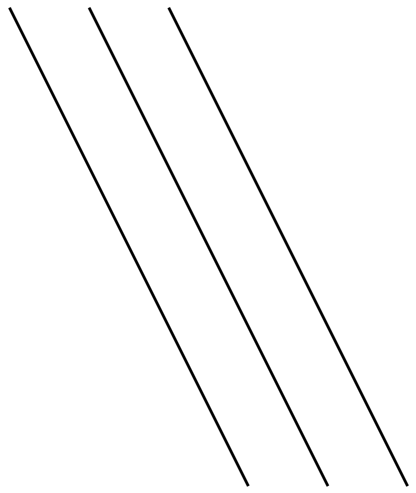
- Infinite number of solutions.

Singular Case: Row Picture with n=3

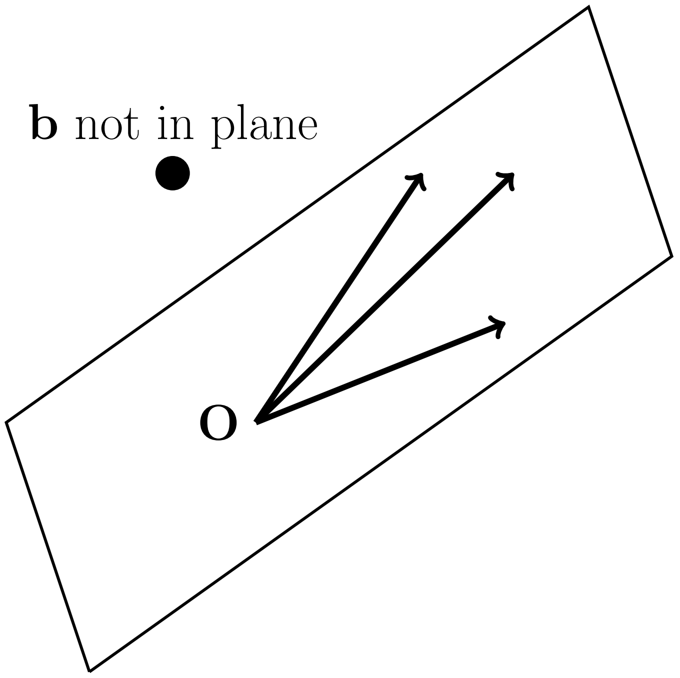

(a) two parallel planes

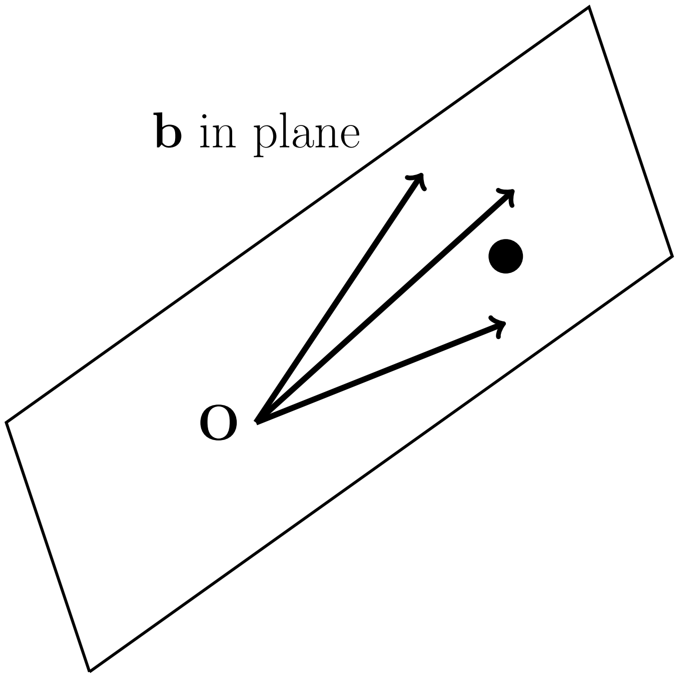

(b) no intersection


(c) line of intersection


(d) all planes parallel

Singular Case: Column Picture with n=3





- In this case, the three columns of the system matrix lie in the same plane.

$$\text{Example:} \quad u \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + v \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} + w \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \mathbf{b}.$$

Matrix Form and Matrix-Vector Products.

- We start with the familiar (row) form

$$\begin{array}{rcrcrcr} 2u & + & v & + & w & = & 5 \\ 4u & - & 6v & & & = & -2 \\ -2u & + & 7v & + & 2w & = & 9 \end{array}$$

- In matrix form, this is

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}, \quad \text{or } A\mathbf{u} = \mathbf{b}.$$

- Of course, this must equal our column form,

$$u \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} + v \begin{bmatrix} 1 \\ -6 \\ 7 \end{bmatrix} + w \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix} = \mathbf{b}.$$

Matrix Form and Matrix-Vector Products, 2.

- So, if A is the matrix with columns $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$,

$$A \;:=\; \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \;=:\; \begin{bmatrix} \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \\ \\ \end{bmatrix}, \qquad \text{and } \; \mathbf{u} := \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

- Then

$$A\mathbf{u} \;=\; u\,\mathbf{a}_1 \;+\; v\,\mathbf{a}_2 \;+\; w\,\mathbf{a}_3$$

Matrix Form and Matrix-Vector Products, 3.

- In general, if $\mathbf{x}$ is the n-vector

$$\mathbf{x} := \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},$$

and A is an $m \times n$ matrix, then

$$\begin{aligned} A\mathbf{x} &= x_1\,\mathbf{a}_1 + x_2\,\mathbf{a}_2 + \cdots + x_n\,\mathbf{a}_n \\ &= \textbf{linear combination of the columns of } A\textbf{.} \end{aligned}$$

- **Always.**

Matrix-Vector Products, Example.

$$\text{If} \qquad \hat{\mathbf{x}} := V \left(V^T A V\right)^{-1} V^T \mathbf{b}$$

$$= V\mathbf{y}.$$

Then $\hat{\mathbf{x}}$ = **linear combination of the columns of V.**

- $\hat{\mathbf{x}}$ lies in the *column space* of V.
- $\hat{\mathbf{x}}$ lies in the *range* of V.
- $\hat{\mathbf{x}} \in \text{span}(V)$

Sigma Notation

- Let A be an $m \times n$ matrix,

$$A = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_j & \cdots & \mathbf{a}_n \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix}.$$

- Then

$$\mathbf{w} = A\mathbf{x} = \sum_{j=1}^{n} x_j \, \mathbf{a}_j = \sum_{j=1}^{n} \mathbf{a}_j \, x_j$$

$$w_i = (A\mathbf{x})_i = \sum_{j=1}^{n} a_{ij} \, x_j$$

Matrix Multiplication

$$\text{If} \quad B = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 \end{bmatrix},$$

$$\text{Then} \quad C = AB = \begin{bmatrix} A\mathbf{b}_1 & A\mathbf{b}_2 \end{bmatrix}.$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

Q: (Important.) Suppose A and B are $n \times n$ matrices.

- How many floating point operations (flops) are required to compute $C = AB$?
- What is the number of memory accesses?

Systems of Linear Equations

- Given $m \times n$ matrix $\boldsymbol{A}$ and m-vector $\boldsymbol{b}$, find unknown n-vector $\boldsymbol{x}$ satisfying $\boldsymbol{Ax} = \boldsymbol{b}$
- System of equations asks “Can $\boldsymbol{b}$ be expressed as linear combination of columns of $\boldsymbol{A}$?”
- If so, coefficients of linear combination are given by components of solution vector $\boldsymbol{x}$
- Solution may or may not exist, and may or may not be unique
- For now, we consider only *square* case, $m = n$

Singularity and Nonsingularity

$n \times n$ matrix $\boldsymbol{A}$ is *nonsingular* if it has any of following equivalent properties

1. Inverse of $\boldsymbol{A}$, denoted by $\boldsymbol{A}^{-1}$, exists
2. $\det(\boldsymbol{A}) \neq 0$
3. $\text{rank}(\boldsymbol{A}) = n$
4. For any vector $\boldsymbol{z} \neq \boldsymbol{0}$, $\boldsymbol{A}\boldsymbol{z} \neq \boldsymbol{0}$

Existence and Uniqueness

- Existence and uniqueness of solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ depend on whether $\boldsymbol{A}$ is singular or nonsingular
- Can also depend on $\boldsymbol{b}$, but only in singular case
- If $\boldsymbol{b} \in \text{span}(\boldsymbol{A})$, system is *consistent*

$\boldsymbol{A}$	$\boldsymbol{b}$	# solutions
nonsingular	arbitrary	one (unique)
singular	$\boldsymbol{b} \in \text{span}(\boldsymbol{A})$	infinitely many
singular	$\boldsymbol{b} \notin \text{span}(\boldsymbol{A})$	none

Geometric Interpretation

- In two dimensions, each equation determines straight line in plane
- Solution is intersection point of two lines
- If two straight lines are not parallel (nonsingular), then intersection point is unique
- If two straight lines are parallel (singular), then lines either do not intersect (no solution) or else coincide (any point along line is solution)
- In higher dimensions, each equation determines hyperplane; if matrix is nonsingular, intersection of hyperplanes is unique solution

Example: Nonsingularity

- 2×2 system

$$\begin{aligned} 2x_1 + 3x_2 &= b_1 \\ 5x_1 + 4x_2 &= b_2 \end{aligned}$$

  or in matrix-vector notation

$$\boldsymbol{Ax} = \begin{bmatrix} 2 & 3 \\ 5 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \boldsymbol{b}$$

  is nonsingular regardless of value of $\boldsymbol{b}$

- For example, if $\boldsymbol{b} = \begin{bmatrix} 8 & 13 \end{bmatrix}^T$, then $\boldsymbol{x} = \begin{bmatrix} 1 & 2 \end{bmatrix}^T$ is unique solution

Example: Singularity

- 2×2 system

$$\boldsymbol{Ax} = \begin{bmatrix} 2 & 3 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \boldsymbol{b}$$

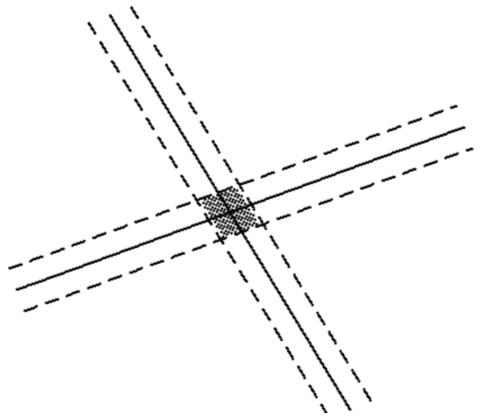
  is singular regardless of value of $\boldsymbol{b}$

- With $\boldsymbol{b} = \begin{bmatrix} 4 & 7 \end{bmatrix}^T$, there is no solution

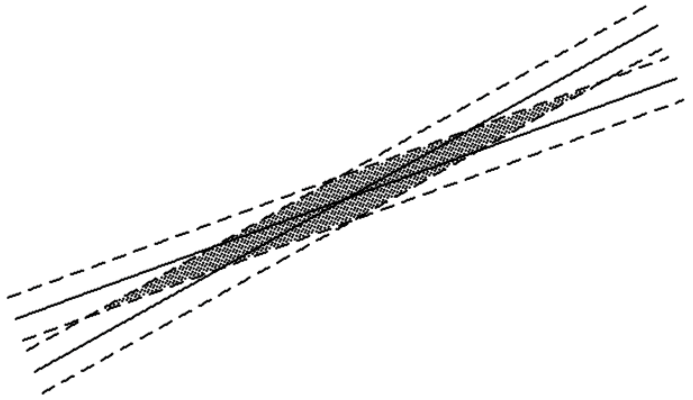
- With $\boldsymbol{b} = \begin{bmatrix} 4 & 8 \end{bmatrix}^T$, $\boldsymbol{x} = \begin{bmatrix} \gamma & (4 - 2\gamma)/3 \end{bmatrix}^T$ is solution for any real number γ, so there are infinitely many solutions

Nearly Singular Matrices

- In two dimensions, uncertainty in intersection point of two lines depends on whether lines are nearly parallel



Well-Conditioned



Ill-Conditioned
(nearly singular)

[An interesting question: For the 2x2 case, can you relate the angle to the condition number ?]

Conditioning of Linear Systems: $A\underline{x} = \underline{b}$

- As before, we ask the question,

  "If we perturb $\underline{b}$, how much change do we see in $\underline{x}$?"

$$A(\underline{x} + \Delta\underline{x}) = (\underline{b} + \Delta\underline{b})$$

  To pursue the answer to this question, we need a measure of the size of $\Delta\,\underline{x}$.

- We introduce ***vector norms***, $||\underline{x}||$, which measure the magnitude of a vector $\underline{x}$.

- Vector norms are also useful in measuring closeness of approximate solutions.

- Their closely-associated ***matrix norms*** are valuable in predicting how easy it is to solve a system, either directly (via *LU* factorization) or iteratively.

Vector Norms

- Magnitude, modulus, or absolute value for scalars generalizes to *norm* for vectors
- We will use only p-norms, defined by

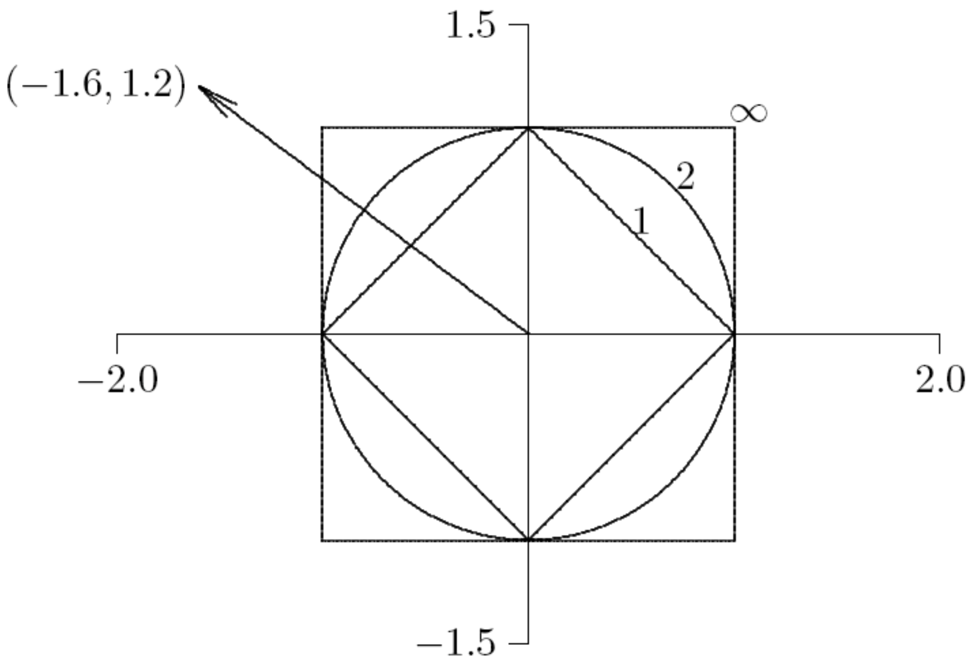
$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

for integer $p > 0$ and n-vector $\boldsymbol{x}$

- Important special cases
  - 1-norm: $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|$
  - 2-norm: $\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$
  - ∞-norm: $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$

Example: Vector Norms

- Drawing shows unit sphere in two dimensions for each norm



- Norms have following values for vector shown

$$\|\boldsymbol{x}\|_1 = 2.8 \quad \|\boldsymbol{x}\|_2 = 2.0 \quad \|\boldsymbol{x}\|_\infty = 1.6$$

Equivalence of Norms

- In general, for any vector $\boldsymbol{x}$ in $\mathbb{R}^n$, $\|\boldsymbol{x}\|_1 \geq \|\boldsymbol{x}\|_2 \geq \|\boldsymbol{x}\|_\infty$
- However, we also have

$$\|\boldsymbol{x}\|_1 \leq \sqrt{n}\,\|\boldsymbol{x}\|_2, \quad \|\boldsymbol{x}\|_2 \leq \sqrt{n}\,\|\boldsymbol{x}\|_\infty, \quad \|\boldsymbol{x}\|_1 \leq n\,\|\boldsymbol{x}\|_\infty$$

- Thus, for given n, norms differ by at most a constant, and hence are equivalent: if one is small, they must all be proportionally small.

❑ *Important Point:* ***Equivalence of Norms*** *(for n fixed):*

For all vector norms $\|\underline{x}\|_m$ and $\|\underline{x}\|_M$ $\exists$ constants c and C such that

$$c\,\|\underline{x}\|_m \leq \|\underline{x}\|_M \leq C\,\|\underline{x}\|_m$$

Allows us to work with the norm that is *most convenient.*

Properties of Vector Norms

- For any vector norm
  - $\|\boldsymbol{x}\| > 0$ if $\boldsymbol{x} \neq \boldsymbol{0}$
  - $\|\gamma\boldsymbol{x}\| = |\gamma| \cdot \|\boldsymbol{x}\|$ for any scalar γ
  - $\|\boldsymbol{x}+\boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ (triangle inequality)
- In more general treatment, these properties taken as *definition* of vector norm
- Useful variation on triangle inequality
  - $|\,\|\boldsymbol{x}\| - \|\boldsymbol{y}\|\,| \leq \|\boldsymbol{x}-\boldsymbol{y}\|$

Matrix Norms

- *Matrix norm* corresponding to given vector norm is defined by

$$\|\boldsymbol{A}\| = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$$

- Norm of matrix measures maximum stretching matrix does to any vector in given vector norm

Example….

