# module 05
# solving $Ax = b$: linear system of equations

**ahmad f. taha**

**ce 2989 — numerical methods in civil & env. engineering**

*email:* ahmad.taha@vanderbilt.edu

*webpage:* http://lab.vanderbilt.edu/taha


March 2, 2026

# module outline

this module covers methods to solve linear systems of equation

$$f(x) = Ax - b = 0 \qquad \text{or} \qquad Ax = b \qquad x \in \mathbb{R}^n$$

- gaussian elimination
- lu decomposition
- gauss-jordan
- matrix inversion
- gauss-seidel, sor, jacobi

# linear systems of equations $Ax = b$

- this module is about solving $Ax = b$ where $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$
- so we have $m$ equations, $n$ unknowns: sometimes $m >> n$ (overdetermined) or $n >> m$ (underdetermined) or $n = m$
- **example:** find a parabola that goes through the points $(x, y) = \{(1, 1), (2, 2), (3, 5)\}$
- we first focus on systems of $n$ equations $n$ unknowns $Ax = b$
- this can be solved via writing $x = A^{-1}b$ but inverse computation is difficult

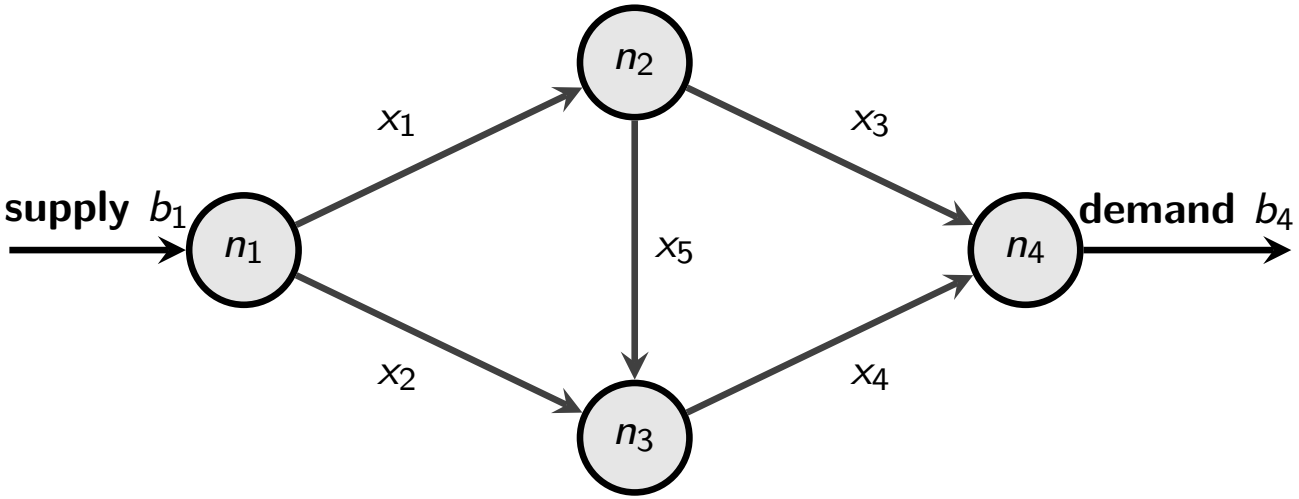
## matrix inverse theorem

for a matrix $A \in \mathbb{R}^{n \times n}$, the following are all equivalent:

1. $A$ is an invertible matrix
2. columns or rows of $A$ are linearly independent
3. $Ax = 0$ has only the trivial solution $x = 0$
4. $Ax = b$ has unique solution for each $b \in \mathbb{R}^n$
5. the number 0 is **not** an evalue of $A$
6. $\det(A) \neq 0$
7. $A^\top$ is invertible and $(A^\top)^{-1} = (A^{-1})^\top$

# solving $Ax = b$: static water flow networks

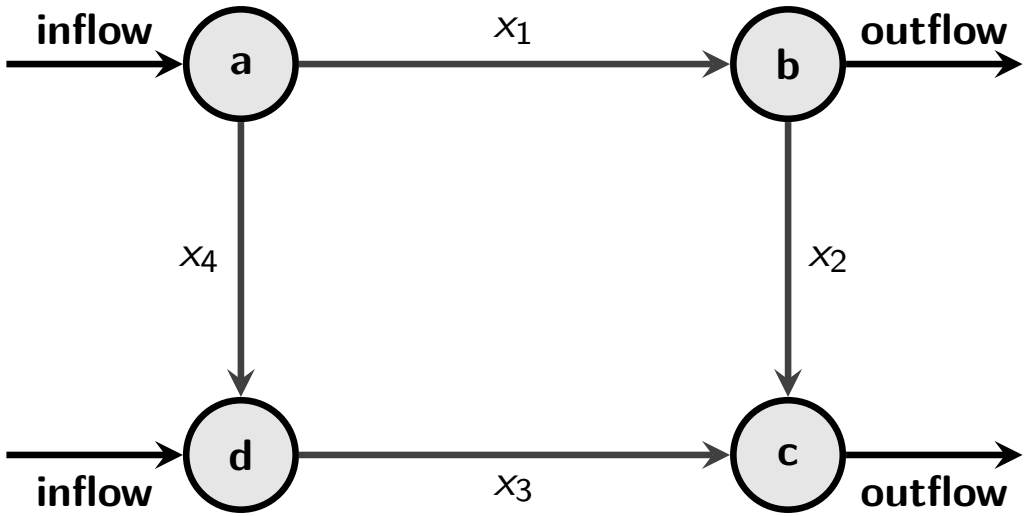
- **concept:** mass balance at pipe junctions (flow in = flow out)
- **small network:** simple neighborhood loop. easily solvable by hand to find internal pipe flows $x$




