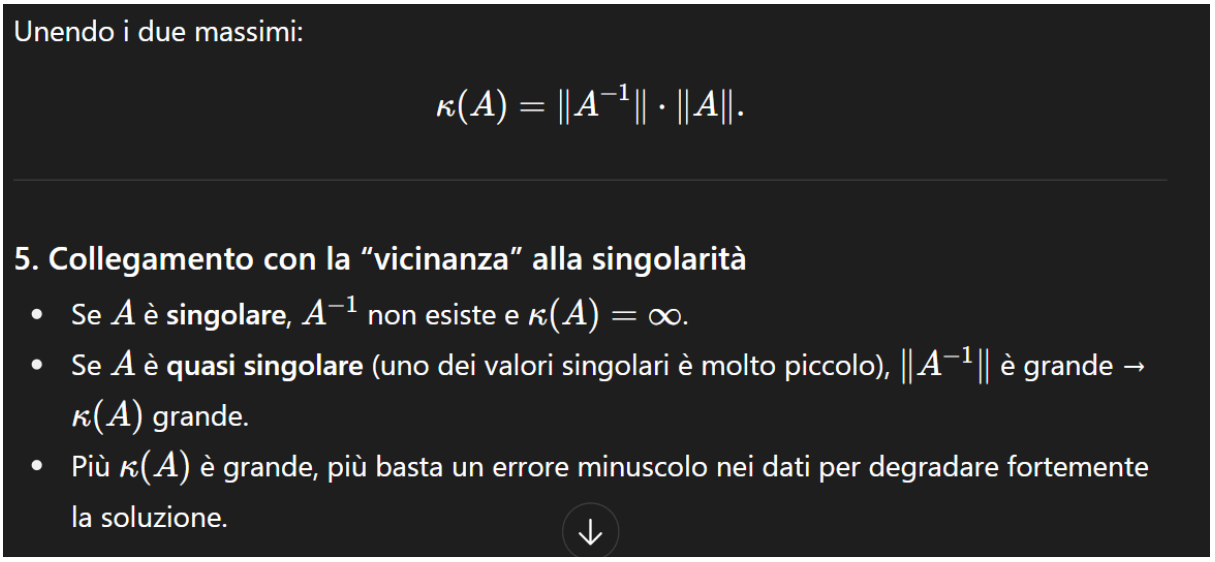
Calcolo (bello) 😊 - per finta 🥹

Fattore di condizionamento

Il condition number misura quanto una matrice sia "vicina" alla singolarità - cioè al non essere risolvibile!
È il **fattore di amplificazione massimo** dell'errore relativo.


Unendo i due massimi:

$$\kappa(A) = \|A^{-1}\| \cdot \|A\|.$$

5. Collegamento con la "vicinanza" alla singolarità

- Se A è **singolare**, A^{-1} non esiste e $\kappa(A) = \infty$.
- Se A è **quasi singolare** (uno dei valori singolari è molto piccolo), $\|A^{-1}\|$ è grande → $\kappa(A)$ grande.
- Più $\kappa(A)$ è grande, più basta un errore minuscolo nei dati per degradare fortemente la soluzione.

▼ Nota bene: Noi usiamo le norme perché assicurano che oggetti microscopici nello spazio (i vettori) siano in grado di trovare la benché minima variazione nel condizionamento

Errore relativo di condizionamento

L'errore relativo **normalizza** rispetto alla grandezza della soluzione.

L'errore relativo cattura la realtà

Caso A: $||\delta x||/||x|| \approx 10^{-6}$ (eccellente precisione)
Caso B: $||\delta x||/||x|| \approx 10^{3}$ (completamente inutile)

L'errore relativo ti dice **quante cifre significative** hai perso.

Perché il condition number usa errori relativi

Il bound fondamentale è:

$||\delta x||/||x|| \leq \kappa(A) \cdot ||\delta b||/||b||$

Questo ti dice: "Se i tuoi dati hanno precisione relativa ε, la soluzione avrà precisione relativa al massimo κ(A)·ε".

L'esempio che spiega tutto

Sistema: **Ax = b** con A = $[10^{-6}, 0; 0, 1]$, b = $[10^{-6}, 1]$
Soluzione: x = [1, 1]

Perturbazione: δb = $[10^{-9}, 10^{-9}]$

Analisi con errore assoluto:

- $||\delta b|| \approx 1.4 \times 10^{-9}$
- $||\delta x|| \approx 1.4 \times 10^{-3}$
- Fattore di amplificazione: $\sim 10^{6}$

Analisi con errore relativo:

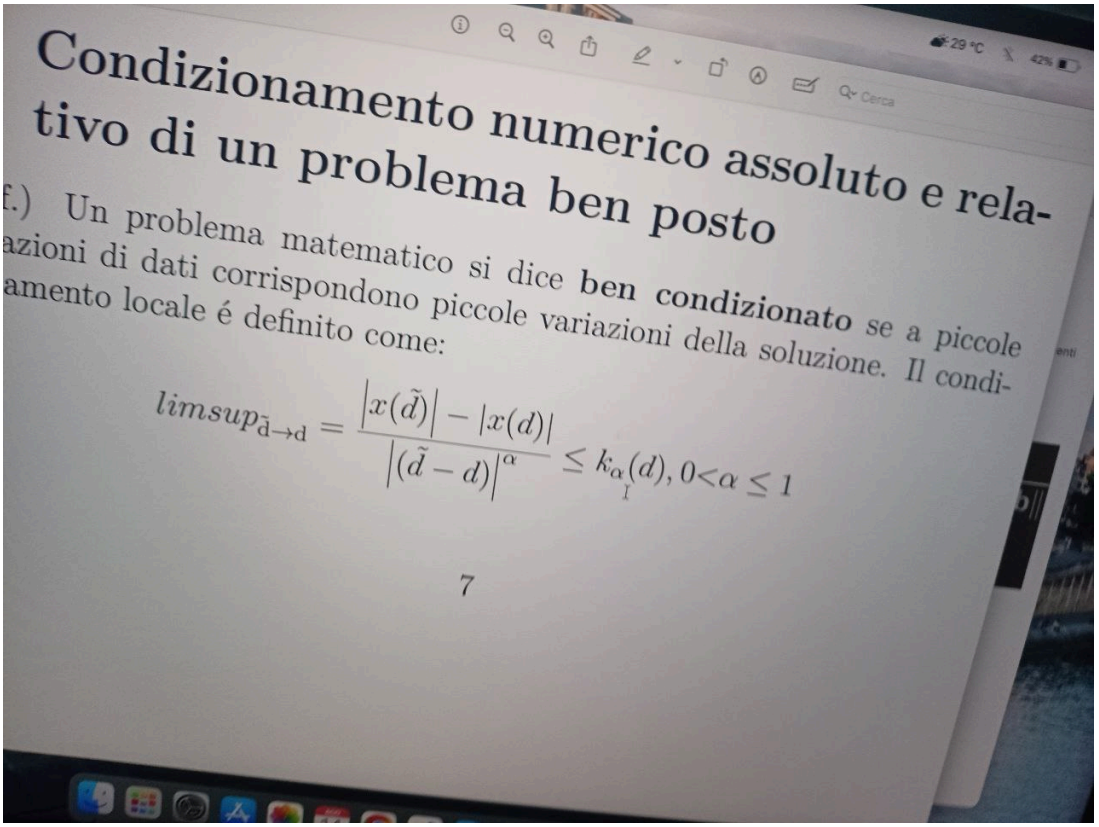
- $||\delta b||/||b|| \approx 1.4 \times 10^{-9}$
- $||\delta x||/||x|| \approx 10^{-3}$
- Fattore di amplificazione: ~1 (κ(A) = 1!)

Il paradosso rivelatore

La matrice A è **perfettamente condizionata** (κ(A) = 1) secondo l'analisi relativa, ma sembra amplificare l'errore di 10^{6} volte secondo l'analisi assoluta.

Chi ha ragione? L'analisi relativa. Il "grande" errore assoluto in δx è in realtà **proporzionalmente corretto** rispetto alla grandezza di x.

▼ Dubbissimo



Ah, questa è la **generalizzazione elegante** del condition number che estende il concetto oltre i sistemi lineari. Ti spiego da dove salta fuori quell'α.

Il problema di fondo

Il condition number classico funziona solo per **problemi lineari**. Ma che succede se hai una funzione generica f(d) e vuoi capire come si propaga l'errore?

L'idea geniale: linearizzazione locale

Per una funzione generale **x = f(d)**, non puoi usare $\kappa(A) = ||A||\,||A^{-1}||$. Però puoi **linearizzare localmente** usando la derivata:

$f(\tilde{d}) \approx f(d) + f'(d)(\tilde{d} - d)$

Quindi: **$\tilde{x} - x \approx f'(d)(\tilde{d} - d)$**

Da dove spunta l'α

L'α nasce dalla necessità di **pesare** errore assoluto vs relativo. La formula generale è:

$\kappa_a(d) = |f'(d)| \times |d|^\alpha / |f(d)|^\alpha$

Dove:

- **α = 0**: condition number **assoluto** (errore assoluto in output vs errore assoluto in input)
- **α = 1**: condition number **relativo** (errore relativo in output vs errore relativo in input)

Il ragionamento matematico

Per α generico, stai misurando:

[Errore in x normalizzato da |x|^α] / [Errore in d normalizzato da |d|^α]

Nel limite:

$$\lim[|\tilde{x} - x|/|x|^\alpha] / [|\tilde{d} - d|/|d|^\alpha] = |f'(d)| \times |d|^\alpha / |f(d)|^\alpha$$

Perché α = 1 è speciale

Con α = 1 ottieni:
$\kappa_1(d) = |f'(d)| \times |d| / |f(d)| = |d\, f'(d) / f(d)| = |d(\ln f(d))'|$

Questo è il **logarithmic condition number** - misura la sensibilità relativa ed è **invariante per scaling**.

Esempio illuminante

Prendi f(d) = d^n:

- **$\kappa_0(d) = |nd^{(n-1)}| = n|d|^{(n-1)}$** (dipende da d!)
- **$\kappa_1(d) = |nd^{(n-1)} \times d / d^n| = n$** (indipendente da d!)

Con α = 1, il condition number diventa una **proprietà intrinseca** della funzione, non del punto specifico.

