Lec 3: Linear Systems and LU Decomposition

15-369/669/769: Numerical Computing

Instructor: Minchen Li

Table of Content

Table of Content

Solvability of Linear Systems

Definition of Linear Systems

systems of linear equations like

$$\begin{aligned} 3x + 2y &= 6 \\ -4x + y &= 7 \end{aligned}$$

can be written in matrix form as in

$$\begin{pmatrix} 3 & 2 \\ -4 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 6 \\ 7 \end{pmatrix}$$

More generally, we can write linear systems in the form $A\mathbf{x} = \mathbf{b}$ for $A \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$. Algorithms for *solving* the linear system $A\mathbf{x} = \mathbf{b}$ input a matrix A and a vector $\mathbf{b}$, and they output a vector $\mathbf{x}$ satisfying the equation—should such an $\mathbf{x}$ exist.

Solvability of Linear Systems

The 3 Cases

The solvability of $A\mathbf{x} = \mathbf{b}$ must fall into one of three cases:

1. The system may not admit any solutions, as in:

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}. \qquad \textit{Overdetermined}$$

   This system enforces two incompatible conditions simultaneously: $x = -1$ and $x = 1$.

2. The system may admit a single solution; for instance, the system at the beginning of this section is solved by $(x, y) = (-8/11, 45/11)$.

3. The system may admit infinitely many solutions, e.g., $0\mathbf{x} = \mathbf{0}$. If a system $A\mathbf{x} = \mathbf{b}$ admits two distinct solutions $\mathbf{x}_0$ and $\mathbf{x}_1$, then it automatically has infinitely many solutions of the form $t\mathbf{x}_0 + (1-t)\mathbf{x}_1$ for all $t \in \mathbb{R}$, since

$$A(t\mathbf{x}_0 + (1-t)\mathbf{x}_1) = tA\mathbf{x}_0 + (1-t)A\mathbf{x}_1 = t\mathbf{b} + (1-t)\mathbf{b} = \mathbf{b}.$$

   Because it has multiple solutions, this linear system is labeled *underdetermined*.

Solvability of Linear Systems

A Geometric View

The solvability of the system $A\mathbf{x} = \mathbf{b}$ depends both on A and on $\mathbf{b}$. For instance, if we modify the unsolvable system above to

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

then the system changes from having no solutions to infinitely many solutions of the form $(1, y)$. Every matrix A admits a right-hand side $\mathbf{b}$ such that $A\mathbf{x} = \mathbf{b}$ is solvable, since $A\mathbf{x} = \mathbf{0}$ always can be solved by $\mathbf{x} = \mathbf{0}$ regardless of A.

Since Ax can be viewed as a linear combination of A's column vectors with coefficients in x:

$$\begin{pmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \coloneqq c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n$$

Ax = b is solvable exactly when b is in the column space of A

Solvability of Linear Systems

Assumptions on A

The shape of the matrix $A \in \mathbb{R}^{m \times n}$ has considerable bearing on the solvability of $A\mathbf{x} = \mathbf{b}$.

No wide matrix system admits a unique solution.

For every tall matrix A, there exists a $\mathbf{b}_0$ such that $A\mathbf{x} = \mathbf{b}_0$ is not solvable for $\mathbf{x}$.

- We will consider only *square* $A \in \mathbb{R}^{n \times n}$.
- We will assume that A is *nonsingular*, that is, that $A\mathbf{x} = \mathbf{b}$ is solvable for any $\mathbf{b}$.

Solvability of Linear Systems

Practical Considerations

We do *not* have to solve $A\mathbf{x} = \mathbf{b}$ by explicitly computing the entire inverse matrix $A^{-1} \in \mathbb{R}^{n \times n}$ and then multiplying to find $\mathbf{x} = A^{-1}\mathbf{b}$. While this procedure is valid mathematically, it can yield a considerable amount of overkill and potential for numerical instability for several reasons:

- The matrix A^{-1} may contain values that are difficult to express in floating-point precision, in the same way that $1/\varepsilon \to \infty$ as $\varepsilon \to 0$.