- **big network:** municipal distribution (e.g., the nashville metro water grid)
- requires assembling a massive, sparse matrix $A$ for thousands of junctions

# solving $Ax = b$: static traffic flow

- **concept:** flow conservation at intersections (vehicles entering = vehicles leaving)
- **small network:** a single urban city block with one-way streets




- **big network:** urban transportation network modeling
- tracking hundreds of thousands of steady-state daily trips relies heavily on iterative linear algebra methods nested inside larger non-linear formulations

# outline of methods to solve $Ax = b$

many, many methods to solve $Ax = b$: the most solved class of math problems ever?

- graphical methods
- finite-step methods (tolerance-free, exact, terminate after fixed number of steps) for small $n$:
  - cramer's rule
  - method of elimination
  - naive gauss
  - gauss-jordan
  - lu decomposition
  - ...
- iterative methods with tolerance-defined stopping criteria, works well for large $n$

# graphical methods and cramer's rule

- graphical methods: basically find the point intersection of lines for $n = 2$ or planes for $n = 3$
- for $n > 3$, graphical method is not applicable
- **cramer's rule:** for any linear system $Ax = b$ with $n$ equations, $n$ unknowns, the solution can be computed as:

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where $A_i$ is the matrix formed out of replacing the $i$-th column of $A$ by vector $b$

- examples: solutions for the generic 2by2 and 3by3 $Ax = b$ given as follows:

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \Rightarrow x_1 = \frac{\det\left(\begin{bmatrix} b_1 & a_2 \\ b_2 & a_4 \end{bmatrix}\right)}{a_1 a_4 - a_2 a_3} = \frac{b_1 a_4 - b_2 a_2}{a_1 a_4 - a_2 a_3}, x_2 = \dots$$

- this method is computationally inefficient for $n > 3$
- it requires computing $n + 1$ determinants, while we will see many other methods only require computing one determinant for the n-by-n matrix $A$
- numerically unstable too so this method is useless

# (naive) gauss method

- gauss elimination, named after carl fiiedrich gauss
- so anyway: what's naive gauss? simple method that has two steps:
  1. forward elimination of unknowns (computationally heavy)
  2. back substitution (computationally lightweight)

$$\left[\begin{array}{ccc:c} a_{11} & a_{12} & a_{13} & c_1 \\ a_{21} & a_{22} & a_{23} & c_2 \\ a_{31} & a_{32} & a_{33} & c_3 \end{array}\right] \Downarrow \left[\begin{array}{ccc:c} a_{11} & a_{12} & a_{13} & c_1 \\ & a'_{22} & a'_{23} & c'_2 \\ & & a''_{33} & c''_3 \end{array}\right] \Downarrow \begin{array}{c} x_3 = c''_3/a''_{33} \\ x_2 = (c'_2 - a'_{23}x_3)/a'_{22} \\ x_1 = (c_1 - a_{12}x_2 - a_{13}x_3)/a_{11} \end{array}$$

Forward elimination

Back substitution

## naive gauss + example

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n &= b'_2 \\ a''_{33}x_3 + \cdots + a''_{3n}x_n &= b''_3 \\ \ddots \quad \vdots \quad &\vdots \\ a^{(n-1)}_{nn}x_n &= b^{(n-1)}_n \end{aligned}$$

- eventually, we get the above system of equations with updated $a'_{ij}$'s and $b'_j$'s via forward elimination
- then we apply back substitution and compute $x_i$'s for the now-upper triangular system of equations $Ux = b'$ which results in this solution

$$x_n = \frac{b^{(n-1)}_n}{a^{(n-1)}_{nn}}, \quad x_i = \frac{b^{(i-1)}_i - \sum_{j=i+1}^{n} a^{(i-1)}_{ij} x_j}{a^{(i-1)}_{ii}}$$

- solve this system of equations via naive gauss method

$$2x_1 + x_2 + 3x_3 = 1,\ 4x_1 + 4x_2 + 7x_3 = 1,\ 2x_1 + 5x_2 + 9x_3 = 3$$

# problems with naive gauss

- division by zero-pivot: if $a_{11} = 0$, then the first elimination doesn't work
- round-off errors are huuuge
- the method also performs poorly for ill-conditioned matrices with big $\kappa(A)$
  - **example:** solve this system of equations

$$x_1 + 2x_2 = 10, \quad 1.1x_1 + 2x_2 = 10.4, \qquad A = \begin{bmatrix} 1 & 2 \\ 1.1 & 2 \end{bmatrix}$$

  - compute condition number: calculating $A^{-1}$

$$A^{-1} = \frac{1}{-0.2} \begin{bmatrix} 2 & -2 \\ -1.1 & 1 \end{bmatrix} = \begin{bmatrix} -10 & 10 \\ 5.5 & -5 \end{bmatrix}$$

