

Chapter 2, Linear Systems

Outline

- 1 Existence, Uniqueness, and Conditioning
- 2 Solving Linear Systems
- 3 Special Types of Linear Systems
- 4 Software for Linear Systems



The Geometry of Linear Equations¹

- Example, 2×2 system:

$$2x - y = 1$$

$$x + y = 5$$

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

¹Gilbert Strang: *Linear Algebra and Its Applications*

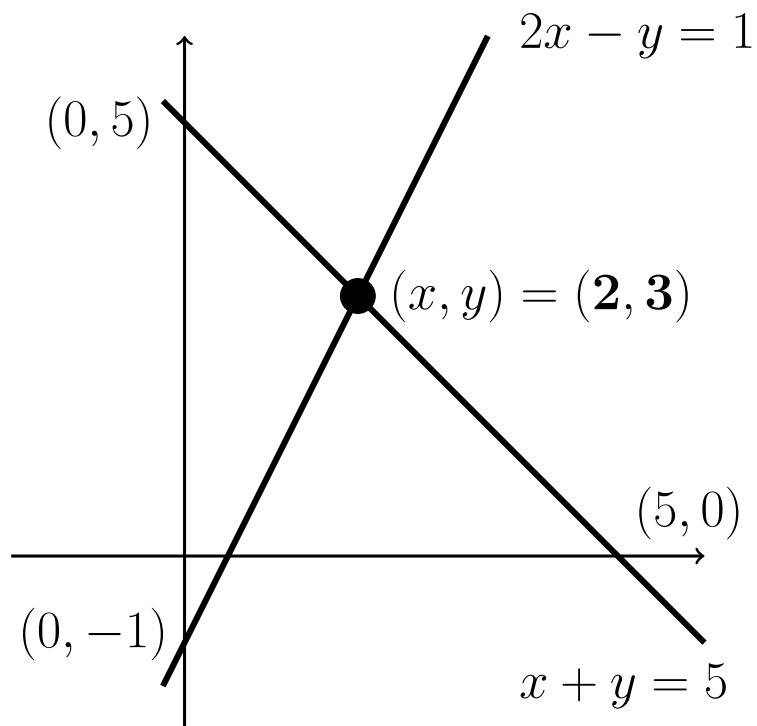
Row Form

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

$$2x - y = 1 \quad \text{line 1}$$

$$x + y = 5 \quad \text{line 2}$$

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

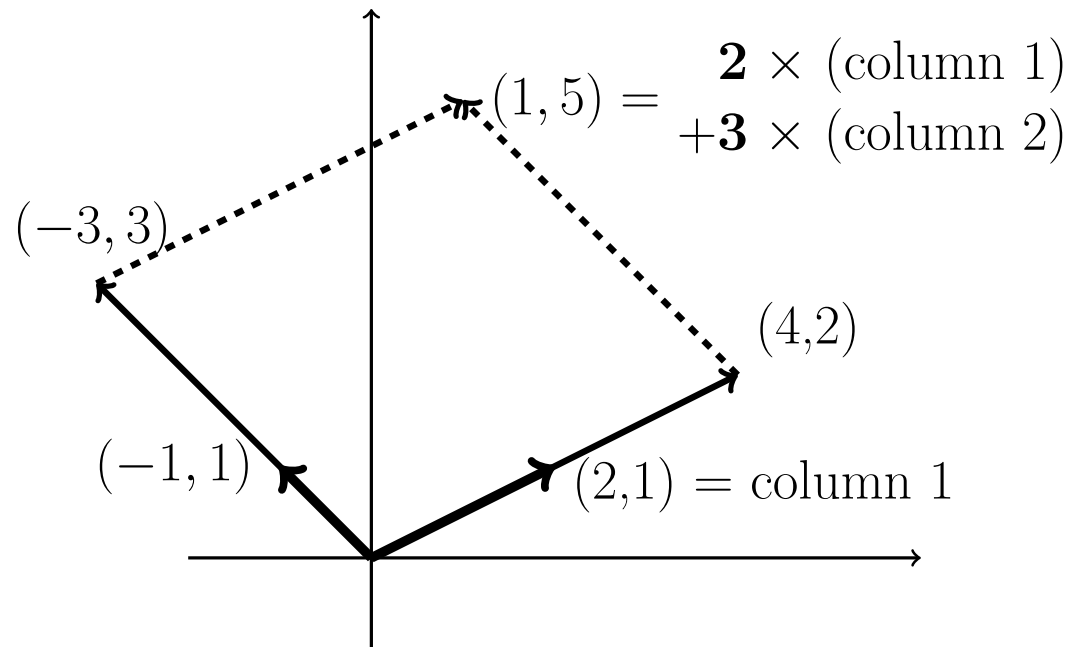


Column Form

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

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

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



Row Form: A Case with $n=3$.

$$2u + v + w = 5$$

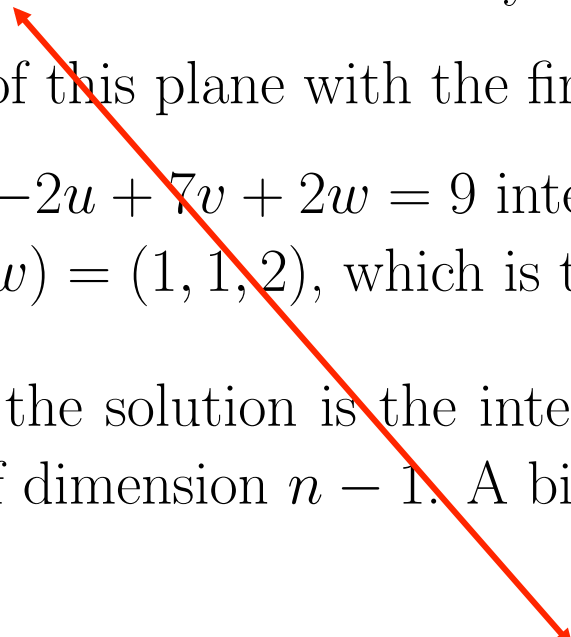
Three planes $4u - 6v = -2$

$$-2u + 7v + 2w = 9$$

- 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.

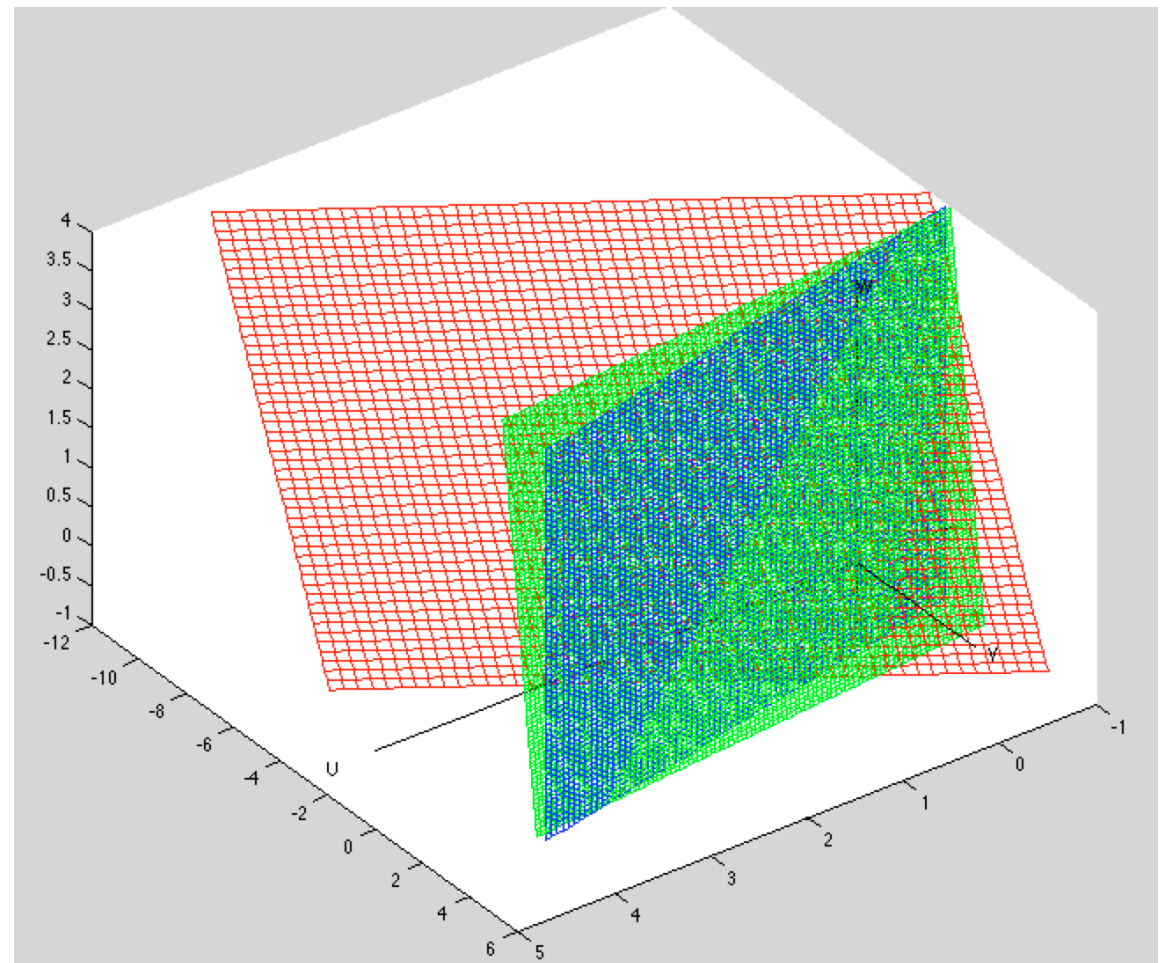


Note that $u=5$ is also a plane....

Row Form

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

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



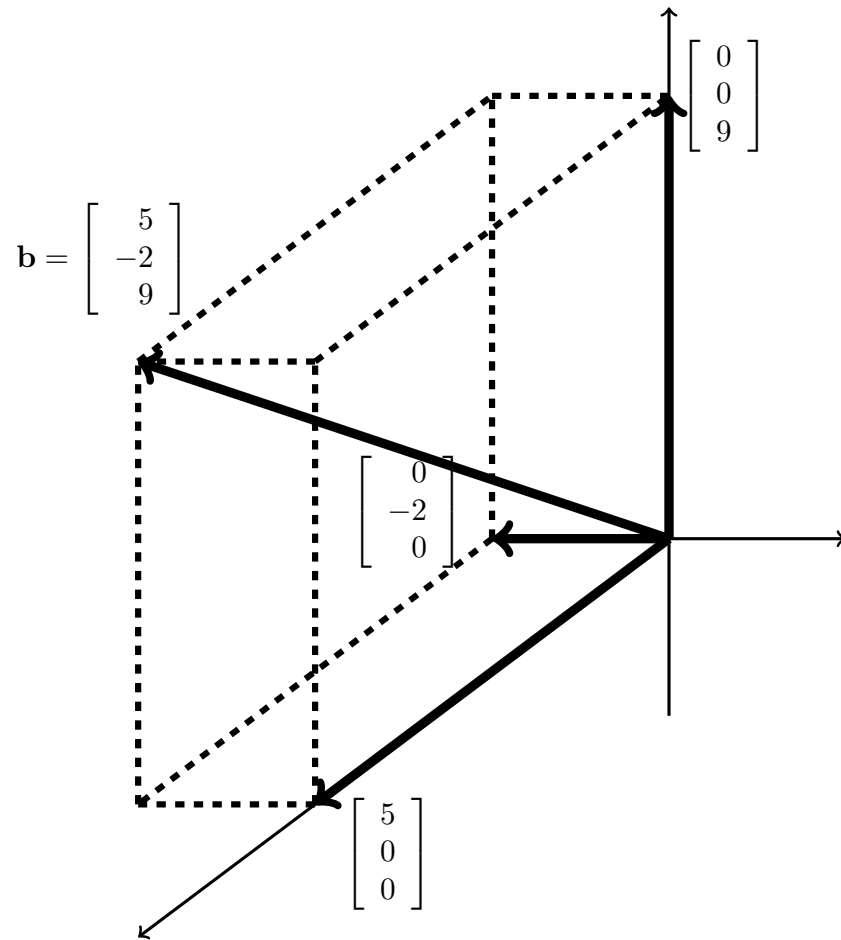
Column Vectors and Linear Combinations

- The preceding system is viewed as the vector equation

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

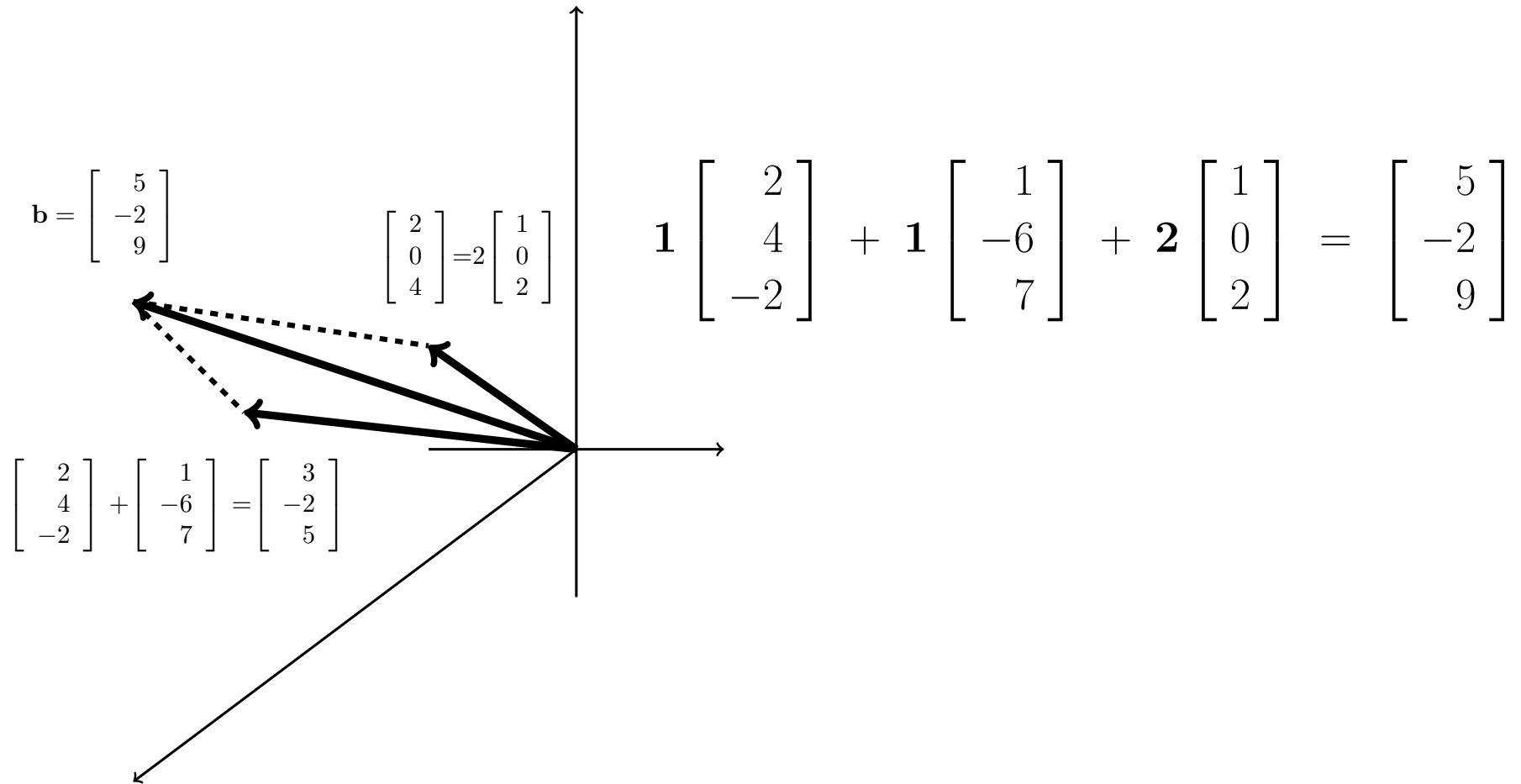
- Our task is to find the multipliers, u , v , and w .
- The vector $\mathbf{b}$ is identified with the point (5,-2,9).
- We can view $\mathbf{b}$ as a list of numbers, a point, or an arrow.
- For $n > 3$, it's probably best to view it as a list of numbers.

