1 Iterative and Multigrid Solvers

- We are concerned with solution of the $n \times n$ system,

$$A\underline{x} \;=\; \underline{b}, \tag{1}$$

where we should think of A as being *sparse* (i.e., it has $O(n)$ nonzero entries) and n as being *large* (e.g., 10^4–10^{14}).

- Ultimately, computational scientists (not developers) are generally interested in methods that are the *fastest* possible for their given application, so we will also review a variety of *direct methods* before delving into *iterative methods*, which are the topic of this course.

- **Q:** What is the complexity (i.e., cost) for solving (1) when A is tridiagonal?

  1. Do you know any systems where A is tridiagonal?
  2. How many nonzeros in A?
  3. How many nonzeros in A^{-1}?
  4. How many nonzeros in the LU factors, $A = LU$?
  5. How many nonzeros in the inverse LU factors, L^{-1}, U^{-1}?
  6. Suppose you reorder the equation numbering; do these answers change?

2 Vector/Parallel Performance

- Time-to-solution is governed not only by the number of nonzeros in the relevant system matrices, but also by the number of operations and the *order* in which values are retrived by memory.

- Poorly-structured algorithms can easily take a factor of 10 longer to execute than well-structured ones, so it's important to understand some basic issues when developing algorithms.

- We begin with a brief overview of performance considerations on high-performance architectures (like your laptop).

- We also present a few concerns for communication in large-scale parallel computing applications, as this can frequently dictate algorithmic modifications.

Direct Solvers Review

- We begin with a brief overview of direct solvers, a.k.a., Gaussian Elimination (*GE*).

- These differ from iterative solvers in that they terminate in a finite number of steps. (Technically, conjugate gradient iteration also terminates in a finite number of steps—but we rarely need to take that many steps to have a converged solution.)

- We will see that direct solvers are advantageous for systems coming from low-dimensional PDEs in $\mathbb{R}^d$ (i.e., $d = 1$), but generally not competitive for $d > 2$. For $d = 2$, the winning approach is largely determined by the condition number of the system matrix.

- Direct methods also form the basis for some preconditioning strategies known as ILU methods, which are based on incomplete LU factorizations.

- We'll start with GE for general (i.e., *dense*) matrices so that we internalize the central ideas.

- We'll then extend these to sparse systems that are the focus of this course.

Direct Solvers Outline

- Triangular solve example
- Gaussian elimination example
  - Partial pivoting
- Geometric interpretations of linear algebra
  - Row-based interpretation
  - Column-vector interpretation
- Implementations of LU factorization
  - General case
  - Banded-matrix case

Triangular Solves Example

- Upper- or lower-triangular systems are straightforward to solve.
- Consider the following upper-triangular system governing the unknown, $\underline{x} = [x_1\ x_2\ x_3]^T$.

$$
\begin{array}{rcrcrcl}
1 \cdot x_1 & + & 2 \cdot x_2 & + & 3 \cdot x_3 & = & 16 \\
 & & 4 \cdot x_2 & + & 5 \cdot x_3 & = & 14 \\
 & & & & 6 \cdot x_3 & = & 12
\end{array}
\tag{2}
$$

- To solve this, we use the well-known *backward substitution* approach of working from the bottom equation (which is trivial) up to the first equation.

- From the bottom, we have

$$x_3 \;=\; 12/6 = \; 2\;. \tag{3}$$

- Next up, we can find x_2 as

$$4 \cdot x_2 \;=\; 14 \;-\; 5 \cdot x_3 \;=\; 14 - 5 \cdot 2 \;=\; 4, \tag{4}$$

  so $x_2 = 1$.

- Finally, from the first equation, we have:

$$1 \cdot x_1 \;=\; 16 \;-\; 3 \cdot x_3 \;-\; 2 \cdot x_2 \;=\; 16 \;-\; 3 \cdot 2 \;-\; 2 \cdot 1 \;=\; 8. \tag{5}$$

- Note that we can permute the rows of this system without changing the answer:

$$\begin{array}{rcrcrcl} & & & & 6 \cdot x_3 & = & 12 \\ & & 4 \cdot x_2 & + & 5 \cdot x_3 & = & 14 \\ 1 \cdot x_1 & + & 2 \cdot x_2 & + & 3 \cdot x_3 & = & 16 \end{array} \tag{6}$$

- We can also permute the columns:

$$\begin{array}{rcrcrcl} 6 \cdot x_3 & & & & & = & 12 \\ 5 \cdot x_3 & + & 4 \cdot x_2 & & & = & 14 \\ 3 \cdot x_3 & + & 2 \cdot x_2 & + & 1 \cdot x_1 & = & 16 \end{array} \tag{7}$$

  Here, nothing has changed, save for the positions on the page.

- The equations and, hence the solution, are the same.
  The solution process follows in precisely the same way as before.

- We conclude that solving a lower-triangular system is essentially the same as solving an upper-triangular system.

  One starts with the trivial entry, computes that value and subtracts a multiple of it from the RHS for the next equation.

  This process is repeated as each unknown (x_3, x_2, etc.) becomes known.

A More General Example

- For more general systems, the convention is to effect a sequence of transformations such that the result is an equivalent *upper triangular system*.

- Because we work in finite-precision arithmetic, "equivalent" must be tempered by the expectation that there will be round-off errors.

- Good (i.e., *stable*) algorithms, however, will mitigate these round-off errors to the extent possible.

- In general, if the condition number of the system matrix is 10^k, we can expect to lose k digits of accuracy.

- For example, if we are working in FP64, we have 16 digits of accuracy in the representation of most numbers. If the condition number of the system matrix is 10^5, we can expect only 11 digits of accuracy in the final result.

- **Q:** For the same system, what accuracy should we expect if working in
  - FP32?
  - FP16?

- The transformation of a general matrix to upper triangular form is known as *Gaussian Elimination* and it is equivalent to what is known as *LU* factorization.

- Equivalence-preserving operations used in Gaussian elimination include
  - row interchanges
  - column interchanges (relatively rare; used only for "full pivoting")
  - addition of a multiple of another row to a given row

  Notice that we do not include "multiplication of a row by a constant" because, while valid for any nonzero constant, it is generally not needed for Gaussian elimination.

- We have already seen how row/column interchanges can transform a system from lower-triangular form to upper-triangular form and can understand that reversing that procedure would take us back to our targeted upper-triangular form.

- Let's now look at the row-addition process for a more general example.

- Consider the 3×3 system,

$$\begin{array}{rcrcrcl} 3\cdot x_1 & + & 9\cdot x_2 & + & 6\cdot x_3 & = & 15 \\ 2\cdot x_1 & + & 8\cdot x_2 & + & 6\cdot x_3 & = & 12 \\ 8\cdot x_1 & + & 2\cdot x_2 & + & 5\cdot x_3 & = & 18 \end{array} \tag{8}$$

- To convert this to *upper-triangular form*, we start with the following steps:
  - Leave the first equation unchanged.

