

# Mathematical Foundations of Computing

Serhii Denysov

February 2026

## One-dimensional space - 1D

- What is an equation in 1D?
- What is a linear equation in 1D?
- Is it hard to solve a linear equation in 1D?
- Geometrical interpretation
- What is a system of equations?

- What is an equation in 1D?

- What is an equation in 1D?

**Find  $x \in \mathbb{R}$  such that  $f(x) = 0$**

- What is a linear equation in 1D?

- What is an equation in 1D?

**Find**  $x \in \mathbb{R}$  **such that**  $f(x) = 0$

- What is a linear equation in 1D?

**Find**  $x \in \mathbb{R}$  **such that**  $f(x) = 0$ , **where**  $f(x) = ax + b$ ;  $a, b \in \mathbb{R}$

- Is it hard to solve a linear equation in 1D?

- What is an equation in 1D?

**Find  $x \in \mathbb{R}$  such that  $f(x) = 0$**

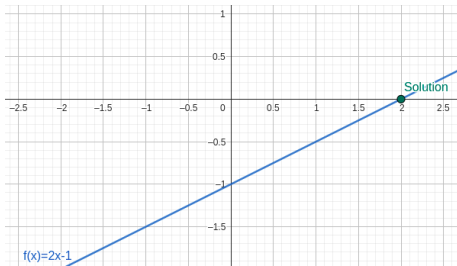
- What is a linear equation in 1D?

**Find  $x \in \mathbb{R}$  such that  $f(x) = 0$ , where  $f(x) = ax + b$ ;  $a, b \in \mathbb{R}$**

- Is it hard to solve a linear equation in 1D?

**Closed form solution  $x = -\frac{b}{a}$**

- Geometrical interpretation



- What is a system of equations?

**Several equations to be solved together. Boring in 1D.**

- What about 2D?

- What is a system of equations?  
**Several equations to be solved together. Boring in 1D.**

- What about 2D?

$$\begin{cases} 2x + 5y = 8, \\ 2x - y = 1. \end{cases}$$

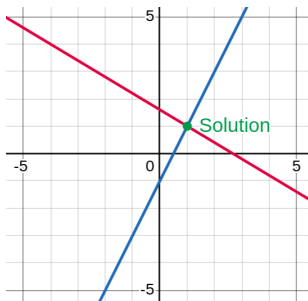
### **Algebraic interpretation:**

- We have two unknowns:  $x$  and  $y$ .
- We need to find their values such that both equations are satisfied.
- Here we can “easily” guess, that  $x^* = 1, y^* = 1$  is a solution.

$$\begin{cases} 3x + 5y = 8, \\ 2x - y = 1. \end{cases}$$

### Geometric interpretation:

- Each equation corresponds to a straight line.
- A solution  $(x^*, y^*)$  is the intersection point of these lines.



## Geometric Picture:

- If the lines intersect at unique point, there is a unique solution.
- If the lines are parallel (no intersection), there is no solution.
- If the lines overlap (coincide), there are infinitely many solutions.

## Solving by substitution or elimination in 2D:

- **Substitution:** Solve one equation for  $y$  in terms of  $x$ , and substitute.
- **Elimination:** Combine equations to remove one variable at a time.

## Generic 2D SLE:

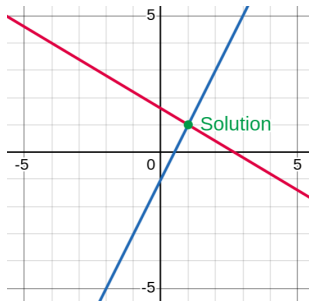
$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1, \\ a_{21}x_1 + a_{22}x_2 = b_2. \end{cases}$$

## Gaussian Elimination

- 1 Choose a pivot equation (e.g., the first).
- 2 Multiply that equation by a factor  $k_{12} = -\frac{a_{21}}{a_{11}}$ . If you add it to the second equation,  $x_1$  variable will be eliminated.
- 3 Solve for  $x_2$ , then back-substitute to find  $x_1$ .