Matrix Norms

For any vector norm $||\underline{x}||_*$, define

$$||A||_* = \max_{\underline{x} \neq 0} \frac{||A\underline{x}||_*}{||\underline{x}||_*} = \boxed{\max_{||\underline{x}||_*=1} ||A\underline{x}||_*}$$

- Often called the induced or subordinate matrix norm associated with the vector norm $||\underline{x}||_*$

Matrix Norms

- Matrix norm corresponding to vector 1-norm is maximum absolute *column* sum

$$\|\boldsymbol{A}\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$

- Matrix norm corresponding to vector ∞-norm is maximum absolute *row* sum

$$\|\boldsymbol{A}\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$

Example….

- Handy way to remember these is that matrix norms agree with corresponding vector norms for $n \times 1$ matrix

Matrix Norms: 2-norm

- The 2-norm of a symmetric matrix is $\max_i \; |\lambda_i|$
- Here, λ_i is the ith eigenvalue of A
- We say A is symmetric if $a_{ij} = a_{ji}$ for I,j $\in \{1,2,\ldots,n\}^2$
- That is, $A = A^T$ (A is equal to its transpose)

Symmetric Matrices

$$A = \begin{bmatrix} 1 & 4 & -2 \\ 4 & 2 & -5 \\ -2 & -5 & 3 \end{bmatrix} = A^T$$

$$B = \begin{bmatrix} 1 & 4 & -2 \\ 4 & 2 & -5 \\ 0 & -5 & 3 \end{bmatrix}$$

$$B^T = \begin{bmatrix} 1 & 4 & 0 \\ 4 & 2 & -5 \\ -2 & -5 & 3 \end{bmatrix}$$

- A is *symmetric*: $a_{ij} = a_{ji}$ for all i, j.
- B is *nonsymmetric*: $b_{ij} \neq b_{ji}$ for all i, j.
- Many (many) systems give rise to symmetric matrices.

Properties of Matrix Norms

- Any matrix norm satisfies
  - $\|\boldsymbol{A}\| > 0$ if $\boldsymbol{A} \neq \boldsymbol{0}$
  - $\|\gamma \boldsymbol{A}\| = |\gamma| \cdot \|\boldsymbol{A}\|$ for any scalar γ
  - $\|\boldsymbol{A} + \boldsymbol{B}\| \leq \|\boldsymbol{A}\| + \|\boldsymbol{B}\|$
- Matrix norms we have defined also satisfy
  - $\|\boldsymbol{A}\boldsymbol{B}\| \leq \|\boldsymbol{A}\| \cdot \|\boldsymbol{B}\|$
  - $\|\boldsymbol{A}\boldsymbol{x}\| \leq \|\boldsymbol{A}\| \cdot \|\boldsymbol{x}\|$ for any vector $\boldsymbol{x}$

Matrix Norm Example

- Matrix norms are particularly useful in analyzing *iterative solvers.*
- Consider the system $A\mathbf{x} = \mathbf{b}$ to be solved with the following iterative scheme.
- Start with initial guess $\mathbf{x}_0 = 0$ and, for k=0, 1, ...,

$$\mathbf{x}_{k+1} \;=\; \mathbf{x}_k \;+\; M\,(\,\mathbf{b} - A\mathbf{x}_k\,)\,. \tag{1}$$

- Let $G := I - MA$. We can use the matrix norm of G to bound the error in the above iteration and determine its rate of convergence.
- Begin by defining the error to be $\mathbf{e}_k := \mathbf{x} - \mathbf{x}_k$.
- Note that $\mathbf{b} - A\mathbf{x}_k = A\mathbf{x} - A\mathbf{x}_k = A(\mathbf{x} - \mathbf{x}_k) = A\mathbf{e}_k$.
- Using the preceding result and subtracting (1) from the equation $\mathbf{x} = \mathbf{x}$ yields the error equation

$$\mathbf{e}_{k+1} \;=\; \mathbf{e}_k \;-\; M\,A\,\mathbf{e}_k \;=\; [\,I - MA]\,\mathbf{e}_k \;=\; G\,\mathbf{e}_k.$$

Matrix Norm Example

- Error equation

$$\mathbf{e}_{k+1} \;=\; \mathbf{e}_k \;-\; M\,A\,\mathbf{e}_k \;=\; [\,I - MA]\,\mathbf{e}_k \;=\; G\,\mathbf{e}_k.$$

- From the definition of the matrix norm, we have

$$||\mathbf{e}_k|| \;\leq\; ||G||\,||\mathbf{e}_{k-1}|| \;\leq\; ||G||^2\,||\mathbf{e}_{k-2}|| \;\ldots\; \leq\; ||G||^k\,||\mathbf{e}_0||$$

- With $\mathbf{x}_0 = 0$, we have $\mathbf{e}_0 = \mathbf{x}$ and thus the *relative error*

$$\frac{||\mathbf{e}_k||}{||\mathbf{x}||} \;\leq\; ||G||^k$$

- If $||G|| < 1$, the scheme (1) is convergent.
- By the equivalence of norms, if $||G|| < 1$ for *any* matrix norm, it is convergent.
- **Q:** Suppose $||G|| \leq 0.25$. What is the bound on the number of iterations required to converge to machine precision in IEEE 64-bit arithmetic? (Hint: Think carefully. What is the best base to use in considering this question?)

Matrix Norm Example

- Consider the following example:

$$\begin{aligned} A &= nI + 0.1\,R, \quad R = \text{rand}(n,n) \;\; r_{ij} \in [0,1] \\ M &= \text{diag}(1/a_{ii}) \end{aligned}$$

- In this case,

$$\begin{aligned} g_{ii} &= 0 \\ g_{ij} &= 0.1\,\frac{-r_{ij}}{n + 0.1 r_{ii}} \end{aligned}$$

- The ∞-norm for G is given by

$$||G||_\infty = \max_i \sum_{j=1}^{n} |g_{ij}| \;\le\; \max_i \sum_{i \neq j} M^* = (n-1)M^*,$$

where

$$M^* := \max_{i \neq j} |g_{ij}| < \frac{0.1}{n}.$$

- In this case, we have a relative error bounded by $||G||_\infty^k \le (0.1)^k$.
- **Q:** Estimate the number of iterations required to reduce the error to machine epsilon when using IEEE 64-bit floating point arithmetic.

Condition Number

- *Condition number* of square nonsingular matrix $\boldsymbol{A}$ is defined by

$$\operatorname{cond}(\boldsymbol{A}) = \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^{-1}\|$$

- By convention, $\operatorname{cond}(\boldsymbol{A}) = \infty$ if $\boldsymbol{A}$ is singular

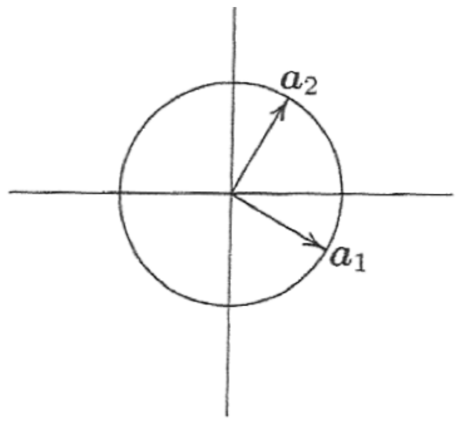
- Since

$$\|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^{-1}\| = \left(\max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}\right) \cdot \left(\min_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}\right)^{-1}$$

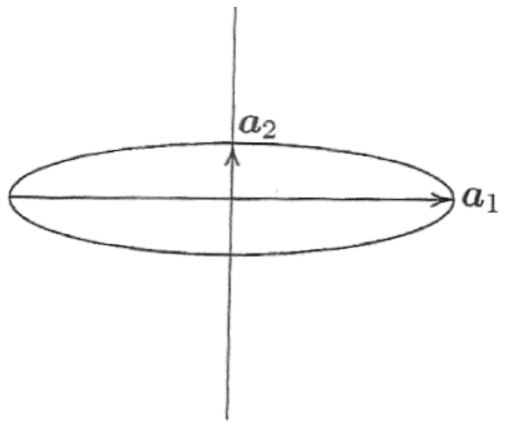
condition number measures ratio of maximum stretching to maximum shrinking matrix does to any nonzero vectors

- Large $\operatorname{cond}(\boldsymbol{A})$ means $\boldsymbol{A}$ is *nearly singular*

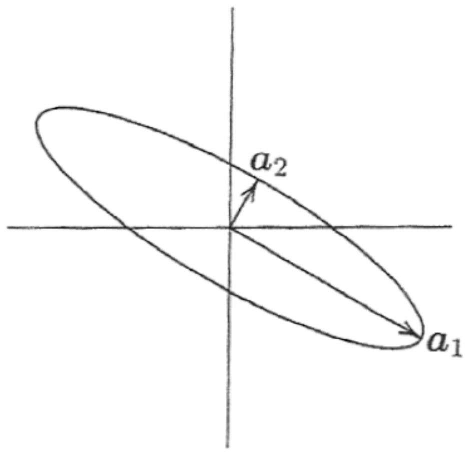
Condition Number Examples



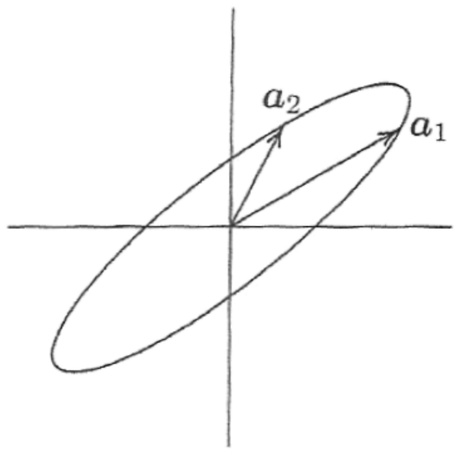
$$A_1 = \begin{bmatrix} 0.87 & 0.5 \\ -0.5 & 0.87 \end{bmatrix}, \quad \mathrm{cond}_2(A_1) = 1$$



$$A_2 = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad \mathrm{cond}_2(A_2) = 4$$



$$A_3 = \begin{bmatrix} 1.73 & 0.25 \\ -1 & 0.43 \end{bmatrix}, \quad \mathrm{cond}_2(A_3) = 4$$



$$A_4 = \begin{bmatrix} 1.52 & 0.91 \\ 0.47 & 0.94 \end{bmatrix}, \quad \mathrm{cond}_2(A_4) = 4$$

Properties of Condition Number

- For any matrix $\boldsymbol{A}$, $\mathrm{cond}(\boldsymbol{A}) \geq 1$
- For identity matrix, $\mathrm{cond}(\boldsymbol{I}) = 1$
- For any matrix $\boldsymbol{A}$ and scalar γ, $\mathrm{cond}(\gamma\boldsymbol{A}) = \mathrm{cond}(\boldsymbol{A})$
- For any diagonal matrix $\boldsymbol{D} = \mathrm{diag}(d_i)$, $\mathrm{cond}(\boldsymbol{D}) = \dfrac{\max |d_i|}{\min |d_i|}$

Computing Condition Number

- Definition of condition number involves matrix inverse, so it is nontrivial to compute
- Computing condition number from definition would require much more work than computing solution whose accuracy is to be assessed
- In practice, condition number is estimated inexpensively as byproduct of solution process
- Matrix norm $\|\boldsymbol{A}\|$ is easily computed as maximum absolute column sum (or row sum, depending on norm used)
- Estimating $\|\boldsymbol{A}^{-1}\|$ at low cost is more challenging

Computing Condition Number, continued

- From properties of norms, if $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{y}$, then

$$\frac{\|\boldsymbol{z}\|}{\|\boldsymbol{y}\|} \le \|\boldsymbol{A}^{-1}\|$$

  and bound is achieved for optimally chosen $\boldsymbol{y}$

- Efficient condition estimators heuristically pick $\boldsymbol{y}$ with large ratio $\|\boldsymbol{z}\|/\|\boldsymbol{y}\|$, yielding good estimate for $\|\boldsymbol{A}^{-1}\|$

- Good software packages for linear systems provide efficient and reliable condition estimator

Error Bounds

- Condition number yields error bound for computed solution to linear system
- Let $\boldsymbol{x}$ be solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, and let $\hat{\boldsymbol{x}}$ be solution to $\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{b} + \Delta\boldsymbol{b}$
- If $\Delta\boldsymbol{x} = \hat{\boldsymbol{x}} - \boldsymbol{x}$, then

$$\boldsymbol{b} + \Delta\boldsymbol{b} = \boldsymbol{A}(\hat{\boldsymbol{x}}) = \boldsymbol{A}(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{A}\Delta\boldsymbol{x}$$

which leads to bound

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le \operatorname{cond}(\boldsymbol{A})\frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}$$

for possible relative change in solution $\boldsymbol{x}$ due to relative change in right-hand side $\boldsymbol{b}$

Error Bounds, continued

- Similar result holds for relative change in matrix: if $(\boldsymbol{A}+\boldsymbol{E})\hat{\boldsymbol{x}} = \boldsymbol{b}$, then

$$\frac{\|\Delta \boldsymbol{x}\|}{\|\hat{\boldsymbol{x}}\|} \le \operatorname{cond}(\boldsymbol{A}) \frac{\|\boldsymbol{E}\|}{\|\boldsymbol{A}\|}$$

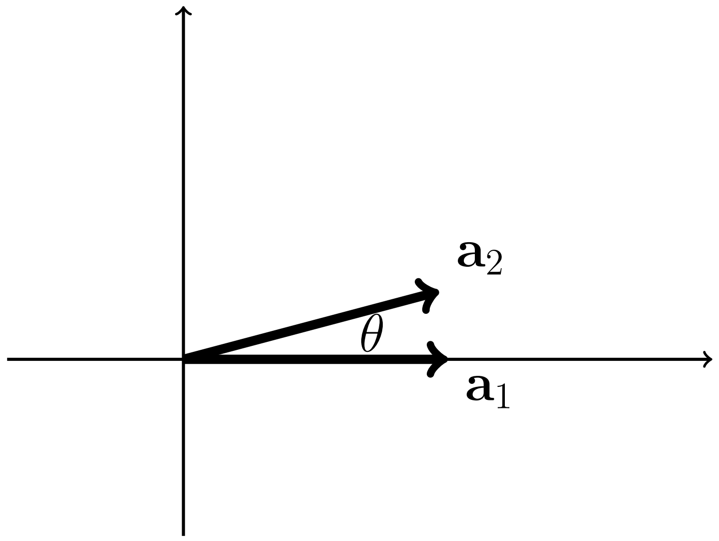
- If input data are accurate to machine precision, then bound for relative error in solution $\boldsymbol{x}$ becomes

$$\frac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le \operatorname{cond}(\boldsymbol{A})\, \epsilon_{\text{mach}}$$

- Computed solution loses about $\log_{10}(\operatorname{cond}(\boldsymbol{A}))$ decimal digits of accuracy relative to accuracy of input

Example

A Nearly Singular Example



$$A = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 \end{bmatrix} = \begin{bmatrix} 1 & c \\ 0 & s \end{bmatrix}$$

$$c = \cos\theta, \qquad s = \sin\theta.$$

- Clearly, as $\theta \longrightarrow 0$ the matrix becomes singular.
- Can show that

$$\begin{aligned} \text{cond} &= \sqrt{\frac{1+|c|}{1-|c|}} \\ &\approx \frac{2}{\theta} \end{aligned}$$

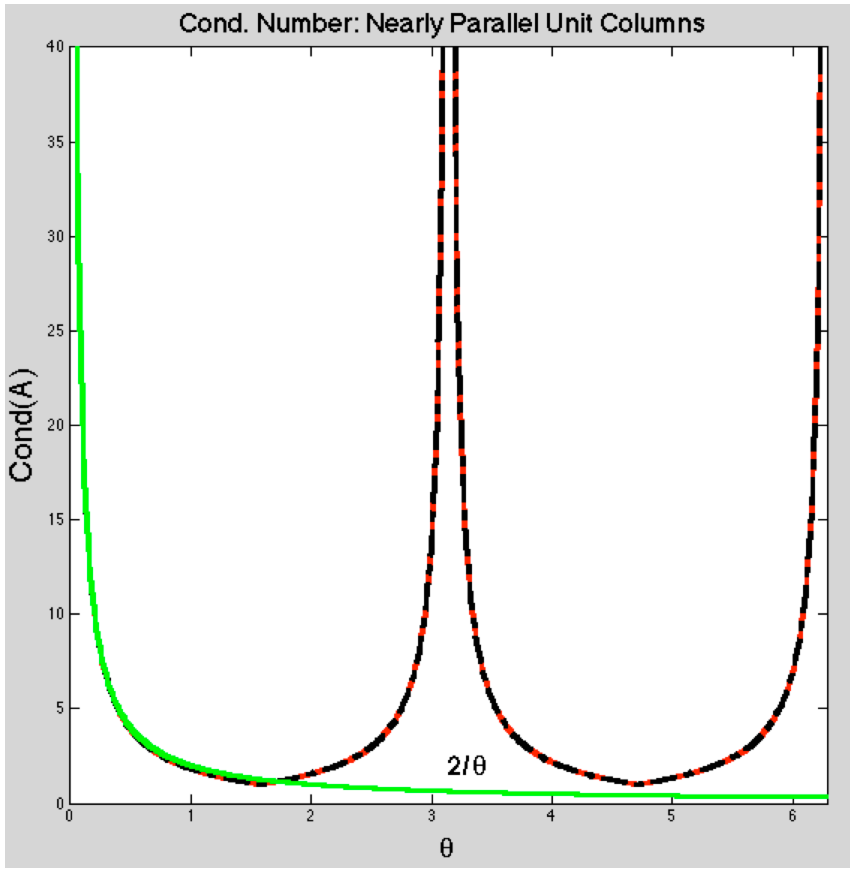
for small θ (by Taylor series!) *matlab demo.*

Matlab Demo cr2.m

This example plots cond(A) as a function of θ, as well as the estimates from the preceding slide.

