# Appunti Calcolo Numerico

Luca Seggiani
2025

## 1 Lezione del 24-02-25

### 1.1 Introduzione al corso

Il corso di **calcolo numerico**, o come viene definito oggi *analisi numerica*, tratta lo studio degli algoritmi per problemi in campi continui (incognite in $\mathbb{R}$, siano queste numeri o funzioni, ecc...) o su grandi moli di dati.

Il programma del corso è così suddiviso:

1. Analisi dell'errore per funzioni scalari;
2. Richiami di algebra lineare (calcolo vettoriale e matriciale, ecc...);
3. Risoluzione di sistemi lineari, cioè forme $A\mathbf{x} = b$;
4. Interpolazione e approssimazione di funzioni nel senso dei minimi quadrati;
5. Metodi per l'integrazione, cioè per forme $\int_a^b f(x)\,dx$;
6. Equazioni non lineari, cioè ricerca dei punti $g(x) = 0$ per $g(x)$ non lineari;
7. Problemi agli autovalori, cioè date matrici $A \in \mathbb{C}^{m\times n}$, trovare $(\lambda, \mathbf{x})$ tali che $A\mathbf{x} = \lambda\mathbf{x}$.

#### 1.1.1 Matematica del continuo

Abbiamo detto che i valori trattati sono continui, che intendiamo per appartenenti ad $\mathbb{R}$. Un problema apparente dei numeri reali è che richiedono teoricamente un numero infinito di cifre per la loro rappresentazione. Si rende quindi necessaria un'approssimazione in modo da ovviare ai problemi:

- Rappresentare oggetti matematici con un numero infinito di parametri;
- Risolvere problemi che non hanno formule chiuse per la soluzione, ma richiedono approcci iterativi (discesa a gradiente, ecc...) e quindi che richiedono un approssimazione data dall'impossibilità di effettuare infiniti passi.

#### 1.1.2 Errori

Avremo quindi bisogno di valutare degli **errori**, che saranno gli *errori di approssimazione* come riportata sopra, uniti agli *errori nei dati* già presenti nella nostra mole di dati.

Una domanda che potremo porci è come questi errori influiscono sul risultato che ci interessa. Una prima distinzione può essere fra algoritmi **instabili** e **stabili**, cioè che *amplificano* l'errore o lo mantengono costante. Una seconda distinzione può essere sul **condizionamento** del problema, cioè la tendenza del problema a reagire in maniera drastica a piccole variazioni delle condizioni iniziali.

### 1.1.3 Efficienza

Un'altra considerazione importante è quella dell'**efficienza** degli algoritmi, cioè il tempo che questi richiedono per convergere ad una soluzione valida (non ottima, in quanto abbiamo visto dobbiamo troncare il numero di passi nel caso di approcci iterativi, che altrimenti potrebbe tendere ad infinito). Questo viene stimato attraverso il *costo computazionale*, tenendo conto di una certa *accuratezza* che vogliamo stabilire.

## 1.2 Rappresentazione dei reali

Introduciamo la classe dei **numeri reali in virgola mobile**, atta a rappresentare numeri $x \in \mathbb{R}$ usando parametri che stanno in $\mathbb{N}$, o al limite in $\mathbb{Z}$, in quanto abbiamo visto possiamo gestire numeri di questo tipo in modo esatto nei calcolatori.

**Teorema 1.1: Rappresentazione dei reali in virgola mobile**

Fissata una base $\beta > 1$, con $\beta \in \mathbb{N}$, si può sempre trovare un esponente $e \in \mathbb{Z}$ e una successione di cifre $\{\alpha_j\}_{j=1,2,...}$ tali per cui:

$$x = \text{sgn}(x) \cdot \beta^e \cdot \sum_{j=1}^{\infty} \alpha_j \beta^{-j}$$

Osserviamo che scelte tipiche per $\beta$ sono 10 (decimale), 2 (binario) e 16 (esadecimale).

Un problema di questa rappresentazione è che non è propriamente **unica**: ad esempio, potremo scrivere un approssimazione di $\pi$ come $3.14$, o come $0.314 \cdot 10$, equivalentemente. Inoltre, possono esistere casi di numeri periodici, come $2.\overline{9}$, che sono effettivamente uguali ad altri (in questo caso 3).