  - infinity norms-based $\kappa(A)$:

$$\|A\|_\infty = \max(|1| + |2|, |1.1| + |2|) = \max(3, 3.1) = 3.1$$

$$\|A^{-1}\|_\infty = \max(|-10| + |10|, |5.5| + |-5|) = \max(20, 10.5) = 20$$

  - **condition number:** $\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = 3.1 \times 20 = 62 >>> 0$
  - solving $Ax = b$ using cramer's rule: $x_1 = 4$ and $x_2 = 3$
  - then solve it again but with the coefficient of $x_1$ in the second equation modified slightly to 1.05
  - using cramer's rule: $x_1 = 8$ and $x_2 = 1$
  - substituting these results in the original system yields 10 and $\approx 10.4$
  - so although $x_1 = 8$ and $x_2 = 1$ are not the true solution to the original system, the error check is close enough to possibly mislead you
  - in general: an ill-conditioned $A$ is one with a $\det(A)$ close to 0

# how to improve naive gauss?

- first things first: **use more sig figs** but this results in heavier computations
- pivoting: if a pivot element is zero, normalization step leads to division by zero
- same problem arises when the pivot element is close to zero
- **solution for pivoting:**
  - **partial pivoting:** switching the rows so that the largest element is the pivot element
  - that is, at the beginning of the *kth* iteration of forward elimination, find the maximum of
    $$|a_{k,k}|, |a_{k+1,k}|, \ldots, |a_{n,k}|$$
  - then if the maximum of the values is in the $p$th row, $k \leq p \leq n$, then switch rows $p$ and $k$
  - gaussian elimination with partial pivoting ensures that at each step of forward elimination the pivoting element $|a_{k,k}|$ has the largest absolute value
  - **complete pivoting:** searching for the largest element in all rows and columns then switching

# examples

solve $Ax = b$ using partial pivoting

$$A = \begin{bmatrix} 10 & -7 & 0 \\ -3 & 2.099 & 6 \\ 5 & -1 & 5 \end{bmatrix}, b = \begin{bmatrix} 7 \\ 3.901 \\ 6 \end{bmatrix}$$

- first step $k = 1$, seeing the values of the first column $|10|$, $|-3|$, and $|5|$ which means switch row 1 with row 1 (do nothing) then:

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2.099 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 3.901 \\ 6 \end{bmatrix} \implies \begin{bmatrix} 10 & -7 & 0 \\ 0 & -0.001 & 6 \\ 0 & 2.5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 6.001 \\ 2.5 \end{bmatrix}$$

- second step $k = 2$, seeing the values of the first column $|-0.001|$ and $|2.5|$ so we need to switch row 2 with row 3:

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & -0.001 & 6 \\ 0 & 2.5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 6.001 \\ 2.5 \end{bmatrix} \implies \begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.001 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2.5 \\ 6.001 \end{bmatrix}$$

then forward elimination:

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.002 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2.5 \\ 6.002 \end{bmatrix}$$

# gauss-jordan method

- a variation of gaussian elimination
- major differences: when an unknown is eliminated, it is eliminated from all other equations
- all rows are normalized by dividing them by their pivot
- elimination step results in an identity matrix
- no need for back substitution to obtain solution
- gj involves approximately 50% more operations than gauss
- gj is like gauss but it keeps going (like me at the buffet)
- **example** for this system: $Ax = b$ written as matrix $[A,\ b]$

$$\left[\begin{array}{ccc|c} 3 & -0.1 & -0.2 & 7.85 \\ 0.1 & 7 & -0.3 & -19.3 \\ 0.3 & -0.2 & 10 & 71.4 \end{array}\right]$$

# matlab's rref command: gauss-jordan elimination

- setup & normalize row 1: express as augmented matrix, divide $r_1$ by 3.

$$\left[\begin{array}{ccc|c} 3 & -0.1 & -0.2 & 7.85 \\ 0.1 & 7 & -0.3 & -19.3 \\ 0.3 & -0.2 & 10 & 71.4 \end{array}\right] \Longrightarrow \left[\begin{array}{ccc|c} 1 & -0.0333 & -0.0667 & 2.6167 \\ 0.1 & 7 & -0.3 & -19.3 \\ 0.3 & -0.2 & 10 & 71.4 \end{array}\right]$$

- eliminate $x_1$ & normalize row 2: reduce $r_2$, $r_3$, then divide $r_2$ by 7.0033.

$$\Longrightarrow \left[\begin{array}{ccc|c} 1 & -0.0333 & -0.0667 & 2.6167 \\ 0 & 7.0033 & -0.2933 & -19.5617 \\ 0 & -0.1900 & 10.0200 & 70.6150 \end{array}\right] \Longrightarrow \left[\begin{array}{ccc|c} 1 & -0.0333 & -0.0667 & 2.6167 \\ 0 & 1 & -0.0419 & -2.7932 \\ 0 & -0.1900 & 10.0200 & 70.6150 \end{array}\right]$$

- eliminate $x_2$ & normalize row 3: reduce $r_1$, $r_3$, then divide $r_3$ by 10.0120.