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1, \\ a_{21}x_1 + a_{22}x_2 = b_2. \end{cases}$$

---



**Why we call it 2D?**

# Vectors

- **1D**:  $x \in \mathbb{R}$
- **2D**:  $x \in \mathbb{R}^2$ ;  $x = (x_1, x_2)$
- **nD**:  $x \in \mathbb{R}^n$ ;  $x = (x_1, \dots, x_n)$

Often we distinguish between row vectors and column vectors, to make notation more consistent:

$$x^T = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Solve **system of linear equations (SLE)**  $\iff$  find a vector  $x \in \mathbb{R}^n$ :

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{cases}$$

# Matrix Definition

- A **matrix** of size  $m \times n$  is a rectangular array of numbers:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

with  $m$  rows and  $n$  columns.

- A row of  $m \times n$  matrix has  $n$  elements, and is a **row vector**, a column has  $m$  entries and is a **column vector**.

# Matrix Addition and Scalar Multiplication

- Matrices of the same size can be added elementwise:

$$(A + B)_{ij} = a_{ij} + b_{ij}.$$

- Multiplication by a scalar  $\alpha \in \mathbb{R}$ :

$$(\alpha A)_{ij} = \alpha a_{ij}.$$

- For  $A \in \mathbb{R}^{m \times n}$  and  $B \in \mathbb{R}^{n \times p}$ , the product  $C = AB$  is an  $m \times p$  matrix,  $C = (c_{ij})$ :

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

- The operation is defined only if the number of columns of  $A$  equals the number of rows of  $B$ .
- Note: matrix multiplication is **not commutative** in general.

# Matrix–Vector Notation

## Why introduce matrices and vectors?

- It's a concise way to write a large system.
- It organizes the coefficients and variables into arrays.
- Leads to efficient algorithms and a rich theory in linear algebra.

**Standard Form** - the same system as above:

$$\mathbf{A} \mathbf{x} = \mathbf{b},$$

where

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

# Gaussian elimination in $n$ dimensions

## Key ideas:

- Same elimination approach: use row operations to systematically eliminate variables.
- The process is repeated from the first row to the last, “sweeping” down to get a triangular form, then back-substitute.
- What if we encounter 0 at the leading (pivot) position?

# Summary of First Steps in Solving SLR

- **2D viewpoint:** lines in the plane, intersection = solution.
- **Algebraic approach:** elimination or substitution.
- **Naive Gaussian elimination:** use row operations to systematically eliminate one variable at a time.
- **Extension to  $n$ -dimensions:** same principle, but repeated to handle all variables.
- **Matrix form  $\mathbf{Ax} = \mathbf{b}$ :** more compact notation to handle large systems.

## Example: A 3x3 System with unique solution

**System of Linear Equations (SLR):**

$$\begin{cases} x_1 + x_2 + x_3 = 6, \\ 3x_1 + 2x_2 + x_3 = 10, \\ 2x_1 - x_2 + 4x_3 = 12. \end{cases}$$

**Write the system as an augmented matrix:**

$$\left[ \begin{array}{ccc|c} 1 & 1 & 1 & 6 \\ 3 & 2 & 1 & 10 \\ 2 & -1 & 4 & 12 \end{array} \right].$$

**Goal:** Use row operations to reduce it to an upper triangular form, and then back-substitute.

## Gaussian Elimination: Step 1 (Pivot in row 1)

$$\left[ \begin{array}{ccc|c} 1 & 1 & 1 & 6 \\ 3 & 2 & 1 & 10 \\ 2 & -1 & 4 & 12 \end{array} \right].$$

Use the **pivot**  $a_{11} = 1$  to eliminate  $x_1$  from rows 2 and 3:

$$R_2 \leftarrow R_2 - 3R_1,$$

$$R_3 \leftarrow R_3 - 2R_1.$$

So, we get:

$$\left[ \begin{array}{ccc|c} 1 & 1 & 1 & 6 \\ 0 & -1 & -2 & -8 \\ 0 & -3 & 2 & 0 \end{array} \right]$$