Sfruttiamo allora il seguente teorema, dato senza dimostrazione:

**Teorema 1.2: Teorema di rappresentazione**

Dati $\beta \in \mathbb{N}, \beta > 1$ base e $x \in \mathbb{R}, x \neq 0$, allora esiste ed è unica la rappresentazione $x = \text{sgn}(x) \cdot \beta^e \cdot \sum_{j=1}^{\infty} \alpha_j \beta^{-j}$ (dal Teorema 1.1) tale che:

- $\alpha_1 \neq 0$;
- $!\exists k \in \mathbb{N} : \alpha_j = \beta - 1 \; \forall j > k$.

Introducendo queste due limitazioni possiamo quindi ovviare al problema dell'unicità.

Diamo quindi alcune definizioni, riguardo alla rappresentazione appena vista:

**Definizione 1.1: Rappresentazione in virgola mobile normalizzata**

L'unica rappresentazione che verifica il Teorema 1.2 si dice **rappresentazione in virgola mobile normalizzata**.

e riguardo alla successione $\{\alpha_j\}$:

**Definizione 1.2: Mantissa**

Prende il nome di **mantissa** la serie $\sum_{j=1}^{\infty} \alpha_j \beta^{-j}$.

Notiamo che vale:

$$\frac{1}{\beta} \leq \sum_{j=1}^{\infty} \alpha_j \beta^{-j} < 1$$

In quanto:

- Per il limite inferiore:

$$\sum_{j=1}^{\infty} \alpha_j \beta^{-j} \geq \beta^{-1} = \frac{1}{\beta}$$

- Per il limite superiore:

$$\sum_{j=1}^{\infty} \alpha_j \beta^{-j} < (\beta - 1) \sum_{j=1}^{\infty} \beta^{-j} = (\beta - 1) \left( \frac{1}{1 - \beta^{-1}} - 1 \right)$$

$$= (\beta - 1) \frac{\beta^{-1}}{1 - \beta^{-1}} = (\beta - 1) \frac{1}{\beta - 1} = 1$$

Vediamo che la mantissa non è effettivamente rappresentabile nella sua interezza in un calcolatore, in quanto richiederebbe un numero infinito di cifre. Si tronca quindi la mantissa, e si considera un range limitato per l'esponente. Si definisce quindi un sottinsieme:

**Definizione 1.3: Numeri di macchina**

Dati: $\beta \in \mathbb{N}$, $m \in \mathbb{N}$, $L, U \in \mathbb{Z}$ tali che $L \leq U$, si definisce $F(\beta, m, L, U)$ come:

$$F = \{x \in \mathbb{R} : x = \text{sgn}(x) \cdot \beta^e \cdot \sum_{j=1}^{m} \alpha_j \beta^{-j},\ L \leq e \leq U,\ \alpha_j = \{0, ..., \beta - 1\},\ \alpha_1 \neq 0\} \cup \{0\}$$

detto **numero di macchina**.

L'inclusione dello zero è necessario in quanto questo non è compreso nel primo insieme dato.

Notiamo che l'insieme dei numeri di macchina non è equispaziato (ci sono più numeri vicino allo zero). Presi intervalli $[\beta^{e-1}, \beta^e)$, questi risulterebbero equispaziati se venissero considerati su una scala logaritmica base $\beta$. All'interno di questi intervalli, invece, si hanno effettivamente numeri equispaziati, con periodo $\beta^e \cdot \beta^{-m} = \beta^{e-m}$.

### 1.2.1 Limiti inferiori e superiori

Potremmo chiederci quali sono i numeri **minimi** e **massimi** rappresentabili.

Osservando la definizione di $F$ si ha che il numero più piccolo possibile è quello che si ha prendendo $\beta = L$, e $\alpha_j = 0$ per ogni $j > 1$, e $\alpha_1 = 1$, quindi:

$$\beta^L \cdot 1 \cdot \beta^{-1} = \beta^{L-1}$$

Il numero più grande si ha invece prendendo $\beta = U$ e $\alpha_j = \beta - 1$, e quindi:

$$\beta^U(\beta-1)\sum_{j=1}^{m}\beta^{-j} = \beta^U\left(\frac{1-\beta^{-m-1}}{1-\beta^{-1}} - 1\right)(\beta-1)$$

$$= \beta^U\left(\frac{1-\beta^{-m}}{\beta-1}\right)(\beta-1) = \beta^U(1-\beta^{-m})$$