Vector Addition Example

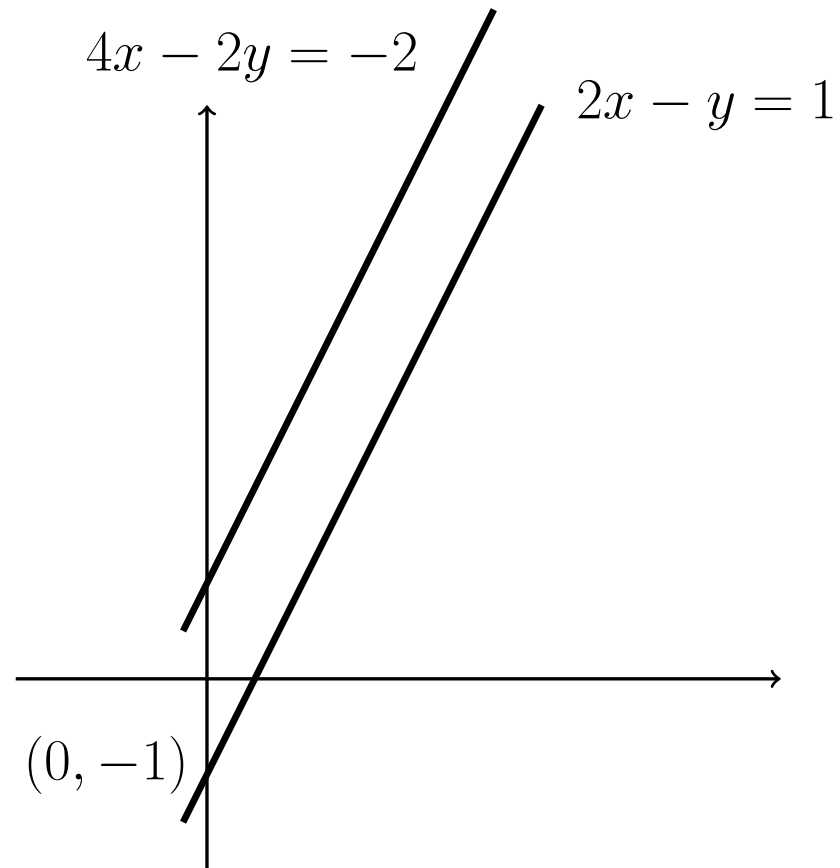


$$\begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 9 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$$

Linear Combination



The Singular Case: Row Picture

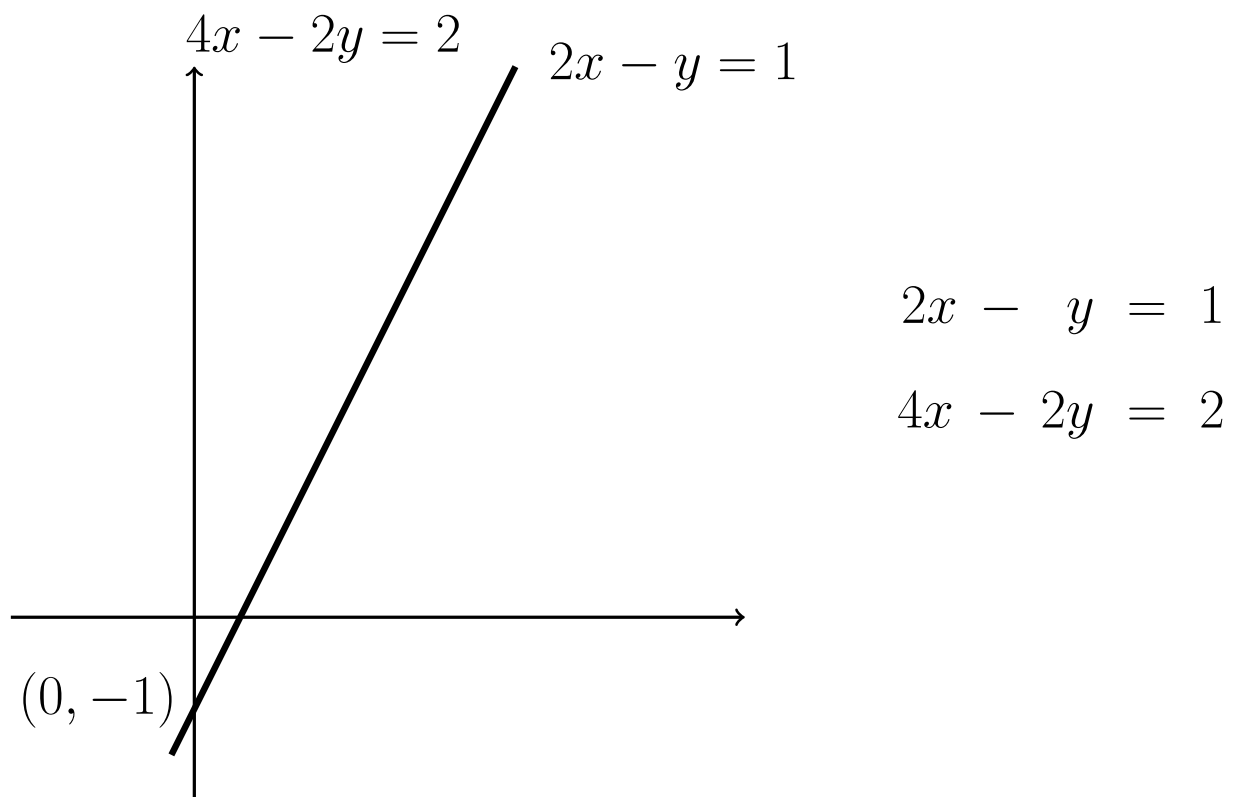


$$2x - y = 1$$

$$4x - 2y = -2$$

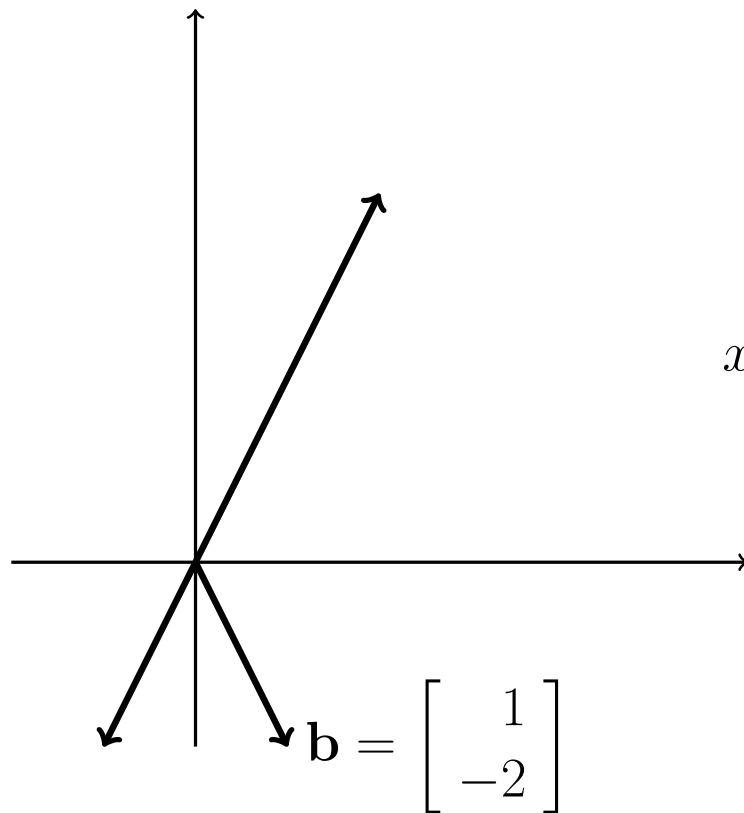
- No solution.

The Singular Case: Row Picture



- Infinite number of solutions.

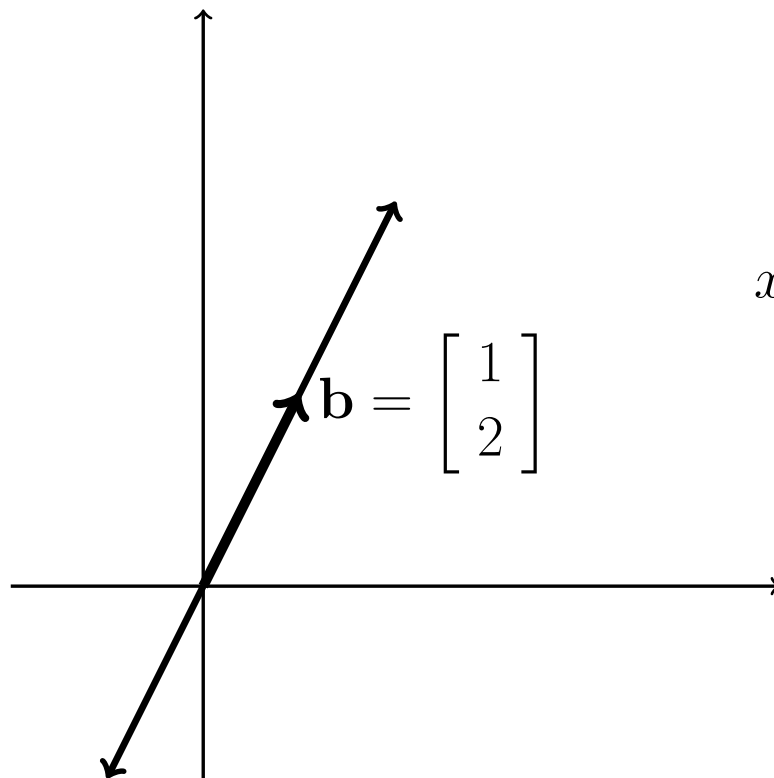
The Singular Case: Column Picture



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

- No solution.

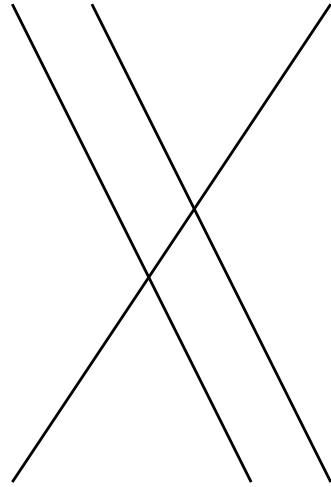
The Singular Case: Column Picture



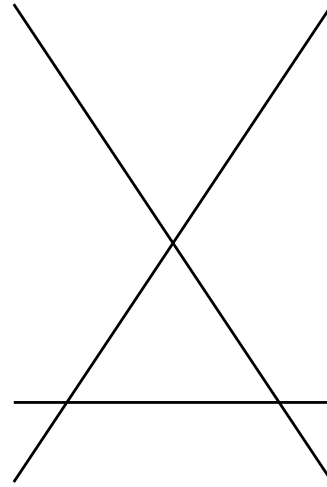
$$x \begin{bmatrix} 2 \\ 4 \end{bmatrix} + y \begin{bmatrix} -1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

- Infinite number of solutions.

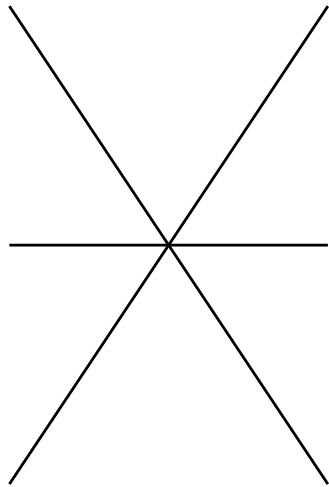
Singular Case: Row Picture with $n=3$



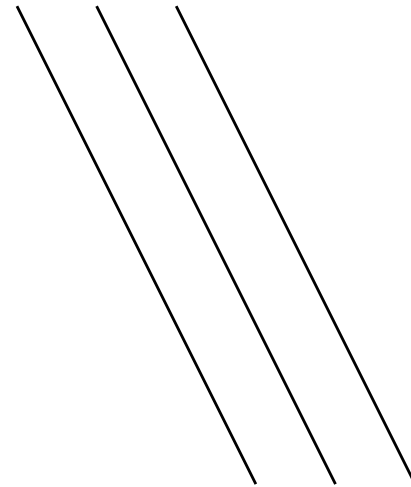
(a) two parallel planes



(b) no intersection

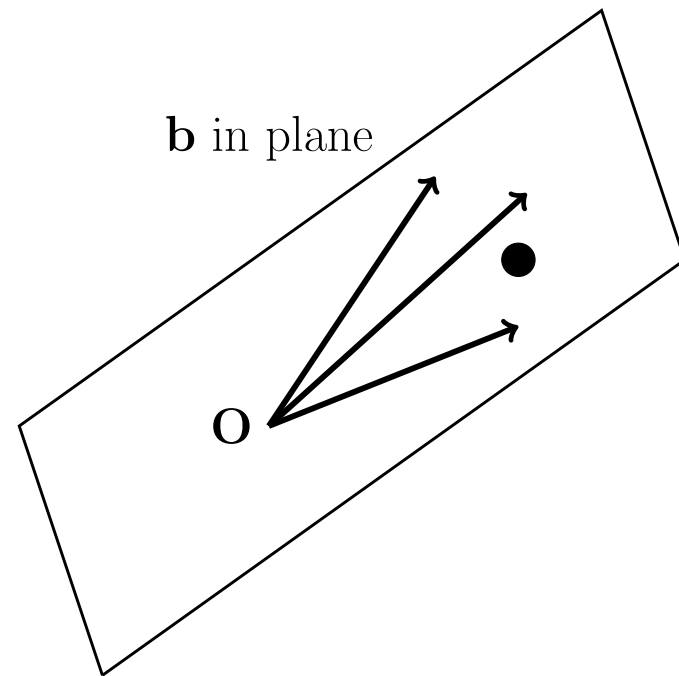
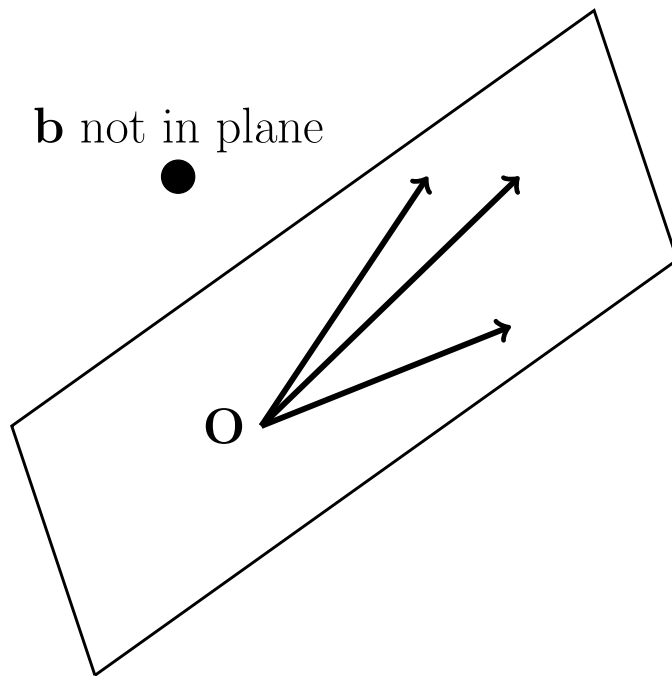


(c) line of intersection



(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.

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

Matrix Form and Matrix-Vector Products.

- We start with the familiar (row) form

$$2u + v + w = 5$$

$$4u - 6v = -2$$

$$-2u + 7v + 2w = 9$$

- In matrix form, this is

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

- 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 $\mathbf{a}_3$,

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

- 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

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

- Always.

Matrix-Vector Products, Example.

$$\begin{aligned}\text{If } \hat{\mathbf{x}} &:= V (V^T A V)^{-1} V^T \mathbf{b} \\ &= V \mathbf{y}.\end{aligned}$$

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,

$$\begin{aligned} A &= \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_j & \cdots & \mathbf{a}_n \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix}. \end{aligned}$$

- 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

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

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

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

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 A and m -vector b , find unknown n -vector x satisfying $Ax = b$
- System of equations asks “Can b be expressed as linear combination of columns of A ?”
- If so, coefficients of linear combination are given by components of solution vector 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 A is *nonsingular* if it has any of following equivalent properties

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



Existence and Uniqueness

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

A	b	# solutions
nonsingular	arbitrary	one (unique)
singular	$b \in \text{span}(A)$	infinitely many
singular	$b \notin \text{span}(A)$	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

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

is nonsingular regardless of value of b

- For example, if $b = [8 \ 13]^T$, then $x = [1 \ 2]^T$ is unique solution



Example: Singularity

- 2×2 system

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

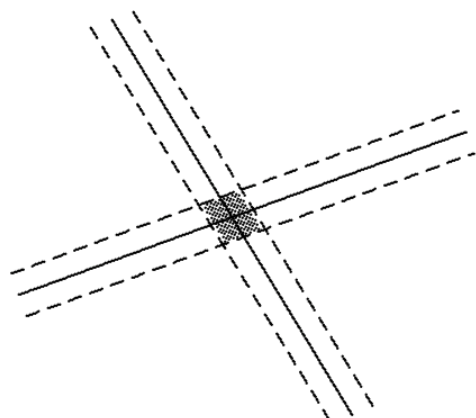
is singular regardless of value of b

- With $b = \begin{bmatrix} 4 & 7 \end{bmatrix}^T$, there is no solution
- With $b = \begin{bmatrix} 4 & 8 \end{bmatrix}^T$, $x = \begin{bmatrix} \gamma & (4 - 2\gamma)/3 \end{bmatrix}^T$ is solution for any real number γ , so there are infinitely many solutions