L'interpretazione fisica

L'α ti permette di scegliere il "peso" che dai all'errore:

- **α = 0**: "Mi interessa l'errore assoluto"
- **α = 1**: "Mi interessa l'errore percentuale"

- **α ∈ (0,1)**: Una via di mezzo pesata

Connessione con i sistemi lineari

Per Ax = b, hai f(A,b) che è lineare. Il condition number relativo (α = 1) ricade esattamente nella definizione κ(A) = ||A|| $||A^{-1}||$.

La potenza del formalismo

Questa generalizzazione ti permette di analizzare il condizionamento di:

- Radici di polinomi
- Autovalori
- Integrali numerici
- Qualsiasi problema computazionale

Domanda stimolante: Se hai un problema dove l'errore assoluto e relativo hanno importanza diversa (es. calcolo di una massa vs una probabilità), quale α sceglieresti e perché?

Se $0 < \alpha < 1$ → **continuità di Hölder**:
la soluzione varia meno regolarmente:

$$|x(\tilde{d})| - |x(d)| \lesssim k_\alpha(d)\, |\tilde{d} - d|^\alpha$$

Significa che anche se $|\tilde{d} - d|$ è piccolo, l'effetto sulla soluzione può essere "più brusco" rispetto al caso lineare, e la relazione non è proporzionale, ma sottolineare.

---

3. Da dove "arriva" α

Matematicamente, α viene da **una generalizzazione del concetto di Lipschitz**:

- Per $\alpha = 1$ → condizionamento "normale" (errore ∝ perturbazione).
- Per $\alpha < 1$ → il problema non è Lipschitz continuo, ma **solo Hölder continuo**, quindi il condizionamento si degrada: piccole perturbazioni possono avere effetti peggiori.

In analisi numerica, questo serve per includere **problemi non lineari o debolmente continui** in un'unica definizione di condizionamento, senza limitarsi alla continuità lineare.

Fattorizzazione LU

La fattorizzazione LU è il **cuore pulsante** di quasi tutti i solver per sistemi lineari. Ti spiego come funziona e perché è così potente.

L'idea di base

Obiettivo: Scomporre una matrice A in due fattori:

- **L** = Lower triangular (triangolare inferiore)
- **U** = Upper triangular (triangolare superiore)

Tale che **A = LU**

Il collegamento con Gauss

La fattorizzazione LU è **esattamente** l'eliminazione di Gauss, ma **organizzata intelligentemente**. Invece di modificare direttamente A, **registri** le operazioni che fai.

L'algoritmo passo-passo

Passo 1: Prima colonna

```
A = [a11  a12  a13]    L = [1   0   0]    U = [u11  u12  u13]
    [a21  a22  a23]        [l21 1   0]        [0    u22  u23]
    [a31  a32  a33]        [l31 l32 1]        [0    0    u33]
```

Prima riga di U: u11 = a11, u12 = a12, u13 = a13
Prima colonna di L: l21 = a21/a11, l31 = a31/a11

Passo 2: Elimina prima colonna

```
A⁽¹⁾ = A - [a21/a11] [a11 a12 a13] = [a11    a12       a13    ]
           [a31/a11]                 [0   a22-l21u12  a23-l21u13]
                                     [0   a32-l31u12  a33-l31u13]
```

Passo 3: Continua ricorsivamente

Applichi lo stesso processo alla sottomatrice 2×2 rimanente.

Il codice essenziale## La magia dietro le quinte

Perché funziona?

Ogni passo dell'eliminazione di Gauss corrisponde a moltiplicare A per una **matrice elementare** M_k:

$$\mathbf{A^{(k)} = M\_k \times A^{(k-1)}}$$

Alla fine: $\mathbf{U = M\_\{n-1\} \times \ldots \times M\_1 \times A}$

Quindi: $\mathbf{A = M\_1^{-1} \times \ldots \times M\_\{n-1\}^{-1} \times U = L \times U}$

Le matrici elementari

$$M_k = \begin{bmatrix} 1 & 0 & 0 \\ -m_1 & 1 & 0 \\ -m_2 & 0 & 1 \end{bmatrix} \quad M_k^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ m_1 & 1 & 0 \\ m_2 & 0 & 1 \end{bmatrix}$$

Le inverse si **accumulano** in L!

I vantaggi computazionali

1. **Costo**: $O(2n^3/3)$ per la fattorizzazione

2. **Multiple RHS**: Fattorizzazione una volta, solve $O(n^2)$ per ogni b

3. **Determinante**: $\det(A) = \prod_i U[i,i]$

4. **Controllo numerico**: Puoi monitorare i pivot

Il problema del pivoting

Senza pivoting: L'algoritmo può fallire o essere instabile se incontri pivot piccoli.

Soluzione: **Partial pivoting** - scambia righe per avere il pivot più grande possibile.

Confronto dei metodi

1. LU senza pivoting (solo Gauss)

Limitazioni

- Serve **a_{kk} ≠ 0** per ogni passo kkk (nessuno zero pivot), altrimenti divisione per zero.
- Anche se akk≠0a_{kk} \neq 0akk≠0, se è **molto piccolo** rispetto agli altri elementi della colonna → rischio di **instabilità numerica** (amplificazione degli errori di arrotondamento).
- Funziona bene senza pivoting solo per matrici che sono **diagonalmente dominanti** o **definite positive** (queste ultime si preferisce decomposizione di Cholesky).