Potremmo poi chiederci quanti numeri macchina esistono fissati $\beta, m, L$ e $U$. Si hanno intanto due scelte di segno, $(U-L+1)$ scelte di esponenti e $(\beta^m - \beta^{m-1})$ scelte di mantisse (tutti i numeri rappresentabili su $m$ cifre base $\beta$ meno quelli con prima cifra nulla) più lo zero, da cui:

$$2 \cdot (U-L+1) \cdot (\beta^m - \beta^{m-1}) + 1$$

Le rappresentazioni più comuni dei numeri macchina sono definite dallo standard IEEE 754, che definisce:

| **Precisione** | $\beta$ | $m$ | $L$ | $U$ | **Dimensione** |
|---|---|---|---|---|---|
| Singola | 2 | 24 | -126 | 127 | 4 byte (32 bit) |
| Doppia | 2 | 53 | -1022 | 1023 | 8 byte (64 bit) |
| Quadrupla | 2 | 113 | -16382 | 16383 | 16 byte (128 bit) |

Casi particolari potrebbero richiedere precisioni più grandi o più lasche. Ultimamente, in particolare, si sono diffuse rappresentazioni a precisione più bassa (su 16 o addirittura 8 bit), in particolare nel campo delle reti neurali.

Il pacchetto MATLAB utilizza di default la precisione **doppia** come definita dallo standard IEEE 754.

### 1.3 Arrotondamento

Abbiamo visto come ci stiamo spostando dalla matematica esatta a una serie di approssimazioni. In generale, vorremo partire da un certo numero reale $x \in \mathbb{R}$, ma non appartenente all'insieme dei numeri macchina ($x \notin F(\beta, m, L, U)$). L'obiettivo è quello di riportare $x$ ad una sua *approssimazione* appartenente ad $F(\beta, m, L, U)$.

Esistono 3 possibili situazioni:

- $|x| > \beta^U(1-\beta^{-m})$, cioè $x$ maggiore del massimo rappresentabile (**overflow**);
- $|x| < \beta^{L-1}$, cioè $x$ minore del minimo rappresentabile (**underflow**);
- $\beta^{L-1} \leq x \leq \beta^U(1-\beta^{-m})$. Se nei casi precedenti si poteva scegliere, rispettivamente, un $M$ molto grande, o $\infty$, e 0, qui si può effettivamente procedere con l'arrotondamento.

**Definizione 1.4: Arrotondamento**

Un arrotondamento è una funzione $RD : \mathbb{R} \to F(\beta, m, L, U)$, con $RD(x)$ uno dei numeri di macchina in $F(\beta, m, L, U)$ "vicini" ad $x$.

Preso un certo reale $x$, avremo un numero di macchina a sinistra è uno a destra, cioè il primo più piccolo e il primo più grande. Possiamo allora definire i seguenti arrotondamenti:

- **Troncamento** (*round down*):
$$Tr(x) = \lfloor x \rfloor$$
- **Round-up:**
$$Ru(x) = \lceil x \rceil$$
- **Round-to-nearest:**
$$Rn(x) = \begin{cases} \lceil x \rceil, & \alpha_{m+1} \geq \frac{\beta}{2} \\ \lfloor x \rfloor, & \alpha_{m+1} < \frac{\beta}{2} \end{cases}$$
con $\alpha_{m+1}$ la prima cifra che viene scartata dall'arrotondamento.

Notiamo poi che preso il troncamento:

$$Tr(x) = \text{sgn}(x)\beta^e \sum_{j=1}^{m} \alpha_j \beta^{-j}$$

il round up corrispondente sarà:

$$Ru(x) = \text{sgn}(x)\beta^e \left( \sum_{j=1}^{m} \alpha_j \beta^{-j} + \beta^{-m} \right)$$

### 1.3.1 Errore di arrotondamento

Valgono le disuguaglianze:

- $|Tr(x) - x| \leq \beta^{e-m}$
- $|Rn(x) - x| \leq \frac{\beta^{e-m}}{2}$, che è il minimo errore possibile cioè:
$$|Rn(x) - x| = \min_{RD(x)}(RD(x) - x)$$
Per questo motivo da qui in poi assumeremo quindi di prendere sempre $Rn(x)$.