- It may be possible to tune the solution strategy both to A and to $\mathbf{b}$, e.g., by working with the columns of A that are the closest to $\mathbf{b}$ first. Strategies like these can provide higher numerical stability.

- Even if A is sparse, meaning it contains many zero values that do not need to be stored explicitly, or has other special structure, the same may not be true for A^{-1}.

Avoid computing A^{-1} explicitly unless you have a strong justification for doing so.

Table of Content

Gaussian Elimination (A Simple Example)

Overview

We will consider the following system:

$$\begin{aligned} y - z &= -1 \\ 3x - y + z &= 4 \\ x + y - 2z &= -3. \end{aligned}$$

- **Strategy:** "isolate" variables, iteratively simplifying individual equalities until each is of the form x = const.

- Alongside each step, we maintain a matrix encoding the current state:

$$\left(\begin{array}{ccc|c} 0 & 1 & -1 & -1 \\ 3 & -1 & 1 & 4 \\ 1 & 1 & -2 & -3 \end{array}\right)$$

Gaussian Elimination (A Simple Example)

Simplification Steps

Perhaps we wish to deal with the variable x first. For convenience, we can *permute* the rows of the system so that the third equation appears first:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3x - y + z &= 4 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 3 & -1 & 1 & 4 \end{array}\right)$$

We then *substitute* the first equation into the third to eliminate the $3x$ term. This is the same as scaling the relationship $x + y - 2z = -3$ by -3 and adding the result to the third equation:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ -4y + 7z &= 13 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & -4 & 7 & 13 \end{array}\right)$$

Gaussian Elimination (A Simple Example)

Simplification Steps

Similarly, to eliminate y from the third equation, we scale the second equation by 4 and add the result to the third:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3z &= 9 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 3 & 9 \end{array}\right)$$

We have now isolated z! We scale the third row by $1/3$ to yield an expression for z:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ z &= 3 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Gaussian Elimination (A Simple Example)

Simplification Steps

Now, we substitute $z = 3$ into the other two equations to remove z from all but the final row:

$$\begin{aligned} x + y &= 3 \\ y &= 2 \\ z &= 3 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Finally, we make a similar substitution for y to reveal the solution:

$$\begin{aligned} x &= 1 \\ y &= 2 \\ z &= 3 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Gaussian Elimination (A Simple Example)

Summary

Revisiting the steps above yields a few observations about how to solve linear systems:

- We wrote successive systems $A_i x = b_i$ that can be viewed as simplifications of the original $Ax = b$.
- We solved the system without ever writing down A^{-1}.
- We repeatedly used a few elementary operations: scaling, adding, and permuting rows.
- The same operations were applied to A and b.
- The steps did not depend on b. All our decisions were motivated by eliminating nonzero values in A.
- We terminated when we reached the simplified system $I_{n \times n} x = b_k$.

We will use all of these general observations about solving linear systems to our advantage.

Table of Content

Gaussian Elimination (Encoding Row Operations)

Matrix Representation

- Like the previous example, we can solve any linear systems using these 3 **row operations**:
  - permutation,
  - row scaling, and
  - adding a multiple of one row to another
- For convenience of analysis, we can use **matrices** to represent them.
- But in the algorithms, constructing matrices may not be necessary

- The following are equivalent:
  1. Scale the first row of A by 2.
  2. Replace A with $S_2 A$, where S_2 is defined by:

$$S_2 := \begin{pmatrix} 2 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Gaussian Elimination (Encoding Row Operations)

Permutation

- Indexing the rows of a matrix using the integers 1, . . . , m, a permutation of them is a function:

  $\sigma : \{1, \ldots, m\} \to \{1, \ldots, m\}$ such that $\{\sigma(1), \ldots, \sigma(m)\} = \{1, \ldots, m\}$

  where σ maps every index to a different target.

> If $\mathbf{e}_k$ is the k-th standard basis vector, then the product $\mathbf{e}_k^\top A$ is the k-th row of the matrix A. We can "stack" or concatenate these row vectors vertically to yield a matrix permuting the rows according to σ:
>
> $$P_\sigma := \begin{pmatrix} - & \mathbf{e}_{\sigma(1)}^\top & - \\ - & \mathbf{e}_{\sigma(2)}^\top & - \\ & \vdots & \\ - & \mathbf{e}_{\sigma(m)}^\top & - \end{pmatrix}.$$
>
> The product $P_\sigma A$ is the matrix A with rows permuted according to σ.