  - Modify the second equation by subtracting a multiple of the first, in such a way that we **generate a zero in column 1 of the output.**

    For example,

$$\begin{array}{ll} -\frac{2}{3} & [\;3\cdot x_1 \;+\; 9\cdot x_2 \;+\; 6\cdot x_3 \;=\; 15\;] \\ + & [\;2\cdot x_1 \;+\; 8\cdot x_2 \;+\; 6\cdot x_3 \;=\; 12\;] \\ \hline \longrightarrow & \;\;0\cdot x_1 \;+\; 2\cdot x_2 \;+\; 2\cdot x_3 \;=\; 2 \end{array} \tag{9}$$

  - Do the same with the third equation, i.e., subtract from it a multiple of the first equation that will **yield a zero coefficient for the x_1 term in row 3:**

$$\begin{array}{ll} -\frac{8}{3} & [\;3\cdot x_1 \;+\; 9\cdot x_2 \;+\; 6\cdot x_3 \;=\; 15\;] \\ + & [\;8\cdot x_1 \;+\; 2\cdot x_2 \;+\; 5\cdot x_3 \;=\; 18\;] \\ \hline \longrightarrow & \;\;0\cdot x_1 \;-\; 22\cdot x_2 \;-\; 11\cdot x_3 \;=\; -22 \end{array} \tag{10}$$

- After the first round of Gaussian elimination, the system looks like

$$\begin{array}{rcrcrcr} 3\cdot x_1 & + & 9\cdot x_2 & + & 6\cdot x_3 & = & 15 \\ & & 2\cdot x_2 & + & 2\cdot x_3 & = & 2 \\ & - & 22\cdot x_2 & - & 11\cdot x_3 & = & -22 \end{array} \tag{11}$$

- Notice that the row multipliers in (9) and (10) are respectively $\frac{2}{3}$ and $\frac{8}{3}$, which correspond to the ratio of the leading coefficients in rows 2 and 3 to the leading coefficient in row 1.
  - If the leading coefficient of **row 1** (here, the *pivot row*) is 0 we clearly have a problem.
  - Even if it is just relatively small, this can cause difficulties, because we are then adding a large multiple of row 1 to each of row 2 and 3 and the information in these rows can be lost because of round-off effects.

- The primary remedy for the small-pivot scenario is simply to **reorder the rows** so that row 1 has the largest leading coefficient (in absolute value) of all rows.

- **Every row multiplier will then be of magnitude** ≤ 1 and the original row information will dominate, under the assumption that the row coefficients are initially comparable in scale.

- For example, applying this idea to the preceding case, we would first swap rows 1 and 3,

$$\begin{array}{ccccccc} 8 \cdot x_1 & + & 2 \cdot x_2 & + & 5 \cdot x_3 & = & 18 \\ 2 \cdot x_1 & + & 8 \cdot x_2 & + & 6 \cdot x_3 & = & 12 \\ 3 \cdot x_1 & + & 9 \cdot x_2 & + & 6 \cdot x_3 & = & 15 \end{array} \tag{12}$$

  and follow this with the elimination steps for row 2,

$$\begin{array}{ccccccccc} -\frac{2}{8} & [& 8 \cdot x_1 & + & 2 \cdot x_2 & + & 5 \cdot x_3 & = & 18 \;] \\ + & [& 2 \cdot x_1 & + & 8 \cdot x_2 & + & 6 \cdot x_3 & = & 12 \;] \\ \hline \longrightarrow & & 0 \cdot x_1 & + & 7\frac{1}{2} \cdot x_2 & + & 4\frac{3}{4} \cdot x_3 & = & 7\frac{1}{2} \end{array} \tag{13}$$

  and for row 3,

$$\begin{array}{ccccccccc} -\frac{3}{8} & [& 8 \cdot x_1 & + & 2 \cdot x_2 & + & 5 \cdot x_3 & = & 18 \;] \\ + & [& 3 \cdot x_1 & + & 9 \cdot x_2 & + & 6 \cdot x_3 & = & 15 \;] \\ \hline \longrightarrow & & 0 \cdot x_1 & + & 8\frac{1}{4} \cdot x_2 & + & 4\frac{1}{8} \cdot x_3 & = & 8\frac{1}{8} \end{array} \tag{14}$$

- Note that the row multipliers in (13) and (14) are now both < 1 in magnitude.

- This row-exchange idea can be performed at each round of Gaussian elimination and is referred to as *partial pivoting* or *row pivoting.*

- It is fast and provides stability in many cases. It is not needed, however, if the system matrix is *symmetric positive definite* (SPD).

- For some reason, there is a myth about the overhead associated with actually swapping rows, as opposed to simply keeping track of which row is used as the pivot-row.

  For example, Wikipedia states,

  > *Pivoting might be thought of as swapping or sorting rows or columns in a matrix, and thus it can be represented as multiplication by permutation matrices. However, algorithms* **rarely** *move the matrix elements because this would cost too much time; instead, they just keep track of the permutations.*

- In fact, **fast vector implementations** seek *unit-stride addressing* and *minimal loop clutter* (i.e., branching), which can be realized most effectively by avoiding the indirect addressing associated with index-based row-swapping.

  Indeed, LaPack's `dgetf2.f` **explicitly row swaps,** so that the active submatrix can be addressed in a *contiguous, unit-stride, fashion.*

- **Summary 1:** Explicitly row-swap within a processor.

- **Summary 2:** Do not row-swap between processors in distributed-memory (message-passing) applications.

- Coming back to our (unpivoted) system after the first round of elimination, we have

$$\begin{array}{rcrcrcr} 3\cdot x_1 & + & 9\cdot x_2 & + & 6\cdot x_3 & = & 15 \\ & & 2\cdot x_2 & + & 2\cdot x_3 & = & 2 \\ & - & 22\cdot x_2 & - & 11\cdot x_3 & = & -22 \end{array} \tag{15}$$

- To complete the process, we proceed with another round to generate a zero in column 2 of the last row.

  - Here, we leave rows 1 and 2 intact and update row 3 by subtracting a multiple of row 2 from it.

  - The update of row 3 looks like:

$$\begin{array}{lcrcrcrl} +\frac{22}{2}\,[& & 2\cdot x_2 & + & 2\cdot x_3 & = & 2 &] \quad \longleftarrow \textbf{final row 2} \\ +\quad [& - & 22\cdot x_2 & - & 11\cdot x_3 & = & -22 &] \quad \longleftarrow \textbf{old row 3} \\ \hline \longrightarrow & & 0\cdot x_2 & + & 11\cdot x_3 & = & 0 & \quad\;\; \longleftarrow \textbf{new row 3} \end{array} \tag{16}$$

  - **Q:** Could we update row 3 by substracting a multiple of row 1, instead of row 2??

- Note that here the row multiplier is $-\frac{22}{2}$, which is again > 1 in absolute value.

  Pivoting would have led to a row swap before the elimination step in which we generate a 0-coefficient for column 2, row 3.

- The final full system, in upper-triangular form, now reads

$$\begin{array}{rcrcrcr} 3\cdot x_1 & + & 9\cdot x_2 & + & 6\cdot x_3 & = & 15 \\ & & 2\cdot x_2 & + & 2\cdot x_3 & = & 2 \\ & & & & 11\cdot x_3 & = & 0 \end{array} \tag{17}$$

- Using backwards substitution we find,

$$\begin{array}{rcl} x_1 & = & 2 \\ x_2 & = & 1 \\ x_3 & = & 0 \end{array} \tag{18}$$

Matrix Factorization

- It's clear from the above exercise that we of course do not need to carry the unknowns x_j in the manipulations, which is why solution of a linear system can be expressed as a sequence of factors.
- Written as a matrix-vector product, our preceding example would read:

$$\begin{bmatrix} 3\cdot x_1 & + & 9\cdot x_2 & + & 6\cdot x_3 \\ 2\cdot x_1 & + & 8\cdot x_2 & + & 6\cdot x_3 \\ 8\cdot x_1 & + & 2\cdot x_2 & + & 5\cdot x_3 \end{bmatrix} = \begin{bmatrix} 3 & 9 & 6 \\ 2 & 8 & 6 \\ 8 & 2 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 15 \\ 12 \\ 18 \end{bmatrix}. \tag{19}$$

- In compact form, we write this equation as

$$A\underline{x} \;=\; \underline{b} \tag{20}$$

$$\tag{21}$$

with

$$A \;:=\; \begin{bmatrix} 3 & 9 & 6 \\ 2 & 8 & 6 \\ 8 & 2 & 5 \end{bmatrix}, \quad \underline{x} \;:=\; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \underline{b} \;:=\; \begin{bmatrix} 15 \\ 12 \\ 18 \end{bmatrix}. \tag{22}$$