- The computed value of cond(A) given by matlab exactly matches $[\,(1+|\cos\theta|)\,/\,(1-|\cos\theta|)\,]^{1/2}$
- The more interesting result is cond(A) ~ $2/\theta$, which is very accurate for small angles.

```
%% Note - eigenvalues of A'*A are evals of C=A'*A =
%%
%%     1 c
%%     c 1
%%
%% (1-lam)*(1-lam) - c^2 , which is z^2 - c^2 with roots
%%
%%   z=c and z=-c
%%
%%   1-lam = c --> lam = 1 - c
%%
%%   1-lam = -c --> lam = 1+c
%%
%%   K2 = 1+c / 1 - c
%%
%%     ~ 2 / (1/2 theta^2) for small theta ~ 4 / theta^2
%%
%%   Therefore:     K(A) = sqrt(K2) ~ 2/theta
%%

format compact

jj=0; for j=.01:.01:(2*pi); cj=cos(j);sj=sin(j); jj=jj+1;
    R=[ cj -sj ; sj cj ];
    a1 = [ 1 ; 0 ]; a2 = R*a1; A = [ a1 a2 ];

    C(jj) = cond(A);
    t(jj)=j; aj = abs(cj); z(jj)=sqrt( (1+aj)/(1-aj) );
end;
plot(t,C,'r-',t,z,'k-.',t,2./abs(t),'g-','LineWidth',3);
axis([0 2*pi 0 40]);text(pi,2,'2/\theta','FontSize',18) axis square;
xlabel('\theta','FontSize',18);ylabel('Cond(A)','FontSize',20)
title('Cond. Number: Nearly Parallel Unit Columns','FontSize',18)
```



Error Bounds – Illustration

- In two dimensions, uncertainty in intersection point of two lines depends on whether lines are nearly parallel

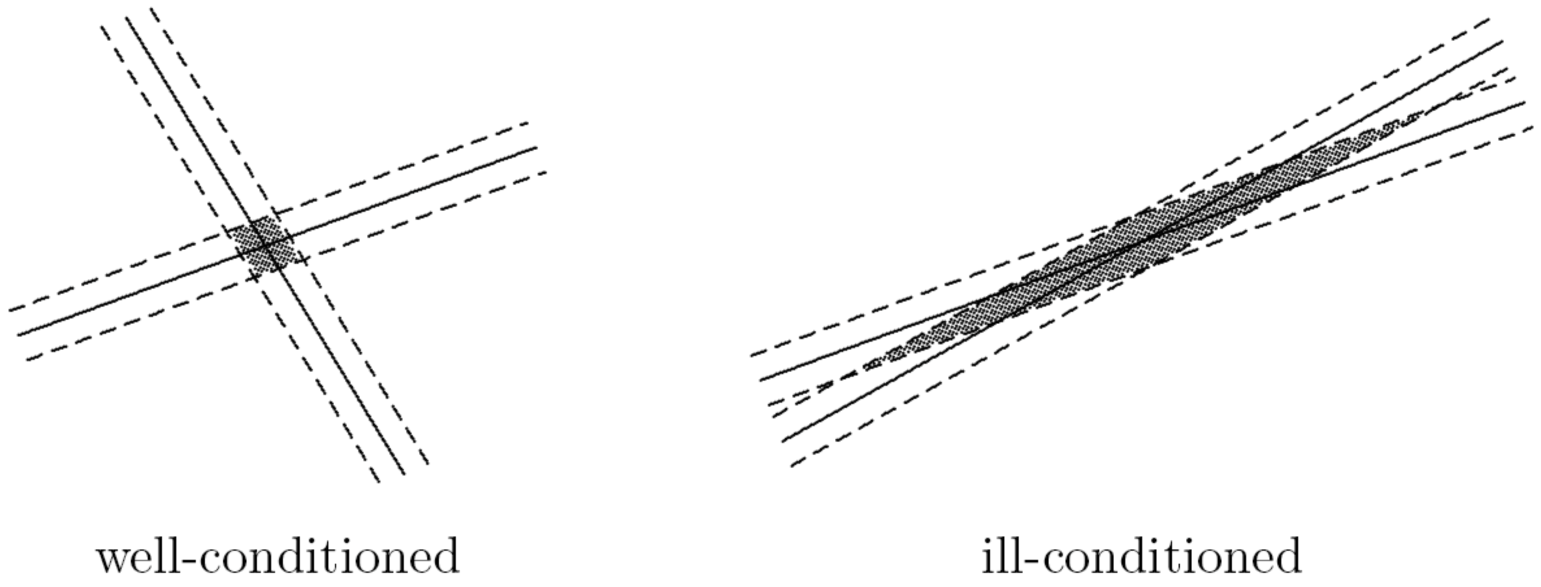
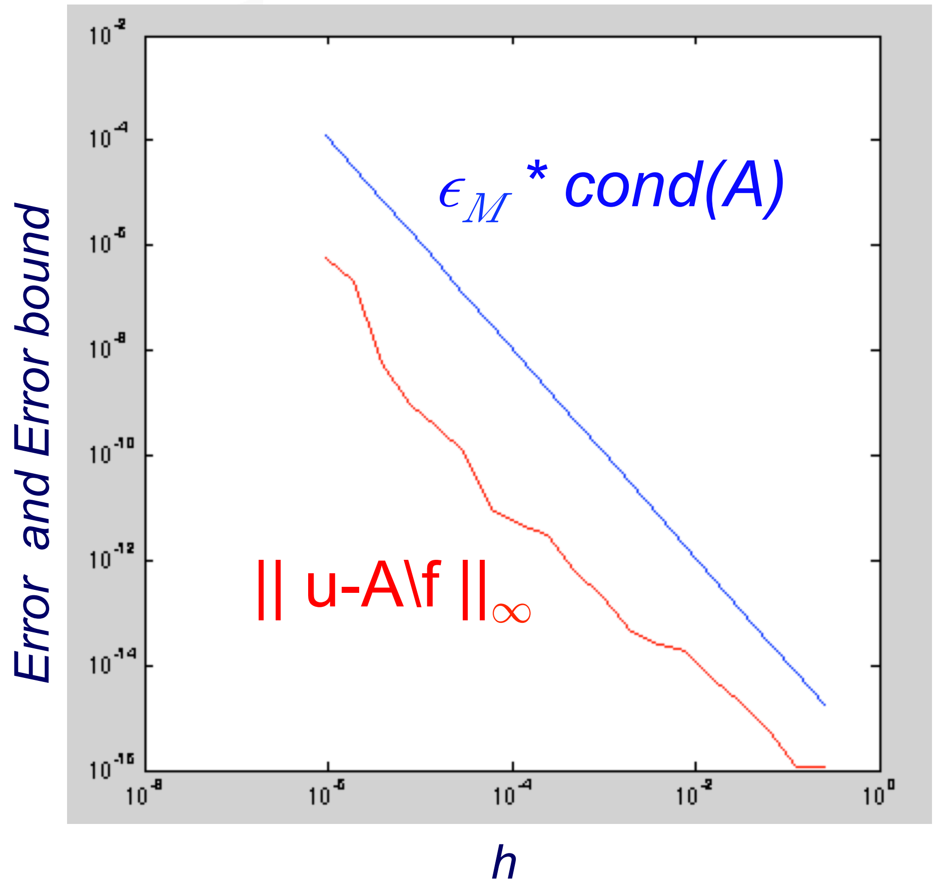


Illustration of Impact of cond(A)

```
%% Check the error in solving Au=f vs eps*cond(A).

%% Test problem is finite difference solution to -u" = f
%% on [0,1] with u(0)=u(1)=0.

for k=2:20; n = (2^k)-1; h=1/(n+1);

  e = ones(n,1);
  A = spdiags([-e 2*e -e],-1:1, n,n)/(h*h);
  x=1:n; x=h*x';
  ue=1+sin(pi*(8*x.*x));

  f=A*ue;
  u=A\f;

  hk(k)=h; ck(k)=cond(A);
  ek(k)=max(abs(u-ue))/max(ue);
end;
loglog(hk,ek,'r-',hk,eps*ck,'b-');
axis square
```

*Here, we see that \epsilon_M * cond(A) bounds the error in the solution to Au=f, as expected.*



Error Bounds – Caveats

- Normwise analysis bounds relative error in *largest* components of solution; relative error in smaller components can be much larger
  - Componentwise error bounds can be obtained, but somewhat more complicated
- Conditioning of system is affected by relative scaling of rows or columns
  - Ill-conditioning can result from poor scaling as well as near singularity
  - Rescaling can help the former, but not the latter

Residual

- *Residual vector* of approximate solution $\hat{\boldsymbol{x}}$ to linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is defined by

$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}}$$

- In theory, if $\boldsymbol{A}$ is nonsingular, then $\|\hat{\boldsymbol{x}} - \boldsymbol{x}\| = 0$ if, and only if, $\|\boldsymbol{r}\| = 0$, but they are not necessarily small simultaneously

- Since

$$\frac{\|\Delta \boldsymbol{x}\|}{\|\hat{\boldsymbol{x}}\|} \le \text{cond}(\boldsymbol{A}) \frac{\|\boldsymbol{r}\|}{\|\boldsymbol{A}\| \cdot \|\hat{\boldsymbol{x}}\|}$$

small relative residual implies small relative error in approximate solution *only if* $\boldsymbol{A}$ is well-conditioned

Residual, continued

- If computed solution $\hat{\boldsymbol{x}}$ exactly satisfies

  $$(\boldsymbol{A} + \boldsymbol{E})\hat{\boldsymbol{x}} = \boldsymbol{b}$$

  then

  $$\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{A}\|\;\|\hat{\boldsymbol{x}}\|} \le \frac{\|\boldsymbol{E}\|}{\|\boldsymbol{A}\|}$$

  so large *relative residual* implies large backward error in matrix, and algorithm used to compute solution is unstable

- Stable algorithm yields small relative residual regardless of conditioning of nonsingular system

- Small residual is easy to obtain, but does not necessarily imply computed solution is accurate

Solving Linear Systems

- To solve linear system, transform it into one whose solution is same but easier to compute
- What type of transformation of linear system leaves solution unchanged?
- We can *premultiply* (from left) both sides of linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by any *nonsingular* matrix $\boldsymbol{M}$ without affecting solution
- Solution to $\boldsymbol{M}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}\boldsymbol{b}$ is given by

$$\boldsymbol{x} = (\boldsymbol{M}\boldsymbol{A})^{-1}\boldsymbol{M}\boldsymbol{b} = \boldsymbol{A}^{-1}\boldsymbol{M}^{-1}\boldsymbol{M}\boldsymbol{b} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

Example: Permutations

- *Permutation matrix* $\boldsymbol{P}$ has one 1 in each row and column and zeros elsewhere, i.e., identity matrix with rows or columns permuted
- Note that $\boldsymbol{P}^{-1} = \boldsymbol{P}^T$ ***Matlab Demo: perm.m***
- Premultiplying both sides of system by permutation matrix, $\boldsymbol{PAx} = \boldsymbol{Pb}$, reorders rows, but solution $\boldsymbol{x}$ is unchanged
- Postmultiplying $\boldsymbol{A}$ by permutation matrix, $\boldsymbol{APx} = \boldsymbol{b}$, reorders columns, which permutes components of original solution

$$\boldsymbol{x} = (\boldsymbol{AP})^{-1}\boldsymbol{b} = \boldsymbol{P}^{-1}\boldsymbol{A}^{-1}\boldsymbol{b} = \boldsymbol{P}^T(\boldsymbol{A}^{-1}\boldsymbol{b})$$

Example: Diagonal Scaling

- Row scaling: premultiplying both sides of system by nonsingular diagonal matrix $\boldsymbol{D}$, $\boldsymbol{DAx} = \boldsymbol{Db}$, multiplies each row of matrix and right-hand side by corresponding diagonal entry of $\boldsymbol{D}$, but solution $\boldsymbol{x}$ is unchanged
- Column scaling: postmultiplying $\boldsymbol{A}$ by $\boldsymbol{D}$, $\boldsymbol{ADx} = \boldsymbol{b}$, multiplies each column of matrix by corresponding diagonal entry of $\boldsymbol{D}$, which rescales original solution

$$\boldsymbol{x} = (\boldsymbol{AD})^{-1}\boldsymbol{b} = \boldsymbol{D}^{-1}\boldsymbol{A}^{-1}\boldsymbol{b}$$

Premultiply by Diagonal Matrix: Row Scaling

$$\begin{pmatrix} \boxed{d_1\,a_{11}} & d_1\,a_{12} & d_1\,a_{13} \\ d_2\,a_{21} & d_2\,a_{22} & d_2\,a_{23} \\ d_3\,a_{31} & d_3\,a_{32} & d_3\,a_{33} \end{pmatrix} = \begin{pmatrix} \boxed{d_1 \quad\quad} & & \\ & d_2 & \\ & & d_3 \end{pmatrix} \begin{pmatrix} \boxed{\begin{matrix} a_{11} \\ a_{21} \\ a_{31} \end{matrix}} & \begin{matrix} a_{12} \\ a_{22} \\ a_{32} \end{matrix} & \begin{matrix} a_{13} \\ a_{23} \\ a_{33} \end{matrix} \end{pmatrix}$$

Note on Row Scaling / Permutation

$$D\mathbf{v} \;=\; \text{scale rows of } \mathbf{v}$$

$$P\mathbf{v} \;=\; \text{permute rows of } \mathbf{v}$$

$$DA \;=\; [\, D\mathbf{a}_1 \, D\mathbf{a}_2 \, \cdots \, D\mathbf{a}_n \,] \;=\; \text{scale rows of } A$$

$$PA \;=\; [\, P\mathbf{a}_1 \, P\mathbf{a}_2 \, \cdots \, P\mathbf{a}_n \,] \;=\; \text{permute rows of } A$$

Triangular Linear Systems

- What type of linear system is easy to solve?
- If one equation in system involves only one component of solution (i.e., only one entry in that row of matrix is nonzero), then that component can be computed by division
- If another equation in system involves only one additional solution component, then by substituting one known component into it, we can solve for other component
- If this pattern continues, with only one new solution component per equation, then all components of solution can be computed in succession.
- System with this property is called *triangular*

Triangular Matrices

- Two specific triangular forms are of particular interest
  - *lower triangular*: all entries *above* main diagonal are zero, $a_{ij} = 0$ for $i < j$
  - *upper triangular*: all entries *below* main diagonal are zero, $a_{ij} = 0$ for $i > j$
- Successive substitution process described earlier is especially easy to formulate for lower or upper triangular systems
- Any triangular matrix can be permuted into upper or lower triangular form by suitable row and column permutation

Forward-Substitution

- *Forward-substitution* for lower triangular system $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{b}$

$$x_1 = b_1/\ell_{11}, \quad x_i = \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} x_j \right) / \; \ell_{ii}, \quad i = 2, \ldots, n$$

```
for $j = 1$ to $n$                         { loop over columns }
    if $\ell_{jj} = 0$ then stop           { stop if matrix is singular }
    $x_j = b_j / \ell_{jj}$                { compute solution component }
    for $i = j + 1$ to $n$
        $b_i = b_i - \ell_{ij} x_j$        { update right-hand side }
    end
end
```

Back-Substitution

- *Back-substitution* for upper triangular system $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{b}$

$$x_n = b_n/u_{nn}, \quad x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j\right) \Big/ \, u_{ii}, \quad i = n-1, \ldots, 1$$

```
for $j = n$ to $1$                        { loop backwards over columns }
    if $u_{jj} = 0$ then stop             { stop if matrix is singular }
    $x_j = b_j/u_{jj}$                    { compute solution component }
    for $i = 1$ to $j - 1$
        $b_i = b_i - u_{ij}x_j$           { update right-hand side }
    end
end
```

Solution of Lower Triangular Systems

$$\begin{bmatrix} l_{11} & & & & & \\ l_{21} & l_{22} & & & & \\ l_{31} & l_{32} & l_{33} & & & \\ \vdots & & & \ddots & & \\ \vdots & & & & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

$$\text{for } i = 1, 2, \ldots, n: \quad x_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij}\, x_j \right).$$

As written:

```
for $i = 1 : n$
    $x_i = b_i$
    for $j = 1 : i - 1$
        $x_i = x_i - l_{ij}\, x_j$
    end
    $x_i = x_i / l_{ii}$
end
```

Better memory access (*faster*):

```
for $j = 1 : n$
    if $l_{jj} = 0$, stop - matrix is singular.
    $x_j = b_j / l_{jj}$
    for $i = j + 1 : n$
        $b_i = b_i - l_{ij}\, x_j$
    end
end
```

Solution of Upper Triangular Systems

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & \cdots & u_{1n} \\ & u_{22} & u_{23} & \cdots & \cdots & u_{2n} \\ & & u_{33} & & & u_{33} \\ & & & \ddots & & \vdots \\ & & & & \ddots & \vdots \\ & & & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

$$\text{for } i = n, n-1, \ldots, 1: \quad x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{n} u_{ij}\, x_j \right).$$

As written:

```
for $i = n : 1$
    $x_i = b_i$
    for $j = i + 1 : n$
        $x_i = x_i - u_{ij}\, x_j$
    end
    $x_i = x_i / u_{ii}$
end
```

Better memory access (*faster*):

```
for $j = n : 1$
    if $u_{jj} = 0$, stop - matrix is singular.
    $x_j = b_j / u_{jj}$
    for $i = 1 : j - 1$
        $b_i = b_i - u_{ij}\, x_j$
    end
end
```

What is the cost ??

Solution of Upper Banded Systems

Suppose U is a *banded matrix*: $u_{ij} = 0$, $j > i + \beta$.

For example, $\beta = 2$:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & & & \\ & u_{22} & u_{23} & u_{14} & & \\ & & u_{33} & \ddots & \ddots & \\ & & & \ddots & \ddots & u_{n-2,n} \\ & & & & \ddots & u_{n-1,n} \\ & & & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

$$\text{for } i = n, n-1, \ldots, 1: \quad x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{\min(i+\beta,n)} u_{ij}\, x_j \right).$$

What is the cost ??