Gaussian Elimination (Encoding Row Operations)

Permutation (Example)

Example 3.1 (Permutation matrices). Suppose we wish to permute rows of a matrix in $\mathbb{R}^{3\times 3}$ with $\sigma(1) = 2$, $\sigma(2) = 3$, and $\sigma(3) = 1$. According to our formula we have

$$P_\sigma = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

- P_σ has 1's in positions indexed $(k, \sigma(k))$ and 0's elsewhere
- If we put 1's in $(\sigma(k), k)$ and 0's elsewhere, we get P_σ^T, and it undoes the permutation
- Thus, $P_\sigma^T P_\sigma = P_\sigma P_\sigma^T = I$, or $P_\sigma^T = P_\sigma^{-1}$

Gaussian Elimination (Encoding Row Operations)

Row Scaling

Suppose we write down a list of constants $a_1, \ldots, a_m$ and seek to scale the k-th row of A by a_k for each k. This task is accomplished by applying the scaling matrix S_a:

$$S_a \coloneqq \begin{pmatrix} a_1 & 0 & 0 & \cdots \\ 0 & a_2 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_m \end{pmatrix}.$$

Assuming that all the a_k's satisfy $a_k \neq 0$, it is easy to invert S_a by scaling back:

$$S_a^{-1} = S_{1/a} \coloneqq \begin{pmatrix} 1/a_1 & 0 & 0 & \cdots \\ 0 & 1/a_2 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/a_m \end{pmatrix}.$$

If any a_k equals zero, then S_a is not invertible.

Gaussian Elimination (Encoding Row Operations)

Elimination

- Suppose we wish to scale row k by a constant c, and add the result to row ℓ; ($k \neq l$).

- We first construct the matrix that extract a row of a matrix and move it to another row:

  - $\mathbf{e}_k^T A$ picks out the k-th row of A.

  - Pre-multiplying it by $\mathbf{e}_l$ yields a matrix $\mathbf{e}_l \mathbf{e}_k^T A$ that is 0 except on its ℓ-th row, which is equal to the k-th row of A.

Example:

$$\mathbf{e}_2 \mathbf{e}_3^\top A = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$$= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 7 & 8 & 9 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 0 \\ 7 & 8 & 9 \\ 0 & 0 & 0 \end{pmatrix}.$$

Gaussian Elimination (Encoding Row Operations)

Elimination (Matrix Form)

We have succeeded in isolating row k and moving it to row ℓ. Our original elimination operation was to add c times row k to row ℓ, which we can now carry out using the sum $A + c\mathbf{e}_\ell \mathbf{e}_k^\top A = (I_{n\times n} + c\mathbf{e}_\ell \mathbf{e}_k^\top)A$. Isolating the coefficient of A, the desired elimination matrix is $M := I_{n\times n} + c\mathbf{e}_\ell \mathbf{e}_k^\top$.

The action of M can be reversed: Scale row k by c and *subtract* the result from row ℓ. We can check this formally:

$$\begin{aligned}(I_{n\times n} - c\mathbf{e}_\ell \mathbf{e}_k^\top)(I_{n\times n} + c\mathbf{e}_\ell \mathbf{e}_k^\top) &= I_{n\times n} + (-c\mathbf{e}_\ell \mathbf{e}_k^\top + c\mathbf{e}_\ell \mathbf{e}_k^\top) - c^2 \mathbf{e}_\ell \mathbf{e}_k^\top \mathbf{e}_\ell \mathbf{e}_k^\top \\ &= I_{n\times n} - c^2 \mathbf{e}_\ell (\mathbf{e}_k^\top \mathbf{e}_\ell) \mathbf{e}_k^\top \\ &= I_{n\times n} \text{ since } \mathbf{e}_k^\top \mathbf{e}_\ell = \mathbf{e}_k \cdot \mathbf{e}_\ell, \text{ and } k \neq \ell.\end{aligned}$$

That is, $M^{-1} = I_{n\times n} - c\mathbf{e}_\ell \mathbf{e}_k^\top$.