- We will often write a matrix as a collection of column vectors, e.g.,

$$A \;:=\; [\,\underline{a}_1\ \underline{a}_2\ \underline{a}_3\,]\,, \tag{23}$$

with

$$\underline{a}_1 \;=\; \begin{bmatrix} 3 \\ 2 \\ 8 \end{bmatrix}, \quad \underline{a}_2 \;=\; \begin{bmatrix} 9 \\ 8 \\ 2 \end{bmatrix}, \quad \underline{a}_3 \;=\; \begin{bmatrix} 8 \\ 2 \\ 5 \end{bmatrix}. \tag{24}$$

- **A key idea** in linear algebra that is central to iterative methods is that *every matrix-vector product is a linear combination of the columns of that matrix.*

- Consider an $m \times n$ matrix, V. The matrix-vector product $V\underline{y}$ is

$$\underline{z} \;=\; V\underline{y} \;=\; \underline{v}_1 y_1 + \underline{v}_2 y_2 + \cdots + \underline{v}_n y_n. \tag{25}$$

- **Q:** What can we say about the vector $\underline{z}$ in the following expression?

$$\underline{z} \;=\; V(V^T A V)^{-1} V^T \underline{y} \tag{26}$$

  A: We can say that $\underline{z}$ lies in the *column space of* V, which is also known as the *range of* V, denoted as $\mathcal{R}(V)$.

  That is, $\underline{z}$ is a linear combination of the columns of V. Always.

- We explore the implications of this fact in through geometric interpretations of linear systems in the following examples.

The Geometry of Linear Equations[1]

- Example, 2×2 system:

$$\left.\begin{array}{rcl} 2x - y & = & 1 \\ x + y & = & 5 \end{array}\right\} \iff \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

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

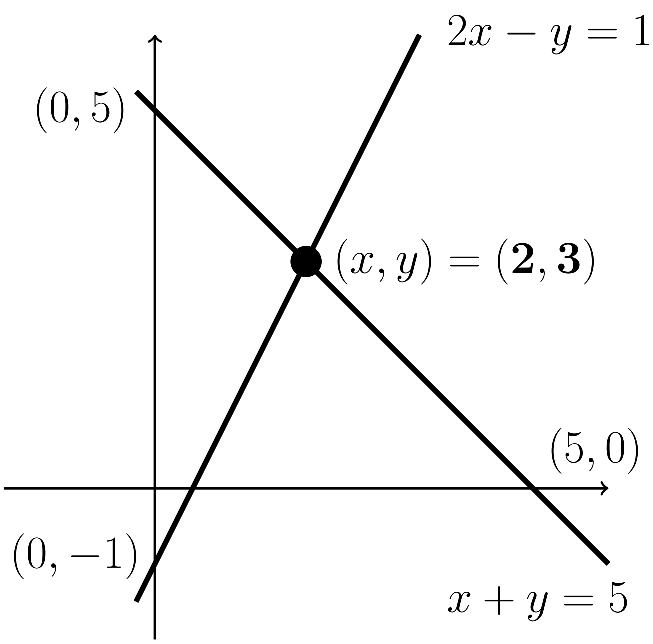
[1] Gilbert Strang: *Linear Algebra and Its Applications*

Row Form

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

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

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



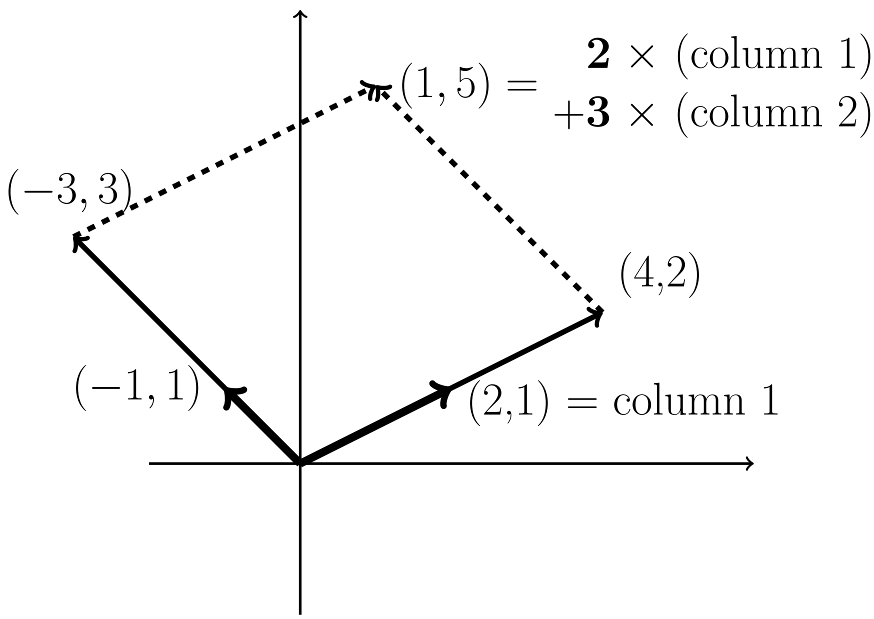
- We remark that the system is relatively *ill-conditioned* if the lines are close to being parallel, that is, if the smallest subtended angle is close to 0.

Column Form

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

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

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



- In this case, the system is *ill-conditioned* if the column vectors are nearly parallel.

  If these vectors are separated by an angle θ, it's relatively easy to show that the condition number scales as $\kappa \sim \frac{2}{\theta}$ as $\theta \longrightarrow 0$.

Row Form: A Case with n=3.

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

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

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

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

Column Vectors and Linear Combinations

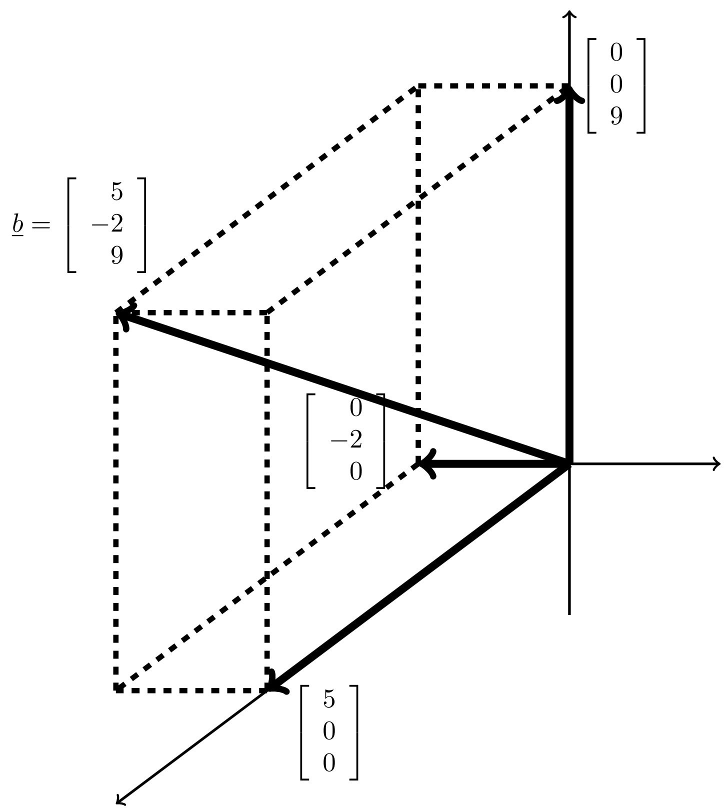
- The preceding system in $\mathbb{R}^3$ can be viewed as the vector equation

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

- Our task is to find the multipliers, u, v, and w.
- The vector $\underline{b}$ is identified with the point (5,-2,9).
- We can view $\underline{b}$ 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.