$$\Longrightarrow \left[\begin{array}{ccc|c} 1 & 0 & -0.0681 & 2.5236 \\ 0 & 1 & -0.0419 & -2.7932 \\ 0 & 0 & 10.0120 & 70.0843 \end{array}\right] \Longrightarrow \left[\begin{array}{ccc|c} 1 & 0 & -0.0681 & 2.5236 \\ 0 & 1 & -0.0419 & -2.7932 \\ 0 & 0 & 1 & 7.0000 \end{array}\right]$$

- eliminate $x_3$: reduce $r_1$, $r_2$ to find the final reduced row echelon form.

$$\Longrightarrow \left[\begin{array}{ccc|c} 1 & 0 & 0 & 3.0000 \\ 0 & 1 & 0 & -2.5000 \\ 0 & 0 & 1 & 7.0000 \end{array}\right]$$

- then $x_1 = 3, x_2 = -2.5, x_3 = 7$ without any back substitution

# how gj gives us matrix inverse

- gj involves approximately 50% more operations than gauss, then why use it?
- computing the matrix inverse: by augmenting the system with an identity matrix $[A \mid I_n]$, gj directly transforms it into $[I_n \mid A^{-1}]$
- standard gaussian elimination cannot compute the inverse directly
- two birds in one slightly more expensive stone
- elimination sweeps in gj rows both above and below the pivot $\Rightarrow$ this structure can be heavily parallelized on modern hardware
- rref uniquely yields the final solution, identifying free variables, or determining the rank of a matrix: you see if the gj process yielded a full row of zeroes meaning matrix dropped rank
- practical note: for simply solving $Ax = b$ in professional software (like matlab's backslash operator), lu decomposition is used

# computing the inverse $A^{-1}$ using gauss-jordan

- setup & normalize row 1: express as augmented matrix $[a \mid i_3]$, divide $r_1$ by 3.

$$\left[\begin{array}{ccc:ccc} 3 & -0.1 & -0.2 & 1 & 0 & 0 \\ 0.1 & 7 & -0.3 & 0 & 1 & 0 \\ 0.3 & -0.2 & 10 & 0 & 0 & 1 \end{array}\right] \Longrightarrow \left[\begin{array}{ccc:ccc} 1 & -0.0333 & -0.0667 & 0.3333 & 0 & 0 \\ 0.1 & 7 & -0.3 & 0 & 1 & 0 \\ 0.3 & -0.2 & 10 & 0 & 0 & 1 \end{array}\right]$$

- eliminate $x_1$ & normalize row 2: reduce $r_2$, $r_3$, then divide $r_2$ by 7.0033.

$$\Longrightarrow \left[\begin{array}{ccc:ccc} 1 & -0.0333 & -0.0667 & 0.3333 & 0 & 0 \\ 0 & 7.0033 & -0.2933 & -0.0333 & 1 & 0 \\ 0 & -0.1900 & 10.0200 & -0.1000 & 0 & 1 \end{array}\right] \Longrightarrow \left[\begin{array}{ccc:ccc} 1 & -0.0333 & -0.0667 & 0.3333 & 0 & 0 \\ 0 & 1 & -0.0419 & -0.0048 & 0.1428 & 0 \\ 0 & -0.1900 & 10.0200 & -0.1000 & 0 & 1 \end{array}\right]$$

- eliminate $x_2$ & normalize row 3: reduce $r_1$, $r_3$, then divide $r_3$ by 10.0120.

$$\Longrightarrow \left[\begin{array}{ccc:ccc} 1 & 0 & -0.0681 & 0.3331 & 0.0048 & 0 \\ 0 & 1 & -0.0419 & -0.0048 & 0.1428 & 0 \\ 0 & 0 & 10.0120 & -0.1009 & 0.0271 & 1 \end{array}\right] \Longrightarrow \left[\begin{array}{ccc:ccc} 1 & 0 & -0.0681 & 0.3331 & 0.0048 & 0 \\ 0 & 1 & -0.0419 & -0.0048 & 0.1428 & 0 \\ 0 & 0 & 1 & -0.0101 & 0.0027 & 0.0999 \end{array}\right]$$

- eliminate $x_3$: reduce $r_1$, $r_2$ to find the final form $[I_3 \mid A^{-1}]$.

$$\Longrightarrow \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & 0.3324 & 0.0050 & 0.0068 \\ 0 & 1 & 0 & -0.0052 & 0.1429 & 0.0042 \\ 0 & 0 & 1 & -0.0101 & 0.0027 & 0.0999 \end{array}\right]$$

- the right side of the dashed line is now our computed $A^{-1}$ matrix.