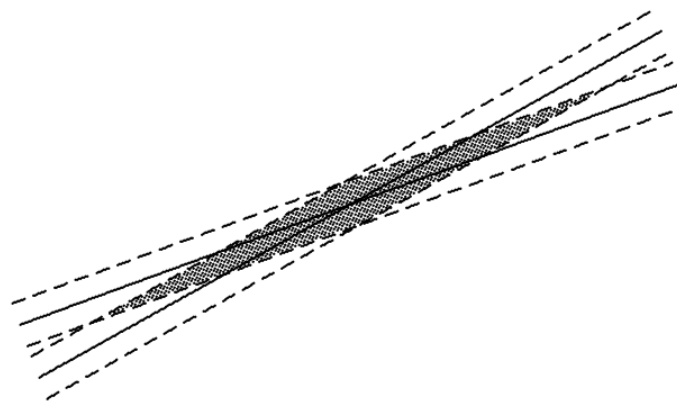


Nearly Singular Matrices

- 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

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

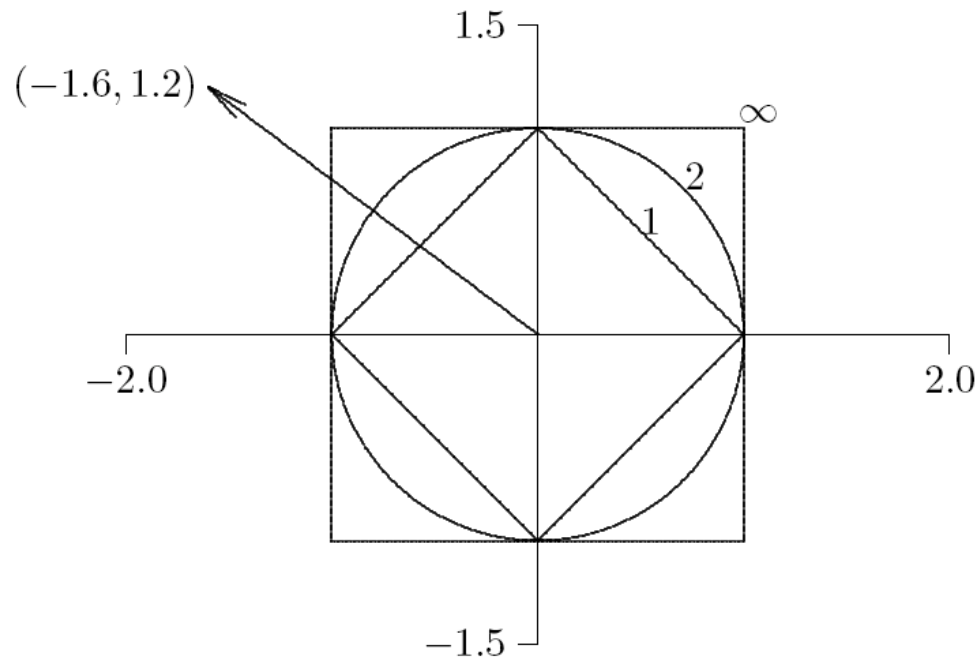
for integer $p > 0$ and n -vector $\mathbf{x}$

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



Example: Vector Norms

- Drawing shows unit sphere in two dimensions for each norm



- Norms have following values for vector shown

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



Equivalence of Norms

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

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

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



Matrix Norms

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

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

- 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

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

- 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

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

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

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

- 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 i th eigenvalue of A
- ❑ We say A is symmetric if $a_{ij} = a_{ji}$ for $i, j \in \{1, 2, \dots, n\}$
- ❑ 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
 - $\|A\| > 0$ if $A \neq 0$
 - $\|\gamma A\| = |\gamma| \cdot \|A\|$ for any scalar γ
 - $\|A + B\| \leq \|A\| + \|B\|$
- Matrix norms we have defined also satisfy
 - $\|AB\| \leq \|A\| \cdot \|B\|$
 - $\|Ax\| \leq \|A\| \cdot \|x\|$ for any vector 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 $k=0, 1, \dots$,

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

- 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

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

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

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

- 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:

$$A = nI + 0.1 R, \quad R = \text{rand}(n, n) \quad r_{ij} \in [0, 1]$$

$$M = \text{diag}(1/a_{ii})$$

- In this case,

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

- The ∞ -norm for G is given by

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

where

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

- In this case, we have a relative error bounded by $\|G\|_{\infty}^k \leq (0.1)^k$.
- 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 $\mathbf{A}$ is defined by

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$

- By convention, $\text{cond}(\mathbf{A}) = \infty$ if $\mathbf{A}$ is singular
- Since

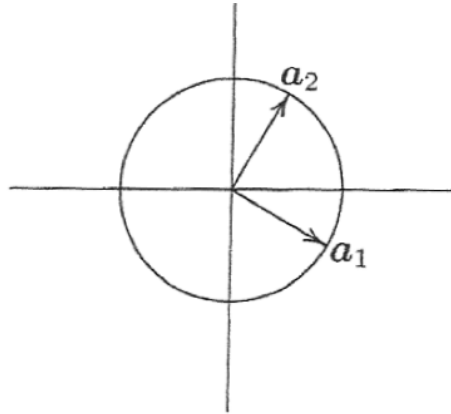
$$\|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\| = \left(\max_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \right) \cdot \left(\min_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \right)^{-1}$$

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

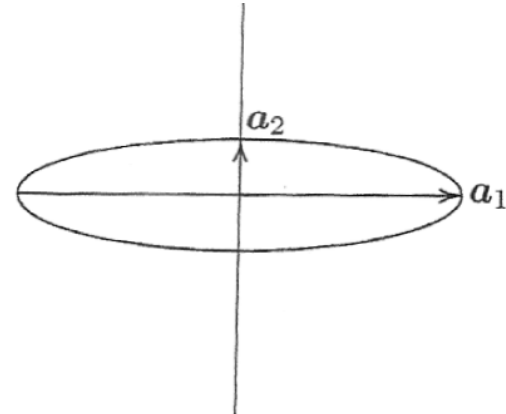
- Large $\text{cond}(\mathbf{A})$ means $\mathbf{A}$ is *nearly singular*



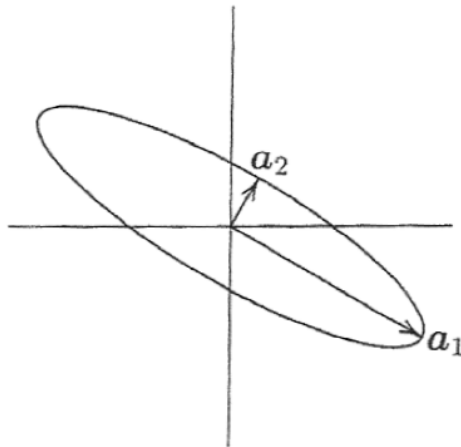
Condition Number Examples



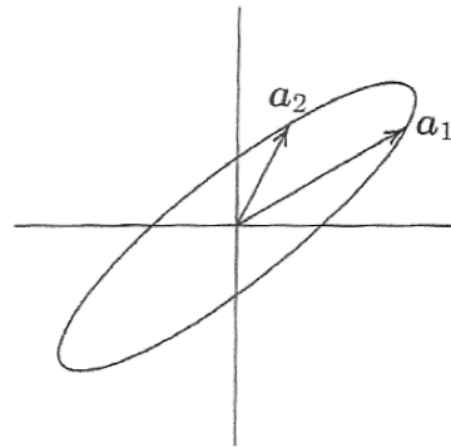
$$A_1 = \begin{bmatrix} 0.87 & 0.5 \\ -0.5 & 0.87 \end{bmatrix}, \quad \text{cond}_2(A_1) = 1$$



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



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



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

Properties of Condition Number

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



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 $\|A\|$ is easily computed as maximum absolute column sum (or row sum, depending on norm used)
- Estimating $\|A^{-1}\|$ at low cost is more challenging



Computing Condition Number, continued

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

$$\frac{\|z\|}{\|y\|} \leq \|\mathbf{A}^{-1}\|$$

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

- Efficient condition estimators heuristically pick $\mathbf{y}$ with large ratio $\|z\|/\|y\|$, yielding good estimate for $\|\mathbf{A}^{-1}\|$
- 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 x be solution to $Ax = b$, and let $\hat{x}$ be solution to $A\hat{x} = b + \Delta b$
- If $\Delta x = \hat{x} - x$, then

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

which leads to bound

$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\Delta b\|}{\|b\|}$$

for possible relative change in solution x due to relative change in right-hand side b



Condition Number and Relative Error: $A\mathbf{x} = \mathbf{b}$.

- Want to solve $A\mathbf{x} = \mathbf{b}$, but computed rhs is:

$$\mathbf{b}' = \mathbf{b} + \Delta\mathbf{b},$$

where we anticipate

$$\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} \approx \leq \epsilon_M.$$

- Net result is we end up solving $A\mathbf{x}' = \mathbf{b}'$ and want to know how large is the relative error, $\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$,

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|}?$$

- Since $A\mathbf{x}' = \mathbf{b}'$ and (by definition) $A\mathbf{x} = \mathbf{b}$, we have:

$$\|\Delta\mathbf{x}\| \leq \|A^{-1}\| \|\Delta\mathbf{b}\|$$

$$\|\mathbf{b}\| \leq \|A\| \|\mathbf{x}\|$$

$$\frac{1}{\|\mathbf{x}\|} \leq \|A\| \frac{1}{\|\mathbf{b}\|}$$

$$\frac{\Delta\mathbf{x}}{\|\mathbf{x}\|} \leq \|A\| \frac{\Delta\mathbf{x}}{\|\mathbf{b}\|}$$

$$\leq \|A\| \|A^{-1}\| \frac{\Delta\mathbf{b}}{\|\mathbf{b}\|}$$

$$= \text{cond}(A) \frac{\Delta\mathbf{b}}{\|\mathbf{b}\|}.$$

- Key point: If $\text{cond}(A)=10^k$, then expected relative error is $\approx 10^k \epsilon_M$, meaning that you will lose k digits (of 16, if $\epsilon_M \approx 10^{-16}$).

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.

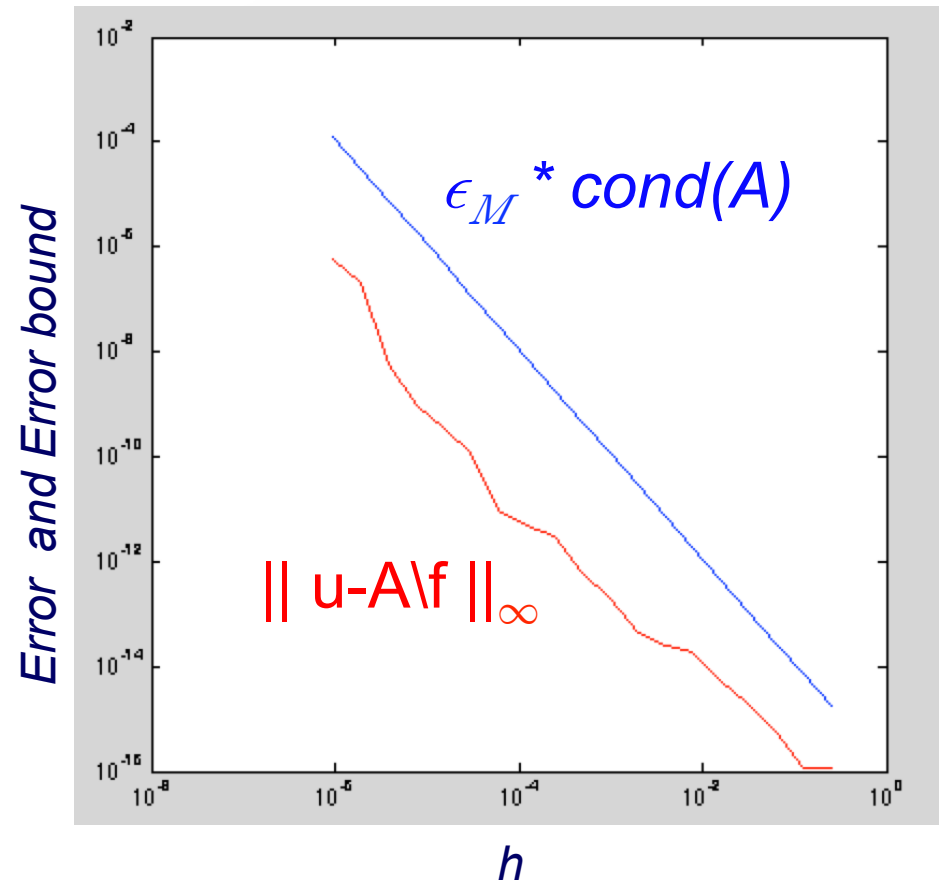
for k=2:20; n = (2^k)-1; h=1/(n+1);

    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 $\epsilon_M^ \text{cond}(A)$ bounds the error in the solution to $Au=f$, as expected.*



Error Bounds, continued

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

$$\frac{\|\Delta \mathbf{x}\|}{\|\hat{\mathbf{x}}\|} \leq \text{cond}(\mathbf{A}) \frac{\|\mathbf{E}\|}{\|\mathbf{A}\|}$$

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

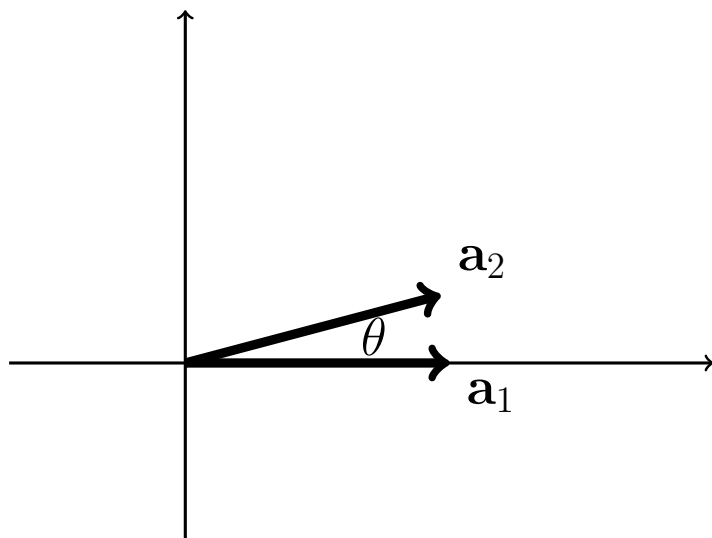
$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \text{cond}(\mathbf{A}) \epsilon_{\text{mach}}$$

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

Example



A Nearly Singular Example



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

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

- Clearly, as $\theta \rightarrow 0$ 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 $\text{cond}(A)$ as a function of θ , as well as the estimates from the preceding slide.

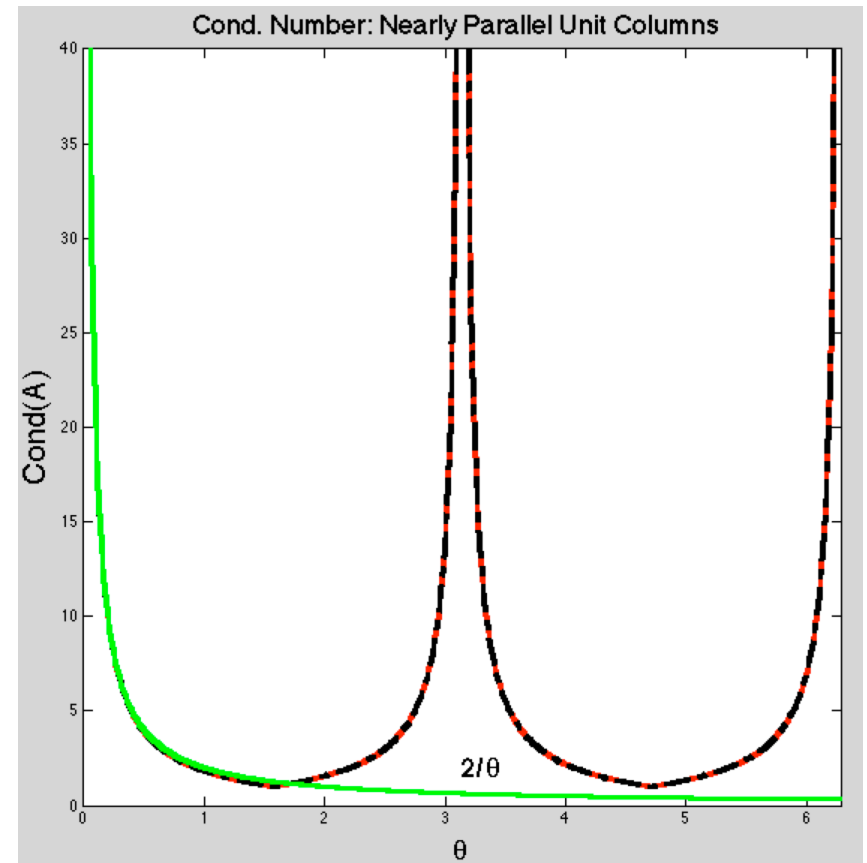
- ❑ The computed value of $\text{cond}(A)$ given by matlab exactly matches $[(1+|\cos \theta|)/(1-|\cos \theta|)]^{1/2}$
- ❑ The more interesting result is $\text{cond}(A) \sim 2/\theta$, which is very accurate for small angles.

```
%% 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

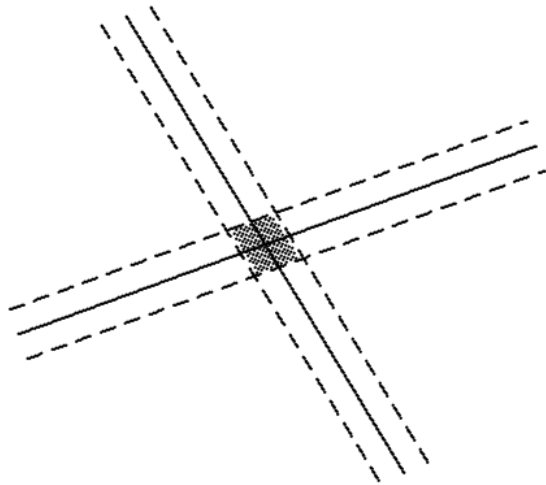
jj=0; for j=.01:.01:(2*pi); cj=cos(j);sj=sin(j); jj=jj+1;
    R=[ cj -sj ; sj cj ];
    a1 = [ 1 ; 0 ]; a2 = R*a1; A = [ a1 a2 ];

    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)
```