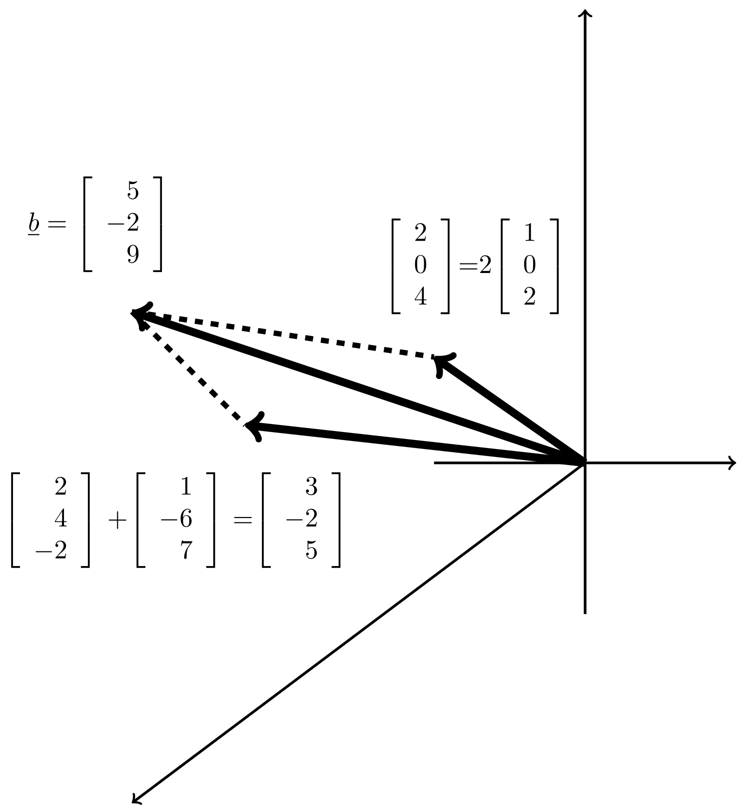
Vector Addition Example

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

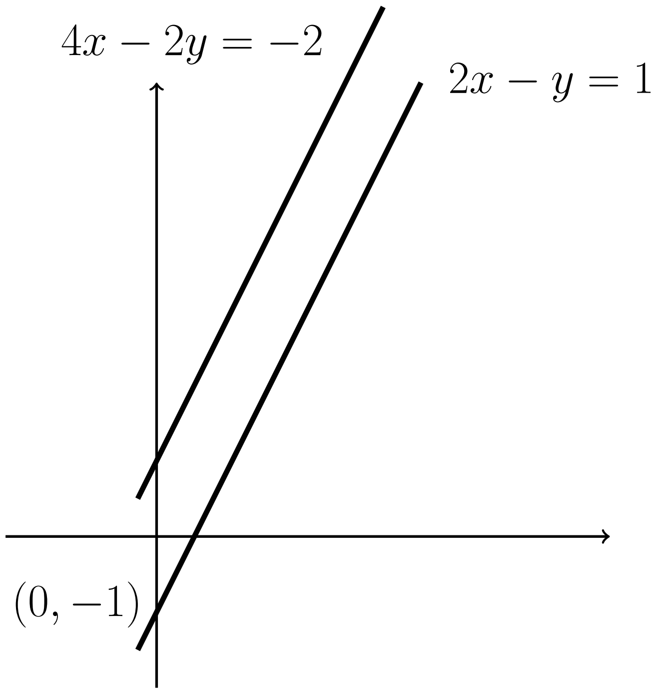


Linear Combination

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



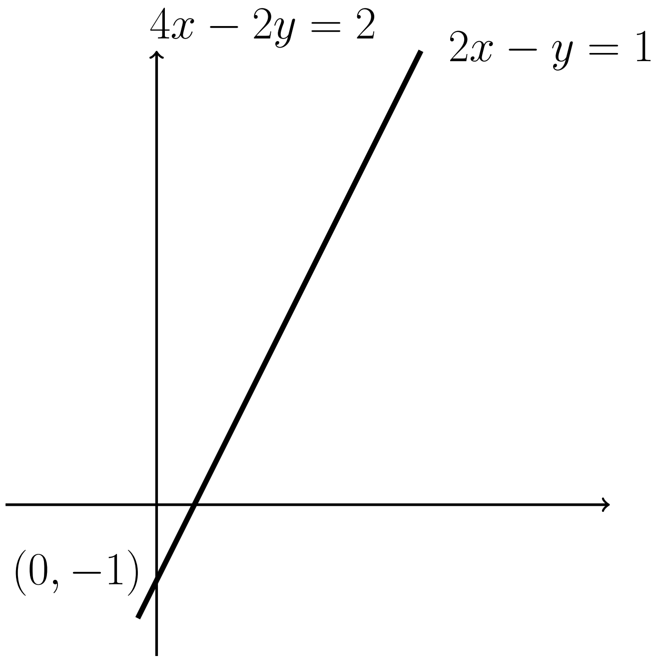
The Singular Case: Row Picture



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

- No solution.

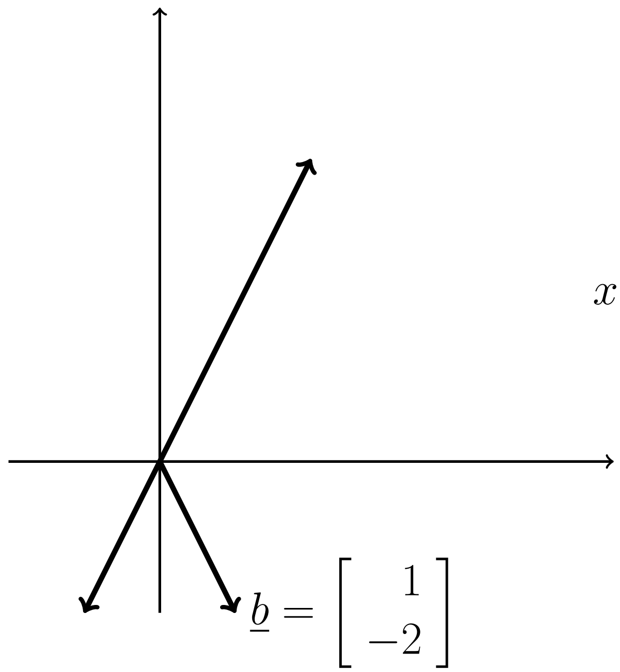
The Singular Case: Row Picture



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

- Infinite number of solutions.

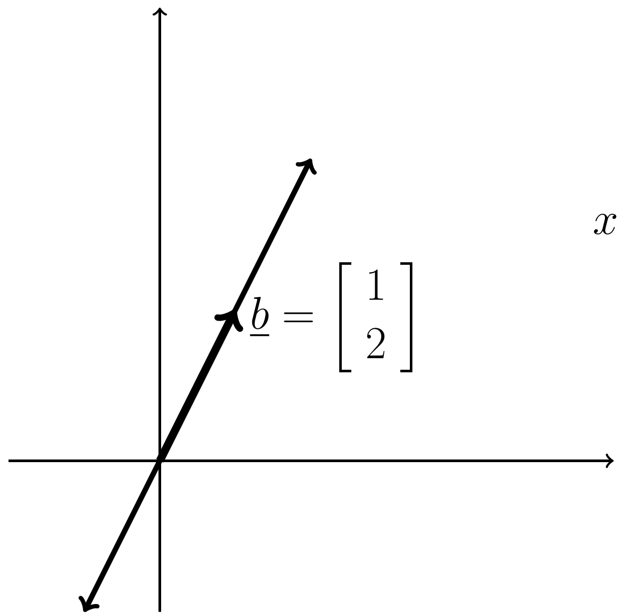
The Singular Case: Column Picture



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

- No solution.