Solution of Upper Banded Systems

$$\text{for } i = n, n-1, \ldots, 1: \quad x_i \; = \; \frac{1}{u_{ii}} \left(b_i \; - \sum_{j=i+1}^{\min(i+\beta,n)} u_{ij}\, x_j \right).$$

As written:

```
for $i = n : 1$
    $x_i \; = \; b_i, \;\; j_{\max} := \min(j+\beta, n)$
    for $j = i+1 : j_{\max}$
        $x_i \; = \; x_i \; - \; u_{ij}\, x_j$
    end
    $x_i \; = \; x_i / u_{ii}$
end
```

Better memory access (*faster*):

```
for $j = n : 1$
    if $u_{jj} \; = \; 0$, stop - matrix is singular.
    $x_j \; = \; b_j / u_{jj}, \;\; i_{\min} := \max(1, j-\beta)$
    for $i = i_{\min} : j-1$
        $b_i \; = \; b_i \; - \; u_{ij}\, x_j$
    end
end
```

- In this case, there are $\sim 2\beta n$ operations and $\sim \beta n$ memory references (one for each u_{ij}).
- Often $\beta \ll n$, which means that the upper-banded system is *much* faster to solve than the full upper triangular system.
- The same savings applies to the lower-banded case.

Generating Triangular Systems: LU Factorization

$$A = LU$$

Generating Upper Triangular Systems: *LU* Factorization

- Example:

$$\begin{bmatrix} 1 & 2 & 3 & & \\ & 4 & 4 & 6 & 1 \\ & 8 & 8 & 9 & 2 \\ & 6 & 1 & 3 & 3 \\ & 4 & 2 & 8 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 4 \\ 4 \\ 4 \end{bmatrix}$$

- First column is already in upper triangular form.
- Eliminate second column:

$$\begin{aligned} \text{row}_3 &\longleftarrow \text{row}_3 - \frac{8}{4} \times \text{row}_2 \\ \text{row}_4 &\longleftarrow \text{row}_4 - \frac{6}{4} \times \text{row}_2 \\ \text{row}_5 &\longleftarrow \text{row}_5 - \frac{4}{4} \times \text{row}_2 \end{aligned} \qquad \begin{bmatrix} 1 & 2 & 3 & & \\ & 4 & 4 & 6 & 1 \\ & & 0 & -3 & 0 \\ & & -5 & -6 & \frac{3}{2} \\ & & -2 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ -4 \\ -2 \\ 0 \end{bmatrix}$$

- $a_{22} = 4$ is the *pivot*
- row_2 is the *pivot row*
- $l_{32} = \frac{8}{4}$, $l_{42} = \frac{6}{4}$, $l_{52} = \frac{4}{4}$, is the *multiplier column.*

Generating Upper Triangular Systems: LU Factorization

- Augmented form. Store $\mathbf{b}$ in $A(:, n+1)$:

$$\left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & 8 & 8 & 9 & 2 & 4 \\ & 6 & 1 & 3 & 3 & 4 \\ & 4 & 2 & 8 & 4 & 4 \end{array}\right] \longrightarrow \left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & 0 & -3 & 0 & -4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & -2 & 2 & 3 & 0 \end{array}\right]$$

This Case.	***General Case.***
pivot $=$ 4	$=$ a_{kk} when zeroing the kth column.
pivot row $=$ $[\ 4\ \ 6\ \ 1 \mid 4\ ]$	$= \mathbf{r}_k^T = a_{kj},\ j = k+1, \dots, n\,[+b_k]$
multiplier column $= \dfrac{1}{4}\begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}$	$= \mathbf{c}_k = \dfrac{a_{ik}}{a_{kk}},\ i = k+1, \dots, n$
$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$	

Generating Upper Triangular Systems: LU Factorization

- Augmented form. Store $\mathbf{b}$ in $A(:, n+1)$:

$$\left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & \boxed{4 & 6 & 1} & \boxed{4} \\ & \boxed{8} & 8 & 9 & 2 & 4 \\ & \boxed{6} & 1 & 3 & 3 & 4 \\ & \boxed{4} & 2 & 8 & 4 & 4 \end{array}\right] \longrightarrow \left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & 0 & -3 & 0 & -4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & -2 & 2 & 3 & 0 \end{array}\right]$$

This Case.		***General Case.***
pivot	$= 4$	$= a_{kk}$ when zeroing the kth column.
pivot row	$= \boxed{[\;4\;\;6\;\;1 \mid 4\;]}$	$= \mathbf{r}_k^T = a_{kj},\ j = k+1, \dots, n\,[+b_k]$
multiplier column	$= \dfrac{1}{4}\boxed{\begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}}$	$= \mathbf{c}_k = \dfrac{a_{ik}}{a_{kk}},\ i = k+1, \dots, n$
	$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$	

Generating Upper Triangular Systems: *LU* Factorization

- Augmented form. Store $\mathbf{b}$ in $A(:, n+1)$:

$$\left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & \boxed{4 & 6 & 1 & 4} \\ & \boxed{8} & 8 & 9 & 2 & 4 \\ & 6 & 1 & 3 & 3 & 4 \\ & 4 & 2 & 8 & 4 & 4 \end{array}\right] \longrightarrow \left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & 0 & -3 & 0 & -4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & -2 & 2 & 3 & 0 \end{array}\right]$$

This Case.			***General Case.***
pivot	$=$	4	$= a_{kk}$ when zeroing the kth column.
pivot row	$=$	$\boxed{[\ 4\ \ 6\ \ 1 \mid 4\ ]}$	$= \mathbf{r}_k^T = a_{kj},\ j = k+1, \ldots, n\,[+b_k]$
multiplier column	$=$	$\frac{1}{4}\boxed{\begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}}$	$= \mathbf{c}_k = \frac{a_{ik}}{a_{kk}},\ i = k+1, \ldots, n$
	$=$	$\begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$	$\boxed{\mathbf{c}_k \longrightarrow \mathbf{l}_k, \text{ store as column } k \text{ of } L.}$

kth Update Step

- Look more closely at the kth update step for Gaussian elimination.
- Assume A is $m \times n$, which covers the case where A is augmented with the right-hand side vector.
- For each row i, with $i > k$, we want to generate a zero in place of a_{ij}.
- We do this by subtracting a multiple of row k from row i.
- This operation can be expressed in several equivalent ways:

$$\begin{aligned} \text{row}_i &= \text{row}_i - \frac{a_{ik}}{a_{kk}} \times \text{row}_k \\ a_{ij} &= a_{ij} - a_{ik}\, a_{kk}^{-1}\, a_{kj} \qquad j = k+1, \dots, n \\ &= a_{ij} - (\mathbf{c}_k)_i \left(\mathbf{r}_k^T\right)_j \qquad j = k+1, \dots, n \\ A^{(k+1)} &= A^{(k)} - \mathbf{c}_k\, \mathbf{r}_k^T, \end{aligned}$$

Matlab: lu_demo_1.m

- Here, $\mathbf{c}_k$ is the column vector with entries $(\mathbf{c}_k)_i = a_{ik}/a_{kk}$, and $\mathbf{r}_k^T$ is the row vector with entries $\left(\mathbf{r}_k^T\right)_j = a_{kj}$.
- Formally, we think of $(\mathbf{c}_k)_i = 0,\ i \leq k$ and $\left(\mathbf{r}_k^T\right)_j = 0,\ j \leq k$, though we would implement as an update only to the active submatrix.
- The $m \times n$ matrix $\mathbf{c}_k\, \mathbf{r}_k^T$ is of rank 1. All columns are multiples of the only linearly independent column, $\mathbf{c}_k$.
- We typically save the entries of the multiplier column as the kth column of a lower triangular matrix: $l_{ik} := (\mathbf{c}_k)_i$.

Multiplier Columns = $\mathbf{l}_k$: $LU = A$

- $A^{(1)} := A, \; A^{(k+1)} = A^{(k)} - \mathbf{c}_k \mathbf{r}_k^T$.

$$LU = \begin{bmatrix} 1 & & \\ a_{21}^{(1)}/a_{11}^{(1)} & 1 & \\ a_{31}^{(1)}/a_{11}^{(1)} & a_{31}^{(2)}/a_{22}^{(2)} & 1 \end{bmatrix} \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ & a_{22}^{(2)} & a_{23}^{(2)} \\ & & a_{33}^{(3)} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(2)} + \frac{a_{21}^{(1)}a_{12}^{(1)}}{a_{11}^{(1)}} & a_{23}^{(2)} + \frac{a_{21}^{(1)}a_{13}^{(1)}}{a_{11}^{(1)}} \\ a_{31}^{(1)} & etc. & etc. \end{bmatrix}$$

- Recall, for example,

$$a_{22}^{(2)} = a_{22}^{(1)} - \frac{a_{21}^{(1)}a_{12}^{(1)}}{a_{11}^{(1)}}, \text{ or}$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}a_{kj}^{(k)}}{a_{kk}^{(k)}}, \text{in general.}$$

- Thus, we see that the 2-2 entry of LU is indeed $a_{22}^{(1)} = a_{22}$, etc.

Using LU Factorization in Practice

- Give $LU = A$, we can solve $A\mathbf{x} = \mathbf{b}$ as follows:

$$\begin{aligned}\text{Given:}\quad A\mathbf{x} &= L\,U\,\mathbf{x} = \mathbf{b} \\ L\,(U\,\mathbf{x}) &= L\mathbf{b} = \mathbf{b}\end{aligned}$$

$$\begin{aligned}\text{Solve:}\quad L\mathbf{y} &= \mathbf{b} \\ U\mathbf{x} &= \mathbf{y}\end{aligned}$$

- We have seen already that the total solve cost (for L and U solves) is $2 \times n^2$.
- What about the factor cost, $A \longrightarrow LU$?

LU Factorization Costs (Important)

- In general, the cost for $A \longrightarrow LU$ is $O(n^3)$.
- It is large (i.e., it is not optimal, which would be $O(n)$), and therefore important.
- The dominant cost comes from the essential update step:

$$A^{(k+1)} \;\; = \;\; A^{(k)} \; - \; \mathbf{c}_k \, \mathbf{r}_k^T,$$

  which is effected for $k = 1, \ldots, n-1$ steps.
- If A is square $(n \times n)$, then $\mathbf{c}_k \, \mathbf{r}_k^T$ is a square matrix with $(n-k)^2$ nonzeros.
- Each entry requires one "*" and its subtraction from $A^{(k)}$ requires one "-".
- Total cost is $2 \times [\,(n-1)^2 \, + \, (n-2)^2 \, + \, \ldots (1)^2] \;\sim\; 2n^3/3$ operations.
- **Example:** $n = 10^3 \;\longrightarrow n^3 = 10^9$. Cost is about 0.6 billion operations.
  With a 3 GHz clock and 2 floating point ops / clock, expect about 0.1 seconds (very fast).
- **Example:** $n = 10^4 \;\longrightarrow n^3 = 10^{12}$. Cost is about 600 billion operations.
  With a 3 GHz clock and 2 floating point ops / clock, expect about 10.0 seconds.

Final Topics

- Pivoting / zeros & stability
  - Approach
  - Permutation Matrices
  - Stability
  - Cost
- SPD / Cholesky Factorization
- Banded Factorization
  - Approach
  - Cost

Pivoting

- We return to our original 5×5 example. The next step would be:

$$\left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & 0 & -3 & 0 & -4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & -2 & 2 & 3 & 0 \end{array}\right]$$

- Here, we have diffiulty because the nominal pivot, a_{33} is zero.
- The remedy is to exchange rows with one of the remaining two, since the order of the equations is immaterial.
- For numerical stability, we choose the row that maximizes $|a_{ik}|$.
- This choice ensures that all entries in the multiplier column are less than one in modulus.

Next Step: $k = k + 1$

- After switching rows, we have

$$\left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & 0 & -3 & 0 & -4 \\ & & -2 & 2 & 3 & 0 \end{array}\right] \longrightarrow \left[\begin{array}{ccccc|c} 1 & 2 & 3 & & & 0 \\ & 4 & 4 & 6 & 1 & 4 \\ & & -5 & -6 & \frac{3}{2} & -2 \\ & & 0 & -3 & 0 & -4 \\ & & 0 & 4\frac{2}{5} & 2\frac{2}{5} & \frac{4}{5} \end{array}\right]$$

$$\begin{aligned} \text{pivot} &= -5 \\ \text{pivot row} &= \left[\begin{array}{cc|c} -6 & \dfrac{3}{2} & -2 \end{array}\right] \\ \text{multiplier column} &= \frac{1}{-5}\begin{bmatrix} 0 \\ -2 \end{bmatrix} \end{aligned}$$

Row Interchanges

- Gaussian elimination breaks down if leading diagonal entry of remaining unreduced matrix is zero at any stage
- Easy fix: if diagonal entry in column k is zero, then interchange row k with some subsequent row having nonzero entry in column k and then proceed as usual
- If there is no nonzero on or below diagonal in column k, then there is nothing to do at this stage, so skip to next column
- Zero on diagonal causes resulting upper triangular matrix $\boldsymbol{U}$ to be singular, but LU factorization can still be completed
- Subsequent back-substitution will fail, however, as it should for singular matrix

Partial Pivoting

- In principle, any nonzero value will do as pivot, but in practice pivot should be chosen to minimize error propagation
- To avoid amplifying previous rounding errors when multiplying remaining portion of matrix by elementary elimination matrix, multipliers should not exceed 1 in magnitude
- This can be accomplished by choosing entry of largest magnitude on or below diagonal as pivot at each stage
- Such *partial pivoting* is essential in practice for numerically stable implementation of Gaussian elimination for general linear systems

LU Factorization with Partial Pivoting

- With partial pivoting, each $\boldsymbol{M}_k$ is preceded by permutation $\boldsymbol{P}_k$ to interchange rows to bring entry of largest magnitude into diagonal pivot position
- Still obtain $\boldsymbol{M}\boldsymbol{A} = \boldsymbol{U}$, with $\boldsymbol{U}$ upper triangular, but now

$$\boldsymbol{M} = \boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}\cdots\boldsymbol{M}_1\boldsymbol{P}_1$$

- $\boldsymbol{L} = \boldsymbol{M}^{-1}$ is still triangular in general sense, but not necessarily *lower* triangular
- Alternatively, we can write

$$\boxed{\boldsymbol{P}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}}$$

where $\boldsymbol{P} = \boldsymbol{P}_{n-1}\cdots\boldsymbol{P}_1$ permutes rows of $\boldsymbol{A}$ into order determined by partial pivoting, and now $\boldsymbol{L}$ is lower triangular

Complete Pivoting

- *Complete pivoting* is more exhaustive strategy in which largest entry in entire remaining unreduced submatrix is permuted into diagonal pivot position
- Requires interchanging columns as well as rows, leading to factorization

$$\boldsymbol{PAQ} = \boldsymbol{LU}$$

  with $\boldsymbol{L}$ unit lower triangular, $\boldsymbol{U}$ upper triangular, and $\boldsymbol{P}$ and $\boldsymbol{Q}$ permutations
- Numerical stability of complete pivoting is theoretically superior, but pivot search is more expensive than for partial pivoting
- Numerical stability of partial pivoting is more than adequate in practice, so it is almost always used in solving linear systems by Gaussian elimination

Example: Permutations

- *Permutation matrix* $\boldsymbol{P}$ has one 1 in each row and column and zeros elsewhere, i.e., identity matrix with rows or columns permuted
- Note that $\boldsymbol{P}^{-1} = \boldsymbol{P}^T$

Matlab Demo: perm.m

- Premultiplying both sides of system by permutation matrix, $\boldsymbol{PAx} = \boldsymbol{Pb}$, reorders rows, but solution $\boldsymbol{x}$ is unchanged
- Postmultiplying $\boldsymbol{A}$ by permutation matrix, $\boldsymbol{APx} = \boldsymbol{b}$, reorders columns, which permutes components of original solution

$$\boldsymbol{x} = (\boldsymbol{AP})^{-1}\boldsymbol{b} = \boldsymbol{P}^{-1}\boldsymbol{A}^{-1}\boldsymbol{b} = \boldsymbol{P}^T(\boldsymbol{A}^{-1}\boldsymbol{b})$$

Comments About Permutation Matrices

- As with A^{-1}, we never actually form them – we simply use pointers to swap rows (or columns).
- However, they are notationally convenient, and can be constructed from elementary permutation matrices that swap just two rows, e.g. If P_{ij} is the identity matrix with rows i and j swapped, then we have:

$$P_{ij}^{-1} = P_{ij}^{T} = P_{ij}$$

So applying P_{ij} twice brings two rows back to their original position.