## Gaussian Elimination: Step 2 (Pivot in row 2)

$$\left[ \begin{array}{ccc|c} 1 & 1 & 1 & 6 \\ 0 & -1 & -2 & -8 \\ 0 & -3 & 2 & 0 \end{array} \right]$$

Use the pivot  $a_{22} = -1$  to eliminate  $x_2$  from row 3:

$$R_3 \leftarrow R_3 - 3R_2.$$

$$\left[ \begin{array}{ccc|c} 1 & 1 & 1 & 6 \\ 0 & -1 & -2 & -8 \\ 0 & 0 & 8 & 24 \end{array} \right]$$

Now we have an **upper triangular system**.

## Back Substitution

$$\left[ \begin{array}{ccc|c} 1 & 1 & 1 & 6 \\ 0 & -1 & -2 & -8 \\ 0 & 0 & 8 & 24 \end{array} \right]$$

This corresponds to the system:

$$\begin{cases} x_1 + x_2 + x_3 = 6, \\ -x_2 - 2x_3 = -8, \\ 8x_3 = 24. \end{cases}$$

Solve from the bottom:

$$8x_3 = 24 \Rightarrow x_3 = 3.$$

$$-x_2 - 2(3) = -8 \Rightarrow -x_2 - 6 = -8 \Rightarrow x_2 = 2.$$

$$x_1 + 2 + 3 = 6 \Rightarrow x_1 = 1.$$

## Solution and Quick Check

$$x = (x_1, x_2, x_3) = (1, 2, 3).$$

**Check in the original equations:**

$$1 + 2 + 3 = 6 \checkmark$$

$$3 \cdot 1 + 2 \cdot 2 + 3 = 3 + 4 + 3 = 10 \checkmark$$

$$2 \cdot 1 - 2 + 4 \cdot 3 = 2 - 2 + 12 = 12 \checkmark$$

---

**Let's code it!**

---

# When does a system have a solution? (2D intuition)

**In 2D, each linear equation is a line.**

- **Unique solution:** two lines intersect at exactly one point.
- **No solution:** lines are parallel.
- **Infinitely many solutions:** lines coincide (same line).

**In higher dimensions:** same idea, but with planes / hyperplanes.  
Solutions depend on how these objects intersect.

# Gaussian elimination actually answers the question

Gaussian elimination transforms the augmented matrix

$$[\mathbf{A} \mid \mathbf{b}]$$

by **row operations** (which do not change the solution set).

After elimination, three outcomes are possible:

- **Consistent + pivots everywhere  $\Rightarrow$  unique solution.**
- **Inconsistent row appears  $\Rightarrow$  no solutions.**
- **Consistent but some free variables  $\Rightarrow$  infinitely many solutions.**

# Unique solution: pivot in every variable

After elimination (row echelon form), we might get:

$$\left[ \begin{array}{ccc|c} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{array} \right]$$

## Interpretation:

- There is a pivot (leading nonzero) in each column of variables.
- No free variables  $\Rightarrow$  exactly one solution.

## Equivalent linear algebra statement:

$$\det(\mathbf{A}) \neq 0 \quad \Leftrightarrow \quad \mathbf{A} \text{ is invertible} \quad \Leftrightarrow \quad \text{unique solution for every } \mathbf{b}.$$

# No solution: inconsistent row appears

Sometimes elimination produces a row like:

$$\left[ \begin{array}{ccc|c} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & 0 & 1 \end{array} \right]$$

The last row means:

$$0x_1 + 0x_2 + 0x_3 = 1,$$

which is impossible.