Problematiche numeriche

- **Zero pivot** → algoritmo fallisce.
- **Pivot piccolo** → grandi moltiplicatori → propagazione e amplificazione degli errori floating-point.
- Mancanza di stabilità in senso backward per matrici generiche.

2. LU con pivoting parziale per righe

Idea del pivoting

A ogni passo kkk:

- **Cerca nella colonna kkk**, a partire dalla riga kkk, l'elemento con valore assoluto massimo (**pivot**).
- **Scambia** la riga kkk con la riga che contiene questo elemento.
- Procedi con l'eliminazione usando quel pivot.

Motivazione

- Evitare **divisioni per zero** (pivot nullo).
- Ridurre il rischio di **pivot troppo piccoli** → migliora la stabilità numerica.
- Controllare la crescita degli elementi di UUU (growth factor).

Prima le basi: punti fissi e contrazioni

Punto fisso = dove la funzione "si ferma"

Definizione brutale: x* è un **punto fisso** della funzione f se *f(x) = x***

Esempio visivo: Hai la funzione f(x) = x/2 + 1

- Prova x = 2: f(2) = 2/2 + 1 = 2 ✓
- x = 2 è punto fisso perché "la funzione restituisce se stesso"

Perché ci fregano: Il nostro metodo iterativo è **x^(k+1) = Gx^(k) + c**
Se converge a x*, allora x* = Gx* + c → *x è punto fisso!**

Contrazioni = funzioni che "stringono"

Contrazione: Una funzione f è una contrazione se **"avvicina sempre i punti"**

Matematicamente: **||f(x) - f(y)|| ≤ L||x - y||** con **L < 1**

Esempio concreto: f(x) = x/2 + 1

- Prendi x = 10, y = 0
- f(10) = 6, f(0) = 1
- |f(10) - f(0)| = |6 - 1| = 5
- |10 - 0| = 10
- 5 ≤ (1/2) × 10 ✓ → L = 1/2 < 1

Teorema magico: Ogni contrazione ha **un unico punto fisso** e **qualsiasi successione x^(k+1) = f(x^(k)) converge** a quel punto fisso.

Link con i metodi iterativi: Se G ha norma < 1, allora T(x) = Gx + c è una contrazione → il metodo converge sempre.

I personaggi storici

Carl Gustav Jacob Jacobi (1804-1851)

Chi era: Matematico prussiano, uno dei più grandi dell'800.

Cosa fece:

- Rivoluzionò la teoria delle funzioni ellittiche

- Contributi enormi all'algebra lineare (determinante jacobiano)
- Inventò il metodo che porta il suo nome quasi per caso

Il suo metodo: Risolvi ogni equazione per la sua incognita principale, usando i valori "vecchi" delle altre.

Aneddoto: Non lo inventò per i sistemi lineari, ma per sistemi di equazioni non lineari. L'applicazione ai sistemi lineari fu quasi un effetto collaterale.

Carl Friedrich Gauss (1777-1855)

Chi era: IL matematico. Principe della matematica. Quello della distribuzione normale, del teorema fondamentale dell'algebra, dell'eliminazione gaussiana...

Genio assoluto: A 19 anni dimostrò che il poligono regolare a 17 lati si può costruire con riga e compasso. Inventò praticamente tutto.

Perché è nei metodi iterativi: Lui faceva già eliminazione gaussiana, ma il metodo "di Gauss-Seidel" non è propriamente suo.

Philipp Ludwig von Seidel (1821-1896)

Chi era: Matematico e astronomo tedesco, meno famoso degli altri due ma molto pratico.

Il suo contributo: Migliorò il metodo di Jacobi con un'idea semplicissima: **"perché non usare i valori appena calcolati nella stessa iterazione?"**

Gauss-Seidel: In realtà Gauss usava già questa idea, ma Seidel la formalizzò e la studiò sistematicamente.

Come funzionano davvero questi metodi

Sistema di esempio

$$
\begin{aligned}
4x_1 + x_2 + x_3 &= 6 \\
x_1 + 4x_2 + x_3 &= 6 \\
x_1 + x_2 + 4x_3 &= 6
\end{aligned}
$$

Soluzione: $x_1 = x_2 = x_3 = 1$ (ma facciamo finta di non saperlo)

Metodo di Jacobi ("tutti insieme")

Idea: Risolvi ogni equazione per la sua incognita:

- $x_1 = (6 - x_2 - x_3)/4$
- $x_2 = (6 - x_1 - x_3)/4$
- $x_3 = (6 - x_1 - x_2)/4$

Iterazione: Parti da x^(0) = [0, 0, 0] e calcola **tutto insieme**:

k=0: x^(0) = [0, 0, 0]
k=1:

- x_1^(1) = (6 - 0 - 0)/4 = 1.5
- x_2^(1) = (6 - 0 - 0)/4 = 1.5
- x_3^(1) = (6 - 0 - 0)/4 = 1.5

k=2: x^(1) = [1.5, 1.5, 1.5]

- x_1^(2) = (6 - 1.5 - 1.5)/4 = 0.75
- x_2^(2) = (6 - 1.5 - 1.5)/4 = 0.75
- x_3^(2) = (6 - 1.5 - 1.5)/4 = 0.75

Caratteristica: Usi sempre i valori della **iterazione precedente** per tutti.