- We can construct a compound permutation matrix as the product of these swaps, e.g., $P = P_{21}P_{43}$
- The compound permutation matrix is not symmetric, but we still have

$$P^{-1} = P^{T} = P_{43}^{T}\,P_{21}^{T} = P_{43}\,P_{21}$$

perm.m

```
%%   perm.m - permutation demo

     A= [ 1  2  3  4  ;
          2  3  4  5  ;
          3  4  5  6  ;
          4  5  6  7  ];

     p = [  4  ;  %    Row  4  will  go  to  Row  1
            1  ;  %    Row  1  will  go  to  Row  2
            2  ;  %    Row  2  will  go  to  Row  3
            3  ];%    Row  3  will  go  to  Row  4

     I=eye(4);   P = I(p,:);
     A, P
     display('Row permutation: P*A'), PA=P*A
     display('Col permutation: A*P'), AP=A*P

display('Permutation of vector:')
     c          = [ b   P*b ];
     b1         = b(p); b2(p,1) = b;
     [ c b1 b2 ]
```

```
A =
          1          2          3          4
          2          3          4          5
          3          4          5          6
          4          5          6          7
P =
          0          0          0          1
          1          0          0          0
          0          1          0          0
          0          0          1          0
Row permutation: P*A
PA =
          4          5          6          7
          1          2          3          4
          2          3          4          5
          3          4          5          6
Col permutation: A*P
AP =
          2          3          4          1
          3          4          5          2
          4          5          6          3
          5          6          7          4
Permutation of vector:
ans =
          1          4          4          2
          2          1          1          3
          3          2          2          4
          4          3          3          1
```

Example: Pivoting

- Need for pivoting has nothing to do with whether matrix is singular or nearly singular
- For example,

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

is nonsingular yet has no LU factorization unless rows are interchanged, whereas

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

is singular yet has LU factorization

Example: Small Pivots

- To illustrate effect of small pivots, consider

$$\boldsymbol{A} = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}$$

where ϵ is positive number smaller than ϵ_{mach}

- If rows are not interchanged, then pivot is ϵ and multiplier is $-1/\epsilon$, so

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 \\ -1/\epsilon & 1 \end{bmatrix}, \quad \boldsymbol{L} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix},$$

$$\boldsymbol{U} = \begin{bmatrix} \epsilon & 1 \\ 0 & 1 - 1/\epsilon \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}$$

in floating-point arithmetic, but then

$$\boldsymbol{L}\boldsymbol{U} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} \neq \boldsymbol{A}$$

Example, continued

- Using small pivot, and correspondingly large multiplier, has caused loss of information in transformed matrix
- If rows interchanged, then pivot is 1 and multiplier is $-\epsilon$, so

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 \\ -\epsilon & 1 \end{bmatrix}, \quad \boldsymbol{L} = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix},$$

$$\boldsymbol{U} = \begin{bmatrix} 1 & 1 \\ 0 & 1-\epsilon \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

  in floating-point arithmetic
- Thus,

$$\boldsymbol{LU} = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix}$$

  which is correct after permutation

Pivoting:

- Moving small pivots down moves us closer to upper triangular form, with ***no round-off.***

$$PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix}$$

- A general principle in numerical computing regarding round-off:

  Small corrections are preferred to large ones.

- Failure to pivot can result in all subsequent rows looking like multiples of the kth row → ***singular submatrix.***

Failure to pivot can result in all subsequent rows looking like multiples of the kth row:

- Consider

$$A = \begin{pmatrix} \epsilon & \text{——} \underline{r}_1^T \text{——} \\ a_{21} & \text{——} \underline{r}_2^T \text{——} \\ a_{31} & \text{——} \underline{r}_3^T \text{——} \\ \vdots & \text{——} \vdots \text{——} \end{pmatrix}$$

Gaussian elimination leads to

$$\underline{r}_i \longleftarrow \underline{r}_i - \frac{a_{i1}}{\epsilon}\underline{r}_1 \approx -\frac{a_{i1}}{\epsilon}\underline{r}_1 .$$

- Matlab example “pivot.m”

pivot_gui.m

```
1.0e-18    1.0000    2.0000    3.0000    4.0000

 1.0000    4.0000    4.0000    6.0000    1.0000

 2.0000    8.0000    7.0000    9.0000    2.0000

 3.0000    6.0000    1.0000    3.0000    3.0000

 4.0000    4.0000    2.0000    8.0000    4.0000
```

Failure to Pivot, Noncatastrophic Case

- In cases where the nominal pivot is small but > ϵ_M, we simply are driving down the number of significant digits that represent the remainder of the matrix A.

- In essence, we are driving the rows (or columns) to be similar, which is equivalent to saying that we have nearly parallel columns.

- We saw already a 2 x 2 example where the condition number of the matrix with 2 unit-norm vectors scales like 2 / θ , where θ is the (small) angle between the column vectors.

Partial Pivoting: Costs

Procedure:

- For each k, pick k' such that $|a_{k'k}| \geq |a_{ik}|$, $i \geq k$.
- Swap rows k and k'.
- Proceed with central update step: $A^{(k+1)} \;=\; A^{(k)} \;-\; \mathbf{c}_k\,\mathbf{r}_k^T$

Costs:

- For each step, search is $O(n-k)$, total cost is $\approx n^2/2$.
- For each step, row swap is $O(n-k)$, total cost is $\approx n^2/2$.
- Total cost for partial pivoting is $O(n^2) \ll 2n^3/3$.
- If we use *full pivoting*, total search cost such that $|a_{k'k''}| \geq |a_{ij}|$, $i, j \geq k$, is $O(n^3)$.
- Row and column exchange costs still total only $O(n^2)$.

Note: Partial (row) pivoting ensures that multiplier column entries have modulus ≤ 1. (Good.)

Partial Pivoting: LU=PA

- Note: If we swap rows of A, we are swapping equations.
- We must swap rows of $\mathbf{b}$.
- LU routines normally return the pivot index vector to effect this exchange.
- Nominally, it looks like a permutation matrix P, which is simply the identity matrix with rows interchanged.
- If we swap equations, we must also swap rows of L
- If we are consistent, we can swap rows at any time (i.e., A, or L) and get the same final factorization: $LU = PA$.
- Most codes swap $A^{(k+1)}$, but not the factors in L that have already been stored.
- Swapping rows of $A^{(k+1)}$ helps with speed (vectorization) of $A^{(k+1)} = A^{(k)} - \mathbf{c}_k \, \mathbf{r}_k^T$.
- In parallel computing, one would *not* swap the pivot row. Just pass the pointer to the processor holding the new pivot row, where the swap would take place locally.

Pivoting, continued

- Although pivoting is generally required for stability of Gaussian elimination, pivoting is *not* required for some important classes of matrices
  - *Diagonally dominant*

$$\sum_{i=1,\, i\neq j}^{n} |a_{ij}| < |a_{jj}|, \quad j = 1, \dots, n$$

  - *Symmetric positive definite*

$$\boldsymbol{A} = \boldsymbol{A}^T \quad \text{and} \quad \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} > 0 \ \text{ for all } \ \boldsymbol{x} \neq \boldsymbol{0}$$

- The following slides present the book’s derivation of the LU factorization process.
- I’ll highlight a few of them that show the equivalence between the outer product approach and the elementary elimination matrix approach.

Example: Triangular Linear System

$$\begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$$

- Using back-substitution for this upper triangular system, last equation, $4x_3 = 8$, is solved directly to obtain $x_3 = 2$
- Next, x_3 is substituted into second equation to obtain $x_2 = 2$
- Finally, both x_3 and x_2 are substituted into first equation to obtain $x_1 = -1$

Elimination

- To transform general linear system into triangular form, we need to replace selected nonzero entries of matrix by zeros
- This can be accomplished by taking linear combinations of rows
- Consider 2-vector $\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$
- If $a_1 \neq 0$, then

$$\begin{bmatrix} 1 & 0 \\ -a_2/a_1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ 0 \end{bmatrix}$$

Elementary Elimination Matrices

- More generally, we can annihilate *all* entries below kth position in n-vector $\boldsymbol{a}$ by transformation

$$\boldsymbol{M}_k\boldsymbol{a} = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_n & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $m_i = a_i/a_k$, $i = k+1, \ldots, n$

- Divisor a_k, called *pivot*, must be nonzero

Elementary Elimination Matrices, continued

- Matrix $\boldsymbol{M}_k$, called *elementary elimination matrix*, adds multiple of row k to each subsequent row, with *multipliers* m_i chosen so that result is zero
- $\boldsymbol{M}_k$ is unit lower triangular and nonsingular
- $\boldsymbol{M}_k = \boldsymbol{I} - \boldsymbol{m}_k \boldsymbol{e}_k^T$, where $\boldsymbol{m}_k = [0, \ldots, 0, m_{k+1}, \ldots, m_n]^T$ and $\boldsymbol{e}_k$ is kth column of identity matrix
- $\boldsymbol{M}_k^{-1} = \boldsymbol{I} + \boldsymbol{m}_k \boldsymbol{e}_k^T$, which means $\boldsymbol{M}_k^{-1} =: \boldsymbol{L}_k$ is same as $\boldsymbol{M}_k$ except signs of multipliers are reversed

Elementary Elimination Matrices, continued

- If $\boldsymbol{M}_j$, $j > k$, is another elementary elimination matrix, with vector of multipliers $\boldsymbol{m}_j$, then

$$\begin{aligned}\boldsymbol{M}_k\boldsymbol{M}_j &= \boldsymbol{I} - \boldsymbol{m}_k\boldsymbol{e}_k^T - \boldsymbol{m}_j\boldsymbol{e}_j^T + \boldsymbol{m}_k\boldsymbol{e}_k^T\boldsymbol{m}_j\boldsymbol{e}_j^T \\ &= \boldsymbol{I} - \boldsymbol{m}_k\boldsymbol{e}_k^T - \boldsymbol{m}_j\boldsymbol{e}_j^T\end{aligned}$$

which means product is essentially “union,” and similarly for product of inverses, $\boldsymbol{L}_k\boldsymbol{L}_j$

Comment on update step and $\underline{m}_k \underline{e}^T_k$

- Recall, $\underline{v} = C\,\underline{w} \in \text{span}\{C\}$.
- $\therefore\ V = (\,\underline{v}_1\,\underline{v}_2 \ldots \underline{v}_n) = C\,(\,\underline{w}_1\,\underline{w}_2 \ldots \underline{w}_n) \in \text{span}\{C\}$.
- If $C = \underline{c}$, i.e., C is a column vector and therefore of rank 1, then V is in span{C} and is of rank 1.
- All columns of V are multiples of $\underline{c}$.
- Thus, $W = \underline{c}\,\underline{r}^T$ is an n x n matrix of rank 1.
  - All columns are multiples of the first column and
  - All rows are multiples of the first row.

Elementary Elimination Matrices, continued

- Matrix $\boldsymbol{M}_k$, called *elementary elimination matrix*, adds multiple of row k to each subsequent row, with *multipliers* m_i chosen so that result is zero
- $\boldsymbol{M}_k$ is unit lower triangular and nonsingular
- $\boldsymbol{M}_k = \boldsymbol{I} - \boldsymbol{m}_k \boldsymbol{e}_k^T$, where $\boldsymbol{m}_k = [0, \dots, 0, m_{k+1}, \dots, m_n]^T$ and $\boldsymbol{e}_k$ is kth column of identity matrix
- $\boldsymbol{M}_k^{-1} = \boldsymbol{I} + \boldsymbol{m}_k \boldsymbol{e}_k^T$, which means $\boldsymbol{M}_k^{-1} =: \boldsymbol{L}_k$ is same as $\boldsymbol{M}_k$ except signs of multipliers are reversed

Example: Elementary Elimination Matrices

- For $\boldsymbol{a} = \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix}$,

$$\boldsymbol{M}_1\boldsymbol{a} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

and

$$\boldsymbol{M}_2\boldsymbol{a} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}$$

Example, continued

- Note that

$$\boldsymbol{L}_1 = \boldsymbol{M}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{L}_2 = \boldsymbol{M}_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/2 & 1 \end{bmatrix}$$

and

$$\boldsymbol{M}_1\boldsymbol{M}_2 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix}, \quad \boldsymbol{L}_1\boldsymbol{L}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1/2 & 1 \end{bmatrix}$$

Gaussian Elimination

- To reduce general linear system $\boldsymbol{Ax} = \boldsymbol{b}$ to upper triangular form, first choose $\boldsymbol{M}_1$, with a_{11} as pivot, to annihilate first column of $\boldsymbol{A}$ below first row
  - System becomes $\boldsymbol{M}_1\boldsymbol{Ax} = \boldsymbol{M}_1\boldsymbol{b}$, but solution is unchanged
- Next choose $\boldsymbol{M}_2$, using a_{22} as pivot, to annihilate second column of $\boldsymbol{M}_1\boldsymbol{A}$ below second row
  - System becomes $\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{Ax} = \boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{b}$, but solution is still unchanged
- Process continues for each successive column until all subdiagonal entries have been zeroed

Gaussian Elimination

- To reduce general linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ to upper triangular form, first choose $\boldsymbol{M}_1$, with a_{11} as pivot, to annihilate first column of $\boldsymbol{A}$ below first row
  - System becomes $\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}_1\boldsymbol{b}$, but solution is unchanged
- Next choose $\boldsymbol{M}_2$, using a_{22} as pivot, to annihilate second column of $\boldsymbol{M}_1\boldsymbol{A}$ below second row
  - System becomes $\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{b}$, but solution is still unchanged
- *Technically, this should be a'$_{22}$, the 2-2 entry in A' := M$_1$A. Thus, we don't know all the pivots in advance.*

Gaussian Elimination, continued

- Resulting upper triangular linear system

$$\begin{aligned} \boldsymbol{M}_{n-1} \cdots \boldsymbol{M}_1 \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{M}_{n-1} \cdots \boldsymbol{M}_1 \boldsymbol{b} \\ \boldsymbol{M}\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{M}\boldsymbol{b} \end{aligned}$$

  can be solved by back-substitution to obtain solution to original linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

- Process just described is called *Gaussian elimination*

LU Factorization

- Product $\boldsymbol{L}_k\boldsymbol{L}_j$ is unit lower triangular if $k < j$, so

$$\boldsymbol{L} = \boldsymbol{M}^{-1} = \boldsymbol{M}_1^{-1} \cdots \boldsymbol{M}_{n-1}^{-1} = \boldsymbol{L}_1 \cdots \boldsymbol{L}_{n-1}$$

  is unit lower triangular

- By design, $\boldsymbol{U} = \boldsymbol{M}\boldsymbol{A}$ is upper triangular
- So we have

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

  with $\boldsymbol{L}$ unit lower triangular and $\boldsymbol{U}$ upper triangular

- Thus, Gaussian elimination produces *LU factorization* of matrix into triangular factors

LU Factorization, continued

- Having obtained LU factorization, $\boldsymbol{Ax} = \boldsymbol{b}$ becomes $\boldsymbol{LUx} = \boldsymbol{b}$, and can be solved by forward-substitution in lower triangular system $\boldsymbol{Ly} = \boldsymbol{b}$, followed by back-substitution in upper triangular system $\boldsymbol{Ux} = \boldsymbol{y}$

- Note that $\boldsymbol{y} = \boldsymbol{Mb}$ is same as transformed right-hand side in Gaussian elimination

- Gaussian elimination and LU factorization are two ways of expressing same solution process

Example: Gaussian Elimination

- Use Gaussian elimination to solve linear system

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \boldsymbol{b}$$

- To annihilate subdiagonal entries of first column of $\boldsymbol{A}$,

$$\boldsymbol{M}_1\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix},$$

$$\boldsymbol{M}_1\boldsymbol{b} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}$$

Example, continued

- To annihilate subdiagonal entry of second column of $\boldsymbol{M}_1\boldsymbol{A}$,

$$\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = \boldsymbol{U},$$

$$\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{b} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} = \boldsymbol{M}\boldsymbol{b}$$

Example, continued

- We have reduced original system to equivalent upper triangular system

$$\boldsymbol{U}\boldsymbol{x} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} = \boldsymbol{M}\boldsymbol{b}$$

which can now be solved by back-substitution to obtain

$$\boldsymbol{x} = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}$$

Example, continued

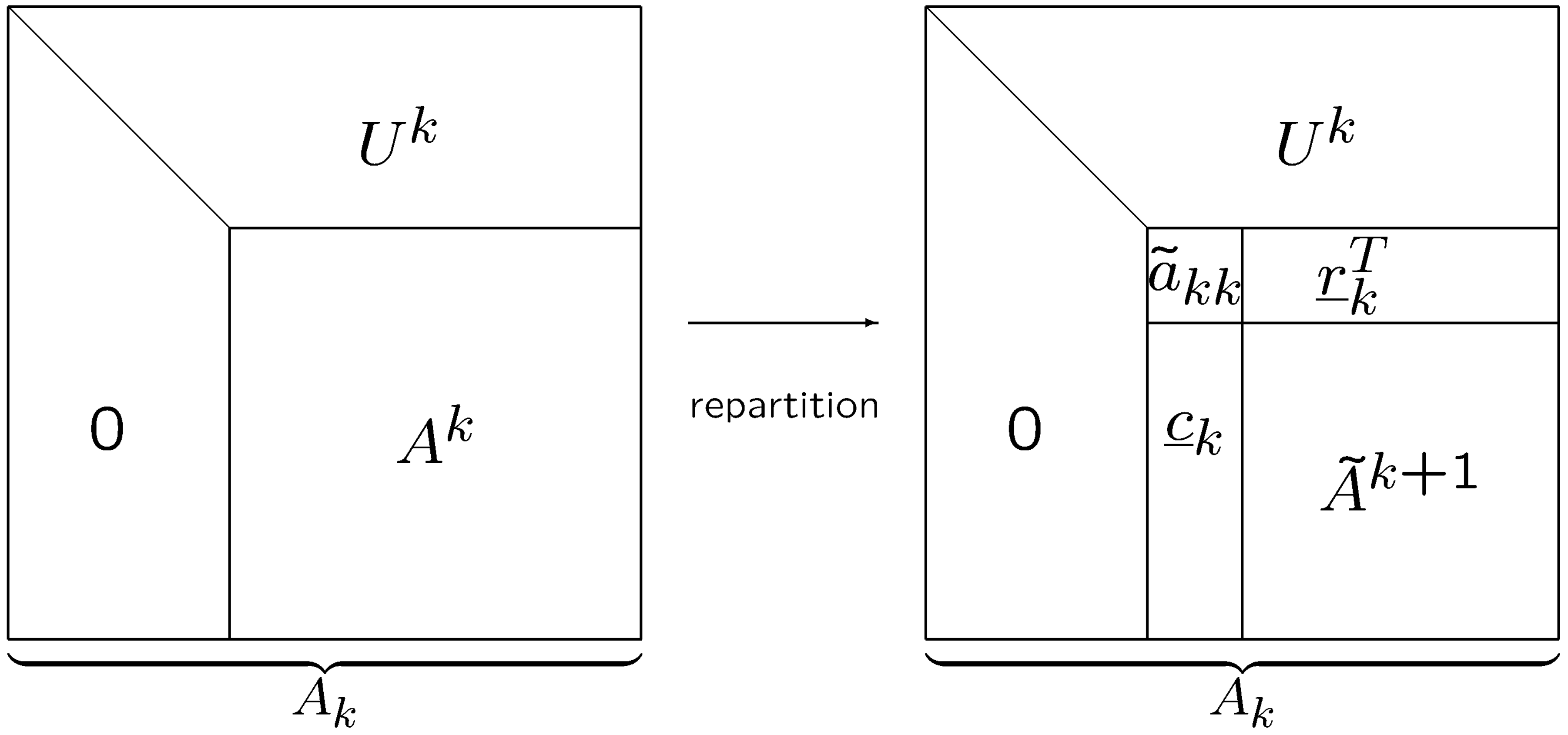
- To write out LU factorization explicitly,

$$\boldsymbol{L}_1\boldsymbol{L}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} = \boldsymbol{L}$$