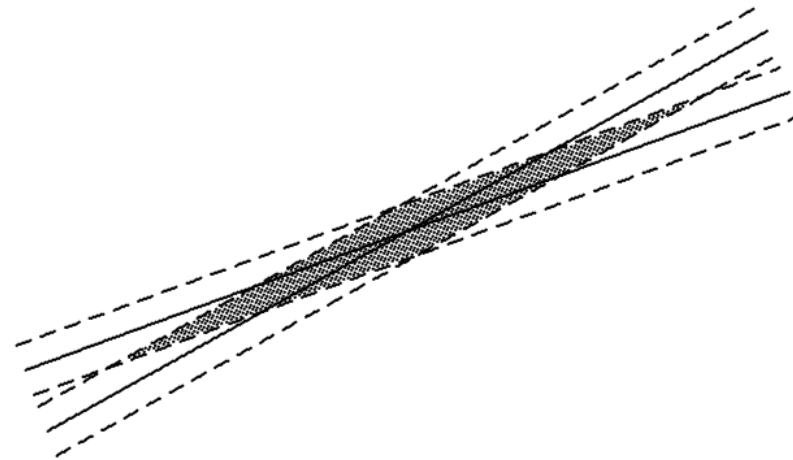


Error Bounds – Illustration

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



well-conditioned



ill-conditioned



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 $\hat{x}$ to linear system $Ax = b$ is defined by

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

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

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

small relative residual implies small relative error in approximate solution *only if* A is well-conditioned



Residual, continued

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

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

then

$$\frac{\|r\|}{\|A\| \|\hat{x}\|} \leq \frac{\|E\|}{\|A\|}$$

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 $Ax = b$ by any *nonsingular* matrix M without affecting solution
- Solution to $MAx = Mb$ is given by

$$x = (MA)^{-1}Mb = A^{-1}M^{-1}Mb = A^{-1}b$$



Example: Permutations

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

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



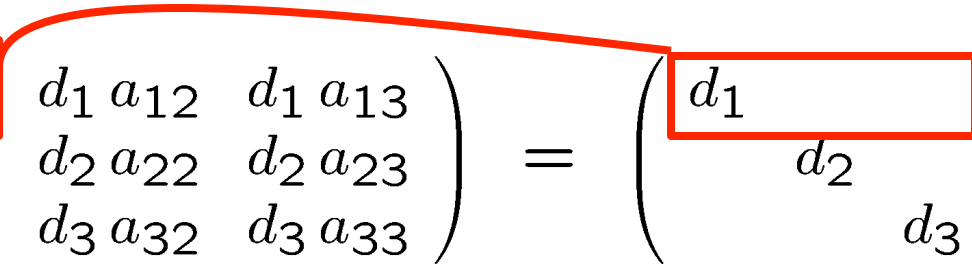
Example: Diagonal Scaling

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

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



Premultiply by Diagonal Matrix: Row Scaling

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


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 $Lx = b$

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

```
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 $Ux = b$

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

```
for  $j = n$  to 1
    if  $u_{jj} = 0$  then stop
     $x_j = b_j / u_{jj}$ 
    for  $i = 1$  to  $j - 1$ 
         $b_i = b_i - u_{ij} x_j$ 
    end
end
```

{ loop backwards over columns }
{ stop if matrix is singular }
{ compute solution component }

{ update right-hand side }



Solution of Lower Triangular Systems

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

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

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
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
end

```

Solution of Upper Triangular Systems

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & \cdots & u_{1n} \\ & u_{22} & u_{23} & \cdots & \cdots & u_{2n} \\ & & u_{33} & & & u_{3n} \\ & & & \ddots & & \vdots \\ & & & & \ddots & \vdots \\ & & & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

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

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, $\beta = 2$:

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

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

What is the cost ??