**Conclusion:** the system is **inconsistent**  $\Rightarrow$  **no solutions**.

# Infinitely many solutions: free variable(s)

If elimination produces something like:

$$\left[ \begin{array}{ccc|c} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & 0 & 0 \end{array} \right]$$

## Interpretation:

- The last row means  $0 = 0$  (no new information).
- At least one variable does not have a pivot  $\Rightarrow$  **free variable**.
- Free variable(s)  $\Rightarrow$  infinitely many solutions (a line/plane/etc. of solutions).

# Rank criterion

Let  $\text{rank}(\mathbf{A})$  be the number of pivots in  $\mathbf{A}$ , and  $\text{rank}([\mathbf{A}|\mathbf{b}])$  the number of pivots in the augmented matrix.

**No solutions**  $\iff \text{rank}(\mathbf{A}) < \text{rank}([\mathbf{A}|\mathbf{b}])$

**At least one solution**  $\iff \text{rank}(\mathbf{A}) = \text{rank}([\mathbf{A}|\mathbf{b}])$

**Unique solution**  $\iff \text{rank}(\mathbf{A}) = \text{rank}([\mathbf{A}|\mathbf{b}]) = n$

**Infinite solutions**  $\iff \text{rank}(\mathbf{A}) = \text{rank}([\mathbf{A}|\mathbf{b}]) < n$

**Here  $n$  is the number of unknowns.**

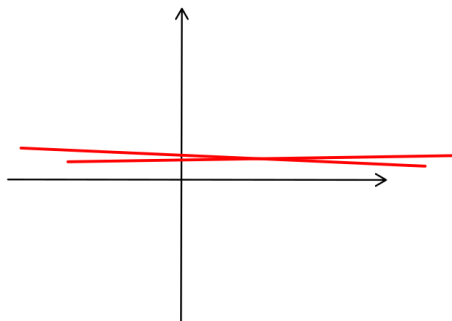
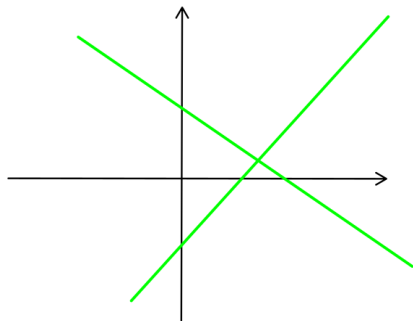
# Good or bad?

## Good versus Evil

- Are all SLEs with an unique solution are equally good?
- Let's come with idea of a “bad” solvable system!
- Easiest intuition is based on geometry.

# Good / bad SLE geometry

- Well-conditioned SLE - nearly orthogonal lines
- Ill-conditioned SLE - nearly parallel lines



# Hilbert matrix

Hilbert matrix:

$$H_n = (h_{ij})_{i,j=1}^n \quad \text{with} \quad h_{ij} = \frac{1}{i+j-1}.$$

$$H_4 = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}.$$

Let's test how our algorithm works here (try different sizes!)

# Condition Number of a Matrix

- For the linear system  $Ax = b$ , the *condition number* quantifies how errors in  $A$  or  $b$  are amplified in the solution  $x$ .
- One can show

$$\frac{\|\delta x\|}{\|x\|} \leq K(A) \frac{\|\delta b\|}{\|b\|}, \quad K(A) = \|A\| \|A^{-1}\|.$$

- By definition  $K(A) \geq 1$ : the closer to 1, the *better-conditioned*; large  $K \Rightarrow$  *ill-conditioned*.
- But what is a **norm**?

# Vector and Matrix Norms

Recall: **vector norm**  $\|\cdot\|$  on  $\mathbb{R}^n$  assigns a length to each vector and satisfies positivity, homogeneity and triangle inequality. Widely used norms:

- $\|\mathbf{x}\|_1 = \sum_i |x_i|$  (1-norm).
- $\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$  (Euclidean norm).
- $\|\mathbf{x}\|_\infty = \max_i |x_i|$  (infinity norm).