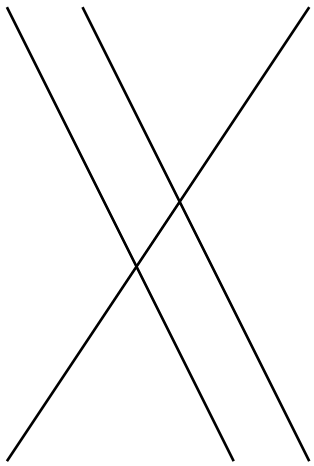
The Singular Case: Column Picture



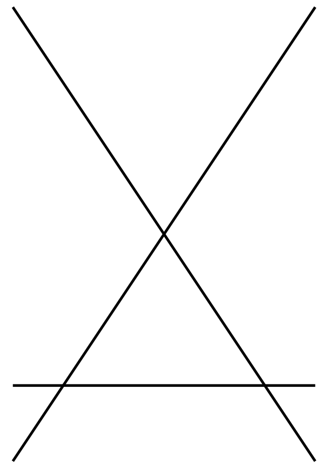
$$x \begin{bmatrix} 2 \\ 4 \end{bmatrix} + y \begin{bmatrix} -1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

- Infinite number of solutions. ($\underline{b}$ coincident with $\underline{a}_1$ and $\underline{a}_2$.)

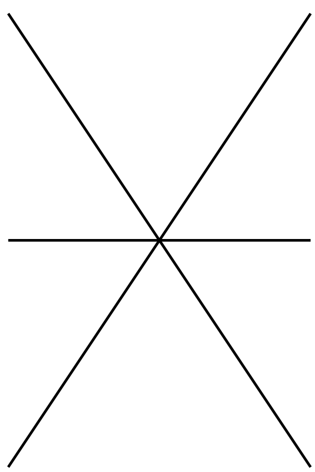
Singular Case: Row Picture with n=3



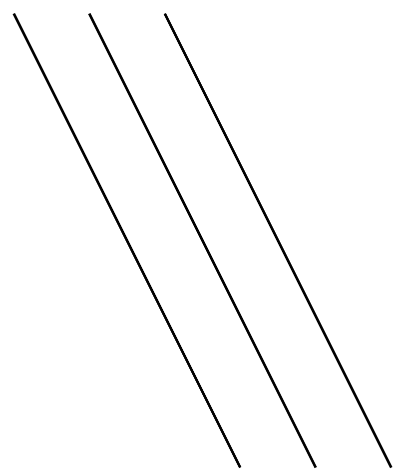
(a) two parallel planes



(b) no intersection



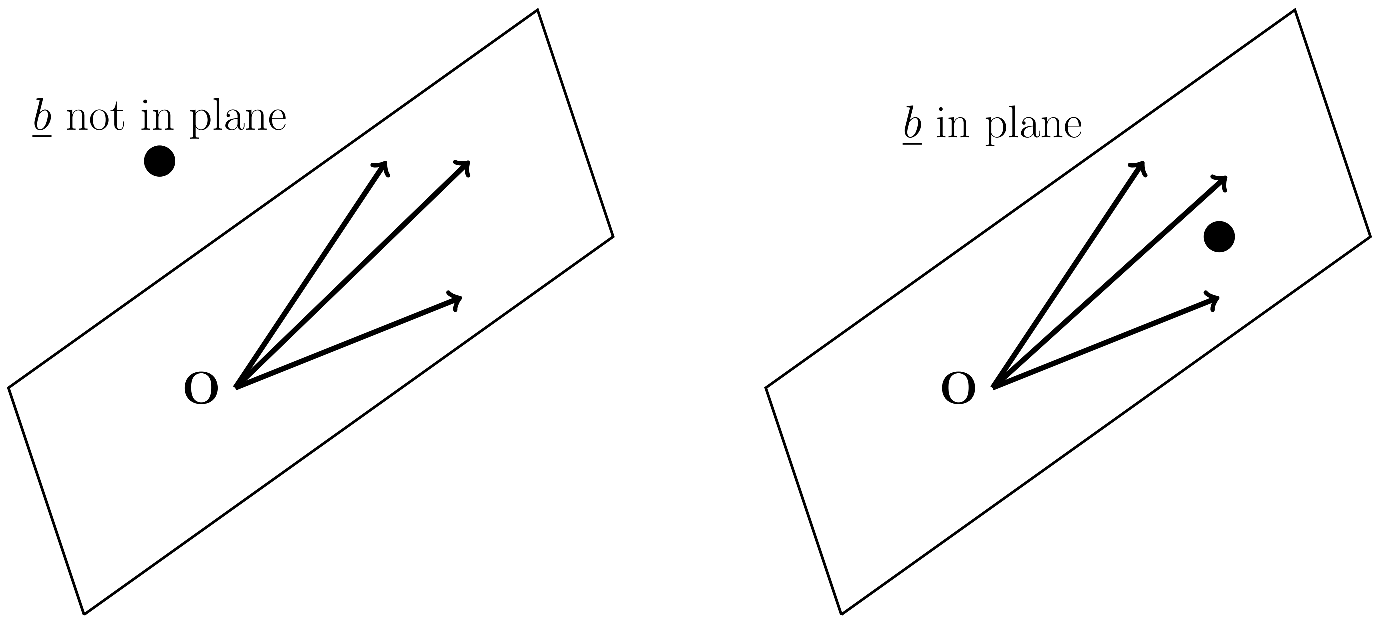
(c) line of intersection



(d) all planes parallel

End-on view of 3 planes.

Singular Case: Column Picture with n=3



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

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

- On the left, $\underline{b}$ is not in the plane $\longrightarrow$ *no solution.*
- On the right, $\underline{b}$ is in the plane $\longrightarrow$ *an inifnite number of solutions.*
- Our system is *solvable* (we can get to any point in $\mathbb{R}^3$) for **any** $\underline{b}$ if the three columns are *linearly independent.*

Matrix Form and Matrix-Vector Products.

- We start with the familiar (row) form

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

- In matrix form, this is

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

- Of course, this must equal our column form,

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

Matrix Form and Matrix-Vector Products, 2.

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

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

- Then

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

Matrix Form and Matrix-Vector Products, 3.

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

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

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

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

- **Always.**

Sigma Notation

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

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

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

- Then

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

- Components of the output,

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

Matrix Multiplication

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

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

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

Quiz Questions

1. 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?

2. Suppose $D := \text{diag}(d_{ii})$ is a diagonal matrix of the form

$$D = \begin{bmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_{nn} \end{bmatrix},$$

   and $C := DA$.

   How do the entries of C relate to those of A?

3. For the same D, how do the entries of $C = AD$ relate to those of A?

Ans. for Q2: Think of a matrix-matrix product as a sequence of matrix vector products, one each for $\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_n$. That is $DA = [D\underline{a}_1 \; D\underline{a}_2 \; \cdots \; D\underline{a}_n]$.

Matrix-Vector Products, Example.

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

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

$$\text{Then } \underline{\hat{x}} = \textbf{linear combination of the columns of } V\textbf{.}$$

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

Some Special Matrix-Vector Products, 1/2.

- Suppose $V = \underline{v}$ and $W = \underline{w}$ are $n \times 1$ matrices (i.e., vectors).
- Then

$$C \;=\; V^T W \;=\; \underline{v}^T \underline{w} \;=\; \sum_{j=1}^{n} v_j w_j \;=\; c$$

  is a 1×1 matrix (i.e., a scalar).
- We refer to $\underline{v}^T \underline{w}$ as the "dot" or *inner* product of $\underline{v}$ and $\underline{w}$.

Some Special Matrix-Vector Products, 2/2.