Solution of Upper Banded Systems

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

As written:

```

for i = n : 1
    x_i = b_i, j_max := min(j + β, n)
    for j = i + 1 : j_max
        x_i = x_i - u_ij x_j
    end
    x_i = x_i / u_ii
end
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 - β)
    for i = i_min : j - 1
        b_i = b_i - u_ij x_j
    end
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:

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

- $a_{22} = 4$ 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 $A(:, n + 1)$:

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

This Case.

$$\text{pivot} = 4$$

$$\text{pivot row} = [4 \ 6 \ 1 \mid 4]$$

$$\text{multiplier column} = \frac{1}{4} \begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

General Case.

$$= a_{kk} \text{ when zeroing the } k\text{th column.}$$

$$= \mathbf{r}_k^T = a_{kj}, j = k + 1, \dots, n [+ b_k]$$

$$= \mathbf{c}_k = \frac{a_{ik}}{a_{kk}}, i = k + 1, \dots, n$$

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.

$$\text{pivot} = 4$$

$$\text{pivot row} = [4 \ 6 \ 1 \ 4]$$

$$\text{multiplier column} = \frac{1}{4} \begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

General Case.

$$= a_{kk} \text{ when zeroing the } k\text{th column.}$$

$$= \mathbf{r}_k^T = a_{kj}, j = k + 1, \dots, n [+ b_k]$$

$$= \mathbf{c}_k = \frac{a_{ik}}{a_{kk}}, i = k + 1, \dots, n$$

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.

$$\text{pivot} = 4$$

$$\text{pivot row} = [4 \ 6 \ 1 \ 4]$$

$$\text{multiplier column} = \frac{1}{4} \begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

General Case.

$$= a_{kk} \text{ when zeroing the } k\text{th column.}$$

$$= \mathbf{r}_k^T = a_{kj}, j = k + 1, \dots, n [+ b_k]$$

$$= \mathbf{c}_k = \frac{a_{ik}}{a_{kk}}, i = k + 1, \dots, n$$

$$\mathbf{c}_k \longrightarrow \mathbf{l}_k, \text{ store as column } k \text{ of } L.$$

k th Update Step

- Look more closely at the k th 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_{ik} .
- We do this by subtracting a multiple of row k from row i .
- This operation can be expressed in several equivalent ways:

$$\text{row}_i = \text{row}_i - \frac{a_{ik}}{a_{kk}} \times \text{row}_k$$

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

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

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 $(\mathbf{r}_k^T)_j = a_{kj}$.
- Formally, we think of $(\mathbf{c}_k)_i = 0$, $i \leq k$ and $(\mathbf{r}_k^T)_j = 0$, $j \leq k$, 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 k th column of a lower triangular matrix: $l_{ik} := (\mathbf{c}_k)_i$.

k th Update Step

- Look more closely at the k th 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_{ik} .
- We do this by subtracting a multiple of row k from row i .
- This operation can be expressed in several equivalent ways:

$$\text{row}_i = \text{row}_i - \frac{a_{ik}}{a_{kk}} \times \text{row}_k$$

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

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

- 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 $(\mathbf{r}_k^T)_j = a_{kj}$.
- Formally, we think of $(\mathbf{c}_k)_i = 0$, $i \leq k$ and $(\mathbf{r}_k^T)_j = 0$, $j \leq k$, 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 k th column of a lower triangular matrix: $l_{ik} := (\mathbf{c}_k)_i$.

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

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

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

- Recall, for example,

$$\begin{aligned}
 a_{22}^{(2)} &= a_{22}^{(1)} - \frac{a_{21}^{(1)}a_{12}^{(1)}}{a_{11}^{(1)}}, \text{ or} \\
 a_{ij}^{(k+1)} &= a_{ij}^{(k)} - \frac{a_{ik}^{(k)}a_{kj}^{(k)}}{a_{kk}^{(k)}}, \text{ in general.}
 \end{aligned}$$

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

***LU* Factorization as a Sequence of Matrix-Matrix Products**

(Following notation in the text.)

- Consider solution of $A\mathbf{x} = \mathbf{b}$ via Gaussian elimination.
- Let $A^{(1)} := A$ and $\mathbf{b}^{(1)} := \mathbf{b}$.
- Take $n = 4$ for purposes of illustration.
- Apply one-step of Gaussian elimination to the augmented system $[A^{(1)} \mid \mathbf{b}^{(1)}]$.
- After one round, we have:

$$\begin{aligned}
 [A^{(2)} \mid \mathbf{b}^{(2)}] &= M_1 [A^{(1)} \mid \mathbf{b}^{(1)}] \\
 &= M_1 \left[\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & \vdots & a_{14}^{(1)} & b_1^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \vdots & a_{24}^{(1)} & b_2^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & \vdots & a_{34}^{(1)} & b_3^{(1)} \\ a_{41}^{(1)} & a_{42}^{(1)} & \vdots & a_{44}^{(1)} & b_4^{(1)} \end{array} \right] \\
 &=: \left[\begin{array}{cccc|c} a_{11}^{(2)} & a_{12}^{(2)} & \vdots & a_{14}^{(2)} & b_1^{(2)} \\ 0 & a_{22}^{(2)} & \vdots & a_{24}^{(2)} & b_2^{(2)} \\ 0 & a_{32}^{(2)} & \vdots & a_{34}^{(2)} & b_3^{(2)} \\ 0 & a_{42}^{(2)} & \vdots & a_{44}^{(2)} & b_4^{(2)} \end{array} \right].
 \end{aligned}$$

- That is, $M_1 [A^{(1)} \mid \mathbf{b}^{(1)}] = [A^{(2)} \mid \mathbf{b}^{(2)}]$, where $A^{(2)}$ is zero in column 1 for $i > 1$.

- That is, $M_1 [A^{(1)} | \mathbf{b}^{(1)}] = [A^{(2)} | \mathbf{b}^{(2)}]$, where $A^{(2)}$ is zero in column 1 for $i > 1$.
- The matrix that zeros out these entries in column one is given by:

$$M_1 = I - \mathbf{m}_1 \mathbf{e}_1^T, \quad \mathbf{m}_1 = \frac{1}{a_{11}^{(1)}} \begin{bmatrix} 0 & a_{21}^{(1)} & a_{31}^{(1)} & a_{41}^{(1)} \end{bmatrix}^T,$$

and $\mathbf{e}_1$ = the 1st column of the identity matrix.

- **Test:** Apply M_1 to each column of $[A^{(1)} | \mathbf{b}^{(1)}]$:

$$M_1 \cdot \mathbf{a}_1^{(1)} = \mathbf{a}_1^{(1)} - \mathbf{m}_1 \mathbf{e}_1^T \mathbf{a}_1^{(1)}$$

$$\left[M_1 \mathbf{a}_1^{(1)} \right]_i = a_{i1}^{(1)} - \left(\frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \right) a_{11}^{(1)} = 0, \quad i > 1.$$

- **Test:** Apply M_1 to each column of $[A^{(1)} \mid \mathbf{b}^{(1)}]$:

$$M_1 \cdot \mathbf{a}_1^{(1)} = \mathbf{a}_1^{(1)} - \mathbf{m}_1 \mathbf{e}_1^T \mathbf{a}_1^{(1)}$$

$$\left[M_1 \mathbf{a}_1^{(1)} \right]_i = a_{i1}^{(1)} - \left(\frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \right) a_{11}^{(1)} = 0, \quad i > 1.$$

For any $\mathbf{z} \in \mathbb{R}^n$,

$$[M_1 \mathbf{z}]_i = z_i - \left(\frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \right) z_1 \quad i > 1.$$

For any matrix $V \in \mathbb{R}^{n \times n'}$,

$$[M_1 V]_{ij} = V_{ij} - \left(\frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \right) V_{1j} \quad i > 1, \quad j = 1, \dots, n'.$$

$$i\text{th row} \longrightarrow i\text{th row} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \times 1\text{st row}.$$

Elimination Step!!

- Now, we take next step, $[A^{(3)} | \mathbf{b}^{(3)}] = M_2 [A^{(2)} | \mathbf{b}^{(2)}]$:

$$\begin{aligned}
[A^{(3)} | \mathbf{b}^{(3)}] &= M_2 \left[\begin{array}{cccc|c} a_{11}^{(2)} & a_{12}^{(2)} & \vdots & a_{14}^{(2)} & b_1^{(2)} \\ 0 & a_{22}^{(2)} & \vdots & a_{24}^{(2)} & b_2^{(2)} \\ 0 & a_{32}^{(2)} & \vdots & a_{34}^{(2)} & b_3^{(2)} \\ 0 & a_{42}^{(2)} & \vdots & a_{44}^{(2)} & b_4^{(2)} \end{array} \right] \\
&= \left[\begin{array}{cccc|c} a_{11}^{(3)} & a_{12}^{(3)} & \vdots & a_{14}^{(3)} & b_1^{(3)} \\ 0 & a_{22}^{(3)} & \vdots & a_{24}^{(3)} & b_2^{(3)} \\ 0 & 0 & \vdots & a_{34}^{(3)} & b_3^{(3)} \\ 0 & 0 & \vdots & a_{44}^{(3)} & b_4^{(3)} \end{array} \right],
\end{aligned}$$

with

$$M_2 = I - \mathbf{m}_2 \mathbf{e}_2^T, \quad \mathbf{m}_2 = \frac{1}{a_{11}^{(2)}} \begin{bmatrix} 0 & 0 & a_{31}^{(2)} & a_{41}^{(2)} \end{bmatrix}^T,$$

and $\mathbf{e}_2$ = the 2nd column of the identity matrix.

- After $n - 1$ rounds, we have

$$\left[A^{(n-1)} \mid \mathbf{b}^{(n-1)} \right] = M_{n-1} M_{n-2} \cdots M_2 M_1 \left[A \mid \mathbf{b} \right],$$

with $U = A^{(n-1)}$ being upper triangular, and

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

the k th **elementary elimination matrix**.

- It's easy to show that $M_k^{-1} = I + \mathbf{m}_k \mathbf{e}_k^T$.

Gaussian Elimination and Elementary Elimination Matrices

$$\begin{aligned} U &= M_{n-1}M_{n-2}\cdots M_2M_1A \\ &= L^{-1}A \longrightarrow \boxed{LU = A.} \end{aligned}$$

$$\begin{aligned} L^{-1} &= M_{n-1}M_{n-2}\cdots M_2M_1 \\ L &= M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1} \\ &= L_1L_2\cdots L_{n-1}, \end{aligned}$$

with

$$L_k := M_k^{-1} = I + \mathbf{m}_k \mathbf{e}_k^T.$$

- With more work, can show

$$L = \begin{bmatrix} 1 & & & & & \\ m_{21} & 1 & & & & \\ m_{31} & m_{32} & 1 & & & \\ \vdots & & & \ddots & & \\ \vdots & & & & \ddots & \\ m_{n1} & m_{n2} & m_{n3} & \cdots & \cdots & 1 \end{bmatrix}.$$

That is, the entries of L are just the entries of the multiplier columns!

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:

$$M \underline{A} \underline{x} = M \underline{b},$$

$$M := M_{n-1} M_{n-2} \dots M_1 =: L^{-1}.$$

Elementary Elimination Matrices

- More generally, we can annihilate *all* entries below k th position in n -vector a by transformation

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

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

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



Using LU Factorization in Practice

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

$$\text{Given: } A\mathbf{x} = LU\mathbf{x} = \mathbf{b}$$

$$L(U\mathbf{x}) = L\mathbf{y} = \mathbf{b}$$

$$\text{Solve: } L\mathbf{y} = \mathbf{b}$$

$$U\mathbf{x} = \mathbf{y}$$

- 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)

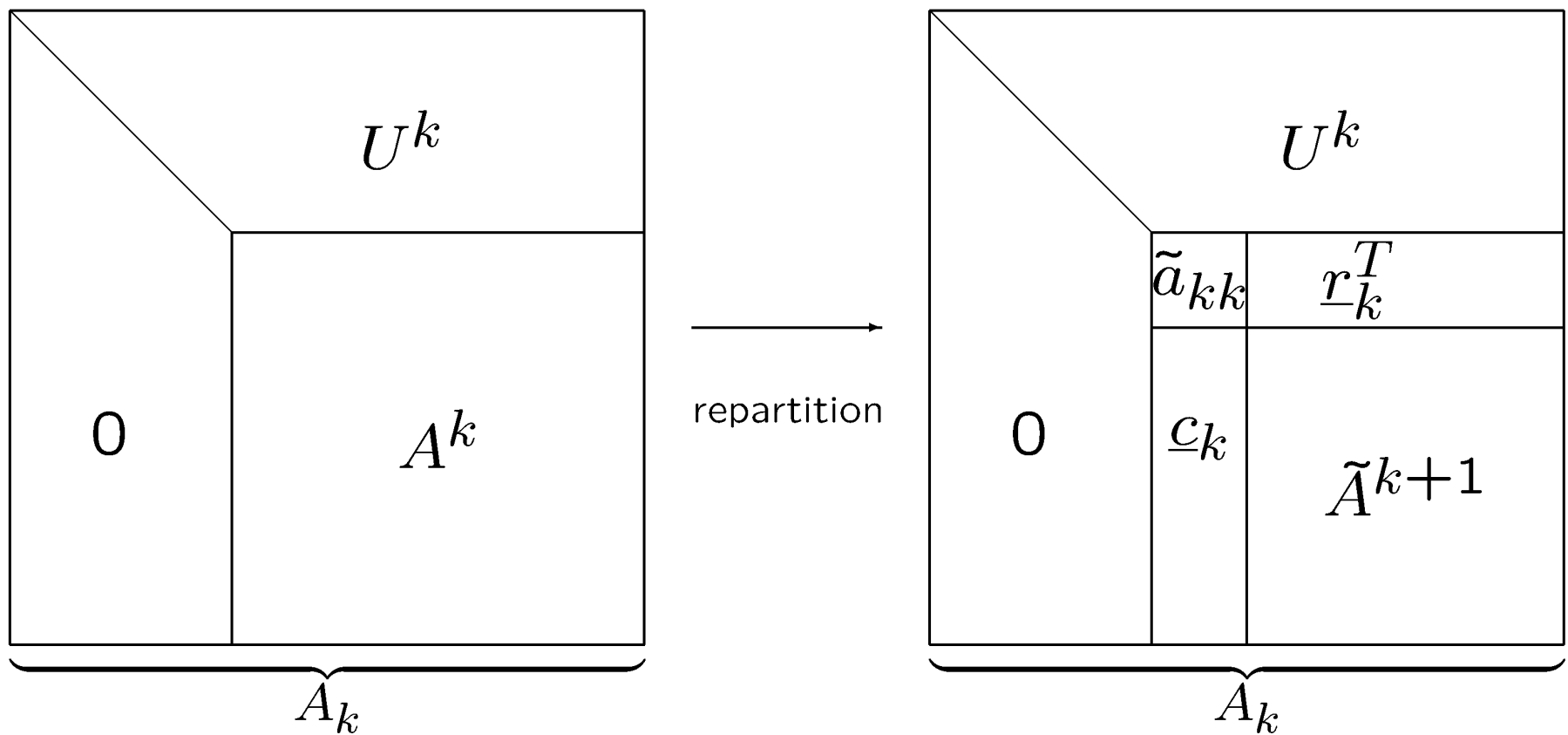
- In general, the cost for $A \longrightarrow LU$ is $O(n^3)$.
- It is large (i.e., it is not optimal, which would be $O(n)$), and therefore important.
- The dominant cost comes from the essential update step:

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

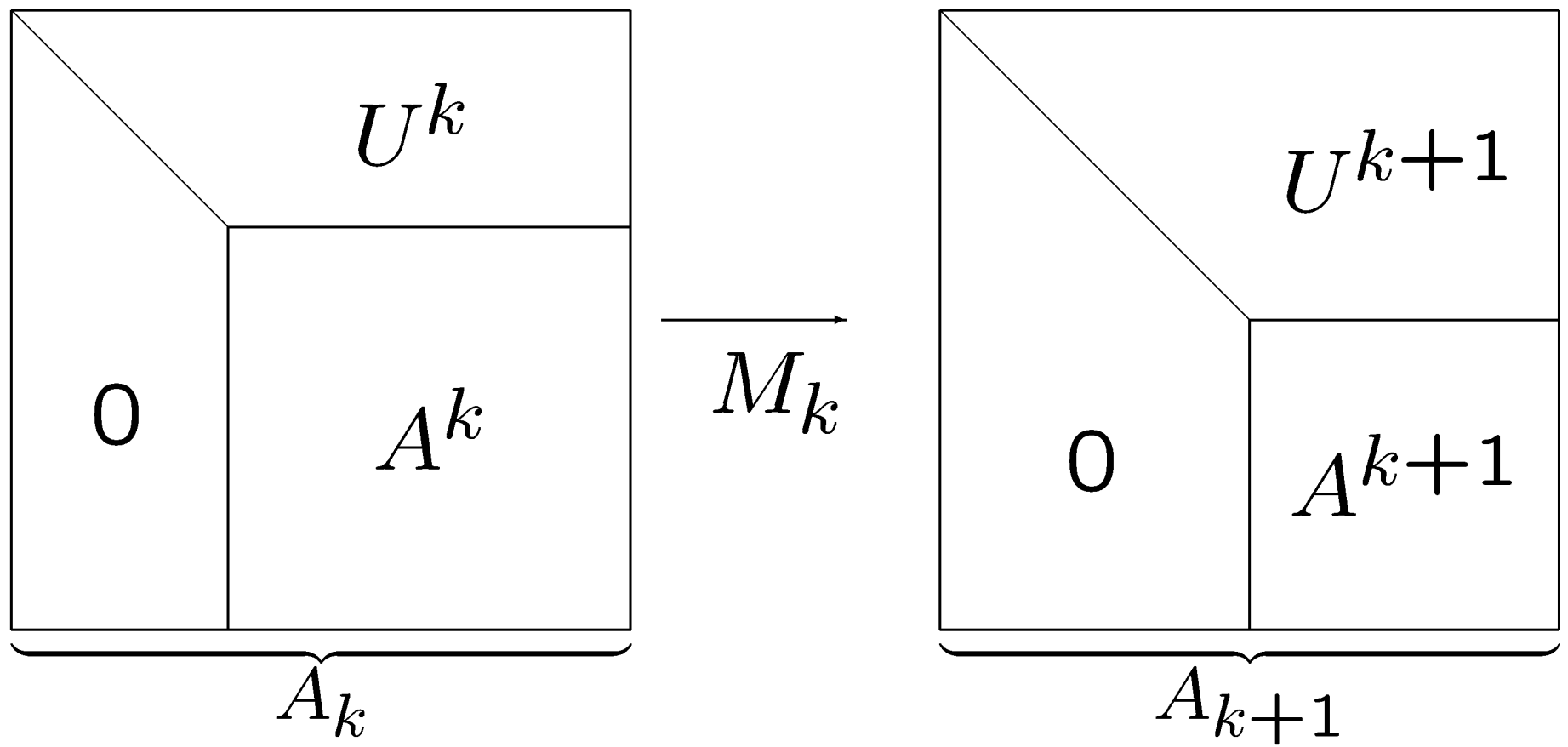
which is effected for $k = 1, \dots, n - 1$ steps.