A **matrix norm**  $\|A\|$  measures the "magnitude" of a matrix. An *induced* (or operator) norm is defined by

$$\|A\| = \sup_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$$

for a given vector norm – so, every vector norm produces a matrix norm this way. It measures how much  $A$  stretches vectors.

# Common Induced Matrix Norms

- **1-norm:**  $\|A\|_1 = \max_j \sum_i |a_{ij}|$  — the maximum absolute column sum.
- **Infinity-norm:**  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$  — the maximum absolute row sum.
- **Spectral (2-)norm:**  $\|A\|_2 = \sqrt{\lambda_{\max}(A^\top A)} = \sigma_{\max}$ , the largest singular value (eigenvalue of the matrix  $A^\top A$ ).

These norms are *submultiplicative*, meaning  $\|AB\| \leq \|A\| \|B\|$  and they satisfy  $\|I\| = 1$ , where  $I$  is identity matrix.

# Hilbert Matrix Condition Numbers

- So,  $H_n = \frac{1}{i+j-1}$  is famously ill-conditioned.
- Asymptotically,  $K(H_n) \sim O(e^{3.5n}/\sqrt{n})$ .
- Feel the evil:

$$K(H_4) \approx 1.55 \times 10^4, \quad K(H_8) \approx 1.5 \times 10^{10}.$$

# Pros and cons of Gaussian elimination

- Is it fast?
- Is it universal?
- Is it easy to implement?
- What else can we tell?

# Pros and cons of Gaussian elimination

- $O(n^3)$  asymptotic complexity. Problematic for very large systems.
- Direct (non-iterative) algorithm.
- Solves any system with unique solution, stops with obvious error if it has no or infinite number of solutions.

## **What about other algorithms?**

- Other direct methods
- Iterative algorithms. Can it be useful?

Let's consider our SLE solution search problem again:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{cases}$$

**Goal:** Find the vector  $\mathbf{x}^* = (x_1^*, x_2^*, \dots, x_n^*)$  that satisfies all these equations simultaneously.

### Common Approaches:

- **Direct Methods:** e.g. Gaussian elimination. We find a solution in a finite number of steps. Exact solution if calculations are accurate.
- **Iterative Methods:** Build a sequence of vectors  $\mathbf{x}^k$  that converges to the true solution  $\mathbf{x}^*$ .

## Jacobi Iteration: naive $2 \times 2$ Form

Consider the  $2 \times 2$  system

$$\begin{cases} a_{11} x_1 + a_{12} x_2 = b_1, \\ a_{21} x_1 + a_{22} x_2 = b_2. \end{cases}$$

And let's assume we have an initial approximation (initial guess of the solution vector):

$$x^{(0)} = (x_1^{(0)}, x_2^{(0)})$$

In Jacobi's algorithm we “solve each row for its own variable,” using only the previous approximation:

$$\begin{cases} a_{11} x_1^{(k+1)} + a_{12} x_2^{(k)} = b_1, \\ a_{21} x_1^{(k)} + a_{22} x_2^{(k+1)} = b_2, \end{cases}$$

so

$$x_1^{(k+1)} = \frac{b_1 - a_{12} x_2^{(k)}}{a_{11}}, \quad x_2^{(k+1)} = \frac{b_2 - a_{21} x_1^{(k)}}{a_{22}}.$$

# Jacobi Iteration: general $n$ -dimensional Form

For a general system  $Ax = b$ ,  $A = (a_{ij}) \in \mathbb{R}^{n \times n}$ , define the iterate

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

You start from any initial guess  $x^{(0)}$ , then update all components “in parallel.”

Here  $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})$  is an approximate solution vector after  $k$  iterations.

**Will it always converge to the solution?**

# Jacobi convergence requirements

**Sufficient** convergence conditions – **diagonal dominance**:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \quad \text{for all } i = 1 \dots n$$

**Why such a condition?** *Open question for now...*

# Jacobi Algorithm: Pseudocode

- 1 Choose initial vector  $x^{(0)}$  and error tolerance  $\varepsilon$ .
- 2 Starting with  $k = 0$ , iterate

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}, \quad i = 1 \dots n$$

- 3 Stop when  $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$ .