- Suppose $V = \underline{v}$ and $W = \underline{w}$ are $n \times 1$ matrices (i.e., vectors).
- Then

$$C \;=\; VW^T \;=\; \underline{v}\,\underline{w}^T \;=\; \underline{v}\,[\,w_1\ w_2\ \cdots\ w_n\,]$$

$$= \left[\begin{array}{cccc} \underline{v}w_1 & \underline{v}w_2 & \cdots & \underline{v}w_n \end{array}\right]$$

  is an $n \times n$ matrix, with each column a multiple of $\underline{v}$.

- We refer to $\underline{v}\,\underline{w}^T$ as the *outer* product of $\underline{v}$ and $\underline{w}$.
- It is a matrix of rank 1 and not invertible (unless $n = 1$).
  - *every column is a multiple of* $\underline{v}$
  - *every row is a multiple of* $\underline{w}^T$

- **Q:** Suppose $C = \underline{v}\,\underline{w}^T$ is an $n \times n$ matrix.

  What subset of $\mathbb{R}^n$ is reachable by the matrix-vector product $\underline{z} = C\underline{y}$?

- **A:** ?

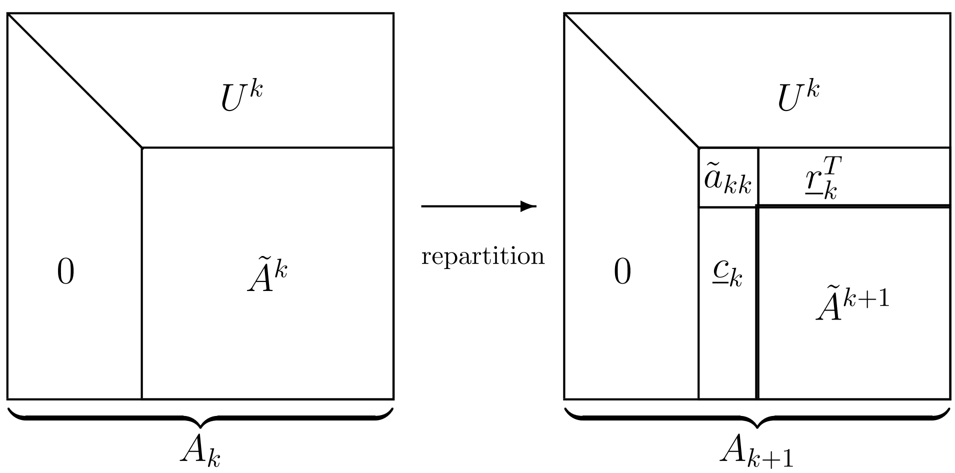
Code for the general case, without pivoting:

As written, in *row* form:

```
for k = 1 : min(m,n)
    piv = a_kk
    for i = k+1 : m
        a_ik = a_ik/piv
        for j = k+1 : n
          a_ij = a_ij - a_ik * a_kj
        end
    end
end
```

Better memory access (much faster):

```
for k = 1 : min(m,n)
    piv = a_kk
    for i = k+1 : m        % put multiplier column
        a_ik = a_ik/piv    % in lower part of A
    end
    for j = k+1 : n        % Ã^{k+1} = Ã^{k+1} - c_k r_k^T
    for i = k+1 : m
        a_ij = a_ij - a_ik * a_kj
    end
    end
end
```



- Remarkably, L is now resident in the overwritten lower part of A.
- To retrieve L and U, we use the following:

```
l = min(m,n);   L = zeros(m,l);   U = zeros(l,n);
for k = 1 : l
    L(k : end, k) = A(k : end, k);   L(k,k) = 1;
    U(k, k : end) = A(k, k : end);
end
```

Solution of Upper Triangular Systems

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

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

As written:

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

Better memory access (*faster*):

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

Solution of Lower Triangular Systems

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

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

As written:

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

Better memory access (*faster*):

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

Solution of Upper Banded Systems

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

For example, $\beta = 2$:

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

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

Solution of Upper Banded Systems

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

As written:

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

Better memory access (*faster*):

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

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

Generating Upper Triangular Systems: LU Factorization

- Example:

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

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

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

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

Generating Upper Triangular Systems: LU Factorization

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

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

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

Next Step: $k = k + 1$

- We now move to eliminate the next column, $k = 3$.

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

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

Next Step: $k = k + 1$

- After switching rows, we have

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

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

Code for the general case, without pivoting:

As derived, in *row* form:

$$
\begin{array}{l}
\text{for } k = 1 : \min(m,n) \\
\quad piv \ = \ a_{kk} \\
\quad \text{for } i = k+1 : m \\
\quad\quad a_{ik} \ = \ a_{ik}/piv \\
\quad\quad \text{for } j = k+1 : n \\
\quad\quad\quad a_{ij} \ = \ a_{ij} \ - \ a_{ik} * a_{kj} \\
\quad\quad \text{end} \\
\quad \text{end} \\
\text{end}
\end{array}
$$

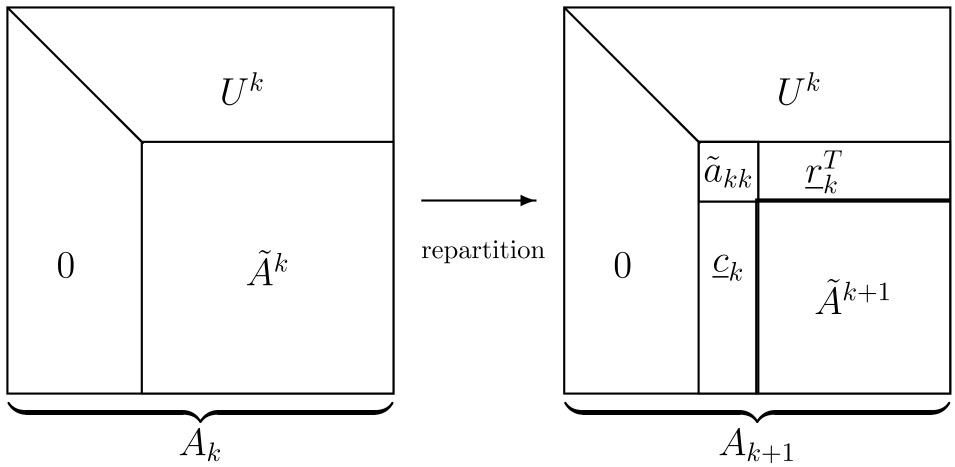
Better memory access (much faster):

$$
\begin{array}{ll}
\text{for } k = 1 : \min(m,n) & \\
\quad piv \ = \ a_{kk} & \\
\quad \text{for } i = k+1 : m & \textbf{\% put multiplier column} \\
\quad\quad a_{ik} \ = \ a_{ik}/piv & \textbf{\% in lower part of } A \\
\quad \text{end} & \\
\quad \text{for } j = k+1 : n & \% \ \tilde{A}^{k+1} = \tilde{A}^{k+1} - \underline{c}_k \, \underline{r}_k^T \\
\quad \text{for } i = k+1 : m & \\
\quad\quad a_{ij} \ = \ a_{ij} \ - \ a_{ik} * a_{kj} & \\
\quad \text{end} & \\
\quad \text{end} & \\
\text{end} &
\end{array}
$$

- Remarkably, L is now resident in the overwritten lower part of A.
- To retrieve L and U, we use the following:

$$
\begin{array}{l}
l = \min(m,n); \quad L = \text{zeros(m,l)}; \quad U = \text{zeros(l,n)}; \\
\text{for } k = 1 : l \\
\quad L(k : end, k) = A(k : end, k); \ \ L(k,k) = 1; \\
\quad U(k, k : end) = A(k, k : end); \\
\text{end}
\end{array}
$$

Illustration of Basic Update Step:



- A_k is the reduced form of A at the start of step k.
- $\tilde{A}^k$ is the active submatrix A^k starting at row k, col k.
- After identifying the