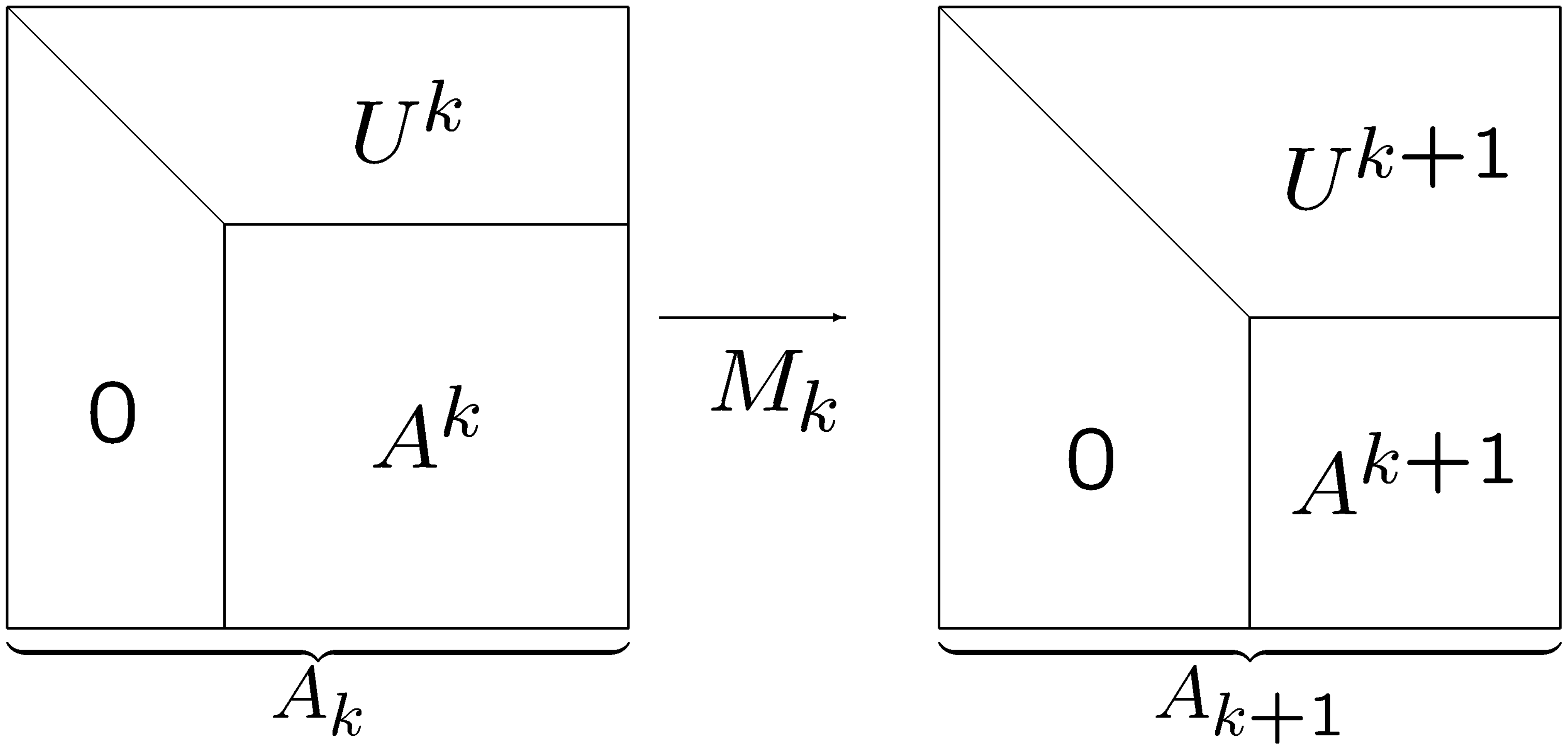
  so that

$$\boldsymbol{A} = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = \boldsymbol{LU}$$

First Step: Define sub-block



Single Gaussian Elimination Step



Update step viewed as matrix-matrix product.

Note that

$$A_{k+1} = A_k - \underline{m}_k \underline{e}_k^T A_k = M_k A_k,$$

with

$$M_k := I - \underline{m}_k \underline{e}_k^T,$$

as defined in the text.

Recall:

$$\begin{aligned} MA\underline{x} &= M\underline{b}, \\ M &:= M_{n-1} M_{n-2} \ldots M_1 =: L^{-1}. \end{aligned}$$

Elementary Elimination Matrices

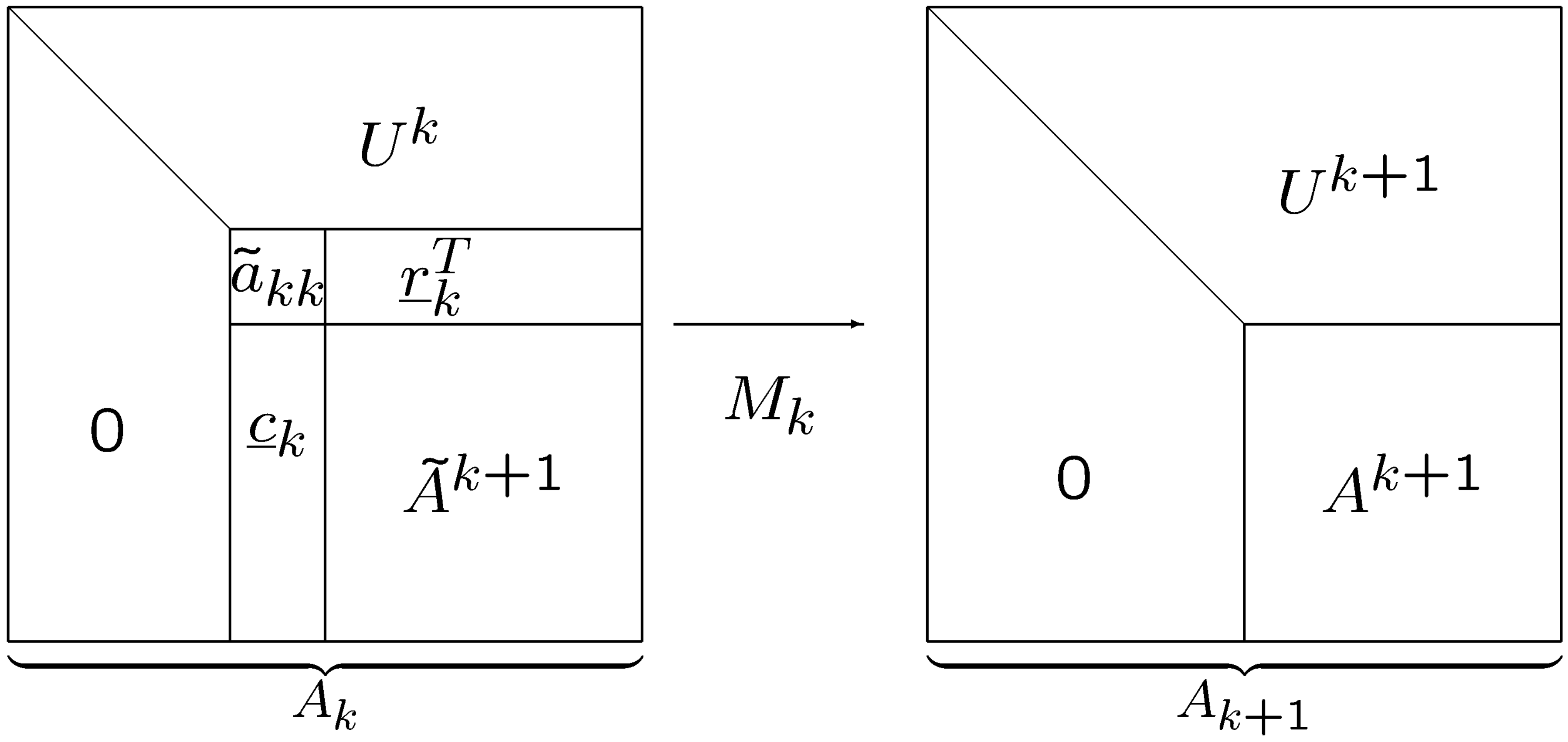
- More generally, we can annihilate *all* entries below kth position in n-vector $\boldsymbol{a}$ by transformation

$$\boldsymbol{M}_k\boldsymbol{a} = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_n & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

  where $m_i = a_i/a_k$, $i = k+1, \ldots, n$

- Divisor a_k, called *pivot*, must be nonzero

Second Step: Annihilate $\underline{c}_k$



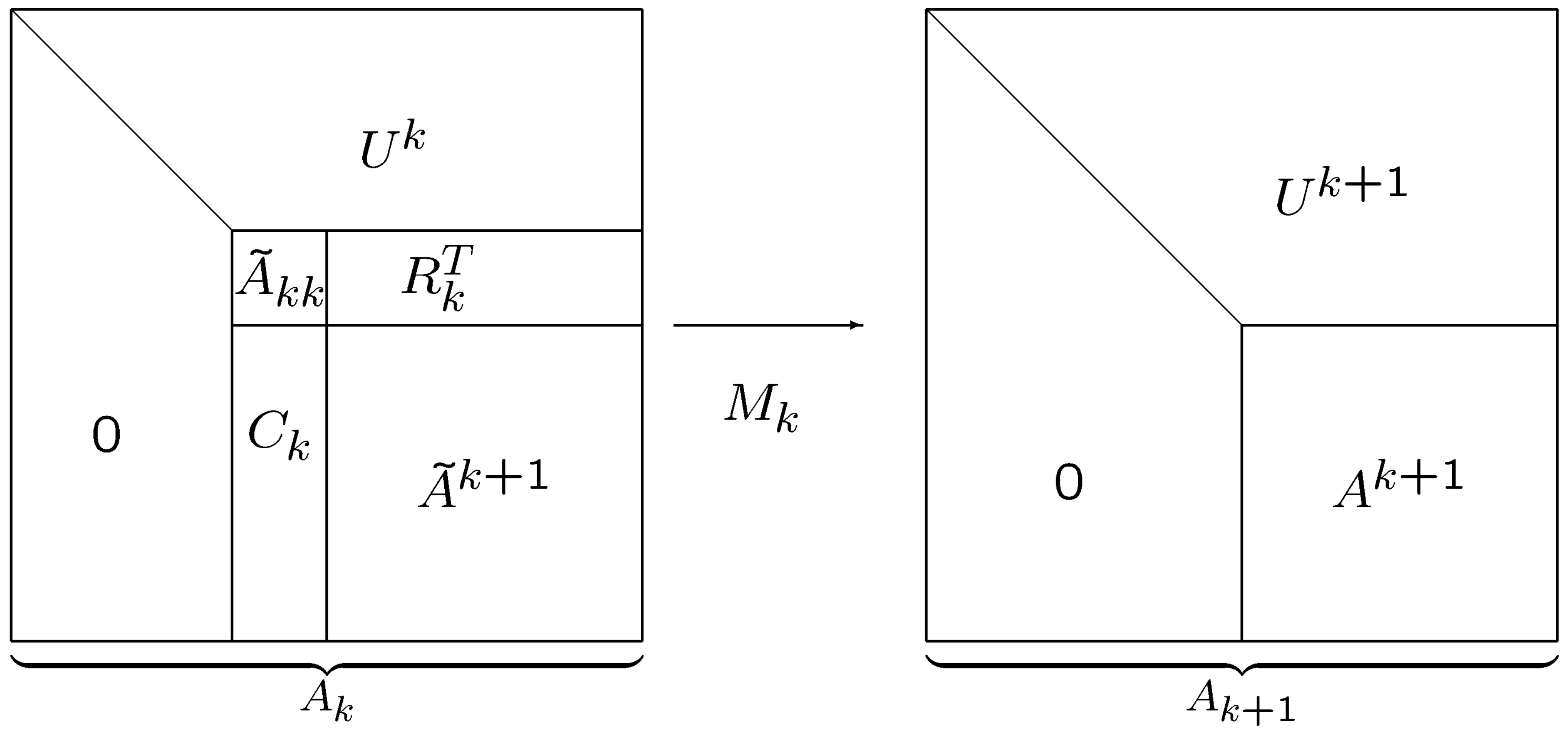
- Update step is:

$$A^{k+1} = \tilde{A}^{k+1} - \underline{c}_k \tilde{a}_{kk}^{-1} \underline{r}_k^T$$

which is a rank one update to A_K:

$$A_{k+1} = A_k - \underline{m}_k \underline{e}_k^T A_k$$

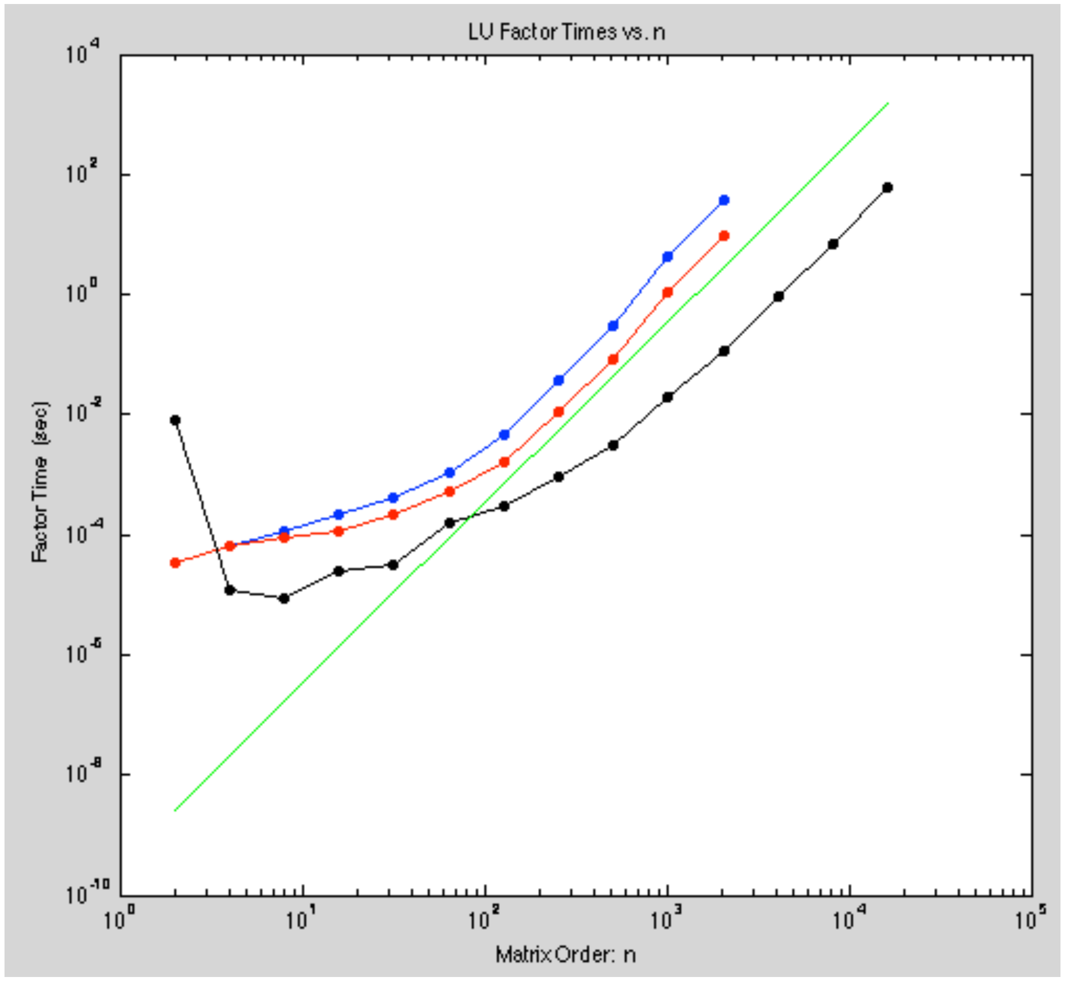
Can also be Implemented in ***Block Form***



$$A^{k+1} = \tilde{A}^{k+1} - C_k \tilde{A}_{kk}^{-1} R_k^T$$

- Advantage is that, if A_{kk} is a b x b block, you revisit the A_k sub-block only n/b times, and thus need fewer memory accesses. ***An order-of-magnitude faster.*** (LAPACK vs. LINPACK)

Matlab demo, gauss2.m



- Blue curve is rank-1 update
- Red curve is rank-4 update
- Black curve is matlab lu() function
  - It uses a 4 CPUs on the Mac and achieves an impressive 50 Gflops, which is certainly near peak (or perhaps beyond?)