# lu decomposition intro

- would be nice to generalize gaussian elimination into a matrix decomposition problem
- here, we study writing $A$ as $A = LU$, multiplication of lower and upper triangular matrices $L$ and $U$
  - here $L$ has diagonal entries all equal to 1
- why is this helpful?
- it would allow us to solve $Ax = b$ easier
- how so? well if $Ax = b \rightarrow LUx = b$
- defining $y = Ux$ then $Ly = b$ can be solved easily via back substitution for $y$
- then you can solve $Ux = y$ via back substitution too, quite easily
- so the lu decomposition is comprised of three steps
  1. step 1: finding $L$ and $U$ such that $A = LU$
  2. step 2: back substitution to find $y$ for $Ly = b$
  3. step 3: back substitution to find $x$ for $Ux = y$
- lu decomposition provides an efficient way to compute matrix inverse too as we shall see soon
- we will see how guassian decomposition is kinda like the lu decomposition

# lu structure

- objective: writing $Ax = LUx = b$ for a $n = 3$ as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- note that you can compute $U = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a'_{33} \end{bmatrix}$ via applying the first step of gaussian elimination that results in $Ux = d$
- we will see how $L$ can be extracted from the first step of gaussian elimination
- how did we perform forward elimination? we multiply row 1 by $l_{21} = a_{21}/a_{11}$ and subtract from row 2, multiply row 1 by $l_{31} = a_{31}/a_{11}$ and subtract from row 3
- then, we multiply the modified second row by $l_{32} = a'_{32}/a'_{22}$
- you can observe that row operations is basically multiplying by a mostly identity matrix besides the weight that is used to multiply specific rows
- you can now check that: $A = LU$

## motivating example

- consider $A = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix}$ then $U = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.003 & -0.293 \\ 0 & 0 & 10.01 \end{bmatrix}$
- hence $l_{21} = a_{21}/a_{11} = 0.033$, $l_{31} = a_{31}/a_{11} = 0.1$, and $l_{32} = a'_{32}/a'_{22} = -0.027$
- now we can write:

$$A = LU = \begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix} \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix}$$

- which is slightly different from $A$

$$LU = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.0999999 & 7 & -0.3 \\ 0.3 & -0.2 & 9.99996 \end{bmatrix}$$

- btw, we restricted structure of $L$ to have 1's on the diagonal so we can get a unique lu decomposition
- in general, any square matrix satisfies: a unique lu factorization, infinitely many LU factorizations, or no LU factorization

# storing entries of $L$ and $U$

- an example on how to store entries of $L$, $U$ without storing zeros:

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow[r_3 \leftarrow r_3 + 2r_1]{r_2 \leftarrow r_2 - 3r_1} \begin{bmatrix} 3 & -1 & 1 \\ \hline 3 & 4 & -1 \\ -2 & 3 & -3 \end{bmatrix} \xrightarrow{r_3 \leftarrow r_3 - \frac{3}{4} r_2} \begin{bmatrix} 3 & -1 & 1 \\ 3 & 4 & -1 \\ \hline -2 & \frac{3}{4} & -\frac{9}{4} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -2 & \frac{3}{4} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix}.$$

- example for rectangular matrices:

$$\begin{bmatrix} 2 & -1 \\ 6 & 5 \\ -10 & 3 \\ 12 & -2 \end{bmatrix} \xrightarrow[r_4 \leftarrow r_4 - 6r_1]{\substack{r_2 \leftarrow r_2 - 3r_1 \\ r_3 \leftarrow r_3 + 5r_1}} \begin{bmatrix} 2 & -1 \\ \hline 3 & 8 \\ -5 & -2 \\ 6 & 4 \end{bmatrix} \xrightarrow[r_4 \leftarrow r_4 - \frac{1}{2} r_2]{r_3 \leftarrow r_3 + \frac{1}{4} r_2} \begin{bmatrix} 2 & -1 \\ 3 & 8 \\ -5 & -\frac{1}{4} \\ \hline 6 & \frac{1}{2} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ -5 & -\frac{1}{4} & 1 & 0 \\ 6 & \frac{1}{2} & 0 & 1 \end{bmatrix}_{4 \times 4}, \quad U = \begin{bmatrix} 2 & -1 \\ 0 & 8 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}_{4 \times 2}.$$

# example on back substitution

- how do you solve for $x$ given the lu decomposition?
- apply steps 2 and step 3
- setting up the system and identifying the $lu$ decomposition:

$$A = \begin{bmatrix} 1 & 4 & -2 \\ 2 & 5 & -3 \\ -3 & -18 & 16 \end{bmatrix}, \quad b = \begin{bmatrix} -12 \\ -14 \\ 64 \end{bmatrix}$$

$$A = LU \triangleq \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & -2 \\ 0 & -3 & 1 \\ 0 & 0 & 8 \end{bmatrix}$$

solve for $x$

- solve $Ly = b$ for $y$: $y = \begin{bmatrix} -12 \\ 10 \\ 8 \end{bmatrix}$
- solve $Ux = y$ for $x$ $y = \begin{bmatrix} 2 \\ -3 \\ 1 \end{bmatrix}$
- computational time for LU decomposition is nearly identical to gaussian elimination
- same issues still exist as in gaussian elimination, so need to perform pivoting or reordering of rows/columns via $P$ such that $PA = LU$

# finding inverse via lu decomposition

- remember matrix property: $(AB)^{-1} = B^{-1}A^{-1}$ if $A$ and $B$ are invertible
- so using lu decomposition, to find matrix inverse:
  1. step 1: find lu decomposition $A = LU$
  2. step 2: find $A^{-1} = U^{-1}L^{-1}$ by inverting matrices $U$ and $L$ which is much easier than inverting a full and not spare matrix $A$
- why is inverting matrices $U$ and $L$ easier than inverting $A$?
- inverse of an upper/lower triangular matrix is another upper/lower triangular matrix
- inverse exists only if none of the diagonal element is zero
- can be computed from first principles: using $L^{-1}L = I_n$
- no need to compute determinant which is awesome
- diagonal elements of $L$ or $U$ are the reciprocal of $L$ or $U$
- other elements are computed such that $L^{-1}L = I_n$ (and there's awesome algorithms for that)
- well, let's take an example:

## pseudo code + example

- consider $L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & 1 \end{bmatrix}$. find $L^{-1}$
- **solution:** let $L^{-1}$ be a lower triang. matrix with entries $x_{ij}$ and $LL^{-1} = I_3$:

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_{11} & 0 & 0 \\ x_{21} & x_{22} & 0 \\ x_{31} & x_{32} & x_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- performing forward substitution row by row to solve for the unknowns:
- from row 1: $1(x_{11}) = 1 \implies x_{11} = 1$
- from row 2:

$$2(x_{11}) + 1(x_{21}) = 0 \implies 2(1) + x_{21} = 0 \implies x_{21} = -2$$
$$2(0) + 1(x_{22}) = 1 \implies x_{22} = 1$$

- from row 3:

$$-1(x_{11}) + 2(x_{21}) + 1(x_{31}) = 0 \implies -1(1) + 2(-2) + x_{31} = 0 \implies x_{31} = 5$$
$$-1(0) + 2(x_{22}) + 1(x_{32}) = 0 \implies 2(1) + x_{32} = 0 \implies x_{32} = -2$$
$$-1(0) + 2(0) + 1(x_{33}) = 1 \implies x_{33} = 1$$

$$\implies L^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 5 & -2 & 1 \end{bmatrix}$$

## cholesky decomposition

- some linear systems of equations $Ax = b$ appear with a symmetric $A$ matrix
- when $A$ is symmetric, this means we can store the unique entries of $A$ in a small data structure
- in this case, we can write $A = LL^\top$, also known as the cholesky decomposition
- we can compute $L$ via the lu decomposition but this requires more time than needed
- alternatively, we can get closed form expressions for the entries of $L$ relative to entries of $A$. how?

# cholesky decomposition

- **example** for a 3-by-3 symmetric matrix $A$:

$$A = LL^\top = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{bmatrix} = \begin{bmatrix} L_{11}^2 & * & * \\ L_{21}L_{11} & L_{21}^2 + L_{22}^2 & * \\ L_{31}L_{11} & L_{31}L_{21} + L_{32}L_{22} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{bmatrix}$$

- hence: $L = \begin{bmatrix} \sqrt{A_{11}} & 0 & 0 \\ A_{21}/L_{11} & \sqrt{A_{22} - L_{21}^2} & 0 \\ A_{31}/L_{11} & (A_{32} - L_{31}L_{21})/L_{22} & \sqrt{A_{33} - L_{31}^2 - L_{32}^2} \end{bmatrix}$
- and in general, for any symmetric matrix of size $n$, we can write:

$$L_{j,j} = (\pm)\sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2}, \quad L_{i,j} = \frac{1}{L_{j,j}} \left( A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right) \quad \text{for } i > j$$

- **example:** compute the cholesky decomposition of this matrix

$$A = \begin{bmatrix} 6 & 15 & 55 \\ 15 & 55 & 225 \\ 55 & 225 & 979 \end{bmatrix} \rightarrow l_{11} = \sqrt{6}, \; l_{21} = \frac{a_{21}}{l_{11}} = 6.123, \; l_{22} = \sqrt{a_{22} - l_{21}^2} = 4.183, \ldots$$

# intro to gauss-seidel

- iterative/approximate methods provide an alternative to the elimination methods that are based on direct algebraic equations and manipulations
- gaus, LU, diagonal, cholesky, etc.. decompositions can be very slow
- so we need an alternative: **iterative solvers**
- iterative methods herein are similar to the ones for finding roots for $f(x) = 0$
- how so? we start by guessing a value and then using a systematic method to obtain a refined estimate of $Ax = b$ or $Ax - b = 0 = f(x)$
- frankly, there are so many of them:
  - jacobi, gauss-seidel, SOR, SSOR; MINRES, GMRES, etc..
  - all are easier to optimize and parallelize than direct methods
  - caveat: iterative methods suffer from convergence issues
- we learn some of these methods: their algorithm, convergence, error analysis, pros and cons

# definitions + preliminaries

- **definition**: the spectral radius of matrix $A \in \mathbb{R}^{n \times n}$ is: $\rho(A) = \max |\text{eig}(A)|$
- **another definition:** matrix $A \in \mathbb{R}^{n \times n}$ is convergent if

$$\lim_{k \to \infty} A_{ij}^k = 0$$

- are these matrices convergent?

$$A = \begin{bmatrix} 1/2 & 0 \\ 1/4 & 1/2 \end{bmatrix} \quad B = \begin{bmatrix} 10 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

## convergent matrices

these statements are equivalent:

- $A$ is convergent
- $\lim_{n \to \infty} ||A^n|| = 0$ for any matrix norm
- spectral radius of $A$ is less than one
- $\lim_{n \to \infty} A^n x = 0$ for all $x \in \mathbb{R}^n$