Note: all updates use only the previous vector  $x^{(k)}$ , making it easy to parallelize!

# Matrix notation

As always, the matrix of our SLE is:

$$A = (a_{ij})_{n \times n} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix},$$

We can always decompose (split) the matrix  $A$  in the next way:

$$A = D + L + U,$$

where

- $D$  is the diagonal part (main diagonal),
- $L$  is the strictly lower-triangular part,
- $U$  is the strictly upper-triangular part.

## Exact forms of $D$ , $L$ , and $U$

$$L = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{pmatrix}, \quad U = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

$$D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

We can directly check, that  $A = D + L + U$ .

# Jacoby in matrix form

With such a decomposition, Jacobi method could be written as:

$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)}),$$

- *What is  $D^{-1}$  ?*
- Let's check the formula!
- Let's code with numpy!

# Jacoby in matrix form

With such a decomposition, Jacobi method could be written as:

$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)}),$$

- *What is  $D^{-1}$  ?*
- Let's check the formula!
- Let's code with numpy!
- **Does it resemble something?**

# Jacoby in matrix form

With such a decomposition, Jacobi method could be written as:

$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)}),$$

- *What is  $D^{-1}$  ?*
- Let's check the formula!
- Let's code with numpy!
- **Does it resemble something?**

It's a particular case of the next iterative procedure:

$$x^{(k+1)} = G(x^{(k)}), \quad x^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})$$

We have a **fixed point iteration** algorithm in  $n$  dimensions!

$G$  is fully determined by the matrix of our SLE. So, we can try to find properties of  $A$  required for convergence, using the technique similar to one dimensional case!

## Fixed point in $\mathbb{R}^n$ : what does “contraction” mean?

We iterate

$$x^{(k+1)} = G(x^{(k)}), \quad G(x) = D^{-1}(b - (L + U)x).$$

**In 1D:**  $|g'(x)| \leq L < 1$  implies contraction.

**In  $n$ D:** choose a norm  $\|\cdot\|$  and require

$$\|G(x) - G(y)\| \leq q \|x - y\|, \quad \text{for all } x, y, \quad \text{with } q < 1.$$

## Fixed point in $\mathbb{R}^n$ : what does “contraction” mean?

We iterate

$$x^{(k+1)} = G(x^{(k)}), \quad G(x) = D^{-1}(b - (L + U)x).$$

**In 1D:**  $|g'(x)| \leq L < 1$  implies contraction.

**In  $n$ D:** choose a norm  $\|\cdot\|$  and require

$$\|G(x) - G(y)\| \leq q \|x - y\|, \quad \text{for all } x, y, \quad \text{with } q < 1.$$

If  $G$  is a contraction, then the iteration converges to the unique fixed point (vector)  $x^* \in \mathbb{R}^n$ .

Jacobi as an affine map:  $G(x) = Bx + c$

Rewrite Jacobi:

$$x^{(k+1)} = D^{-1}b - D^{-1}(L + U)x^{(k)}.$$

So

$$G(x) = Bx + c, \quad B = -D^{-1}(L + U), \quad c = D^{-1}b.$$

## Jacobi as an affine map: $G(x) = Bx + c$

Rewrite Jacobi:

$$x^{(k+1)} = D^{-1}b - D^{-1}(L + U)x^{(k)}.$$

So

$$G(x) = Bx + c, \quad B = -D^{-1}(L + U), \quad c = D^{-1}b.$$

Then the difference of two images is

$$G(x) - G(y) = B(x - y).$$

So contraction reduces to a matrix norm condition:

$$\|G(x) - G(y)\| = \|B(x - y)\| \leq \|B\| \|x - y\|.$$

**Sufficient condition for convergence:**  $\|B\| < 1$

Choose a convenient norm:  $\|\cdot\|_\infty$

We will use the infinity norm:

$$\|x\|_\infty = \max_i |x_i|.$$

Induced matrix norm:

$$\|B\|_\infty = \max_i \sum_{j=1}^n |b_{ij}| \quad (\text{maximum absolute row sum}).$$