Uniqueness of LU Factorization

- Despite variations in computing it, LU factorization is unique up to diagonal scaling of factors
- Provided row pivot sequence is same, if we have two LU factorizations $\boldsymbol{PA} = \boldsymbol{LU} = \hat{\boldsymbol{L}}\hat{\boldsymbol{U}}$, then $\hat{\boldsymbol{L}}^{-1}\boldsymbol{L} = \hat{\boldsymbol{U}}\boldsymbol{U}^{-1} = \boldsymbol{D}$ is both lower and upper triangular, hence diagonal
- If both $\boldsymbol{L}$ and $\hat{\boldsymbol{L}}$ are unit lower triangular, then $\boldsymbol{D}$ must be identity matrix, so $\boldsymbol{L} = \hat{\boldsymbol{L}}$ and $\boldsymbol{U} = \hat{\boldsymbol{U}}$
- Uniqueness is made explicit in LDU factorization $\boldsymbol{PA} = \boldsymbol{LDU}$, with $\boldsymbol{L}$ unit lower triangular, $\boldsymbol{U}$ unit upper triangular, and $\boldsymbol{D}$ diagonal

Storage Management

- Elementary elimination matrices $\boldsymbol{M}_k$, their inverses $\boldsymbol{L}_k$, and permutation matrices $\boldsymbol{P}_k$ used in formal description of LU factorization process are *not* formed explicitly in actual implementation
- $\boldsymbol{U}$ overwrites upper triangle of $\boldsymbol{A}$, multipliers in $\boldsymbol{L}$ overwrite strict lower triangle of $\boldsymbol{A}$, and unit diagonal of $\boldsymbol{L}$ need not be stored
- Row interchanges usually are not done explicitly; auxiliary integer vector keeps track of row order in original locations

Inversion vs. Factorization

- Even with many right-hand sides $\boldsymbol{b}$, inversion never overcomes higher initial cost, since each matrix-vector multiplication $\boldsymbol{A}^{-1}\boldsymbol{b}$ requires n^2 operations, similar to cost of forward- and back-substitution
- Inversion gives less accurate answer; for example, solving $3x = 18$ by division gives $x = 18/3 = 6$, but inversion gives $x = 3^{-1} \times 18 = 0.333 \times 18 = 5.99$ using 3-digit arithmetic
- Matrix inverses often occur as convenient notation in formulas, but explicit inverse is rarely required to implement such formulas
- For example, product $\boldsymbol{A}^{-1}\boldsymbol{B}$ should be computed by LU factorization of $\boldsymbol{A}$, followed by forward- and back-substitutions using each column of $\boldsymbol{B}$

Scaling Linear Systems

- In principle, solution to linear system is unaffected by diagonal scaling of matrix and right-hand-side vector
- In practice, scaling affects both conditioning of matrix and selection of pivots in Gaussian elimination, which in turn affect numerical accuracy in finite-precision arithmetic
- It is usually best if all entries (or uncertainties in entries) of matrix have about same size
- Sometimes it may be obvious how to accomplish this by choice of measurement units for variables, but there is no foolproof method for doing so in general
- Scaling can introduce rounding errors if not done carefully

Example: Scaling

- Linear system

$$\begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \epsilon \end{bmatrix}$$

  has condition number $1/\epsilon$, so is ill-conditioned if ϵ is small

- If second row is multiplied by $1/\epsilon$, then system becomes perfectly well-conditioned

- Apparent ill-conditioning was due purely to poor scaling

- In general, it is usually much less obvious how to correct poor scaling

- Sherman Morrison Formula

Solving Modified Problems

- If right-hand side of linear system changes but matrix does not, then LU factorization need not be repeated to solve new system
- Only forward- and back-substitution need be repeated for new right-hand side
- This is substantial savings in work, since additional triangular solutions cost only $\mathcal{O}(n^2)$ work, in contrast to $\mathcal{O}(n^3)$ cost of factorization

Sherman-Morrison Formula

- Sometimes refactorization can be avoided even when matrix *does* change
- *Sherman-Morrison formula* gives inverse of matrix resulting from rank-one change to matrix whose inverse is already known

  $$(\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{u}(1 - \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1}$$

  where $\boldsymbol{u}$ and $\boldsymbol{v}$ are n-vectors
- Evaluation of formula requires $\mathcal{O}(n^2)$ work (for matrix-vector multiplications) rather than $\mathcal{O}(n^3)$ work required for inversion

Rank-One Updating of Solution

- To solve linear system $(\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)\boldsymbol{x} = \boldsymbol{b}$ with new matrix, use Sherman-Morrison formula to obtain

$$\begin{aligned} \boldsymbol{x} &= (\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)^{-1}\boldsymbol{b} \\ &= \boldsymbol{A}^{-1}\boldsymbol{b} + \boldsymbol{A}^{-1}\boldsymbol{u}(1 - \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{b} \end{aligned}$$

  which can be implemented by following steps
  - Solve $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{u}$ for $\boldsymbol{z}$, so $\boldsymbol{z} = \boldsymbol{A}^{-1}\boldsymbol{u}$
  - Solve $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$ for $\boldsymbol{y}$, so $\boldsymbol{y} = \boldsymbol{A}^{-1}\boldsymbol{b}$
  - Compute $\boldsymbol{x} = \boldsymbol{y} + ((\boldsymbol{v}^T\boldsymbol{y})/(1 - \boldsymbol{v}^T\boldsymbol{z}))\boldsymbol{z}$

- If $\boldsymbol{A}$ is already factored, procedure requires only triangular solutions and inner products, so only $\mathcal{O}(n^2)$ work and no explicit inverses

Example: Rank-One Updating of Solution

- Consider rank-one modification

$$\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -1 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}$$

(with $3, 2$ entry changed) of system whose LU factorization was computed in earlier example

Original Matrix

$$\boxed{\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix}}$$

- One way to choose update vectors is

$$\boldsymbol{u} = \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{v} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

so matrix of modified system is $\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T$

Example, continued

- Using LU factorization of $\boldsymbol{A}$ to solve $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{u}$ and $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$,

$$\boldsymbol{z} = \begin{bmatrix} -3/2 \\ 1/2 \\ -1/2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{y} = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}$$

- Final step computes updated solution

Q: Under what circumstances could the denominator be zero ?

$$\boldsymbol{x} = \boldsymbol{y} + \frac{\boldsymbol{v}^T\boldsymbol{y}}{1 - \boldsymbol{v}^T\boldsymbol{z}}\boldsymbol{z} = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix} + \frac{2}{1 - 1/2}\begin{bmatrix} -3/2 \\ 1/2 \\ -1/2 \end{bmatrix} = \begin{bmatrix} -7 \\ 4 \\ 0 \end{bmatrix}$$

- We have thus computed solution to modified system without factoring modified matrix

Sherman Morrison

[1] Solve $A\underline{\tilde{x}} = \underline{\tilde{b}}$:

$A \longrightarrow LU$ ($O(n^3)$ work)

Solve $L\underline{\tilde{y}} = \underline{\tilde{b}}$,

Solve $U\underline{\tilde{x}} = \underline{\tilde{y}}$ ($O(n^2)$ work).

[2] New problem:

$\left(A - \underline{u}\underline{v}^T\right)\underline{x} = \underline{b}$. (different $\underline{x}$ and $\underline{b}$)

Key Idea: $\left(A - \underline{u}\underline{v}^T\right)\underline{x}$ differs from $A\underline{x}$ by only a small amount of information.

Rewrite as:

$$A\underline{x} + \underline{u}\gamma = \underline{b}$$

$$\gamma := -\underline{v}^T\underline{x} \quad \longleftrightarrow \quad \underline{v}^T\underline{x} + \gamma = 0$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\underline{x} + \gamma\underline{u} &= \underline{b} \\ \underline{v}^T\underline{x} + \gamma &= 0 \end{aligned}$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\underline{x} + \gamma\underline{u} &= \underline{b} \\ \underline{v}^T\underline{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \underline{u} \\ \underline{v}^T & 1 \end{bmatrix} \begin{pmatrix} \underline{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \underline{b} \\ 0 \end{pmatrix}$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\underline{x} + \gamma \underline{u} &= \underline{b} \\ \underline{v}^T \underline{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \underline{u} \\ \underline{v}^T & 1 \end{bmatrix} \begin{pmatrix} \underline{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \underline{b} \\ 0 \end{pmatrix}$$

Eliminate for γ:

$$\begin{bmatrix} A & \underline{u} \\ 0 & 1 - \underline{v}^T A^{-1} \underline{u} \end{bmatrix} \begin{pmatrix} \underline{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \underline{b} \\ -\underline{v}^T A^{-1} \underline{b} \end{pmatrix}$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\underline{x} + \gamma\underline{u} &= \underline{b} \\ \underline{v}^T\underline{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \underline{u} \\ \underline{v}^T & 1 \end{bmatrix} \begin{pmatrix} \underline{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \underline{b} \\ 0 \end{pmatrix}$$

Eliminate for γ:

$$\begin{bmatrix} A & \underline{u} \\ 0 & 1 - \underline{v}^T A^{-1}\underline{u} \end{bmatrix} \begin{pmatrix} \underline{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \underline{b} \\ -\underline{v}^T A^{-1}\underline{b} \end{pmatrix}$$

$$\gamma = -\left(1 - \underline{v}^T A^{-1}\underline{u}\right)^{-1} \underline{v}^T A^{-1}\underline{b}$$

$$\underline{x} = A^{-1}\left(\underline{b} - \underline{u}\gamma\right) = A^{-1}\left[\underline{b} + \underline{u}\left(1 + \underline{v}^T A^{-1}\underline{u}\right)^{-1} \underline{v}^T A^{-1}\underline{b}\right]$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\underline{x} + \gamma \underline{u} &= \underline{b} \\ \underline{v}^T \underline{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \underline{u} \\ \underline{v}^T & 1 \end{bmatrix} \begin{pmatrix} \underline{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \underline{b} \\ 0 \end{pmatrix}$$

Eliminate for γ:

$$\begin{bmatrix} A & \underline{u} \\ 0 & 1 - \underline{v}^T A^{-1} \underline{u} \end{bmatrix} \begin{pmatrix} \underline{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \underline{b} \\ -\underline{v}^T A^{-1} \underline{b} \end{pmatrix}$$

$$\gamma = -\left(1 - \underline{v}^T A^{-1} \underline{u}\right)^{-1} \underline{v}^T A^{-1} \underline{b}$$

$$\underline{x} = A^{-1}\left(\underline{b} - \underline{u}\gamma\right) = A^{-1}\left[\underline{b} + \underline{u}\left(1 + \underline{v}^T A^{-1}\underline{u}\right)^{-1} \underline{v}^T A^{-1} \underline{b}\right]$$

$$\left(A - \underline{u}\underline{v}^T\right)^{-1} = A^{-1}\left(I + \underline{u}\left(1 - \underline{v}^T A^{-1} \underline{u}\right)^{-1} \underline{v}^T A^{-1}\right]$$

Special Types of Linear Systems

- Work and storage can often be saved in solving linear system if matrix has special properties
- Examples include
  - *Symmetric*: $\boldsymbol{A} = \boldsymbol{A}^T$, $a_{ij} = a_{ji}$ for all i, j
  - *Positive definite*: $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq \boldsymbol{0}$
  - *Band*: $a_{ij} = 0$ for all $|i - j| > \beta$, where β is *bandwidth* of $\boldsymbol{A}$
  - *Sparse*: most entries of $\boldsymbol{A}$ are zero

Symmetric Positive Definite (SPD) Matrices

- Very common in optimization and physical processes
- Easiest example:
  - If B is invertible, then $A := B^T B$ is SPD.
- SPD systems of the form $A \underline{x} = \underline{b}$ can be solved using
  - (stable) Cholesky factorization $A = LL^T$, or
  - iteratively with the most robust iterative solver, conjugate gradient iteration (generally with preconditioning, known as preconditioned conjugate gradients, PCG).

Symmetric Positive Definite Matrices

- If $\boldsymbol{A}$ is symmetric and positive definite, then LU factorization can be arranged so that $\boldsymbol{U} = \boldsymbol{L}^T$, which gives *Cholesky factorization*

$$\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{L}^T$$

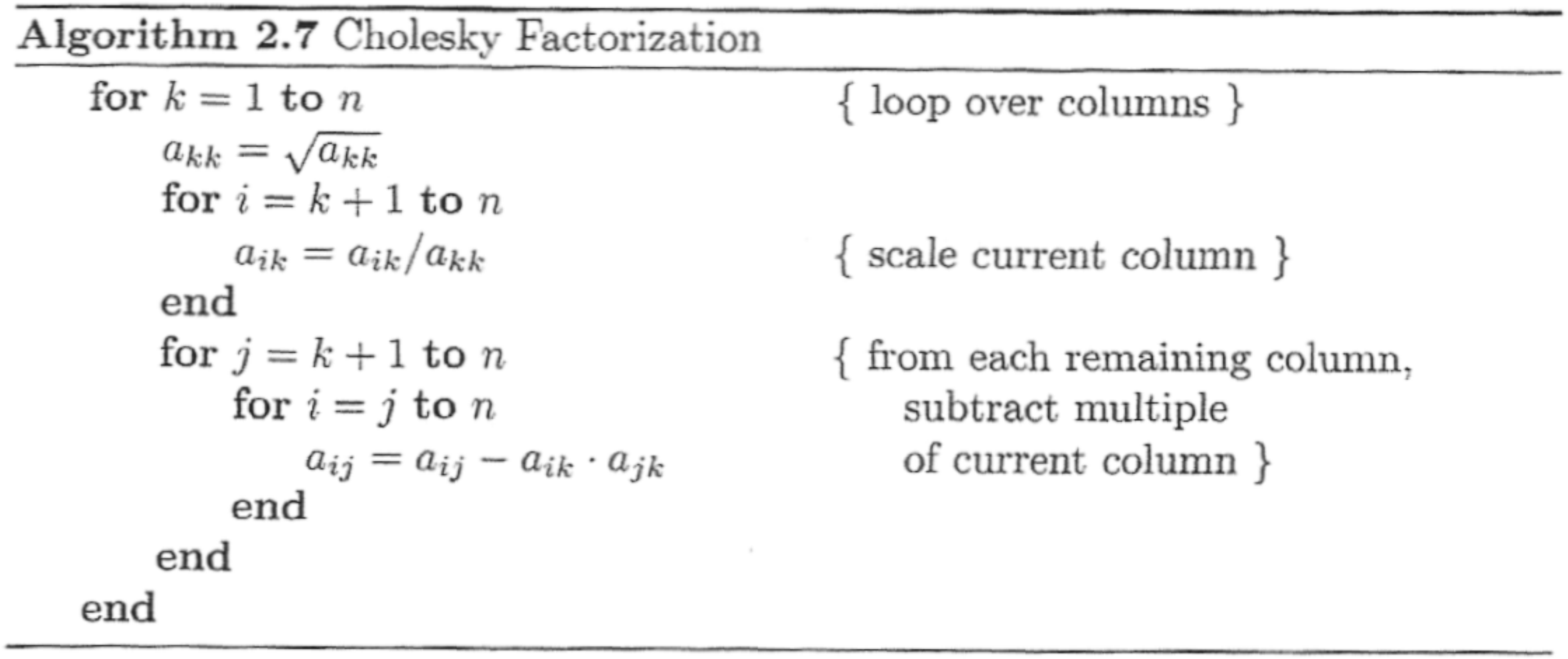
  where $\boldsymbol{L}$ is lower triangular with positive diagonal entries
- Algorithm for computing it can be derived by equating corresponding entries of $\boldsymbol{A}$ and $\boldsymbol{L}\boldsymbol{L}^T$
- In 2×2 case, for example,

$$\begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{bmatrix}$$

  implies

$$l_{11} = \sqrt{a_{11}}, \quad l_{21} = a_{21}/l_{11}, \quad l_{22} = \sqrt{a_{22} - l_{21}^2}$$

Cholesky Factorization (Text)



Algorithm 2.7 Cholesky Factorization

```
for $k = 1$ to $n$                          { loop over columns }
    $a_{kk} = \sqrt{a_{kk}}$
    for $i = k+1$ to $n$
        $a_{ik} = a_{ik}/a_{kk}$            { scale current column }
    end
    for $j = k+1$ to $n$                    { from each remaining column,
        for $i = j$ to $n$                      subtract multiple
            $a_{ij} = a_{ij} - a_{ik} \cdot a_{jk}$        of current column }
        end
    end
end
```

After a row scaling, this is just standard LU decomposition, exploiting symmetry in the LU factors and A. ($U=L^T$)

Cholesky Factorization

- One way to write resulting general algorithm, in which Cholesky factor $\boldsymbol{L}$ overwrites original matrix $\boldsymbol{A}$, is

```
for $j = 1$ to $n$
    for $k = 1$ to $j - 1$
        for $i = j$ to $n$
            $a_{ij} = a_{ij} - a_{ik} \cdot a_{jk}$
        end
    end
    $a_{jj} = \sqrt{a_{jj}}$
    for $k = j + 1$ to $n$
        $a_{kj} = a_{kj} / a_{jj}$
    end
end
```

Cholesky Factorization, continued

- Features of Cholesky algorithm for symmetric positive definite matrices
  - All n square roots are of positive numbers, so algorithm is well defined
  - No pivoting is required to maintain numerical stability
  - Only lower triangle of $\boldsymbol{A}$ is accessed, and hence upper triangular portion need not be stored
  - Only $n^3/6$ multiplications and similar number of additions are required
- Thus, Cholesky factorization requires only about half work and ***half storage*** compared with LU factorization of general matrix by Gaussian elimination, and also avoids need for pivoting

Band Matrices

- Gaussian elimination for band matrices differs little from general case — only ranges of loops change
- Typically matrix is stored in array by diagonals to avoid storing zero entries
- If pivoting is required for numerical stability, bandwidth can grow (but no more than double)
- General purpose solver for arbitrary bandwidth is similar to code for Gaussian elimination for general matrices
- For fixed small bandwidth, band solver can be extremely simple, especially if pivoting is not required for stability