Metodo di Gauss-Seidel ("mano a mano")

Idea di Seidel: "Se ho appena calcolato x_1^(k+1), perché non usarlo subito per x_2^(k+1)?"

Stessa iterazione, strategia diversa:

k=1: Parti sempre da [0, 0, 0]

- x_1^(1) = (6 - 0 - 0)/4 = 1.5
- x_2^(1) = (6 - **1.5** - 0)/4 = 1.125 ← **usa il nuovo x_1!**
- x_3^(1) = (6 - **1.5** - **1.125**)/4 = 0.844 ← **usa i nuovi x_1, x_2!**

k=2: x^(1) = [1.5, 1.125, 0.844]

- x_1^(2) = (6 - 1.125 - 0.844)/4 = 1.008
- x_2^(2) = (6 - **1.008** - 0.844)/4 = 1.037
- x_3^(2) = (6 - **1.008** - **1.037**)/4 = 0.989

Caratteristica: Appena calcoli un componente, lo usi **immediatamente**.

Perché Gauss-Seidel è spesso migliore

Intuizione: Usi informazione "più fresca" → convergi più veloce.

Matematicamente: La matrice di iterazione di Gauss-Seidel spesso ha raggio spettrale più piccolo di quella di Jacobi.

Svantaggio: Jacobi si parallelizza facilmente (tutti i calcoli sono indipendenti), Gauss-Seidel no.

Il trucco per capire se convergono

La condizione pratica

Dominanza diagonale: Se ogni elemento diagonale è più grande (in valore assoluto) della somma di tutti gli altri nella sua riga:

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

Nel nostro esempio:

- Riga 1: $|4| > |1| + |1| = 2$ ✓
- Riga 2: $|4| > |1| + |1| = 2$ ✓
- Riga 3: $|4| > |1| + |1| = 2$ ✓

Risultato: Entrambi i metodi convergono garantito.

Quando NON funzionano

Sistema del cazzo:

$$x_1 + 4x_2 + 4x_3 = 9$$
$$4x_1 + x_2 + 4x_3 = 9$$
$$4x_1 + 4x_2 + x_3 = 9$$

Dominanza diagonale:

- Riga 1: $|1| > |4| + |4| = 8$? NO!
- Tutti gli elementi diagonali sono "troppo piccoli"

Risultato: I metodi probabilmente **divergono** o convergono lentissimamente.

Il punto finale

Jacobi e Gauss-Seidel non sono formule magiche. Funzionano quando la matrice ha **struttura favorevole** (diagonale "dominante" in qualche senso).

L'arte sta nel:

1. **Riconoscere** quando applicarli
2. **Trasformare** il sistema (precondizionamento) per renderli applicabili
3. **Scegliere** tra parallelismo (Jacobi) vs velocità (Gauss-Seidel)

La matematica del '800 applicata ai computer del 2025. Alcuni problemi sono eterni.

IL PROBLEMA FONDAMENTALE

Hai **Ax = b** e devi trovare x.

Metodi diretti (LU, Gauss): Calcolano x in un numero finito di passi.
Metodi iterativi: Partono da una guess $x^{(0)}$ e la migliorano passo dopo passo.

PERCHÉ I METODI ITERATIVI ESISTONO

Scenario reale

Hai una matrice 50.000 × 50.000 che viene da una discretizzazione di un'equazione differenziale. È **sparsa** (99% degli elementi sono zero).

Metodo diretto:

- Devi fare LU → riempi di elementi non zero → matrice diventa densa
- Costo: $O(n^3) = 1.25 \times 10^{14}$ operazioni = 3 ore
- Memoria: $50.000^2 \times 8$ byte = 20 GB

Metodo iterativo:

- Lavori sempre con la matrice sparsa originale
- Costo per iterazione: O(elementi non zero) = 10^6 operazioni
- 100 iterazioni = 10^8 operazioni = 0.01 secondi
- Memoria: solo gli elementi non zero

L'IDEA BASE: SPLITTING

La costruzione matematica

Punto di partenza: $Ax = b$

Idea: Riscrivi A come $A = M - N$ dove:

- **M** è "facile da invertire"
- **$N = M - A$**

Trasformazione del sistema:

$$
\begin{aligned}
&Ax = b \\
&(M - N)x = b \\
&Mx - Nx = b \\
&Mx = Nx + b \\
&x = M^{-1}Nx + M^{-1}b
\end{aligned}
$$

Ultimo passaggio cruciale: Invece di calcolare x direttamente, costruisci la successione:
$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b$

Esempio concreto per capire

Sistema 2×2:

$$
\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}
$$

Splitting tipo Jacobi: M = diagonale di A, N = M - A

$$
M = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \quad N = M - A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}
$$

Iterazione:

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

$$
x^{(k+1)} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} + \begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix}
$$

$$
x^{(k+1)} = \begin{bmatrix} 0 & -1/3 \\ -1/2 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} + \begin{bmatrix} 4/3 \\ 3/2 \end{bmatrix}
$$

In componenti:

- $x_1^{(k+1)} = -x_2^{(k)}/3 + 4/3$
- $x_2^{(k+1)} = -x_1^{(k)}/2 + 3/2$

COSA SIGNIFICA "STAZIONARIO"

Stazionario: Le matrici M e N (e quindi $E = M^{-1}N$ e $q = M^{-1}b$) **non cambiano** durante le iterazioni.