- 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 $2 \times [(n - 1)^2 + (n - 2)^2 + \dots (1)^2] \sim 2n^3/3$ 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.

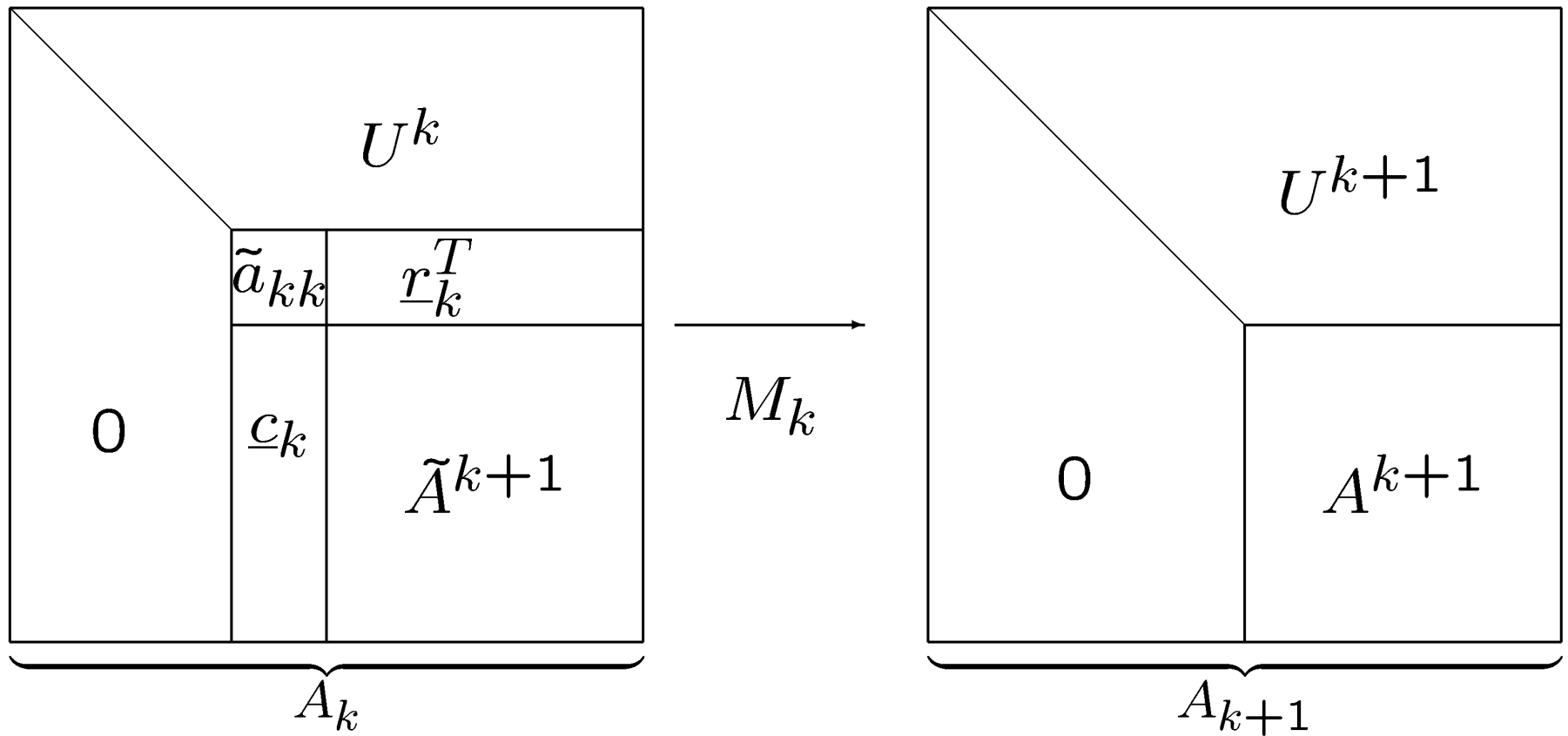
First Step: Define sub-block



Single Gaussian Elimination Step



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



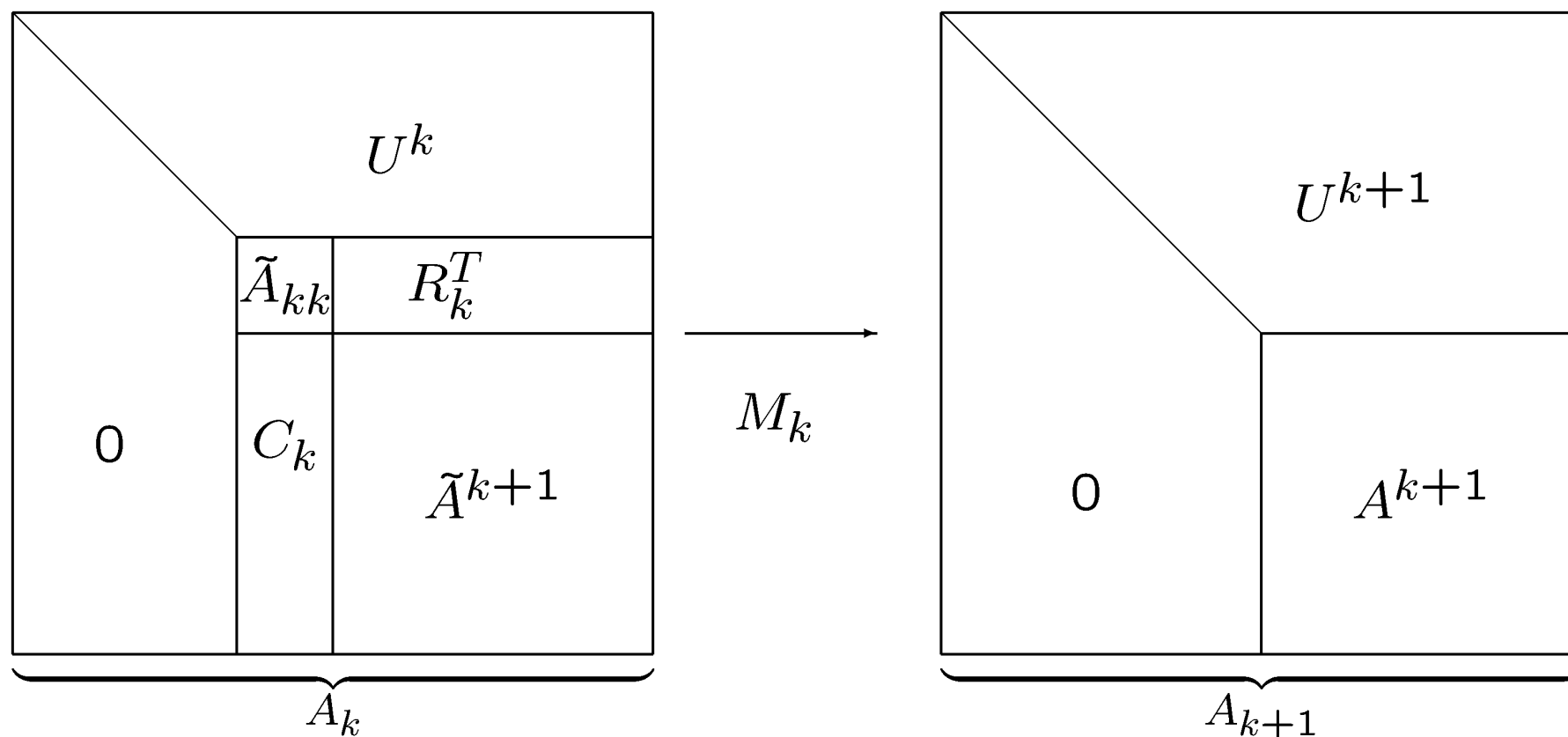
□ Update step is:

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

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**

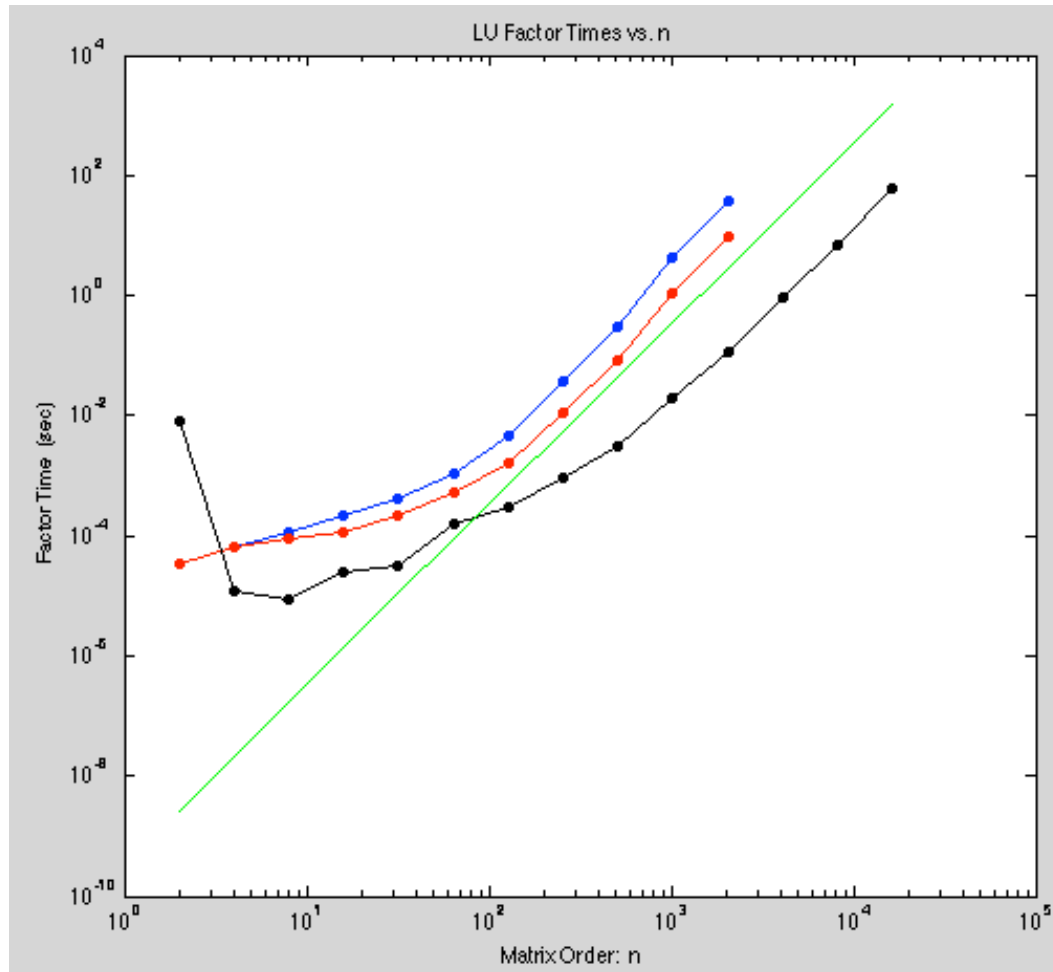


$$A^{k+1} = \tilde{A}^{k+1} - C_k \tilde{A}_{kk}^{-1} R_k^T$$

- Advantage is that, if A_{kk} is a $b \times b$ 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 very near peak
- Note that the black curve represents a **10-20x** speed up over a naïve rank-1 update approach.

Next Topics

- ❑ Pivoting / zeros & stability
 - ❑ Approach
 - ❑ Permutation Matrices
 - ❑ Stability
 - ❑ Cost

- ❑ Sherman Morrison

- ❑ Computing matrix 2-norm

- ❑ SPD / Cholesky Factorization

- ❑ Banded Factorization
 - ❑ Approach
 - ❑ Cost

Recall our earlier 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:

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

- $a_{22} = 4$ 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 $A(:, n + 1)$:

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

This Case.

$$\text{pivot} = 4$$

$$\text{pivot row} = [4 \ 6 \ 1 \mid 4]$$

$$\text{multiplier column} = \frac{1}{4} \begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

General Case.

$$= a_{kk} \text{ when zeroing the } k\text{th column.}$$

$$= \mathbf{r}_k^T = a_{kj}, j = k + 1, \dots, n [+ b_k]$$

$$= \mathbf{c}_k = \frac{a_{ik}}{a_{kk}}, i = k + 1, \dots, n$$

Generating Upper Triangular Systems: LU Factorization

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

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

This Case.

$$\text{pivot} = 4$$

$$\text{pivot row} = [4 \ 6 \ 1 \ 4]$$

$$\text{multiplier column} = \frac{1}{4} \begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

General Case.

$$= a_{kk} \text{ when zeroing the } k\text{th column.}$$

$$= \mathbf{r}_k^T = a_{kj}, j = k + 1, \dots, n [+ b_k]$$

$$= \mathbf{c}_k = \frac{a_{ik}}{a_{kk}}, i = k + 1, \dots, n$$

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.

$$\text{pivot} = 4$$

$$\text{pivot row} = [4 \ 6 \ 1 \ 4]$$

$$\text{multiplier column} = \frac{1}{4} \begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

General Case.

$$= a_{kk} \text{ when zeroing the } k\text{th column.}$$

$$= \mathbf{r}_k^T = a_{kj}, j = k + 1, \dots, n [+ b_k]$$

$$= \mathbf{c}_k = \frac{a_{ik}}{a_{kk}}, i = k + 1, \dots, n$$

$$\mathbf{c}_k \longrightarrow \mathbf{l}_k, \text{ store as column } k \text{ of } L.$$

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 difficulty 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

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

$$\text{pivot} = -5$$

$$\text{pivot row} = \left[-6 \quad \frac{3}{2} \mid -2 \right]$$

$$\text{multiplier column} = \frac{1}{-5} \begin{bmatrix} 0 \\ -2 \end{bmatrix}$$

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 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 M_k is preceded by permutation P_k to interchange rows to bring entry of largest magnitude into diagonal pivot position
- Still obtain $MA = U$, with U upper triangular, but now

$$M = M_{n-1}P_{n-1} \cdots M_1P_1$$

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

$$PA = LU$$

where $P = P_{n-1} \cdots P_1$ permutes rows of A into order determined by partial pivoting, and now 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

$$PAQ = LU$$

with L unit lower triangular, U upper triangular, and P and 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** P has one 1 in each row and column and zeros elsewhere, i.e., identity matrix with rows or columns permuted
- Note that $P^{-1} = P^T$
- Premultiplying both sides of system by permutation matrix, $PAx = Pb$, reorders rows, but solution x is unchanged
- Postmultiplying A by permutation matrix, $APx = b$, reorders columns, which permutes components of original solution

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



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,

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

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

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

is singular yet has LU factorization



Example: Small Pivots

- To illustrate effect of small pivots, consider

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

where ϵ is positive number smaller than ϵ_{mach}

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

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

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

in floating-point arithmetic, but then

$$\mathbf{LU} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} \neq \mathbf{A}$$



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

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

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

in floating-point arithmetic

- Thus,

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

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 exchange a small pivot on the diagonal can result in all subsequent rows looking like multiples of the current pivot row → ***singular submatrix***.

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

□ Consider

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

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 $> \epsilon_M$, we are effectively reducing 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 2×2 example where the condition number of the matrix with 2 unit-norm vectors scales like $2 / \theta$, 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 $O(n^2)\lambda 2n^3/3$.
- 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 $O(n^2)$.

Notes:

- Partial (row) pivoting ensures that multiplier column entries have modulus ≤ 1 . (Good.)
- Full pivoting also destroys band structure, whereas partial pivoting leaves some band structure intact.

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*

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



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 $PA = LU = \hat{L}\hat{U}$, then $\hat{L}^{-1}L = \hat{U}U^{-1} = D$ is both lower and upper triangular, hence diagonal
- If both L and $\hat{L}$ are unit lower triangular, then D must be identity matrix, so $L = \hat{L}$ and $U = \hat{U}$
- Uniqueness is made explicit in LDU factorization $PA = LDU$, with L unit lower triangular, U unit upper triangular, and D diagonal



Storage Management

- Elementary elimination matrices M_k , their inverses L_k , and permutation matrices P_k used in formal description of LU factorization process are *not* formed explicitly in actual implementation
- U overwrites upper triangle of A , multipliers in L overwrite strict lower triangle of A , and unit diagonal of 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 b , inversion never overcomes higher initial cost, since each matrix-vector multiplication $A^{-1}b$ 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 $A^{-1}B$ should be computed by LU factorization of A , followed by forward- and back-substitutions using each column of 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

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

where $\mathbf{u}$ and $\mathbf{v}$ 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 $(A - uv^T)x = b$ with new matrix, use Sherman-Morrison formula to obtain

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

which can be implemented by following steps

- Solve $Az = u$ for z , so $z = A^{-1}u$
 - Solve $Ay = b$ for y , so $y = A^{-1}b$
 - Compute $x = y + ((v^T y)/(1 - v^T z))z$
- If 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

- One way to choose update vectors is

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

Original Matrix

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

so matrix of modified system is $A - uv^T$



Example, continued

- Using LU factorization of A to solve $Az = u$ and $Ay = b$,

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

- Final step computes updated solution

Q: Under what circumstances could the denominator be zero ?

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

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



Sherman Morrison

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

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

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

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

[2] New problem:

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

Key Idea:

- $(A - \mathbf{u}\mathbf{v}^T) \mathbf{x}$ differs from $A\mathbf{x}$ by only a small amount of information.
- Rewrite as: $A\mathbf{x} + \mathbf{u}\gamma = \mathbf{b}$
 $\gamma := -\mathbf{v}^T \mathbf{x} \iff \mathbf{v}^T \mathbf{x} + \gamma = 0$

Sherman Morrison

Extended system:

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

$$\mathbf{v}^T\mathbf{x} + \gamma = 0$$

Sherman Morrison

Extended system:

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

In matrix form:

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

Sherman Morrison

Extended system:

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

In matrix form:

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

Eliminate for γ :

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

Sherman Morrison

Extended system:

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

In matrix form:

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

Eliminate for γ :

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

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

Sherman Morrison

Extended system:

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