Choose a convenient norm:  $\|\cdot\|_\infty$

We will use the infinity norm:

$$\|x\|_\infty = \max_i |x_i|.$$

Induced matrix norm:

$$\|B\|_\infty = \max_i \sum_{j=1}^n |b_{ij}| \quad (\text{maximum absolute row sum}).$$

For Jacobi,

$$B = -D^{-1}(L + U) \quad \Rightarrow \quad b_{ij} = \begin{cases} 0, & i = j, \\ -\frac{a_{ij}}{a_{ii}}, & i \neq j. \end{cases}$$

Therefore,

$$\|B\|_\infty = \max_i \sum_{j \neq i} \left| \frac{a_{ij}}{a_{ii}} \right| = \max_i \frac{\sum_{j \neq i} |a_{ij}|}{|a_{ii}|}.$$

# Diagonal dominance $\Rightarrow$ contraction $\Rightarrow$ convergence

If for every row  $i$  we have **strict diagonal dominance**

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|,$$

then

$$\frac{\sum_{j \neq i} |a_{ij}|}{|a_{ii}|} < 1 \quad \Rightarrow \quad \| -D^{-1}(L + U) \|_{\infty} < 1.$$

# Diagonal dominance $\Rightarrow$ contraction $\Rightarrow$ convergence

If for every row  $i$  we have **strict diagonal dominance**

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|,$$

then

$$\frac{\sum_{j \neq i} |a_{ij}|}{|a_{ii}|} < 1 \quad \Rightarrow \quad \| -D^{-1}(L + U) \|_{\infty} < 1.$$

So  $G$  is a contraction in  $\| \cdot \|_{\infty}$ , and Jacobi converges!

# Convergence speed of Jacobi

Let  $x^*$  be the true solution (fixed point):  $x^* = G(x^*)$ . Define the error  $e^{(k)} = x^{(k)} - x^*$ .

Then

$$e^{(k+1)} = x^{(k+1)} - x^* = G(x^{(k)}) - G(x^*) = B(x^{(k)} - x^*) = Be^{(k)}.$$

And

$$\|e^{(k+1)}\| \leq \|B\| \|e^{(k)}\| \leq \|B\|^{k+1} \|e^{(0)}\|.$$

So if  $\|B\| = q < 1$ , the error decreases with the same speed as in 1D:

$$\|e^{(k)}\| \leq q^k \|e^{(0)}\|.$$

## Check Jacobi convergence via $\|B\|_\infty$

Consider the system  $Ax = b$  with

$$A = \begin{pmatrix} 10 & -1 & 2 \\ 1 & 8 & -1 \\ -1 & 1 & 5 \end{pmatrix}, \quad b = \begin{pmatrix} 11 \\ 8 \\ 5 \end{pmatrix}.$$

- **Check convergence condition**
- Estimate, how many iterations we need to achieve precision  $\varepsilon = 0.01$  in terms of infinity norm of the error  $\|\cdot\|_\infty$

## Another iterative algorithm



**Zendaya?** Seidel!

**Idea:** immediately reuse new values.

$$\begin{cases} a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} = b_1, \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} = b_2. \end{cases}$$

# Seidel (Gauss–Seidel) Iteration

Consider the  $2 \times 2$  system

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1, \\ a_{21}x_1 + a_{22}x_2 = b_2. \end{cases}$$

**Idea:** update variables *sequentially* and immediately reuse new values.

$$\begin{cases} a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} = b_1, \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} = b_2. \end{cases}$$

So

$$x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k)}}{a_{11}}, \quad x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k+1)}}{a_{22}}.$$