Non stazionario: Cambieresti la strategia ad ogni passo (più complicato, a volte più efficace).

Lineare: Il metodo è una combinazione lineare dei valori precedenti (nessuna funzione non lineare tipo sin, exp, etc.).

LA FORMA STANDARD

Ogni metodo iterativo stazionario si può scrivere come:
$x^{(k+1)} = Gx^{(k)} + c$

dove:

- **$G = M^{-1}N$** (matrice di iterazione)
- **$c = M^{-1}b$**

QUANDO CONVERGE: LA TEORIA

Il collegamento con i punti fissi

Se il metodo converge: $x^{(k)} \to x^*$ per $k \to \infty$

Passando al limite: $x^* = Gx^* + c \to$ *x è punto fisso di $F(x) = Gx + c$*

Il lemma delle contrazioni

Contrazione: $||F(x) - F(y)|| \leq L||x - y||$ con $L < 1$

Per la nostra F: $||F(x) - F(y)|| = ||G(x - y)|| \leq ||G||\,||x - y||$

Quindi: Se $||G|| < 1$, allora F è contrazione → **punto fisso unico + convergenza globale**

Il criterio pratico

Teorema fondamentale: Il metodo converge $\iff \rho(G) < 1$

dove **ρ(G)** è il raggio spettrale (autovalore di modulo massimo).

Velocità: $||x^{(k)} - x^*|| \leq \rho(G)^k \, ||x^{(0)} - x^*||$

I METODI SPECIFICI

Metodo di Richardson

Scelta: $M = (1/\alpha)I$, $N = M - A = (1/\alpha)I - A$

Iterazione:

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b$$
$$x^{(k+1)} = \alpha((1/\alpha)I - A)x^{(k)} + \alpha b$$
$$x^{(k+1)} = (I - \alpha A)x^{(k)} + \alpha b$$

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

Interpretazione: Aggiungi una frazione α del **residuo** $r^{(k)} = b - Ax^{(k)}$.

Matrice di iterazione: $G = I - \alpha A$

Convergenza: $\rho(I - \alpha A) < 1 \iff |1 - \alpha\lambda_i| < 1$ per ogni autovalore λ_i di A

Metodo di Jacobi

Splitting: $A = L + D + U$ (L=strictly lower, D=diagonal, U=strictly upper)

Scelta: $M = D$, $N = L + U$

Iterazione:

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

In componenti:

$$x_i^{(k+1)} = (b_i - \Sigma_{j \neq i} \, a_{ij}x_j^{(k)})/a_{ii}$$

Interpretazione: Risolvi ogni equazione per la sua incognita usando i valori "vecchi" delle altre.

Matrice di iterazione: $G = -D^{-1}(L + U)$

Metodo di Gauss-Seidel

Scelta: M = L + D, N = U

Iterazione:

$$(L + D)x^{(k+1)} = Ux^{(k)} + b$$

In componenti:

$$x_i^{(k+1)} = (b_i - \Sigma_{j<i}\, a_{ij}x_j^{(k+1)} - \Sigma_{j>i}\, a_{ij}x_j^{(k)})/a_{ii}$$

Differenza cruciale: Usa i valori **appena calcolati** $x_j^{(k+1)}$ per j < i.

ESEMPIO NUMERICO CHE CHIARISCE TUTTO

Sistema:

$$4x_1 + x_2 = 5$$
$$x_1 + 3x_2 = 4$$

Soluzione esatta: $x_1 = 1$, $x_2 = 1$

Jacobi

Formule:

- $x_1^{(k+1)} = (5 - x_2^{(k)})/4$
- $x_2^{(k+1)} = (4 - x_1^{(k)})/3$

Iterazioni (partendo da [0,0]):

- k=0: [0, 0]
- k=1: [(5-0)/4, (4-0)/3] = [1.25, 1.33]
- k=2: [(5-1.33)/4, (4-1.25)/3] = [0.92, 0.92]
- k=3: [(5-0.92)/4, (4-0.92)/3] = [1.02, 1.03]

Gauss-Seidel

Formule (usa i valori nuovi subito):

- $x_1^{(k+1)} = (5 - x_2^{(k)})/4$
- $x_2^{(k+1)} = (4 - x_1^{(k+1)})/3$ ← **usa il nuovo x_1!**

Iterazioni:

- k=0: [0, 0]
- k=1: $x_1 = (5-0)/4 = 1.25$, $x_2 = (4-1.25)/3 = 0.92$ → [1.25, 0.92]
- k=2: $x_1 = (5-0.92)/4 = 1.02$, $x_2 = (4-1.02)/3 = 0.99$ → [1.02, 0.99]

Osservazione: Gauss-Seidel converge più velocemente perché usa informazione più fresca.