# iterative procedures for $Ax = b$ 101

- consider solving the linear system $Ax = b$ with $n$ unknowns, $n$ equations
- by splitting the matrix $A = M - N$ for an invertible matrix $M$, we can write

$$x^{(k)} = M^{-1}Nx^{(k-1)} + M^{-1}b$$

for $k \geq 1$ and with an initialization $x^{(0)}$

- since $N = M - A$ this iteration can also be written as

$$x^{(k)} = \underbrace{(I_n - M^{-1}A)}_{\text{iteration matrix}} x^{(k-1)} + M^{-1}b \text{ or } x^{(k)} = x^{(k-1)} + M^{-1}(b - Ax^{(k-1)})$$

- important questions:
  - how to pick matrices $M$ and $N$
  - when does this iteration converge? under what conditions on $M$ and $N$?
  - for what initial conditions would this iteration converge?
  - how fast does this iteration converge to a fixed point solution?
- important known results: iteration is quick if $M$ is easily inverted (think diagonal matrices, triangular, etc..) and $M^{-1}$ kinda sorta approximates $A^{-1}$!!!!!

# convergence of iterative procedures

$$x^{(k)} = \underbrace{(I_n - M^{-1}A)}_{\text{iteration matrix}} x^{(k-1)} + M^{-1}b$$

- assume the above iteration converges to $x$
- then it's important to study $e^{(k)} = x - x^{(k)}$, error at iteration $k$
- hence upon convergence, we have: $x = (I_n - M^{-1}A)x + M^{-1}b$
- then we can write $e^{(k)} = (I_n - M^{-1}A)e^{(k-1)}$ and:

$$||e^{(k)}|| \leq ||I_n - M^{-1}A||||e^{(k-1)}|| \leq ||I_n - M^{-1}A||^2||e^{(k-2)}|| \leq \ldots \leq ||I_n - M^{-1}A||^k||e^{(0)}||$$

**convergence theorem**

if the spectral radius of the iteration matrix is less than one:

$$\underbrace{\rho(I_n - M^{-1}A)}_{\text{spectral radius}} < 1$$

then the above iteration converges to the solution of $Ax = b$ for any $x^{(0)}$

- considering $\delta = ||I_n - M^{-1}A||$ then $||e^{(k)}|| \leq \frac{\delta}{1-\delta}||x^{(k)} - x^{(k-1)}||$

# iterative method 1: the richardson method

- this is the simplest method to solve $Ax = b$
- choose $M = I_n$ and hence $N = I_n - A$
- this results in this iteration:

$$x^{(k)} = x^{(k-1)} + (b - Ax^{(k-1)})$$

- recall the theoretical result that the spectral radius of $I_n - M^{-1}A = I_n - A$ has to be less than one
- what happens to the evalues of matrix $X$ when you write $I_n - X$ or $X - I_n$?
- all evalues of $A$ are shifted by one so $|\text{eig}(A - I_n)| = |\text{eig}(A) - 1|$
- so what should the max evalue of $I_n - A$ be in this in case for the richardson method to converge?
- so this method doesn't work if the max evalue of $I_n - A$ are outside the unit disk
- solution: **scaling** as follows $\alpha Ax = \alpha b$ where now the evalues of $A$ satisfy $|\text{eig}(\alpha A) - 1| < 1$
- now we can choose $M = I_n$ and $N = I_n - \alpha A$ resulting in this iteration

$$x^{(k)} = x^{(k-1)} + \alpha(b - Ax^{(k-1)})$$

and the method converges for some $\alpha$

# relaxation methods

- here we cover methods based on *relaxation*
- first, we can split matrix $A$ into $A = D - E - F$ where $D$ is the diagonal entries of $A$, $E$ and $F$ are the lower and upper triangular portions of $A$ with zero diagonals (multiplied by $-1$)
- relaxation methods we will cover here are based on choosing $M$ and $N$ in $A = M - N$ based on this:

| | $M$ | $N$ |
|---|---|---|
| *jacobi method* | $D$ | $E + F$ |
| *gauss-seidel method* | $D - E$ | $F$ |
| *successive over relaxation method* | $\omega^{-1}D - E$ | $\frac{1-\omega}{\omega}D + F \ \ (\omega \in (0, 2))$ |

# method 2: jacobi

- according to the table, jacobi method is performed with $M = D$ and $N = E + F$
- the iteration can then be written as

$$Dx^{(k)} = (E + F)x^{(k-1)} + b$$

or entry-wise at iteration $k$:

$$x_i^{(k)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right)$$

- **example:** solve $Ax = b$ using the jacobi method for this data:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \; b = \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}, \; x^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

- **solution:** $x^* = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$, only reached after $> 10$ iterations