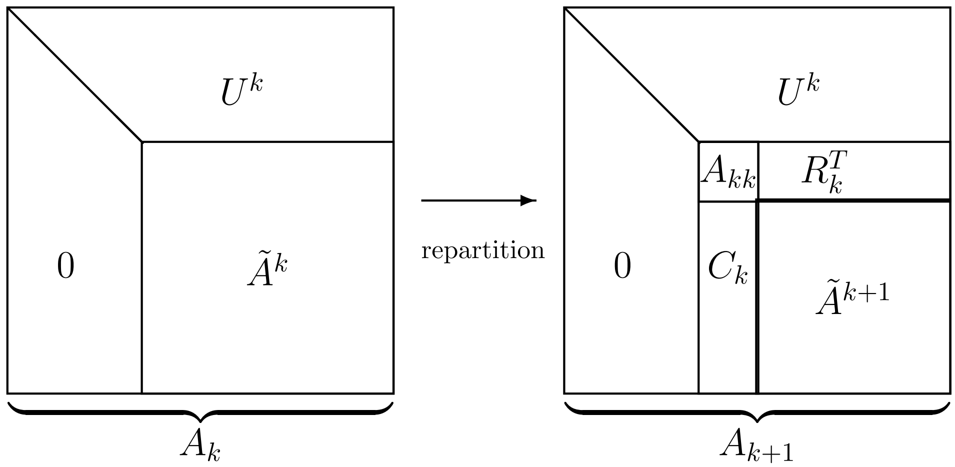
$$\begin{aligned} \text{pivot,} \quad & a_{kk} \\ \text{pivot row,} \quad & \underline{r}_k^T = a_{k:}, \text{ and} \\ \text{multiplier column,} \quad & \underline{c}_k = a_{:k}/a_{kk}, \end{aligned}$$

  the rank-one update step reads:

$$\tilde{A}^{k+1} \; = \; \tilde{A}^{k+1} \; - \; \underline{c}_k \, \underline{r}_k^T.$$

- The memory footprint of each successive submatrix is $(n-1)^2$, $(n-2)^2$, $\ldots$ 1.
- This matrix must be pulled into cache $n-1$ times.
- The total number of memory references (of *non-cached* data) is $\approx \frac{1}{3}n^3$, and the total work $\approx \frac{2}{3}n^3$ ops (one "+" and "*" for each submatrix entry).
- Recall that non-cached memory accesses slow ($\approx 20\times$) compared to an `fma`.
- This observation suggests the idea of **block factorizations** that exploit `BLAS3` matrix-matrix products.
- This is the essential difference between LinPack and LaPack, with the latter being about $20\times$ faster.

Illustration of Block-Update:



- Here, A_{kk} is a $b \times b$ block, where $b \approx 64$ is the block size.
- In this case, the update step is

$$\tilde{A}^{k+1} \;=\; \tilde{A}^{k+1} \;-\; C_k\, A_{kk}^{-1} R_k^T.$$

- Since $A_{kk}^{-1} \;=\; (L_{kk}\, U_{kk})^{-1} \;=\; U_{kk}^{-1}\, L_{kk}^{-1}$, we can rewrite the update step as

$$\begin{aligned} R_k^T \;&=\; L_{kk}^{-1}\, R_k^T \\ C_k \;&=\; C_k U_{kk}^{-1} \\ \tilde{A}^{k+1} \;&=\; \tilde{A}^{k+1} \;-\; C_k\, R_k^T. \end{aligned}$$

- The advantage of the block strategy is that we reduce by a factor of b the number of times that $\tilde{A}^{k+1}$ is dragged into cache from main memory and that the principal work, computation of $C_k\, R_k^T$, is cast as a fast matrix-matrix product.

Matlab Code for LU, with and without Blocking:

```
function [L,U]=plu(A);

%  Unpivoted LU factorization

m=size(A,1);
n=size(A,2);
K=min(m,n);

U=A(1:K,:);
L=zeros(m,K);

for k=1:K;

   piv=U(k,k);              %% pivot
   row=U(k,k:end)';         %% pivot row
   col=U(k+1:end,k)/piv;    %% multiplier column

   U(k+1:end,k:end) = U(k+1:end,k:end)-col*row';

   L(k+1:end,k)     = col;
   L(k,k)           = 1;

end;
```

```
function [L,U]=blu(A,b);

%  Unpivoted Block-LU factorization
%  Blocksize = b

m=size(A,1);
n=size(A,2);
K=min(m,n);

U=A;
L=0*A;

for k=1:b:K; l=k+b-1; l=min(l,K);

   P=U(k:l,k:l);       [PL,PU] = plu(P); %% pivot
   R=U(k:l,k+b:end);  R=PL\R;           %% pivot row
   C=U(k+b:end,k:l);  C=C/PU;           %% multiplier column

   U(k+b:end,k+b:end)  = U(k+b:end,k+b:end) - C*R;

   U(k:l,k+b:end) = R;  U(k:l,k:l) = PU; U(k+b:end,k:l)=0;
   L(k+b:end,k:l) = C;  L(k:l,k:l) = PL;

end;
```

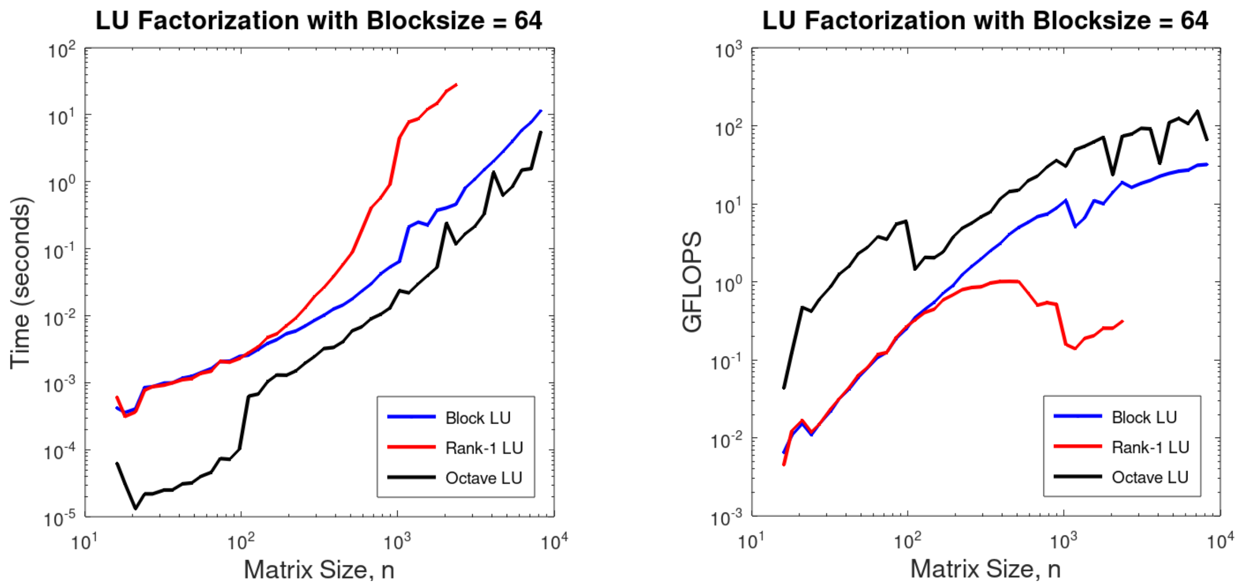


Figure 1: Time and GFLOPS for unblocked rank-1-based LU factorization (red) and blocked LU factorization with blockize $b = 64$ (blue) vs. matrix size, n. For large n, there is a $40\times$ difference in performance between Block-LU and Rank-1 LU. The default Octave LU gains another factor of 5 for large n, and a factor of 70 for $n < 100$. The results show that the dense-matrix factor times for $n = 8192$ are about 6 seconds for Octave when using multiple cores on an M1-based Macbook Pro.