QUANDO FUNZIONANO: CONDIZIONI PRATICHE

Dominanza diagonale stretta

Definizione: $|a_{ii}| > \Sigma_{j \neq i} |a_{ij}|$ per ogni riga i

Risultato: Se A è strettamente diagonalmente dominante → Jacobi converge

Perché: Dal lemma di Gerschgorin, gli autovalori di $G = -D^{-1}(L+U)$ stanno tutti in cerchi di raggio < 1.

Matrici definite positive

Se A è simmetrica definita positiva:

- Richardson converge per $0 < \alpha < 2/\lambda_{max}(A)$
- Gauss-Seidel converge sempre
- Jacobi converge se A è anche diagonalmente dominante

IL PUNTO CHIAVE

I metodi iterativi non sono magia. Funzionano quando la matrice A ha **struttura favorevole** che rende la matrice di iterazione G "piccola".

L'arte sta nel:

1. **Riconoscere** quando la struttura di A è favorevole
2. **Precondizionare** il sistema per rendere G piccola
3. **Scegliere** il metodo giusto per il problema specifico

La matematica del 1800 che risolve problemi computazionali del 2025.

RICHARDSON: L'idea più semplice del mondo

Cosa fa Richardson

L'idea base: "Se ho una soluzione sbagliata, la correggo un po' nella direzione giusta"

Come? Guarda il **residuo** r = b - Ax. Se r ≠ 0, significa che x non è la soluzione. Allora:
x_nuovo = x_vecchio + α × (direzione_giusta)

dove la "direzione giusta" è proprio il residuo r.

Esempio concreto

Sistema: **2x = 6** (soluzione: x = 3)

Guess iniziale: $x^{(0)} = 0$
Parametro: α = 0.5

Iterazione 1:

- Residuo: r = 6 - 2×0 = 6
- Correzione: $x^{(1)} = 0 + 0.5 \times 6 = 3$ ✓

Figata! Ha indovinato subito. Ma se α fosse diverso:

α = 0.1:

- k=0: x = 0, r = 6, $x^{(1)} = 0 + 0.1 \times 6 = 0.6$
- k=1: x = 0.6, r = 6 - 2×0.6 = 4.8, $x^{(2)} = 0.6 + 0.1 \times 4.8 = 1.08$
- k=2: x = 1.08, r = 6 - 2×1.08 = 3.84, $x^{(3)} = 1.08 + 0.1 \times 3.84 = 1.464$

Converge lentamente ma sicuramente verso x = 3.

Perché funziona

Intuizione fisica: È come un sistema massa-molla. Il residuo è la "forza" che spinge x verso la soluzione. α controlla quanto "velocemente" rispondi alla forza.

α troppo piccolo: Convergi lentamente
α troppo grande: Oscilli o divergi

α giusto: Convergi rapidamente

Problema di Richardson puro

Su sistemi reali, Richardson puro converge **lentissimamente** perché non sfrutta la struttura della matrice. È come cercare di aprire una porta spingendo in tutte le direzioni invece di girare la maniglia.

JACOBI: Risolvi equazione per equazione

Cosa fa Jacobi

L'idea: "Prendo ogni equazione del sistema e la risolvo per la sua incognita principale, usando i valori vecchi delle altre"

Esempio 3×3

Sistema:

$$
\begin{aligned}
4x_1 + x_2 + x_3 = 6 &\rightarrow x_1 = (6 - x_2 - x_3)/4 \\
x_1 + 4x_2 + x_3 = 6 &\rightarrow x_2 = (6 - x_1 - x_3)/4 \\
x_1 + x_2 + 4x_3 = 6 &\rightarrow x_3 = (6 - x_1 - x_2)/4
\end{aligned}
$$

Algoritmo Jacobi:

1. Parti da $x^{(0)} = [0, 0, 0]$
2. **Contemporaneamente**, calcola tutti i nuovi valori usando i vecchi:

Iterazione 1 (usando $x^{(0)} = [0, 0, 0]$):

- $x_1^{(1)} = (6 - 0 - 0)/4 = 1.5$
- $x_2^{(1)} = (6 - 0 - 0)/4 = 1.5$
- $x_3^{(1)} = (6 - 0 - 0)/4 = 1.5$

Iterazione 2 (usando $x^{(1)} = [1.5, 1.5, 1.5]$):

- $x_1^{(2)} = (6 - 1.5 - 1.5)/4 = 0.75$
- $x_2^{(2)} = (6 - 1.5 - 1.5)/4 = 0.75$
- $x_3^{(2)} = (6 - 1.5 - 1.5)/4 = 0.75$

Iterazione 3 (usando $x^{(2)} = [0.75, 0.75, 0.75]$):

- $x_1^{(3)} = (6 - 0.75 - 0.75)/4 = 1.125$
- $x_2^{(3)} = (6 - 0.75 - 0.75)/4 = 1.125$
- $x_3^{(3)} = (6 - 0.75 - 0.75)/4 = 1.125$

Sta convergendo verso [1, 1, 1] che è la soluzione esatta.

Perché funziona Jacobi

Intuizione: È come risolvere un puzzle dove ogni pezzo influenza gli altri. Ad ogni iterazione, ogni incognita "si aggiusta" in base a quello che fanno le altre.