Gaussian Elimination (Encoding Row Operations)

A System-Solving Example

Example 3.3 (Solving a system)**.** We can now encode each of our operations from Section 3.2 using the matrices we have constructed above:

1. Permute the rows to move the third equation to the first row:

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

2. Scale row one by -3 and add the result to row three:

$$E_1 = I_{3\times 3} - 3\mathbf{e}_3\mathbf{e}_1^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}.$$

Gaussian Elimination (Encoding Row Operations)

A System-Solving Example

3. Scale row two by 4 and add the result to row three:

$$E_2 = I_{3\times 3} + 4\mathbf{e}_3\mathbf{e}_2^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix}.$$

4. Scale row three by $1/3$:

$$S = \operatorname{diag}(1, 1, 1/3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/3 \end{pmatrix}.$$

5. Scale row three by 2 and add it to row one:

$$E_3 = I_{3\times 3} + 2\mathbf{e}_1\mathbf{e}_3^\top = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Gaussian Elimination (Encoding Row Operations)

A System-Solving Example

6. Add row three to row two:

$$E_4 = I_{3\times 3} + \mathbf{e}_2\mathbf{e}_3^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

7. Scale row two by -1 and add the result to row one:

$$E_5 = I_{3\times 3} - \mathbf{e}_1\mathbf{e}_2^\top = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Gaussian Elimination (Encoding Row Operations)

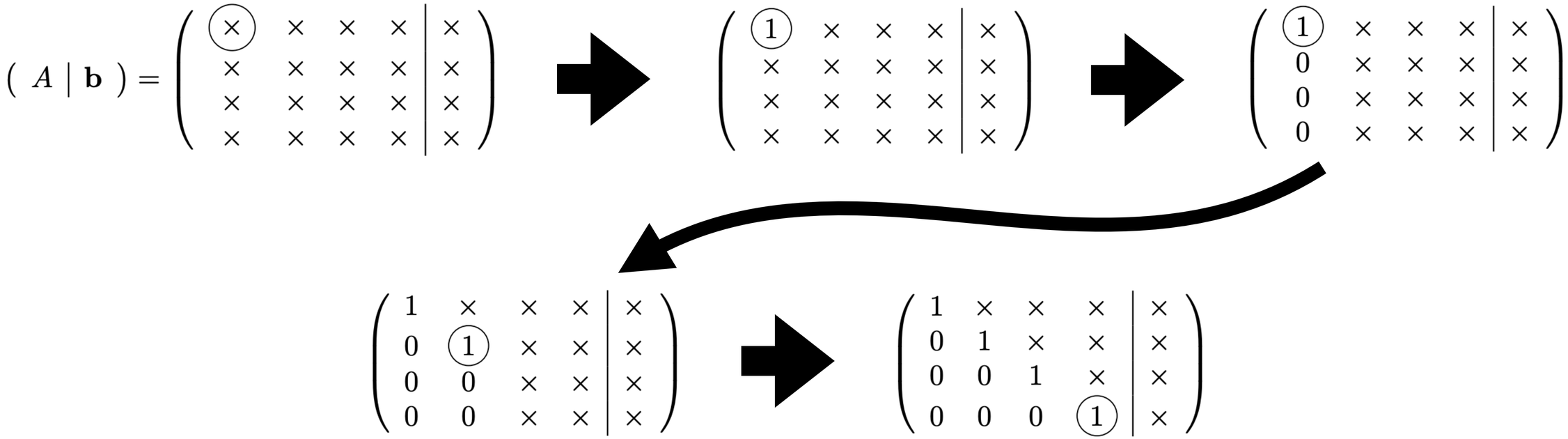
A System-Solving Example

> Combining these matrices, the inverse of A in Section 3.2 satisfies
>
> $$\begin{aligned} A^{-1} &= E_5 E_4 E_3 S E_2 E_1 P \\ &= \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/3 \end{pmatrix} \\ &\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1/3 & 1/3 & 0 \\ 7/3 & 1/3 & -1 \\ 4/3 & 1/3 & -1 \end{pmatrix}. \end{aligned}$$

Table of Content

Gaussian Elimination (The Algorithm)

Forward Substitution



Gaussian Elimination (The Algorithm)

Forward Substitution (Pseudo-Code)

function FORWARD-SUBSTITUTION($A, \mathbf{b}$)
▷ Converts a system $A\mathbf{x} = \mathbf{b}$ to an upper-triangular system $U\mathbf{x} = \mathbf{y}$.
▷ Assumes invertible $A \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$.