- Importantly, the number of operations is $b(n-k)^2$ `fma`'s for the work-intensive matrix-matrix products, while the number of memory references is only $(n-k)^2$, which yields a b-fold increase in *computational intensity* (the ratio of flops to bytes).

- For this reason, LU factorization of large matrices can often realize close to the theoretical peak performance of a machine.

  (Some argue that this so-called Linpack performance number, which is used to score the machines in the Top 500 list, is inflated and artificial. Personally, I view it as an existence proof. The counter-argument is that vendors focus solely on the Linpack benchmark to the detriment of real applications.)

Iterative Solvers (Matrix Norm Example)

- Solve $A\underline{x} = \underline{b}$.
- Consider the following *fixed-point iteration*:
  - *Initial guess:* $\underline{x}_0 = 0$
  - $\underline{x}_{k+1} = \underline{x}_k + (\underline{b} - A\underline{x}_k)$, $k = 0, 1, \ldots, k_{\max}$.
- **Cost:**
  - $[A\underline{x}_k] = 2n^2$ if A is ***full***
  - Total cost $\sim 2n^2 \times$ number of iterations.

  - $[A\underline{x}_k] = O(n)$ if A is ***sparse*** (number of nonzeros per row $< c$, for c a constant independent of n)
  - Total cost $\sim O(n) \times$ number of iterations.

- How many iterations for $||\underline{e}_k|| := ||\underline{x} - \underline{x}_k|| \leq$tol $(= 10^{-8}$, say)?

Iterative Solvers (Matrix Norm Example)

- **Example:**

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{10} \\ \\ \frac{1}{10} & \frac{1}{2} \end{bmatrix}, \qquad \underline{b} = \begin{bmatrix} \frac{4}{5} \\ \\ \frac{8}{5} \end{bmatrix},$$

- **Iteration:**

$$\begin{aligned} \underline{x}_1 &= \mathbf{0} + \underline{b} - A\mathbf{0} \\ \underline{x}_2 &= \underline{x}_1 + \underline{b} - A\underline{x}_1 \\ &\vdots \\ \underline{x}_{k+1} &= \underline{x}_k + \underline{b} - A\underline{x}_k. \end{aligned}$$

- matlab demo: *iter_demo_22a.m*

$$\underline{x} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}.$$

For each iteration k,

```
k          xk         ek
0        0.0000     1.0000
         0.0000     3.0000

1        0.8000     0.2000
         1.6000     1.4000

2        1.0400    -0.0400
         2.3200     0.6800

3        1.0880    -0.0880
         2.6560     0.3440

4        1.0784    -0.0784
         2.8192     0.1808

5        1.0573    -0.0573
         2.9018     0.0982

6        1.0385    -0.0385
         2.9452     0.0548

7        1.0247    -0.0247
         2.9687     0.0313

8        1.0155    -0.0155
         2.9819     0.0181
```

9	1.0096	-0.0096
	2.9894	0.0106

Note on Row Scaling / Permutation

$$\begin{aligned}
D\underline{v} &= \text{ scale rows of } \underline{v} \\
P\underline{v} &= \text{ permute rows of } \underline{v} \\
DA &= [\, D\underline{a}_1 \, D\underline{a}_2 \, \cdots \, D\underline{a}_n \,] \; = \; \text{ scale rows of } A \\
PA &= [\, P\underline{a}_1 \, P\underline{a}_2 \, \cdots \, P\underline{a}_n \,] \; = \; \text{ permute rows of } A
\end{aligned}$$

$$\begin{bmatrix} 2 & & \\ & 3 & \\ & & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix}$$

$$\begin{bmatrix} & & 1 \\ 1 & & \\ & 1 & \end{bmatrix} \begin{bmatrix} 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 4 & 4 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

Note on Column Scaling / Permutation

$$AD = [\, d_1\underline{a}_1 \; d_2\underline{a}_2 \; \cdots \; d_n\underline{a}_n \,] = \text{ scale columns of } A$$

$$AP = \left[\, \underline{a}_{p_1} \; \underline{a}_{p_2} \; \cdots \; \underline{a}_{p_n} \,\right] = \text{ permute columns of } A$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & & \\ & 3 & \\ & & 4 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} & 1 & \\ & & 1 \\ 1 & & \end{bmatrix} = \begin{bmatrix} 4 & 2 & 3 \\ 4 & 2 & 3 \\ 4 & 2 & 3 \end{bmatrix}$$

System Modification by Permutations

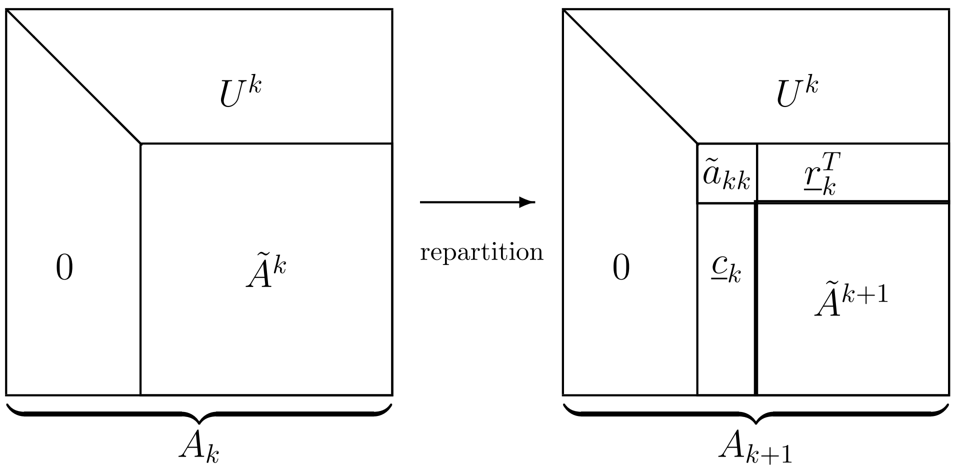
$$
\begin{aligned}
P\,A\,\underline{x} &= P\,\underline{b} \qquad \text{Row Permutation} \\
\longrightarrow A'\,\underline{x} &= \underline{b}'
\end{aligned}
$$

$$
\begin{aligned}
A\,P\,P^T\underline{x} &= \underline{b} \qquad \text{Column Permutation} \\
\longrightarrow A'\,\underline{x}' &= \underline{b}
\end{aligned}
$$

System Modification by Permutations

$$\underbrace{P\,A}_{A'}\,\underline{x} \;=\; P\,\underline{b} \;=\; \underline{b}' \qquad \text{Row Permutation}$$

$$\underbrace{A\,P}_{A'}\,\underbrace{P^T\underline{x}}_{\underline{x}'} \;=\; \underline{b} \qquad \text{Column Permutation}$$



U^k
0
$\tilde{A}^k$
A_k
repartition
U^k
$\tilde{a}_{kk}$
$\underline{r}_k^T$
0
$\underline{c}_k$
$\tilde{A}^{k+1}$
A_{k+1}

Gaussian Elimination as a Sequence of Matrix-Matrix Products

$$
\begin{aligned}
A^{(0)} &:= A \\
A^{(1)} &:= M_1 A^{(0)} \\
A^{(k)} &:= M_k A^{(k-1)} \;=\; M_k \cdots M_1 A \\
&\;\;\vdots \\
A^{(n)} &:= U \text{ upper triangular} \\
&= L^{-1} A \;\;\Longleftrightarrow\;\; LU \;=\; A
\end{aligned}
$$

- LU factorization and Gaussian elimination are equivalent.
- We view the solution process for solving $A\underline{x} = \underline{b}$ in two steps:
  - **Factorization:** $A \longrightarrow LU$
  - **Solve:** $L\,\underline{y} = \underline{b}$, followed by
    $U\underline{x} = \underline{y}$.