# method 3: gauss-seidel

- gauss-seidel method is formulated with $M = D - E$ and $N = F$
- the iteration can be written as

$$Dx^{(k)} = b + Ex^{(x)} + Fx^{(k-1)}$$

or entry-wise at iteration $k$:

$$x_i^{(k)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right)$$

# method 4: successive over relaxation

- the successive over relaxation (SOR) method is formulated with

$$M = \frac{1}{\omega} D - E \quad , \quad N = \frac{1-\omega}{\omega} D + F$$

- the iteration can be written as follows:

$$(D - \omega E)x^{(k)} = \left((1-\omega)D + \omega F\right) x^{(k-1)} + \omega b$$

and the ith component at iteration $k$ is or

$$Dx^{(x)} = (1-\omega)Dx^{(k-1)} + \omega(b + ex^{(k)} + Fx^{(k-1)})$$

- iteration can also be written as

$$x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \omega x_{GS,i}^{(k)}$$

with

$$x_{GS,i}^{(k)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right)$$

- basically the sor is a generalization of the gauss-seidel method (for $\omega = 1$, sor $=$ gs )

# comparison between methods 2–4

- comparison between jacobi, gauss-seidel, sor (with $\omega = 1.2$) for the previous example:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \; b = \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}, \; x^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

- **actual solution:** $x^* = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$

jacobi

| $k$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| 1 | 1.00000 | 1.00000 | 3.00000 |
| 2 | 1.00000 | 2.00000 | 3.00000 |
| 3 | 1.50000 | 2.00000 | 3.50000 |
| 4 | 1.50000 | 2.50000 | 3.50000 |
| 5 | 1.75000 | 2.50000 | 3.75000 |
| 6 | 1.75000 | 2.75000 | 3.75000 |
| 7 | 1.87500 | 2.75000 | 3.87500 |
| 8 | 1.87500 | 2.87500 | 3.87500 |
| 9 | 1.93750 | 2.87500 | 3.93750 |
| 10 | 1.93750 | 2.93750 | 3.93750 |

gauss-seidel

| $k$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| 1 | 1.00000 | 1.00000 | 3.00000 |
| 2 | 1.00000 | 2.00000 | 3.50000 |
| 3 | 1.50000 | 2.50000 | 3.75000 |
| 4 | 1.75000 | 2.75000 | 3.87500 |
| 5 | 1.87500 | 2.87500 | 3.93750 |
| 6 | 1.93750 | 2.93750 | 3.96875 |
| 7 | 1.96875 | 2.96875 | 3.98438 |
| 8 | 1.98438 | 2.98438 | 3.99219 |
| 9 | 1.99219 | 2.99219 | 3.99609 |
| 10 | 1.99609 | 2.99609 | 3.99805 |

sor

| $k$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| 1 | 1.00000 | 1.00000 | 3.40000 |
| 2 | 1.00000 | 2.44000 | 3.78400 |
| 3 | 1.86400 | 2.90080 | 3.98368 |
| 4 | 1.96768 | 2.99066 | 3.99766 |
| 5 | 2.00086 | 3.00098 | 4.00106 |
| 6 | 2.00042 | 3.00069 | 4.00020 |
| 7 | 2.00033 | 3.00018 | 4.00007 |
| 8 | 2.00004 | 3.00003 | 4.00005 |
| 9 | 2.00001 | 3.00000 | 4.00001 |
| 10 | 2.00000 | 3.00000 | 4.00000 |

- gauss-seidel is twice faster than jacobi, and sor is like twice faster than gs
- **important question:** how to compute the optimal $\omega^*$ that results in the fastest convergence?

## convergence of iterative methods

- similar to the previous theoretical result on convergence, and for a sequence governed by

$$x^{(k)} = Tx^{(k-1)} + c$$

where $c$ is a vector and $T$ is the mapping, the iteration converges if $\rho(T) < 1$

- this is btw identical to the fixed point iteration where we wanted $|g'(x)| \leq K < 1$ for the fp iteration to converge
- this is true here because the error bound can be written as

$$||x - x^{(k)}|| \leq ||T||^k ||x - x^{(0)}||$$

where $T$ is a different matrix for jacobi, gs, sor:

| | mapping matrix $T$ |
|---|---|
| *jacobi method* | $T = D^{-1}(E + F)$ |
| *gauss-seidel method* | $T = (D - E)^{-1}F$ |
| *sor method* | $T = (D - \omega E)^{-1}((1 - \omega)D + \omega F)$ |

- for the sor method, the optimal $\boxed{\omega^* = \dfrac{2}{1 + \sqrt{1 - (\rho(D^{-1}(E + F)))^2}}}$

# module summary and final thoughts

- these are not the only methods
- we still have methods based on graphical properties of matrix $A$ (i.e., a network/graph structure)
- and Krylov space-based methods (gradient descent, conjugate gradient)
- no time to cover all of these although they're as important
- in summary: we learned 6+ methods to solve $Ax = b$
- we discussed matrix decomposition, inverse, etc..
- some methods terminate in finite steps; others are iterative terminating for a tolerance

| **Method** | **Stability** | **Precision** | **Breadth of Application** | **Programming Effort** | **Comments** |
|---|---|---|---|---|---|
| Graphical | — | Poor | Limited | — | May take more time than the numerical method, but can be useful for visualization |
| Cramer's rule | — | Affected by round-off error | Limited | — | Excessive computational effort required for more than three equations |
| Gauss elimination (with partial pivoting) | — | Affected by round-off error | General | Moderate | |
| *LU* decomposition | — | Affected by round-off error | General | Moderate | Preferred elimination method; allows computation of matrix inverse |
| Gauss-Seidel | May not converge if not diagonally dominant | Excellent | Appropriate only for diagonally dominant systems | Easy | |