In matrix form:

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

Eliminate for γ :

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

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

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

Sherman Morrison

Extended system:

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

In matrix form:

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

Eliminate for γ :

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

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

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

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

Sherman Morrison: Potential Singularity

- Consider the modified system: $(A - \mathbf{u}\mathbf{v}^T) \mathbf{x} = \mathbf{b}$.
- The solution is

$$\begin{aligned}\mathbf{x} &= (A - \mathbf{u}\mathbf{v}^T)^{-1} \mathbf{b} \\ &= \left[I + A^{-1} \mathbf{u} (1 - \mathbf{v}^T A^{-1} \mathbf{u})^{-1} \mathbf{v}^T A^{-1} \right] A^{-1} \mathbf{b}.\end{aligned}$$

- If $1 - \mathbf{v}^T A^{-1} \mathbf{u} = 0$, failure.
- Why?

Sherman Morrison: Potential Singularity

- Let $\tilde{A} := (A - \mathbf{u}\mathbf{v}^T)$ and consider,

$$\begin{aligned}\tilde{A} A^{-1} &= (A - \mathbf{u}\mathbf{v}^T) A^{-1} \\ &= (I - \mathbf{u}\mathbf{v}^T A^{-1}).\end{aligned}$$

- Look at the product $\tilde{A} A^{-1} \mathbf{u}$,

$$\begin{aligned}\tilde{A} A^{-1} \mathbf{u} &= (I - \mathbf{u}\mathbf{v}^T A^{-1}) \mathbf{u} \\ &= \mathbf{u} - \mathbf{u}\mathbf{v}^T A^{-1} \mathbf{u}.\end{aligned}$$

- If $\mathbf{v}^T A^{-1} \mathbf{u} = 1$, then

$$\tilde{A} A^{-1} \mathbf{u} = \mathbf{u} - \mathbf{u} = \mathbf{0},$$

which means that $\tilde{A}$ is singular since we assume that A^{-1} exists.

- Thus, an unfortunate choice of $\mathbf{u}$ and $\mathbf{v}$ can lead to a singular modified matrix and this singularity is indicated by $\mathbf{v}^T A^{-1} \mathbf{u} = 1$.

Computing $\|A\|_2$ and $\text{cond}_2(A)$.

- Recall: $\text{cond}(A) := \|A^{-1}\| \cdot \|A\|,$

$$\|A\| := \max_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|},$$

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^n x_i^2 \right)^{\frac{1}{2}} = \sqrt{\mathbf{x}^T \mathbf{x}},$$

$$\|\mathbf{x}\|_2^2 = \mathbf{x}^T \mathbf{x}.$$

- From now on, drop the subscript “₂”.

$$\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x}$$

$$\|A\mathbf{x}\|^2 = (A\mathbf{x})^T (A\mathbf{x}) = \mathbf{x}^T A^T A \mathbf{x}.$$

- Matrix norm:

$$\begin{aligned}
 \|A\|^2 &= \max_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|^2}{\|\mathbf{x}\|^2}, \\
 &= \max_{\mathbf{x} \neq 0} \frac{\mathbf{x}^T A^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}} \\
 &= \lambda_{\max}(A^T A) =: \text{spectral radius of } (A^T A).
 \end{aligned}$$

- The symmetric positive definite matrix $B := A^T A$ has positive eigenvalues.
- All symmetric matrices B have a complete set of orthonormal eigenvectors satisfying

$$B\mathbf{z}_j = \lambda_j \mathbf{z}_j, \quad \mathbf{z}_i^T \mathbf{z}_j = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

- **Note:** If $\lambda_i = \lambda_j$, $i \neq j$, then can have $\mathbf{z}_i^T \mathbf{z}_j \neq 0$, but we can orthogonalize $\mathbf{z}_i$ and $\mathbf{z}_j$ so that $\tilde{\mathbf{z}}_i^T \tilde{\mathbf{z}}_j = 0$ and

$$B\tilde{\mathbf{z}}_i = \lambda_i \tilde{\mathbf{z}}_i \quad \lambda_i = \lambda_j$$

$$B\tilde{\mathbf{z}}_j = \lambda_j \tilde{\mathbf{z}}_j.$$

- Assume eigenvalues are sorted with $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$.
- For any $\mathbf{x}$ we have: $\mathbf{x} = c_1 \mathbf{z}_1 + c_2 \mathbf{z}_2 + \dots + c_n \mathbf{z}_n$.
- Let $\|\mathbf{x}\| = 1$.

- Want to find $\max_{\|\mathbf{x}\|=1} \frac{\mathbf{x}^T B \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \max_{\|\mathbf{x}\|=1} \mathbf{x}^T B \mathbf{x}$.

- Note: $\mathbf{x}^T \mathbf{x} = \left(\sum_{i=1}^n c_i \mathbf{z}_i \right)^T \left(\sum_{j=1}^n c_j \mathbf{z}_j \right)$

$$= \sum_{i=1}^n \sum_{j=1}^n c_i c_j \mathbf{z}_i^T \mathbf{z}_j$$

$$= \sum_{i=1}^n \sum_{j=1}^n c_i c_j \delta_{ij}$$

$$= \sum_{i=1}^n c_i^2 = 1.$$

$$\implies c_1^2 = 1 - \sum_{i=2}^n c_i^2.$$

$$\begin{aligned}
\mathbf{x}^T B \mathbf{x} &= \left(\sum_{i=1}^n c_i \mathbf{z}_i \right)^T \left(\sum_{j=1}^n c_j B \mathbf{z}_j \right) \\
&= \left(\sum_{i=1}^n c_i \mathbf{z}_i \right)^T \left(\sum_{j=1}^n c_j \lambda_j \mathbf{z}_j \right) \\
&= \sum_{i=1}^n \sum_{j=1}^n c_i \lambda_j c_j \mathbf{z}_i^T \mathbf{z}_j \\
&= \sum_{i=1}^n \sum_{j=1}^n c_i \lambda_j c_j \delta_{ij} \\
&= \sum_{i=1}^n c_i^2 \lambda_i = c_1^2 \lambda_1 + c_2^2 \lambda_2 + \cdots + c_n^2 \lambda_n \\
&= \lambda_1 [c_1^2 + c_2^2 \beta_2 + \cdots + c_n^2 \beta_n], \quad 0 < \beta_i := \frac{\lambda_i}{\lambda_1} \leq 1, \\
&= \lambda_1 [(1 - c_2^2 - \cdots - c_n^2) + c_2^2 \beta_2 + \cdots + c_n^2 \beta_n] \\
&= \lambda_1 [1 - (1 - \beta_2) c_2^2 + (1 - \beta_3) c_3^2 + \cdots + (1 - \beta_n) c_n^2] \\
&= \lambda_1 [1 - \text{some positive (or zero) numbers}].
\end{aligned}$$

- Expression is maximized when $c_2 = c_3 = \cdots = c_n = 0, \implies c_1 = 1$.
- Maximum value $\mathbf{x}^T B \mathbf{x} = \lambda_{\max}(B) = \lambda_1$.
- Similarly, can show $\min \mathbf{x}^T B \mathbf{x} = \lambda_{\min}(B) = \lambda_n$.

- So, $\|A\|^2 = \max_{\lambda} \lambda(A^T A) = \text{spectral radius of } A^T A$.

- Now,
$$\|A^{-1}\|^2 = \max_{\mathbf{x} \neq 0} \frac{\|A^{-1}\mathbf{x}\|^2}{\|\mathbf{x}\|^2}.$$

- Let $\mathbf{x} = A\mathbf{y}$:

$$\begin{aligned} \|A^{-1}\|^2 &= \max_{\mathbf{y} \neq 0} \frac{\|A^{-1}A\mathbf{y}\|^2}{\|A\mathbf{y}\|^2} = \max_{\mathbf{y} \neq 0} \frac{\|\mathbf{y}\|^2}{\|A\mathbf{y}\|^2} = \left(\min_{\mathbf{y} \neq 0} \frac{\|A\mathbf{y}\|^2}{\|\mathbf{y}\|^2} \right)^{-1} \\ &= \frac{1}{\lambda_{\min}(A^T A)}. \end{aligned}$$

- So, $\text{cond}_2(A) = \|A^{-1}\| \cdot \|A\|,$

$$\text{cond}_2(A) = \sqrt{\frac{\lambda_{\max}(A^T A)}{\lambda_{\min}(A^T A)}}.$$

Special Types of Linear Systems

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

Cholesky Factorization and SPD Matrices.

- A is SPD: $A = A^T$ and $\mathbf{x}^T A \mathbf{x} > 0$ for all $\mathbf{x} \neq 0$.
- Seek a symmetric factorization $A = \tilde{L} \tilde{L}^T$ (not LU).
 - L *not* lower triangular but not *unit* lower triangular.
 - That is, Lt_{ii} not necessarily 1.
- Alternatively, seek factorization $A = LDL^T$, where L is unit lower triangular and D is *diagonal*.

- Start with $LDL^T = A$.
- Clearly, $LU = A$ with $U = DL^T$.
 - Follows from uniqueness of LU factorization.
 - D is a *row scaling* of L^T and thus $D_{ii} = U_{ii}$.
 - A property of SPD matrices is that all pivots are positive.
 - (Another property is that you do not need to pivot.)

- Consider standard update step:

$$\begin{aligned} a_{ij} &= a_{ij} - \frac{a_{ik} a_{kj}}{a_{kk}} \\ &= a_{ij} - \frac{a_{ik} a_{jk}}{a_{kk}} \end{aligned}$$

- Usual multiplier column entries are $l_{ik} = a_{ik}/a_{kk}$.
- Usual pivot row entries are $u_{kj} = a_{kj} = a_{jk}$.
- So, if we factor $1/d_{kk} = 1/a_{kk}$ out of U , we have:

$$\begin{aligned} d_{kk}(a_{kj}/a_{kk}) &= d_{kk}l_{kj} \\ \longrightarrow U &= D(D^{-1}U) \\ &= DL^T. \end{aligned}$$

- For Cholesky, we have

$$A = LDL^T = L\sqrt{D}\sqrt{D}L^T = \tilde{L}\tilde{L}^T,$$

with $\tilde{L} = L\sqrt{D}$.

Symmetric Positive Definite Matrices

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

$$A = L L^T$$

where L is lower triangular with positive diagonal entries

- Algorithm for computing it can be derived by equating corresponding entries of A and $L L^T$
- 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
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 L overwrites original matrix 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 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

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

- 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

Cost is $O(n)$!



Tridiagonal Matrices, continued

- LU factorization of A is then given by

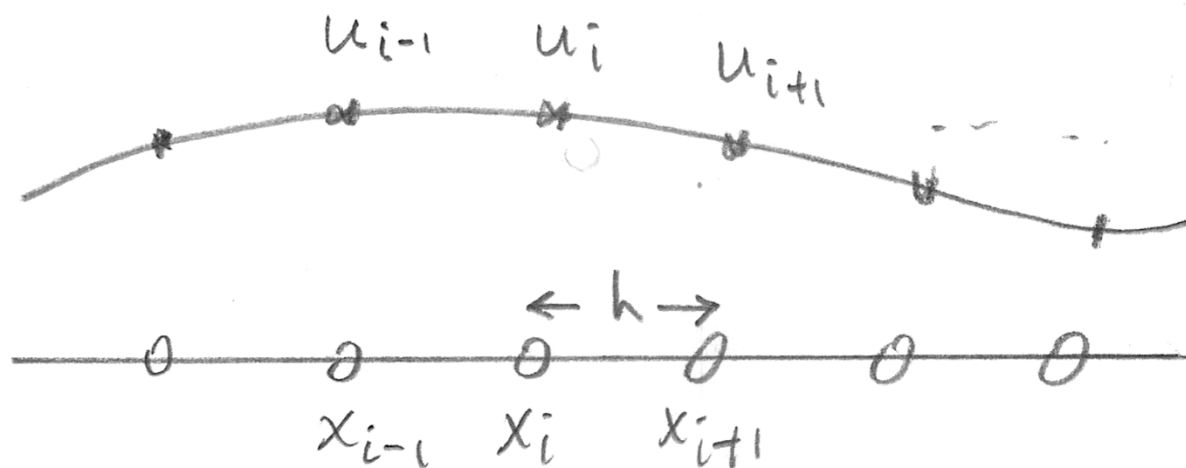
$$L = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ m_2 & 1 & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & m_{n-1} & 1 & 0 \\ 0 & \cdots & 0 & m_n & 1 \end{bmatrix}, \quad U = \begin{bmatrix} d_1 & c_1 & 0 & \cdots & 0 \\ 0 & d_2 & c_2 & \ddots & \vdots \\ \vdots & \ddots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



Example of Banded Systems

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

□ Example:



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

In Matrix Form

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

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

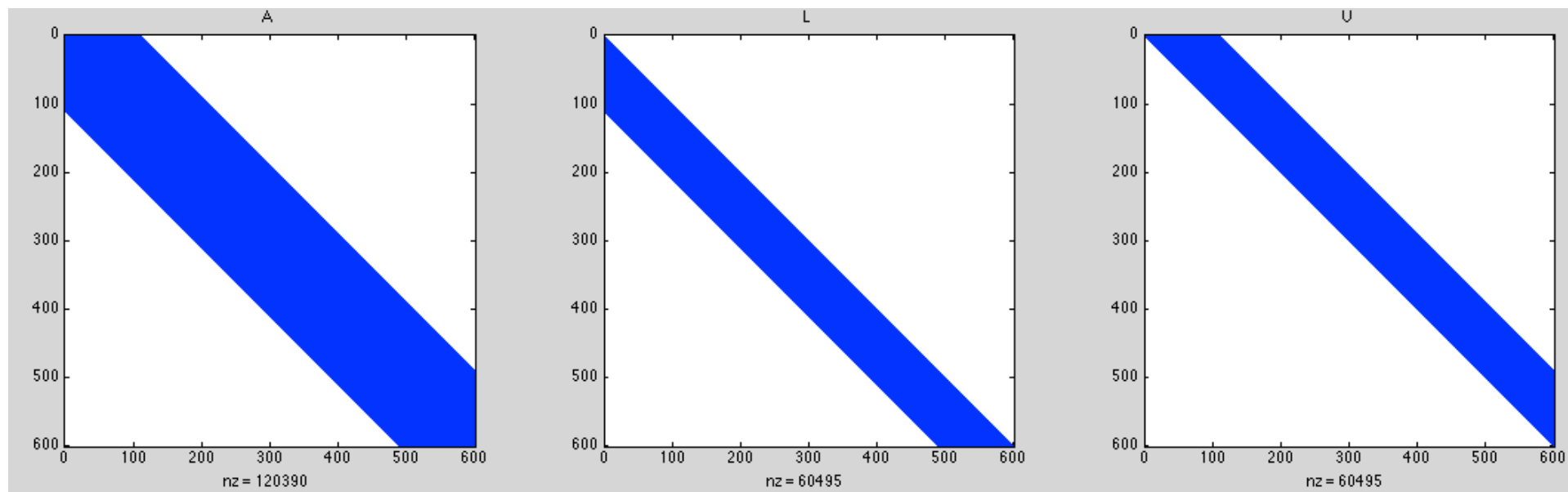
□ Banded, tridiagonal matrix (“1D Poisson Operator”)

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 $\beta \ll n$

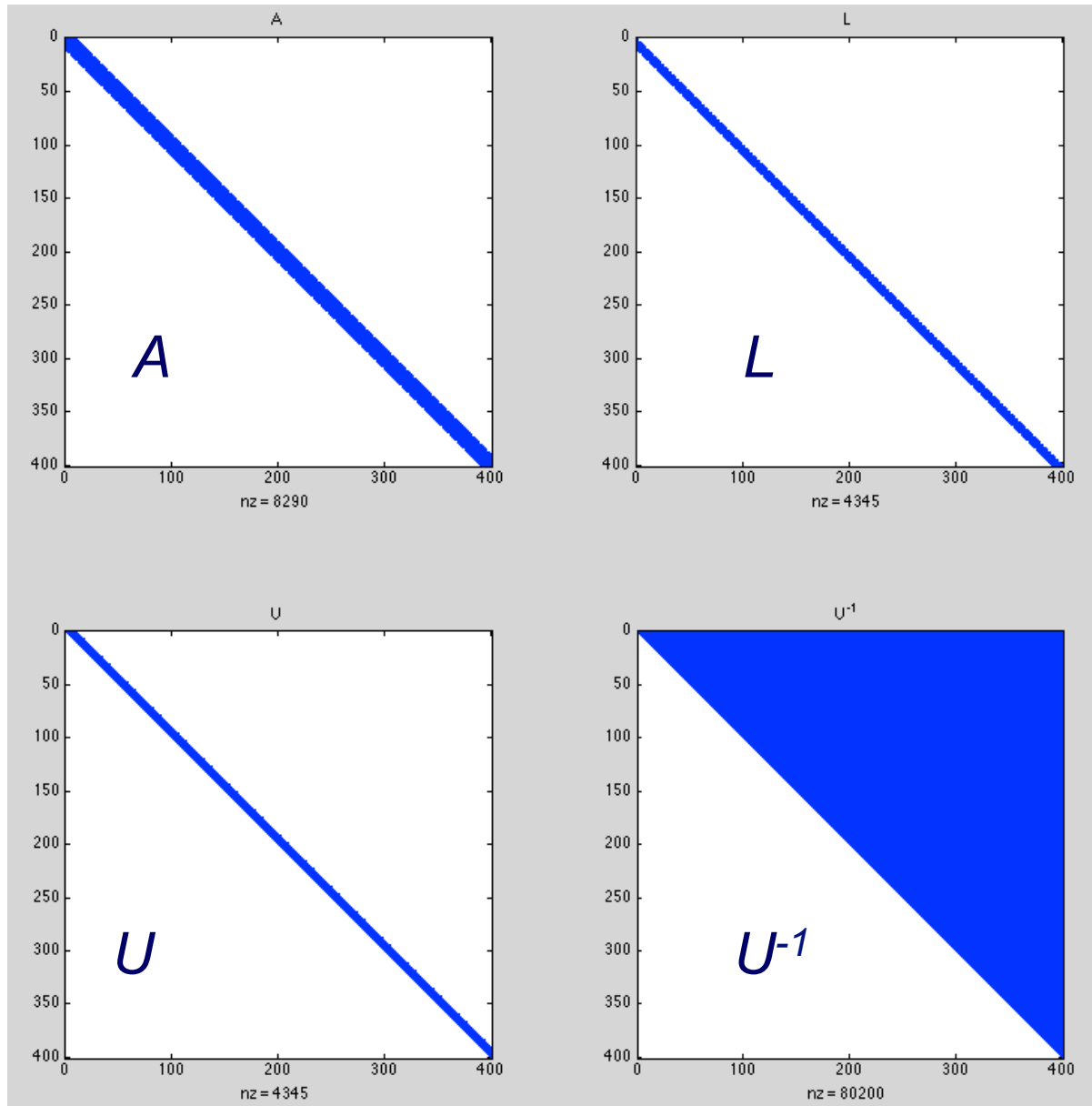


Banded Systems

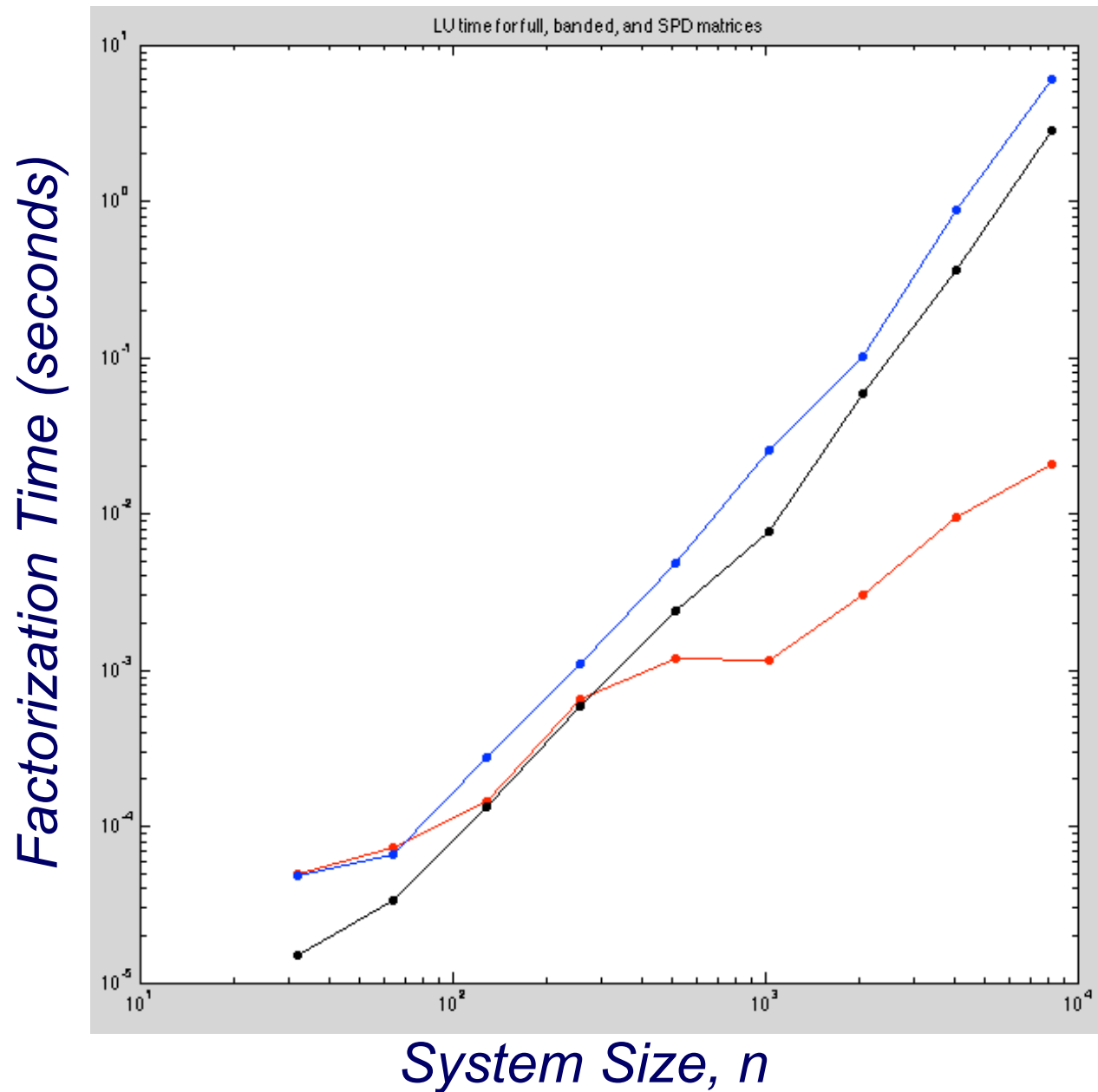


- ❑ Significant savings in storage and work if A is banded $\rightarrow a_{ij} = 0$ if $|i-j| > \beta$
- ❑ The LU factors preserve the nonzero structure of A (unless there is pivoting, in which case, the bandwidth of L can grow by at most $2x$).
- ❑ Storage / solve costs for LU is $\sim 2n \beta$
- ❑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

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



Solver Times, Banded, Cholesky (SPD), Full



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;

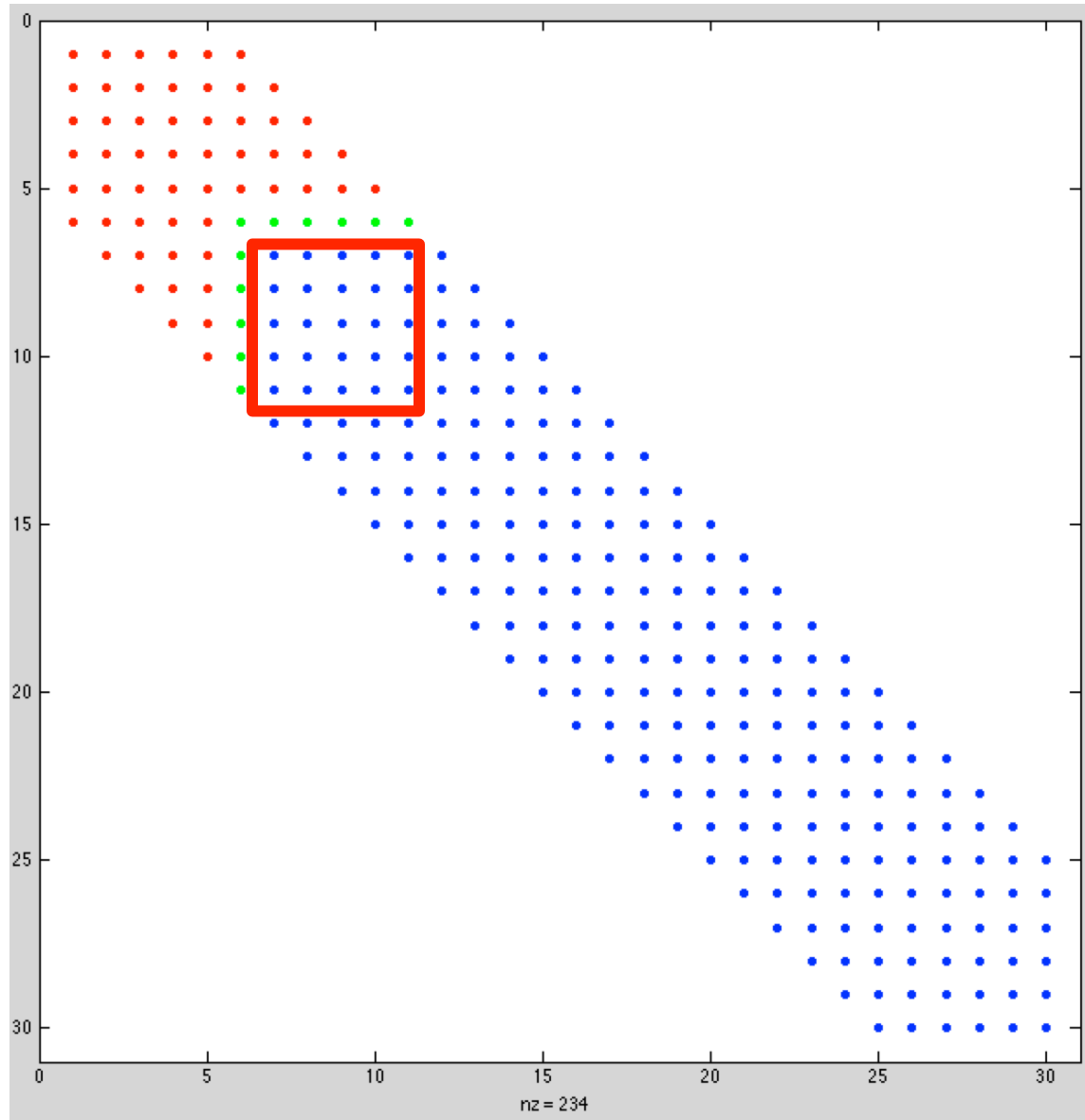
    R=9*eye(n) + rand(n,n); S=R'*R; A=spalloc(n,n,1+2*beta);
    for i=1:n; j0=max(1,i-beta);j1=min(n,i+beta);
        A(i,j0:j1)=R(i,j0:j1);
    end;

    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)]

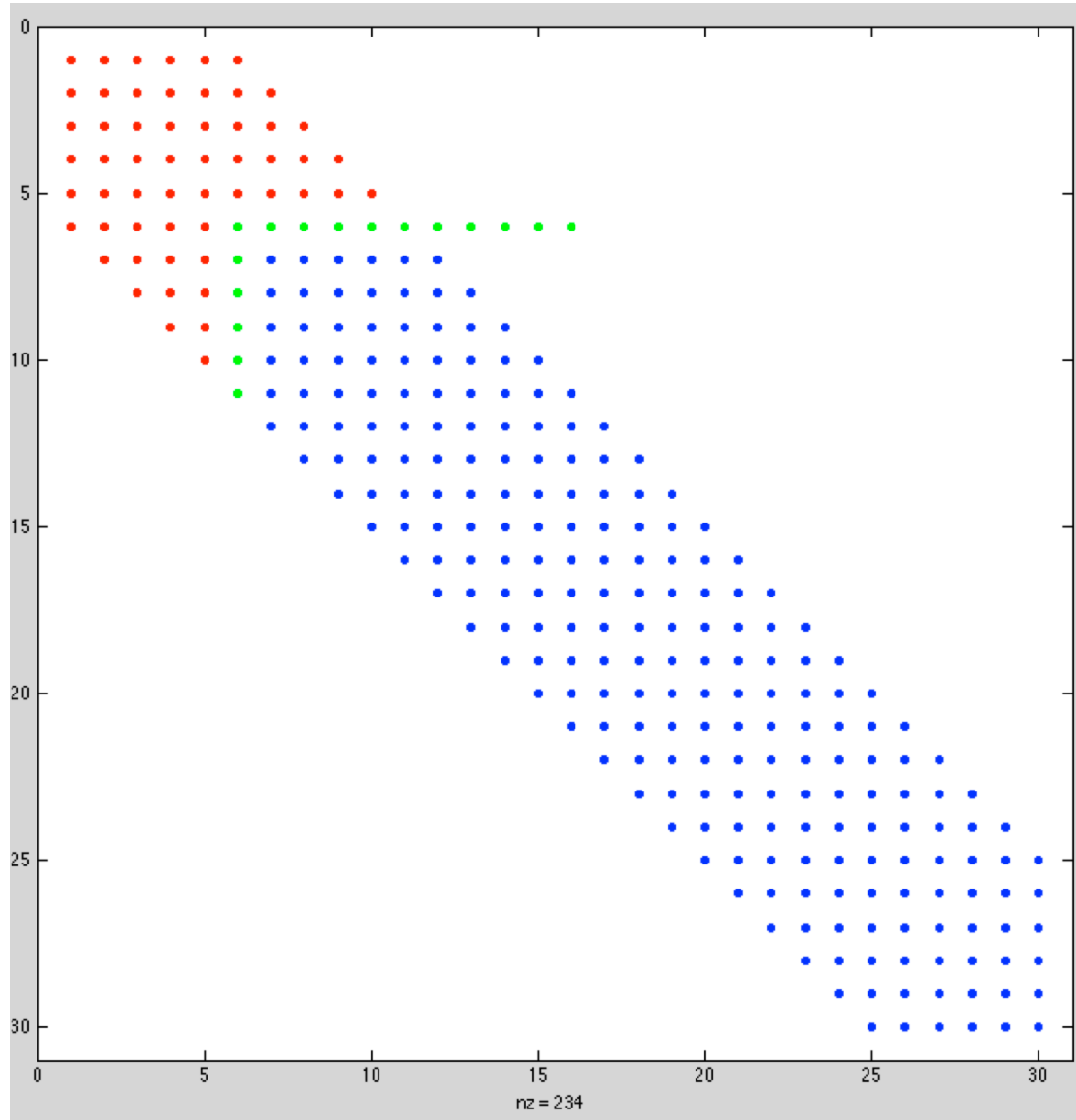
end;
loglog(nk,tsparse,'r.-',nk,tfull,'b.-',nk,tchol,'k.-')
axis square; title('LU time for full, banded, and SPD matrices')
```