Chiave del successo: La diagonale deve essere "dominante". Se l'equazione i-esima dipende molto di più da x_i che dalle altre variabili, allora l'aggiustamento funziona.

Quando Jacobi fallisce

Sistema del cazzo:

$$x_1 + 10x_2 = 11 \quad \rightarrow \quad x_1 = 11 - 10x_2$$
$$10x_1 + x_2 = 11 \quad \rightarrow \quad x_2 = 11 - 10x_1$$

Iterazione da [0, 0]:

- k=1: $x_1 = 11$, $x_2 = 11$
- k=2: $x_1 = 11 - 10\times11 = -99$, $x_2 = 11 - 10\times11 = -99$
- k=3: $x_1 = 11 - 10\times(-99) = 1001$, $x_2 = 1001$

DIVERGE! Perché la diagonale non domina: $|1| < |10|$ e $|1| < |10|$.

GAUSS-SEIDEL: Usa subito i valori nuovi

Cosa fa Gauss-Seidel

L'idea di Seidel: "Jacobi è stupido. Se ho appena calcolato $x_1^{(k+1)}$, perché non usarlo subito per calcolare $x_2^{(k+1)}$?"

Stesso esempio 3×3

Sistema:

$$
\begin{aligned}
4x_1 + x_2 + x_3 &= 6 \\
x_1 + 4x_2 + x_3 &= 6 \\
x_1 + x_2 + 4x_3 &= 6
\end{aligned}
$$

Algoritmo Gauss-Seidel:

1. Parti da $x^{(0)} = [0, 0, 0]$
2. **Sequenzialmente**, calcola i nuovi valori usando sempre i più freschi disponibili:

Iterazione 1:

- $x_1^{(1)} = (6 - x_2^{(0)} - x_3^{(0)})/4 = (6 - 0 - 0)/4 = 1.5$
- $x_2^{(1)} = (6 - x_1^{(1)} - x_3^{(0)})/4 = (6 - 1.5 - 0)/4 = 1.125$ ← **usa il nuovo x_1!**
- $x_3^{(1)} = (6 - x_1^{(1)} - x_2^{(1)})/4 = (6 - 1.5 - 1.125)/4 = 0.844$ ← **usa i nuovi x_1, x_2!**

Iterazione 2 (da [1.5, 1.125, 0.844]):

- $x_1^{(2)} = (6 - 1.125 - 0.844)/4 = 1.008$
- $x_2^{(2)} = (6 - 1.008 - 0.844)/4 = 1.037$
- $x_3^{(2)} = (6 - 1.008 - 1.037)/4 = 0.989$

È già molto più vicino a [1, 1, 1] di Jacobi!

Perché Gauss-Seidel è spesso migliore

Informazione fresca: Appena calcoli un valore migliore, lo usi subito invece di aspettare la prossima iterazione.

Analogia:

- **Jacobi**: "Aspetto che tutti finiscano di parlare, poi parlo io"
- **Gauss-Seidel**: "Appena sento qualcosa di utile, lo uso subito per rispondere meglio"

Svantaggio di Gauss-Seidel

Non parallelizzabile: Devi calcolare x_1 prima di x_2 prima di x_3. Con Jacobi puoi calcolare tutto in parallelo.

Su 1000 core: Jacobi scala benissimo, Gauss-Seidel no.

CONFRONTO DIRETTO: Quando usare cosa

Richardson

Usa quando:

- Hai una matrice generica e vuoi un metodo semplice
- Puoi stimare bene il parametro α ottimale
- La matrice è ben condizionata

Non usare quando:

- La matrice è mal condizionata (κ(A) >> 1)
- Vuoi convergenza veloce
- Non conosci le proprietà spettrali di A

Jacobi

Usa quando:

- A è diagonalmente dominante
- Vuoi parallelizzare su molti core
- La memoria è scarsa (accessi più regolari alla matrice)

Non usare quando:

- A non è diagonalmente dominante
- Hai pochi core e vuoi velocità
- La precisione è critica

Gauss-Seidel

Usa quando:

- A è diagonalmente dominante
- Vuoi convergenza veloce
- Lavori su un singolo core o pochi core

Non usare quando:

- Vuoi parallelizzare massivamente
- A non è diagonalmente dominante (può divergere)

LA VERITÀ BRUTALE

Esempio dove tutti e tre fanno schifo

Matrice di Hilbert 4×4:

```
A[i,j] = 1/(i+j-1)
```

Numero di condizione: $\kappa(A) \approx 15{,}000$

Risultato:

- Richardson: converge lentissimamente (migliaia di iterazioni)
- Jacobi: NON converge (non è diagonalmente dominante)
- Gauss-Seidel: NON converge

Soluzione: **Precondizionamento** → trasforma il sistema in uno equivalente ma più facile.

IL PUNTO FINALE

Questi metodi NON sono generali. Funzionano solo su matrici con struttura specifica. La maggior parte delle matrici "random" li fa fallire miseramente.

Dove brillano: Sistemi che vengono da discretizzazioni di equazioni differenziali, dove la struttura fisica garantisce dominanza diagonale.

L'arte vera: Saper **riconoscere** quando applicarli e **modificare** il sistema (precondizionamento) per renderli applicabili.