**Definizione 1.5: Errori di arrotondamento**

Definiamo:

- **Errore assoluto di arrotondamento:** $x - Rn(x) = \sigma_x$
- **Errore relativo di arrotondamento:** $\frac{x - Rn(x)}{x} = \epsilon_x$

La definizione di errore relativo è utile per avere un errore pressochè costante su tutta la retta dei reali. Si ha quindi che:

- Riguardo all'errore assoluto:
$$|\sigma_x| \leq \frac{\beta^{e-m}}{2}$$
- Riguardo all'errore relativo:
$$|\epsilon_x| \leq \frac{\beta^{e-m}}{2\beta^{e-1}} = \frac{1}{2}\beta^{1-m}$$
visto che $|x| > \beta^{e-1}$. Notiamo che questo errore non dipende dall'esponente $e$, che è quello che desideravamo.

Abbiamo quindi che l'insieme dei numeri in virgola mobile garantiscono un errore relativo limitato in modo uniforme, se non si incombe in overflow o underflow.

**Definizione 1.6: Precisione**

La quantità $U = \beta^{1-m}$ viene detta **precisione di macchina** di una certa rappresentazione a virgola mobile.

Ad esempio, nella precisioni doppia e singola, $U \approx 10^{-16}$ e $U \approx 10^{-8}$, che significa rispettivamente prime 15 o prime 7 cifre esatte.

### 1.3.2 Dettagli di implementazione

Prendiamo ad esempio la doppia precisione. Avremo:

- 1 bit di segno;
- 52 bit di mantissa (con 1 bit implicito impostato ad 1, per $\alpha_1 = 1$);
- 11 bit di esponente in rappresentazione con offset.

Riguardo agli altri formati si ha:

| Precisione | Segno | Esponente | Mantissa | U |
|---|---|---|---|---|
| Singola | 1 | 8 | 23 | $\approx 10^{-8}$ |
| Doppia | 1 | 11 | 52 | $\approx 10^{-16}$ |
| Quadrupla | 1 | 15 | 112 | $\approx 10^{-34}$ |

## 1.4 Numeri sottonormalizzati

Vediamo una tecnica per la rappresentazione di numeri più piccoli di $\beta^{L-1}$, implementata nella maggior parte dei pacchetti software moderni (e nelle implementazioni degli standard a virgola mobile disponibili nei processori moderni). Si ha che se $x \in [0, \beta^{L-1}]$, allora si assume come prima cifra della mantissa $0$, con esponente fisso ad $L$. Questo ci permette di rappresentare più numeri vicino allo zero.

Si avranno quindi numeri equispaziati fra 0 e $\beta^{L-1}$ con distanza $\beta^{L-m}$. Inoltre, sui numeri sottonormalizzati si avrà un errore relativo non limitato da $\frac{1}{2}\beta^{1-m}$, ma che invece aumenta avvicinandosi a 0.

I numeri sottonormalizzati sono indicati da un **valore speciale dell'esponente**, cosa che accade anche per:

- I valori $+\infty$ e $-\infty$;
- I valori NaN (Not a Number) con relativi codici di errore in mantissa.

La presenza di questi valori rende necessaria l'approssimazione delle precisioni per i numeri in virgola mobile.

# 2 Lezione del 28-02-25

## 2.1 Operazioni sui numeri macchina

Abbiamo introdotto l'insieme dei numeri macchina. Vediamo adesso come eseguire **operazioni** fra elementi di questi insiemi.

Notiamo che, di base, dati $x, y \in F(\beta, m, L, U)$, non necessariamente $x \circ y \in F(\beta, m, L, U)$ per le comuni operazioni aritmetiche $+, -, \times, \div$.

Quello che faremo è quindi approssimare tali operazioni, cioè dire:

- $x \oplus y = Rn(x + y)$
- $x \ominus y = Rn(x - y)$
- $x \otimes y = Rn(x \times y)$
- $x \oslash y = Rn(x \div y)$

Effetto di questa approsimazione è negare proprietà note dei reali, come ad esempio l'associativa:

$$(x \oplus y) \oplus z \neq x \oplus (y \oplus z)$$

$$(x \oplus y) \otimes z \neq x \oplus (y \otimes z)$$

Cioè, la valutazione di una formula con ordini diversi ma equivalenti in aritmetica esatta può portare a risultati differenti nell'insieme dei numeri di mmacchina.