Cost of Banded Factorization



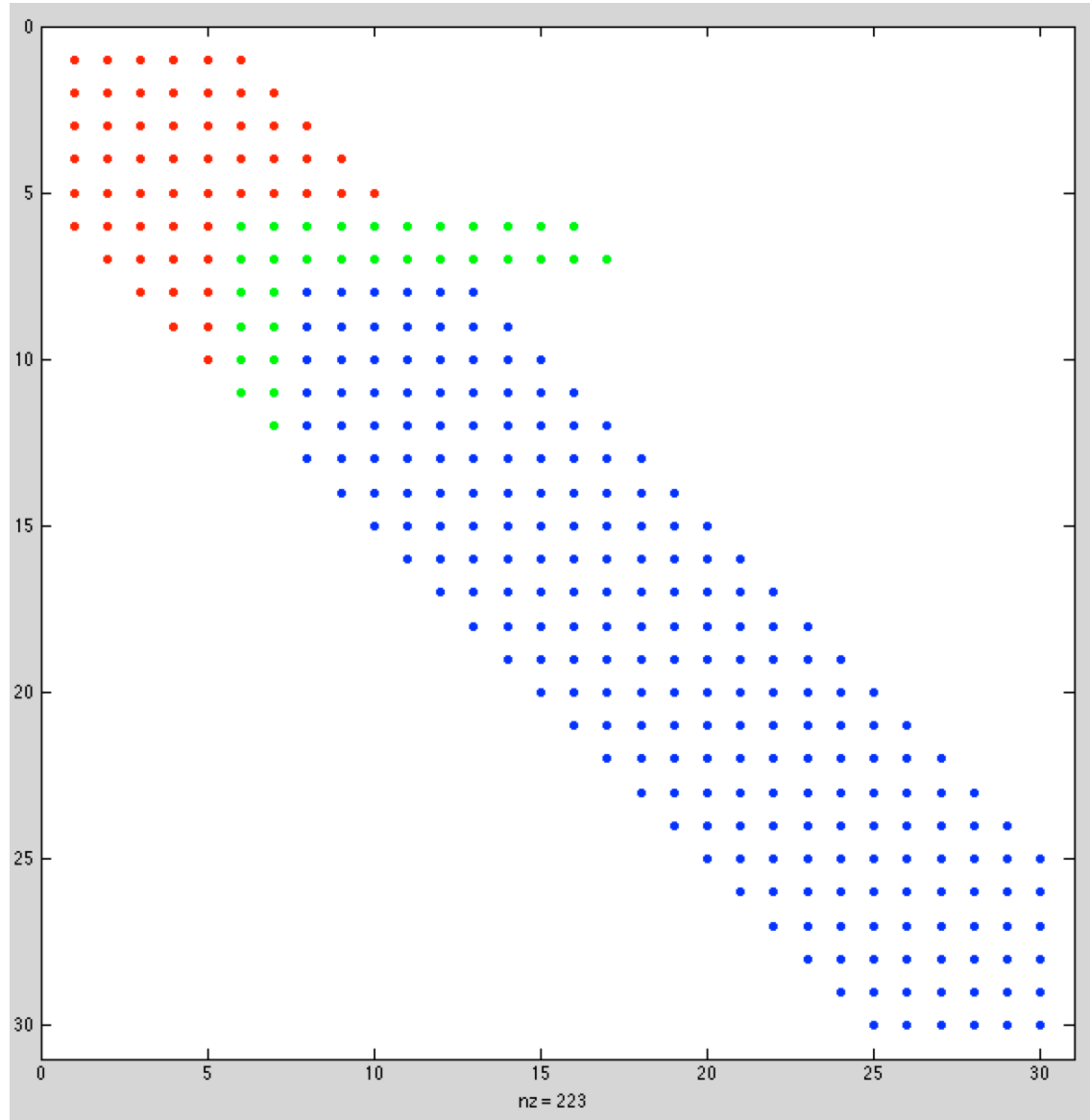
- Active submatrix for matrix with bandwidth b is $(b \times b)$.
- Work for outer product is $\mathbf{c}\mathbf{r}^T$, which is outer product of two vectors of length b .
- So, total work is $\sim n \times (b^2) \times 2$ operations to convert A into LU .
- If we have pivoting, then bandwidth of U can grow by $2x$.

Cost of Banded Factorization



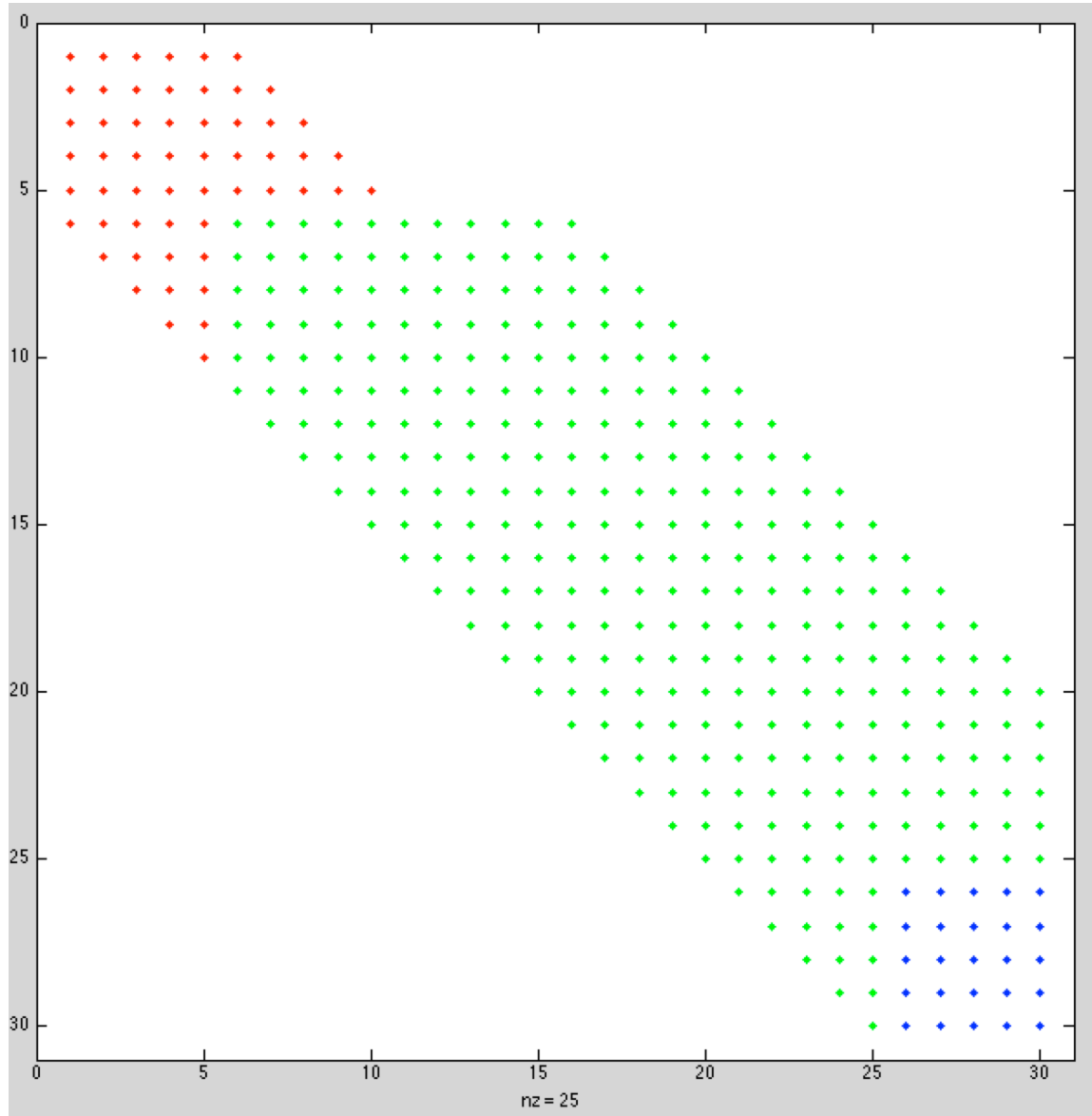
- ❑ Pivoting can pull a row that has $2b$ nonzeros to right of diagonal.
- ❑ U can end up with bandwidth $2b$.

Cost of Banded Factorization



- ❑ Pivoting can pull a row that has $2b$ nonzeros to right of diagonal.
- ❑ U can end up with bandwidth $2b$.

Cost of Banded Factorization



- ❑ Pivoting can pull a row that has $2b$ nonzeros to right of diagonal.
- ❑ U can end up with bandwidth $2b$.

pivot_gui2 demo

0.3808	0.3687	0.9319	0.7159	0	0	0	0	0	0
0	0.6074	0.8979	0.8132	0.8964	0.8443	0	0	0	0
0.0341	0.4704	-0.1058	0.5477	0.2857	-0.3972	0	0	0	0
0.4967	0.2730	-0.0850	-0.5775	-0.2447	-0.2305	0	0	0	0
0	0	0.3564	0.1630	0.1818	0.5544	0.1102	0	0	0
0	0	0	0.0605	0.1366	0.7068	0.0704	0.0576	0	0
0	0	0	0	0.4603	0.5187	0.1690	0.4586	0.1100	0
0	0	0	0	0	0.9951	0.8019	0.8349	0.8467	0.1633
0	0	0	0	0	0	0.4288	0.7628	0.8159	0.2321
0	0	0	0	0	0	0	0.2054	0.3190	0.9207

Partial pivoting

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 sdot snrm2	Scalar $\times$ vector + vector Inner product Euclidean vector norm
2	$\mathcal{O}(n^2)$	sgemv strsv sger	Matrix-vector product Triangular solution Rank-one update
3	$\mathcal{O}(n^3)$	sgemm strsm ssyrk	Matrix-matrix product Multiple triang. solutions 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 $\text{cond}(A)$ bounds our expected accuracy.
 - ❑ e.g., if $\text{cond}(A) \sim 10^5$ we expect at most 11 significant digits in $\underline{x}$.
 - ❑ Why?
 - ❑ We start with IEEE double precision – 16 digits. We lose 5 because $\text{condition}(A) \sim 10^5$, 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.

- ❑ 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 $\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$
- 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 k th position in n -vector a by transformation

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

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

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



Elementary Elimination Matrices, continued

- Matrix 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
- M_k is unit lower triangular and nonsingular
- $M_k = I - m_k e_k^T$, where $m_k = [0, \dots, 0, m_{k+1}, \dots, m_n]^T$ and e_k is k th column of identity matrix
- $M_k^{-1} = I + m_k e_k^T$, which means $M_k^{-1} =: L_k$ is same as M_k except signs of multipliers are reversed



Elementary Elimination Matrices, continued

- If M_j , $j > k$, is another elementary elimination matrix, with vector of multipliers m_j , then

$$\begin{aligned} M_k M_j &= I - m_k e_k^T - m_j e_j^T + m_k e_k^T m_j e_j^T \\ &= I - m_k e_k^T - m_j e_j^T \end{aligned}$$

which means product is essentially “union,” and similarly for product of inverses, $L_k L_j$



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

- Recall, $\underline{v} = C \underline{w} \in \text{span}\{C\}$.
- $\therefore V = (\underline{v}_1 \underline{v}_2 \dots \underline{v}_n) = C (\underline{w}_1 \underline{w}_2 \dots \underline{w}_n) \in \text{span}\{C\}$.
- If $C = \underline{c}$, i.e., C is a column vector and therefore of rank 1, then V is in $\text{span}\{C\}$ and is of rank 1.
- All columns of V are multiples of $\underline{c}$.
- Thus, $W = \underline{c} \underline{r}^T$ is an $n \times n$ 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 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
- M_k is unit lower triangular and nonsingular
- $M_k = I - m_k e_k^T$, where $m_k = [0, \dots, 0, m_{k+1}, \dots, m_n]^T$ and e_k is k th column of identity matrix
- $M_k^{-1} = I + m_k e_k^T$, which means $M_k^{-1} =: L_k$ is same as M_k except signs of multipliers are reversed



Example: Elementary Elimination Matrices

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

$$M_1 a = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

and

$$M_2 a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}$$



Example, continued

- Note that

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

and

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



Gaussian Elimination

- To reduce general linear system $Ax = b$ to upper triangular form, first choose M_1 , with a_{11} as pivot, to annihilate first column of A below first row
 - System becomes $M_1 Ax = M_1 b$, but solution is unchanged
- Next choose M_2 , using a_{22} as pivot, to annihilate second column of $M_1 A$ below second row
 - System becomes $M_2 M_1 Ax = M_2 M_1 b$, 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 $Ax = b$ to upper triangular form, first choose M_1 , with a_{11} as pivot, to annihilate first column of A below first row
 - System becomes $M_1 Ax = M_1 b$, but solution is unchanged
- Next choose M_2 , using a_{22} as pivot, to annihilate second column of $M_1 A$ below second row
 - System becomes $M_2 M_1 Ax = M_2 M_1 b$, 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

$$\begin{aligned}M_{n-1} \cdots M_1 Ax &= M_{n-1} \cdots M_1 b \\MAx &= Mb\end{aligned}$$

can be solved by back-substitution to obtain solution to original linear system $Ax = b$

- Process just described is called *Gaussian elimination*



LU Factorization

- Product $L_k L_j$ is unit lower triangular if $k < j$, so

$$L = M^{-1} = M_1^{-1} \cdots M_{n-1}^{-1} = L_1 \cdots L_{n-1}$$

is unit lower triangular

- By design, $U = MA$ is upper triangular
- So we have

$$A = LU$$

with L unit lower triangular and U upper triangular

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



LU Factorization, continued

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

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

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

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

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



Example, continued

- To annihilate subdiagonal entry of second column of $M_1 A$,

$$M_2 M_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = U,$$

$$M_2 M_1 b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} = Mb$$



Example, continued

- We have reduced original system to equivalent upper triangular system

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

which can now be solved by back-substitution to obtain

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



Example, continued

- To write out LU factorization explicitly,

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

so that

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