$U, \mathbf{y} \leftarrow A, \mathbf{b}$ ▷ U will be upper triangular at completion
for $p \leftarrow 1, 2, \ldots, n$ ▷ Iterate over current pivot row p
▷ Optionally insert pivoting code here

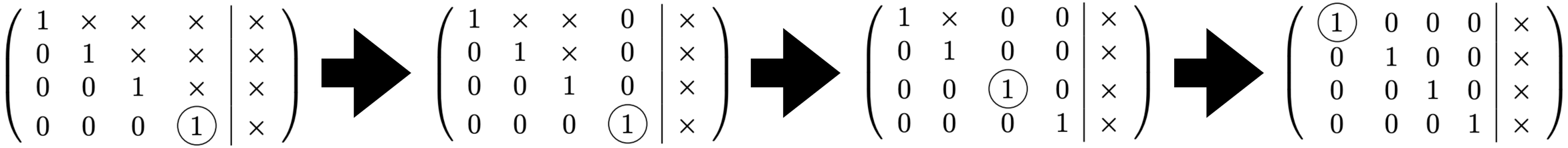
$s \leftarrow {}^1/_{u_{pp}}$ ▷ Scale row p to make element at (p, p) equal one
$y_p \leftarrow s \cdot y_p$
for $c \leftarrow p, \ldots, n : u_{pc} \leftarrow s \cdot u_{pc}$

for $r \leftarrow (p+1), \ldots, n$ ▷ Eliminate from future rows
$s \leftarrow -u_{rp}$ ▷ Scale row p by s and add to row r
$y_r \leftarrow y_r + s \cdot y_p$
for $c \leftarrow p, \ldots, n : u_{rc} \leftarrow u_{rc} + s \cdot u_{pc}$

return $U, \mathbf{y}$

Gaussian Elimination (The Algorithm)

Backward Substitution



function BACK-SUBSTITUTION($U, \mathbf{y}$)
▷ Solves upper-triangular systems $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$.

$\mathbf{x} \leftarrow \mathbf{y}$ ▷ We will start from $U\mathbf{x} = \mathbf{y}$ and simplify to $I_{n\times n}\mathbf{x} = \mathbf{x}$
for $p \leftarrow n, n-1, \ldots, 1$ ▷ Iterate backward over pivots
  for $r \leftarrow 1, 2, \ldots, p-1$ ▷ Eliminate values above u_{pp}
    $x_r \leftarrow x_r - {}^{u_{rp}x_p}/_{u_{pp}}$
return $\mathbf{x}$

Gaussian Elimination (The Algorithm)

Complexity

- Each row operation (scaling, elimination, row swap) takes $O(n)$ time (operates on n elements).
- For each pivot, we perform $O(n)$ operations on n rows $\Rightarrow O(n^2)$ per pivot.
- We perform elimination for n pivots in total.
- Total cost of Gaussian elimination: $O(n^3)$.

Gaussian Elimination (The Algorithm)

Pivoting

Since Gaussian elimination scales by the reciprocal of the pivot, the most numerically stable option is to choose a *large* pivot (in absolute value). Small pivots have large reciprocals, which scale matrix elements to regimes that may lose precision. There are two well-known pivoting strategies:

1. *Partial pivoting* looks through the current column and permutes rows of the matrix so that the element in that column with the largest absolute value appears on the diagonal.

2. *Full pivoting* iterates over the **entire** matrix and permutes rows and columns to place the largest possible value on the diagonal. Permuting columns of a matrix is a valid operation after some added bookkeeping: it corresponds to changing the labeling of the variables in the system, or post-multiplying A by a permutation.