### 2.1.1 Errore nel calcolo di funzione

Sia $f : \mathbb{R}^m \to \mathbb{R}$, e si voglia calcolare $f(P_0)$, con $P_0 = \left(x_1^{(0)}, x_2^{(0)}, ...x_m^{(0)}\right) \in \mathbb{R}^m$. Le operazioni aritmetiche $+, -, \times, \div$ possono essere viste come funzioni di questo tipo. Ci interroghiamo quindi sulla fonte dell'errore nella loro valutazione:

1. Nel caso contenga funzioni irrazionali o trascendenti, $f$ verrà approssimata con una funzione $\overline{f}$ che coinvolge solo operazioni aritmetiche di base $+, -, \times, \div$;
2. $\overline{f}$ viene tradotta in un *algoritmo* $\overline{f}_a$, ovvero in una formula che coinvolge $\oplus, \ominus, \otimes, \oslash \leftarrow +, -, \times, \div$. La formula ottenuta finora viene detta **algoritmo**;
3. Potrebbe essere che $P_0 \neq F(\beta, m, L, U)$, e quindi viene approssimato a $P_1 = Rn(P_0)$.

Quindi, vogliamo $f(P_0)$ ma possiamo solo approssimarla come $\overline{f}_a(P_1)$.

Ad esempio, poniamo di voler calcolare $e^\pi$. I passaggi nell'ordine appena visto saranno:

1. Approssimamo l'esponenziale al secondo grado dello svillupppo di Taylor:

$$e^x \approx 1 + x + \frac{x^2}{2} = \overline{f}(x)$$

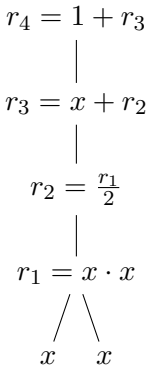
2. Si riporta la $\overline{f}(x)$ a $\overline{f}_a(x)$:

$$1 \oplus (x \oplus ((x \otimes x) \oslash 2))$$

Indicheremo algoritmi di questo tipo anche attraverso **risultati intermedi**:

- $r_1 = x \cdot x$
- $r_2 = \frac{r_1}{2}$
- $r_3 = x + r_2$
- $r_4 = 1 + r_3$

con il risultato finale inteso come l'ultimo risultato intermedio. Un modo di visualizzare i risultati intermedi di un algoritmo può essere un albero:




dove la *radice* rappresenta il **risultato** e le *foglie* rappresentano le variabili della funzione.

3. Si approssima $\pi$ al numero macchina più vicino:

$$Rn(\pi) = 3.1415 = P_1$$

Avremo quindi la formula finale:

$$1 \oplus (P_1 \oplus ((P_1 \otimes P_1) \oslash 2))$$

Diamo quindi la definizione di **errore assoluto**:

**Definizione 2.1: Errore assoluto**

Data $f : \mathbb{R}^m \to \mathbb{R}$, un punto $P_0 \in \mathbb{R}^m$ ed un algoritmo $\overline{f}_a$, l'errore assoluto è dato da:

$$\sigma_f = \overline{f}_a(P_1) - f(P_0), \quad P_1 = Rn(P_0)$$

e di errore relativo:

**Definizione 2.2: Errore relativo**

Date le ipotesi della definizione 2.1, l'errore relativo è dato da:

$$\epsilon_f = \frac{\overline{f}_a(P_1) - f(P_0)}{f(P_0)} = \frac{\sigma_f}{f(P_0)}$$

### 2.1.2 Errore di funzioni razionali

Assumiamo per adesso $f$ **funzione razionale**, ovvero $f$ si può definire con un numero di operazioni in $+, -, \times, \div$. Assumere funzioni razionali ci permette di prendere $f = \overline{f}$ e $f_a = \overline{f}_a$ (non ci sono irrazionali da riportare ai razionali). Potremo allora dire:

$$\sigma_f = f_a(P_1) - f(P_0) = f_a(P_1) - f(P_1) + f(P1) - f(P_0)$$

che possiamo dividere in:

$$\sigma_f = \sigma_a + \sigma_d, \quad \sigma_a = f_a(P_1) - f(P_1), \quad \sigma_d = f(P_1) - f(P_0)$$

che chiamiamo rispettivamente **errore assoluto algoritmico** e **errore assoluto inerente**.