# Seidel (Gauss–Seidel) Iteration

Consider the  $2 \times 2$  system

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1, \\ a_{21}x_1 + a_{22}x_2 = b_2. \end{cases}$$

**Idea:** update variables *sequentially* and immediately reuse new values.

$$\begin{cases} a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} = b_1, \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} = b_2. \end{cases}$$

So

$$x_1^{(k+1)} = \frac{b_1 - a_{12}x_2^{(k)}}{a_{11}}, \quad x_2^{(k+1)} = \frac{b_2 - a_{21}x_1^{(k+1)}}{a_{22}}.$$

**Difference from Jacobi:** the second equation uses  $x_1^{(k+1)}$  (fresh value), not  $x_1^{(k)}$ .

# Gauss–Seidel Iteration: general $n$ -dimensional Form

For  $Ax = b$ , Gauss–Seidel computes components one by one:

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

- Uses **new** values  $x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}$  as soon as they are available.
- Uses **old** values for the remaining components.

**Question:** When does this converge?

## Gauss–Seidel in matrix form

Recall the splitting  $A = D + L + U$  (diagonal + strictly lower + strictly upper).

Starting from

$$(D + L + U)x = b,$$

move the  $Ux$  term to the right:

$$(D + L)x = b - Ux.$$

## Gauss–Seidel in matrix form

Recall the splitting  $A = D + L + U$  (diagonal + strictly lower + strictly upper).

Starting from

$$(D + L + U)x = b,$$

move the  $Ux$  term to the right:

$$(D + L)x = b - Ux.$$

Gauss–Seidel iteration:

$$(D + L)x^{(k+1)} = b - Ux^{(k)}, \quad \Rightarrow \quad x^{(k+1)} = (D + L)^{-1}(b - Ux^{(k)}).$$

So it is again a fixed point iteration  $x^{(k+1)} = G(x^{(k)})$  with

$$G(x) = (D + L)^{-1}(b - Ux).$$

## Fixed point view: contraction via a matrix norm

Write Gauss–Seidel as an affine map:

$$x^{(k+1)} = B_{GS}x^{(k)} + c, \quad B_{GS} = -(D + L)^{-1}U, \quad c = (D + L)^{-1}b.$$

Then for any  $x, y$ :

$$G(x) - G(y) = B_{GS}(x - y).$$

So for any induced norm,

$$\|G(x) - G(y)\| \leq \|B_{GS}\| \|x - y\|.$$

## Fixed point view: contraction via a matrix norm

Write Gauss–Seidel as an affine map:

$$x^{(k+1)} = B_{GS}x^{(k)} + c, \quad B_{GS} = -(D + L)^{-1}U, \quad c = (D + L)^{-1}b.$$

Then for any  $x, y$ :

$$G(x) - G(y) = B_{GS}(x - y).$$

So for any induced norm,

$$\|G(x) - G(y)\| \leq \|B_{GS}\| \|x - y\|.$$

So. convergence condition:  $\|B_{GS}\| < 1$

And **strict diagonal dominance by rows** again is sufficient!

## Example: compute one Gauss–Seidel iteration ( $3 \times 3$ )

Take the same example:

$$A = \begin{pmatrix} 10 & -1 & 2 \\ 1 & 8 & -1 \\ -1 & 1 & 5 \end{pmatrix}, \quad b = \begin{pmatrix} 11 \\ 8 \\ 5 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

# Jacobi vs Gauss–Seidel

Complexity, applicability, ... ?

# Jacobi vs Gauss–Seidel

Complexity, applicability, ... ?

- **Jacobi:** all components use only  $x^{(k)}$  (fully parallel).
- **Gauss–Seidel:** uses newest values immediately (typically faster convergence).
- **Convergence:** both converge under strict diagonal dominance (sufficient condition).

**Cost per iteration:**

- Both are  $O(n^2)$  for dense matrices, but can be much cheaper for sparse matrices.
- Gauss–Seidel is less parallel, but often needs fewer iterations.