Gaussian Elimination (The Algorithm)

Pivoting (Example)

Example 3.4 (Pivoting)**.** Suppose after the first iteration of Gaussian elimination we are left with the following matrix:

$$\begin{pmatrix} 1 & 10 & -10 \\ 0 & \textcircled{0.1} & 9 \\ 0 & 4 & 6.2 \end{pmatrix}.$$

Applying partial pivoting:

$$\begin{pmatrix} 1 & 10 & -10 \\ 0 & \textcircled{4} & 6.2 \\ 0 & 0.1 & 9 \end{pmatrix}$$

Faster speed

Applying full pivoting (rarely needed):

$$\begin{pmatrix} 1 & -10 & 10 \\ 0 & \textcircled{9} & 0.1 \\ 0 & 6.2 & 4 \end{pmatrix}$$

Better numerical stability

Table of Content

LU Factorization

Motivation

- Sometimes we want to solve for a sequence of linear systems with the same coefficient matrix:
  - $A\mathbf{x} = \mathbf{b}_1$, $A\mathbf{x} = \mathbf{b}_2$, …, $A\mathbf{x} = \mathbf{b}_n$
- Solving the system is $O(n^3)$, can we do some precomputation on A to speedup the solve?
- Computing A^{-1} may not be stable, and sparse matrices often have a dense inverse matrix
- Backward-substitution on upper triangular system is actually $O(n^2)$!
- Precompute forward-substitution?

$$\left(\begin{array}{cccc|c} 1 & \times & \times & \times & \times \\ 0 & 1 & \times & \times & \times \\ 0 & 0 & 1 & \times & \times \\ \textcircled{0} & \textcircled{0} & \textcircled{0} & 1 & \times \end{array}\right) \quad \left(\begin{array}{cccc|c} 1 & \times & \times & 0 & \times \\ 0 & 1 & \times & 0 & \times \\ \textcircled{0} & \textcircled{0} & 1 & \textcircled{0} & \times \\ 0 & 0 & 0 & 1 & \times \end{array}\right)$$

LU Factorization

Idea

- We will use Gaussian elimination to factorize A as $A = LU$.

Upper triangular ↓

↑ Lower triangular

- Then $A\mathbf{x} = \mathbf{b}$ becomes $LU\mathbf{x} = \mathbf{b}$.
- This can be solved by:
  - forward-substitution on $L\mathbf{y} = \mathbf{b}$, followed by
  - backward-substitution on $U\mathbf{x} = \mathbf{y}$.
  - They are both triangular systems, and thus $O(n^2)$.

LU Factorization

Construction

- After forward-substitution with row operations $M_1, M_2, \ldots, M_k$:

$$
\begin{aligned}
M_k \cdots M_1 A &= U \\
\implies A &= (M_k \cdots M_1)^{-1} U \\
&= (M_1^{-1} M_2^{-1} \cdots M_k^{-1}) U \text{ from the fact } (AB)^{-1} = B^{-1} A^{-1} \\
&\coloneqq LU, \text{ if we make the definition } L \coloneqq M_1^{-1} M_2^{-1} \cdots M_k^{-1}.
\end{aligned}
$$

- U is upper triangular by design, to show that L is lower triangular:
  - M_i is either a scaling matrix, or $M_i = I + c\mathbf{e}_l \mathbf{e}_k^T$, where $l > k$ (lower triangular)
  - Thus M_i^{-1} is either a scaling matrix, or $M_i^{-1} = I - c\mathbf{e}_l \mathbf{e}_k^T$, where $l > k$ (lower triangular)
  - We just need to proof that the product of lower triangular matrices are still lower triangular

LU Factorization

Product of Triangular Matrices

Proposition 3.1. The product of two or more upper-triangular matrices is upper triangular, and the product of two or more lower-triangular matrices is lower triangular.