Allo stesso modo possiamo definire l'**errore relativo**:

$$\epsilon_f = \frac{\sigma_f}{f(P_0)} = \frac{f_a(P_1) - f(P_0)}{f(P_0)} = \frac{f_a(P_1) - f(P_1)}{f(P_0)} + \frac{f(P_1) - f(P_0)}{f(P_0)}$$

$$= \frac{f_a(P_1) - f(P_1)}{f(P_1)} \cdot \frac{f(P_1)}{f(P_0)} + \frac{f(P_1) - f(P_0)}{f(P_0)}$$

che si divide nuovamente in :

$$\epsilon_f = \epsilon_a + \epsilon_d, \quad \epsilon_a = \frac{f_a(P_1) - f(P_1)}{f(P_1)}, \quad \epsilon_d = \frac{f(P_1) - f(P_0)}{f(P_0)}$$

che chiamiamo rispetivamente **errore relativo algoritmico** e **errore relativo inerente**. Questo viene da:

$$\epsilon_f = [...] = \frac{f_a(P_1) - f(P_1)}{f(P_1)} \cdot \frac{f(P_1)}{f(P_0)} + \frac{f(P_1) - f(P_0)}{f(P_0)}$$

si nota che $\frac{f(P_1)}{f(P_0)} = 1 + \frac{f(P_1) - f(P_0)}{f(P_0)}$, e quindi:

$$= \epsilon_a \cdot \left(1 + \frac{f(P_1) - f(P_0)}{f(P_0)}\right) + \epsilon_d = \epsilon_a(1 + \epsilon_d) + \epsilon_d = \epsilon_a + \epsilon_d + \epsilon_a\epsilon_d \approx \epsilon_a + \epsilon_d$$

assumendo $\epsilon_a\epsilon_d \approx 0$.

Ci interessa dare stime superiori per i valori assoluti di errori assoluti e relativi, come avevamo fatto per gli errori delle funzioni di arrotondamento da reali a numeri macchina. In generale, quindi, per limitare $|\sigma_f|$ cercheremo disuguaglianze $|\sigma_a| < \tau_1$, $|\sigma_d| < \tau_2$, da cui:

$$|\sigma_f| < \tau_1 + \tau_2$$

### 2.1.3 Stima dell'errore inerente

Avevamo quindi definito l'**errore assoluto inerente**:

$$\sigma_d = f(P_1) - f(P_0)$$

Sotto l'ipotesi $f \in C^1(D)$ per $D \subset \mathbb{R}^m$ che contiene $P_0$, si può usare lo sviluppo di Taylor di $f$ in $P_0$, troncato al primo ordine:

$$f(P_1) - f(P_0) = f(P_0) + \nabla f(\overline{P})^T(P_1 - P_0) - f(P_0) = \nabla f(\overline{P})^T(P_1 - P_0)$$

$$= \sum_{j=1}^{m} \frac{\partial f}{\partial x_j}(\overline{P}) \cdot \left(x_j^{(1)} - x_j^{(0)}\right) \approx \sum_{j=1}^{m} \frac{\partial f}{\partial x_j} P_0 \cdot \left(x_j^{(1)} - x_j^{(0)}\right)$$

dove $\overline{P}$ è un punto che sta sul segmento $\overline{P_1 P_0}$. Da questo potremo dire:

$$\sigma_d = \sum_{j=1}^{m} \frac{\partial f}{\partial x_j} P_0 \cdot \sigma_j$$

dove $\sigma_j = \left(x_j^{(1)} - x_j^{(0)}\right)$ è l'**errore di arrotondemento** nella componente $j$ di $P_0$, e $\frac{\partial f}{\partial x_j} P_0$ viene detto **coefficiente di amplificazione**.