Tridiagonal Matrices

- Consider tridiagonal matrix

$$\boldsymbol{A} = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{bmatrix}$$

- Gaussian elimination without pivoting reduces to

```
$d_1 = b_1$
**for** $i = 2$ **to** $n$
    $m_i = a_i / d_{i-1}$
    $d_i = b_i - m_i c_{i-1}$
**end**
```

$\boxed{\textit{Cost is O(n) !}}$

Tridiagonal Matrices, continued

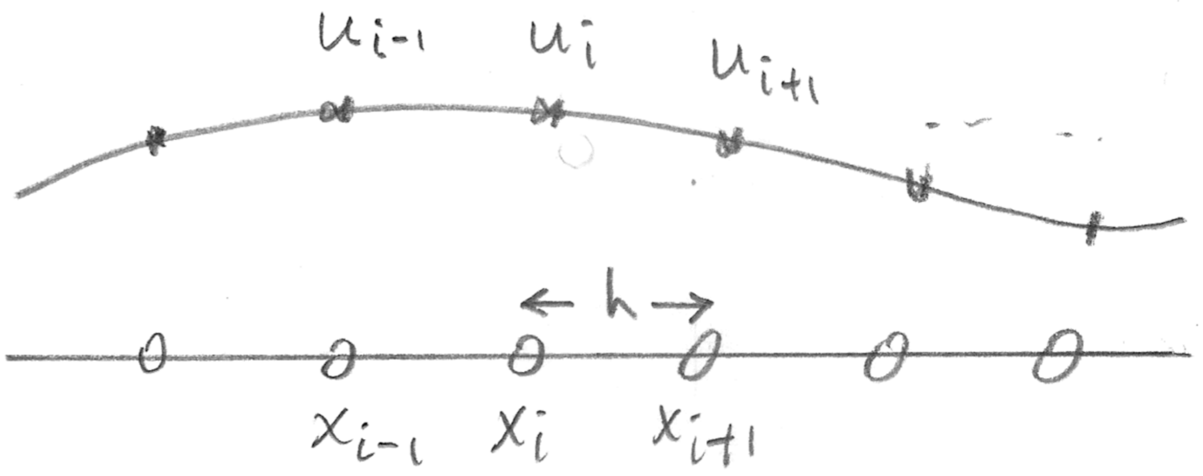
- LU factorization of $\boldsymbol{A}$ is then given by

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ m_2 & 1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & m_{n-1} & 1 & 0 \\ 0 & \cdots & 0 & m_n & 1 \end{bmatrix}, \quad \boldsymbol{U} = \begin{bmatrix} d_1 & c_1 & 0 & \cdots & 0 \\ 0 & d_2 & c_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & d_{n-1} & c_{n-1} \\ 0 & \cdots & \cdots & 0 & d_n \end{bmatrix}$$

Example of Banded Systems

- Graphs (i.e., matrices) arising from differential equations in 1D, 2D, 3D (and higher…) are generally banded and sparse.

- Example:



$$-\frac{d^2u}{dx^2} = f(x) \quad \longrightarrow \quad -\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \approx f_i$$

In Matrix Form

$$-\frac{d^2u}{dx^2} = f(x) \quad \longrightarrow \quad -\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \approx f_i$$

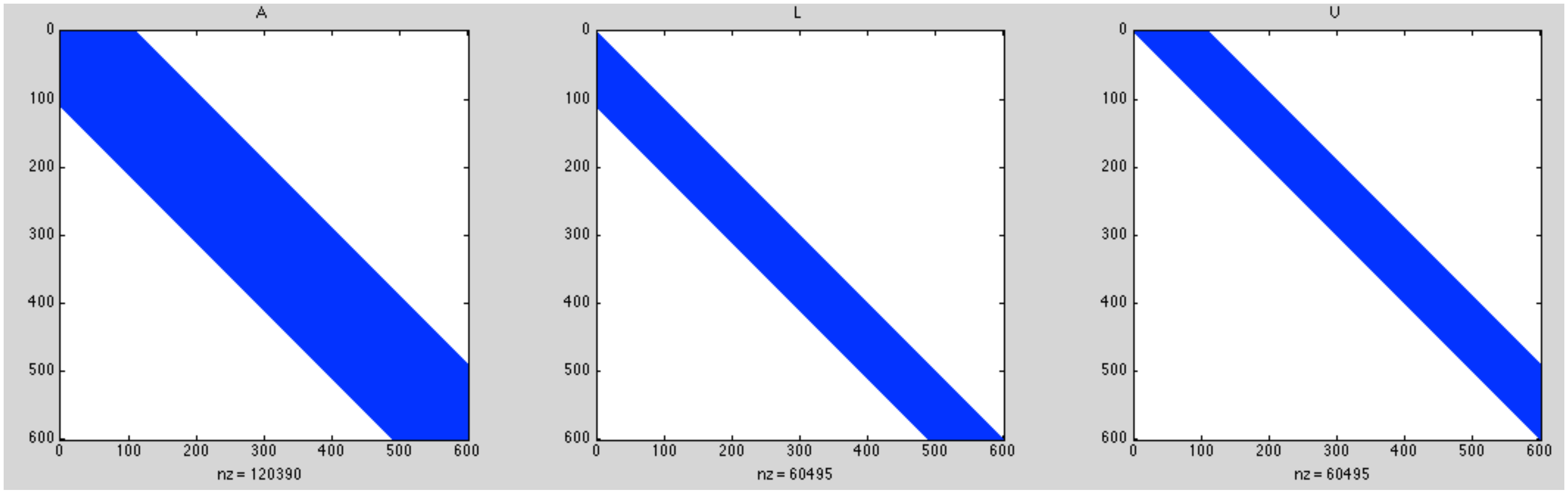
$$A_{1D} = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_m \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_m \end{pmatrix}$$

- Banded, tridiagonal matrix (“1D Poisson Operator”)

General Band Matrices

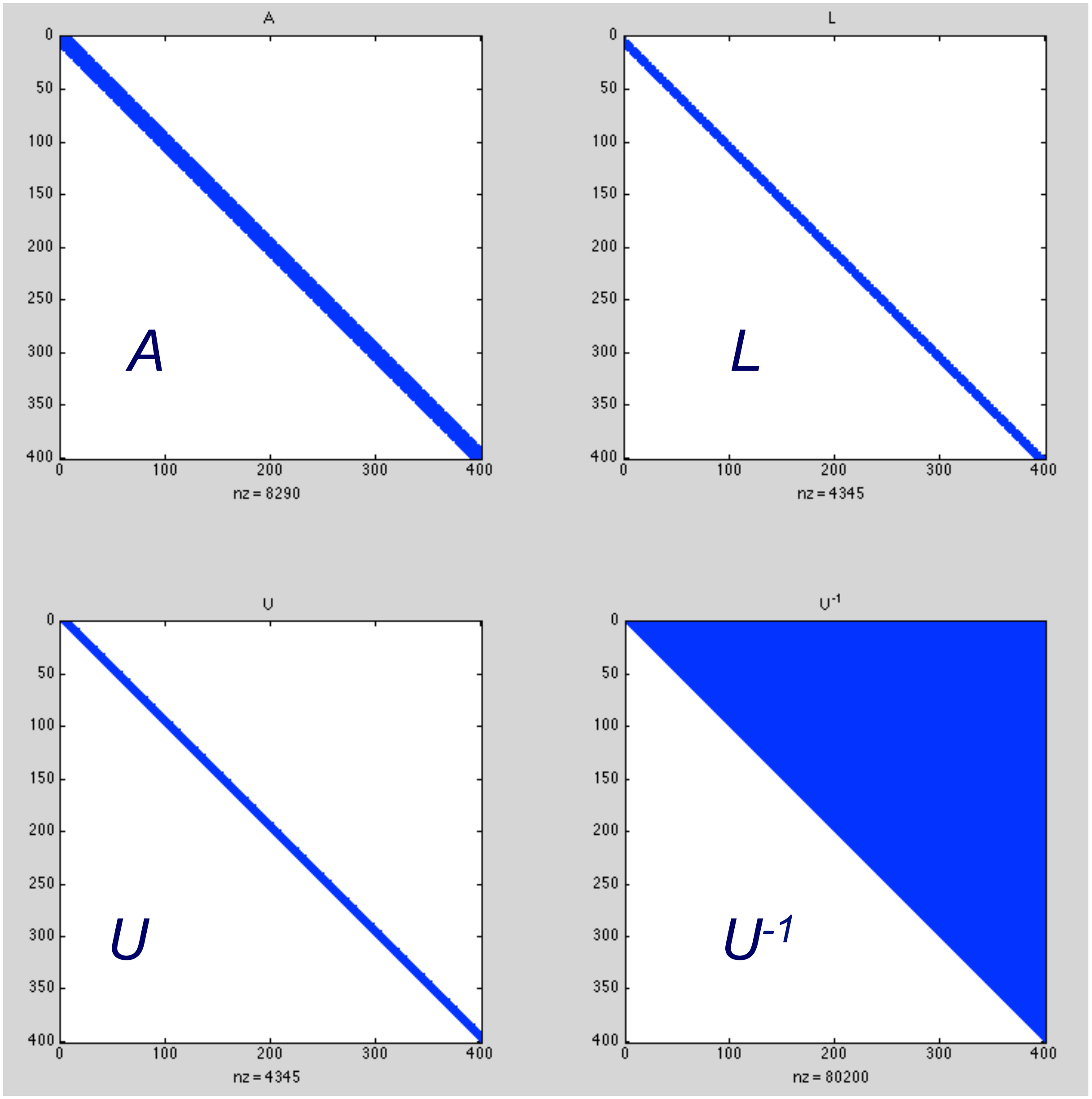
- In general, band system of bandwidth β requires $\mathcal{O}(\beta n)$ storage, and its factorization requires $\mathcal{O}(\beta^2 n)$ work
- Compared with full system, savings is substantial if $\beta \ll n$

Banded Systems

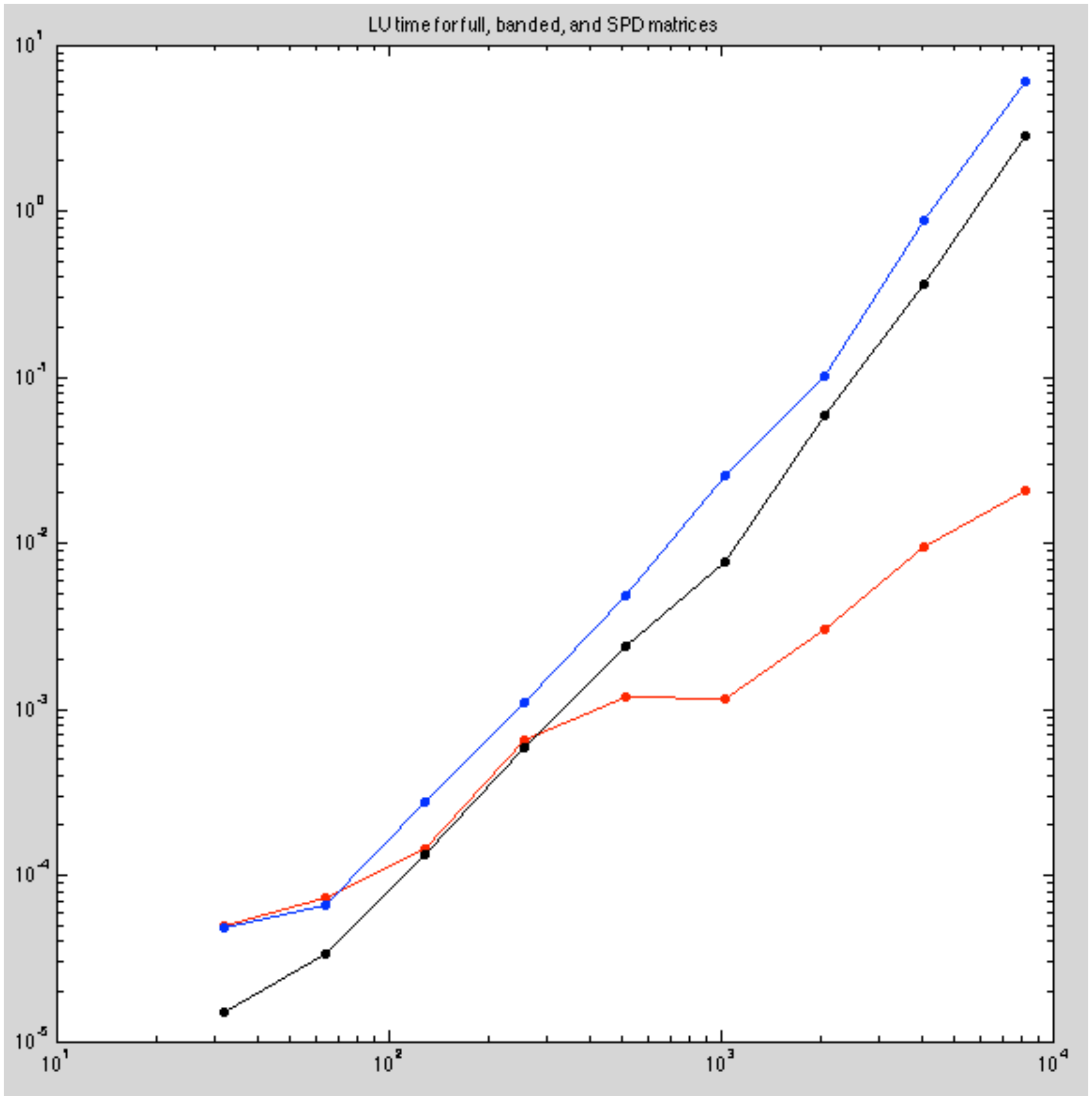


- Significant savings in storage and work if A is banded → $a_{ij} = 0$ if $| i-j | > \beta$
- The LU factors preserve the nonzero structure of A (unless there is pivoting, in which case, the bandwidth of L can grow by at most 2x).
- Storage / solve costs for LU is ~ $2n\,\beta$
- Factor cost is ~ $n\,\beta^2 \ll n^3$

Definitely Do Not Invert A or L or U for Banded Systems



Solver Times, Banded, Cholesky (SPD), Full



Factorization Time (seconds)

System Size, n

Solver Times, Banded, Cholesky (SPD), Full

```
% Demo of banded-matrix costs

clear all;

for pass=1:2;
beta=10;

for k=4:13; n = 2^k;

   R=9*eye(n) + rand(n,n); S=R'*R; A=spalloc(n,n,1+2*beta);
   for i=1:n; j0=max(1,i-beta);j1=min(n,i+beta);
       A(i,j0:j1)=R(i,j0:j1);
   end;

   tstart=tic; [L,U]=lu(A); tsparse(k) = toc(tstart);
   tstart=tic; [L,U]=lu(R); tfull(k) = toc(tstart);
   tstart=tic; [C]=chol(S); tchol(k) = toc(tstart);
   nk(k)=n;
   sk(k)= (2*(n^3)/3)/(1.e9*tfull(k)); % GFLOPS
   ck(k)= (2*(n^3)/3)/(1.e9*tchol(k)); % GFLOPS

   [n tsparse(k) tfull(k) tchol(k)]

end;
loglog(nk,tsparse,'r.-',nk,tfull,'b.-',nk,tchol,'k.-')
axis square; title('LU time for full, banded, and SPD matrices')
```

LINPACK and LAPACK

- `LINPACK` is software package for solving wide variety of systems of linear equations, both general dense systems and special systems, such as symmetric or banded
- Solving linear systems of such fundamental importance in scientific computing that `LINPACK` has become standard benchmark for comparing performance of computers
- `LAPACK` is more recent replacement for `LINPACK` featuring higher performance on modern computer architectures, including some parallel computers
- Both `LINPACK` and `LAPACK` are available from `Netlib`

Basic Linear Algebra Subprograms

- High-level routines in `LINPACK` and `LAPACK` are based on lower-level Basic Linear Algebra Subprograms (`BLAS`)
- `BLAS` encapsulate basic operations on vectors and matrices so they can be optimized for given computer architecture while high-level routines that call them remain portable
- Higher-level `BLAS` encapsulate matrix-vector and matrix-matrix operations for better utilization of memory hierarchies such as cache and virtual memory with paging
- Generic Fortran versions of `BLAS` are available from `Netlib`, and many computer vendors provide custom versions optimized for their particular systems

Examples of BLAS

Level	Work	Examples	Function
1	$\mathcal{O}(n)$	`saxpy`	Scalar $\times$ vector $+$ vector
		`sdot`	Inner product
		`snrm2`	Euclidean vector norm
2	$\mathcal{O}(n^2)$	`sgemv`	Matrix-vector product
		`strsv`	Triangular solution
		`sger`	Rank-one update
3	$\mathcal{O}(n^3)$	`sgemm`	Matrix-matrix product
		`strsm`	Multiple triang. solutions
		`ssyrk`	Rank-k update

- Level-3 `BLAS` have more opportunity for data reuse, and hence higher performance, because they perform more operations per data item than lower-level `BLAS`

Linear Algebra Very Short Summary

Main points:

- Conditioning of matrix cond(A) bounds our expected accuracy.
  - e.g., if cond(A) ~ 10^5 we expect at most 11 significant digits in $\underline{x}$.
  - Why?
  - We start with IEEE double precision – 16 digits. We lose 5 because condition (A) ~ 10^5, so we have 11 = 16-5.

- Stable algorithm (i.e., pivoting) important to realizing this bound.
  - Some systems don't need pivoting (e.g., SPD, diagonally dominant)
  - Unstable algorithms can sometimes be rescued with iterative refinement.

- Costs:
  - Full matrix $\rightarrow$ $O(n^2)$ storage, $O(n^3)$ work (wall-clock time)
  - Sparse or banded matrix, substantially less.