Proof. Suppose A and B are upper triangular, and define $C \coloneqq AB$. By definition of upper-triangular matrices, $a_{ij} = 0$ and $b_{ij} = 0$ when $i > j$. Fix two indices i and j with $i > j$. Then,

$$\begin{aligned} c_{ij} &= \sum_k a_{ik} b_{kj} \text{ by definition of matrix multiplication} \\ &= a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}. \end{aligned}$$

The first $i - 1$ terms of the sum are zero because A is upper triangular, and the last $n - j$ terms are zero because B is upper triangular. Since $i > j$, $(i - 1) + (n - j) > n - 1$ and hence all n terms of the sum over k are zero, as needed.

If A and B are lower triangular, then $A^\top$ and $B^\top$ are upper triangular. By our proof above, $B^\top A^\top = (AB)^\top$ is upper triangular, showing that AB is lower triangular.

LU Factorization

Implementation

Let's examine what happens when we multiply two elimination matrices:

$$(I_{n\times n} - c_\ell \mathbf{e}_\ell \mathbf{e}_k^\top)(I_{n\times n} - c_p \mathbf{e}_p \mathbf{e}_k^\top) = I_{n\times n} - c_\ell \mathbf{e}_\ell \mathbf{e}_k^\top - c_p \mathbf{e}_p \mathbf{e}_k^\top.$$

- The remaining term vanishes by orthogonality of the standard basis vectors $\mathbf{e}_i$ since $k \neq p$.
- Thus, the product of elimination matrices that forward-substitutes a single pivot has the form:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \textcircled{1} & 0 & 0 \\ 0 & \times & 1 & 0 \\ 0 & \times & 0 & 1 \end{pmatrix}$$

- Products of this kind of matrices performed in forward-substitution order combine the values below the diagonal:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 5 & 1 & 0 \\ 0 & 6 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 7 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 5 & 1 & 0 \\ 4 & 6 & 7 & 1 \end{pmatrix}$$

LU Factorization

Implementation (Pseudo-Code)

- Without scaling A, L's diagonal entries are all 1 — no need to explicitly store.

- Thus, we can factorize A in place, the upper triangular part becomes U, and we store L below the diagonal:

function LU-FACTORIZATION-COMPACT(A)
▷ Factors $A \in \mathbb{R}^{n \times n}$ to $A = LU$ in compact format.

for $p \leftarrow 1, 2, \ldots, n$ ▷ Choose pivots like in forward-substitution
for $r \leftarrow p + 1, \ldots, n$ ▷ Forward-substitution row
$s \leftarrow -a_{rp}/a_{pp}$ ▷ Amount to scale row p for forward-substitution
$a_{rp} \leftarrow -s$ ▷ L contains $-s$ because it reverses the forward-substitution
for $c \leftarrow p + 1, \ldots, n$ ▷ Perform forward-substitution
$a_{rc} \leftarrow a_{rc} + s a_{pc}$
return A

LU Factorization

Implementation (Pivoting)

Following the construction in §3.5.1, if we pivot by swapping *columns* of our matrix, in effect we have factored

$$M_k \cdots M_1 A P_1 \cdots P_\ell = U,$$

where the P_i's are permutation matrices used to swap columns. Rearranging this expression leads to a factorization $A = LUP$, where $P = P_\ell^\top \cdots P_1^\top$ is a permutation matrix.

Solving linear systems of equations with this expanded LU factorization is no more difficult asymptotically. In particular, rewriting $A\mathbf{x} = \mathbf{b}$ as $(LUP)\mathbf{x} = \mathbf{b}$ suggests the following algorithm to find $x = A^{-1}b = P^\top U^{-1} L^{-1}$:

1. Solve $L\mathbf{y} = \mathbf{b}$ for y using forward substitution ($O(n^2)$ time).
2. Solve $U\mathbf{z} = y$ for $\mathbf{z}$ using back substitution ($O(n^2)$ time).
3. Permute to compute $\mathbf{x} = P^\top \mathbf{z}$ ($O(n)$ time).

LU Factorization

Implementation (Pivoting Remarks)

- If we pivot rows, then $A = PLU$ with different P, L, and U from column pivoting.
- Pivoting both rows and columns leads to $A = PLUQ$ for two permutations P and Q.
- Any square matrix A (even not invertible) has the above factorizations.
- Beyond stability, some algorithms design P and Q to improve sparsity of L and U, i.e., the number of nonzero entries need to be stored in the matrices.

Table of Content