Per l'**errore relativo inerente**, che avevamo definito come:

$$\epsilon_d = \frac{f(P_1) - f(P_0)}{f(P_0)}$$

potremo fare considerazioni simili:

$$\epsilon_d = \frac{\sum_{j=1}^{m} \frac{\partial f}{\partial x_j}(P_0) \cdot \sigma_j}{f(P_0)} = \sum_{j=1}^{m} \frac{x_j^{(1)} - x_j^{(0)}}{x_j^{(0)}} \cdot \frac{\partial f}{\partial x_j}(P_0) \cdot \frac{x_j^{(0)}}{f(P_0)}$$

dove $\epsilon_j = \frac{x_j^{(1)} - x_j^{(0)}}{x_j^{(0)}}$ sarà l'**errore di arrotondamento relativo** nella componente $j$ di $P_0$ e $P_j = \frac{\partial f}{\partial x_j}(P_0) \cdot \frac{x_j^{(0)}}{f(P_0)}$ viene detto **coefficiente di amplificazione dell'errore relativo**.

La formula finale sarà quindi:

$$\epsilon_d = \sum_{j=1}^{m} \epsilon_j P_j$$

I problemi in cui si devono calcolare $f$ i cui coefficienti di amplificazione degli errori relativi sono grandi in modulo (o ce n'è almeno uno sufficientemente grande) si dicono **malcondizionati**. Viceversa, se $|P_j|$ è vicino a $1$ per ogni $j$ il problema si dice **ben condizionato**, cioè che $\epsilon_d$ è di un ordine di grandezza comparabile a $\max(\epsilon_i)$

Notiamo inoltre che il condizionamento di un problema dipende solamente dalla sua struttura matematica.

### 2.1.4 Errori inerenti delle operazioni aritmetiche

Vediamo gli errori inerenti associati alle 4 operazioni aritmetiche $+, -, \times, \div$:

| **Operazione** | $\sigma_d$ | $\epsilon_d$ |
|---|---|---|
| $x \oplus y$ | $\sigma_x + \sigma_y$ | $\frac{x}{x+y}\epsilon_x + \frac{y}{x+y}\epsilon_y$ |
| $x \ominus y$ | $\sigma_x - \sigma_y$ | $\frac{x}{x-y}\epsilon_x - \frac{y}{x-y}\epsilon_y$ |
| $x \otimes y$ | $y\sigma_x + x\sigma_y$ | $\epsilon_x + \epsilon_y$ |
| $x \oslash y$ | $\frac{1}{y}\sigma_x - \frac{x}{y^2}\sigma_y$ | $\epsilon_x - \epsilon_y$ |

Notiamo come somme e sottrazioni non amplificano gli errori totali, mentre prodotti e divisioni non amplificano gli errori relativi (riguardo agli errori inerenti). Questo significa che somme e sottrazioni possono avere errori relativi grandi quando $|x + y| << \min\{|x|, |y|\}$. Questo effetto viene detto **cancellazione numerica**.

### 2.1.5 Stima dell'errore algoritmico

Avevamo definito un algoritmo $f_a(x)$ di cui vogliamo stimare l'errore algoritmico assoluto $\sigma_a = f_a(P_1) - f(P_1)$. Assumiamo $P_1 = Rn(P_0) \in F(\beta, m, L, U)$, cioè gli operandi come privi di errori iniziali di arrotondamento.

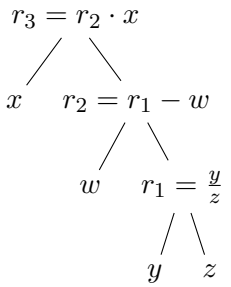
L'idea è di seguire l'errore generato dall'algoritmo sul grafo (o albero) che lo rappresenta sfruttando le relazioni per l'errore inerente nelle 4 operazioni aritmetiche. Prendiamo ad esempio la funzione:

$$f(x, y, z, w) = x \cdot \left( \frac{y}{z} - w \right)$$

Avremo i risultati intermedi:

- $r_1 = \frac{y}{z}$
- $r_2 = r_1 - w$
- $r_3 = r_2 \cdot x$

Di cui riportiamo il grafo:




dove $\epsilon_3$, $\epsilon_2$ e $\epsilon_1$ saranno termini associati ad ogni risultato intermedio che rappresenteranno gli errori di troncamento dei risultati intermedi stessi, e $\epsilon_{r3}$, $\epsilon_{r2}$ e $\epsilon_{r1}$ rappresenteranno gli errori inerenti delle singole operazioni per il calcolo dei risultati intermedi.

Partiamo dalla radice per valutare gli errori:

$$\epsilon_d = \epsilon_{r_3} = \epsilon_3 + \epsilon_x + \epsilon_{r_2} = \epsilon_3 + \epsilon_{r_2}$$

$$= \epsilon_3 + \epsilon_2 + \frac{-zw}{y - zw} \cdot \epsilon_w + \frac{y}{y - zw} \cdot \epsilon_{r_1} = \epsilon_3 + \epsilon_2 + \frac{y}{y - zw} \cdot \epsilon_{r_1}$$

$$= \epsilon_3 + \epsilon_2 + \frac{y}{y - zw} (\epsilon_1 + \epsilon_y - \epsilon_z) = \epsilon_3 + \epsilon_2 + \frac{y}{y - zw} \cdot \epsilon_1 = \epsilon_a$$

Dove abbiamo ignorato i termini di errore relativo $\epsilon_i$ legati ad ogni variabile (come avevamo detto sopra, gli operandi sono considerati come privi di errore di arrotondamento). Per la stima di $\epsilon_3$, $\epsilon_2$ e $\epsilon_1$, vale $\epsilon_i \leq U$ precisione macchina. Nel caso di errori assoluti vale $|\sigma_i| \leq U \cdot \max(x_i)$ considerata ogni variabile $x_i$.

Chiaramente, diversi algoritmi equivalenti in aritmetica esatta potranno avere errori algoritmici diversi fatte tutte le approssimazioni.

Prendiamo ad esempio la funzione $f(x, y) = x^2 - y^2$. Potremmo adottare due algoritmi:

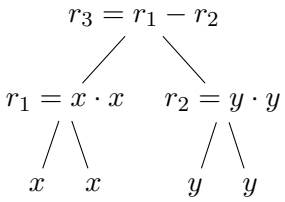
1. Si prendono i risultati intermedi:

$$r_1 = x \cdot x$$

$$r_2 = y \cdot y$$

$$r_3 = x - y$$

Da cui il grafo:




Questo approccio potrebbe risultare il più intuitivo: dalla stima dell'errore si ha:

$$\epsilon_{r_3} = \epsilon_1 + \frac{x^2}{x^2-y^2}\epsilon_{r_1} - \frac{y^2}{x^2-y^2}\epsilon_{r_2} = \epsilon_1 + \frac{x^2}{x^2-y^2}(\epsilon_1+\epsilon_x+\epsilon_x) - \frac{y^2}{x^2-y^2}(\epsilon_2+\epsilon_y+\epsilon_y)$$

$$= \epsilon_1 + \frac{x^2}{x^2-y^2}\epsilon_1 - \frac{y^2}{x^2-y^2}\epsilon_2$$

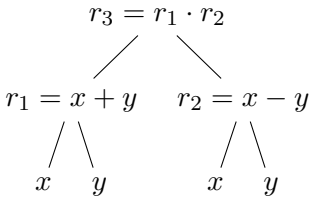
2. Un altro approccio è quello di prendere i risultati intermedi:

$$r_1 = x + y$$

$$r_2 = x - y$$

$$r_3 = r_1 \cdot r_2$$

Da cui il grafo:




Vediamo la stima dell'errore:

$$\epsilon_{r_3} = \epsilon_3 + \epsilon_{r_1} + \epsilon_{r_2} = \epsilon_3 + \epsilon_1 + \frac{x}{x+y}\epsilon_x + \frac{y}{x+y}\epsilon_y + \epsilon_2 + \frac{x}{x-y}\epsilon_x - \frac{y}{x-y}\epsilon_y$$

$$= \epsilon_3 + \epsilon_1 + \epsilon_2$$

Notiamo che la stima dell'errore del secondo approccio è più conveniente, in quanto più strettamente limitata al di sotto di un valore fisso: $\epsilon_1 + \epsilon_2 + \epsilon_3 = 3U$, contro il $\left(1 + \frac{|x^2+y^2|}{|x^2-y^2|}\right) U$ del primo approccio.

Abbiamo visto quindi tenciche per la stima di $\epsilon_a$ e $\epsilon_d$ ($\sigma_a$ e $\sigma_d$), che ci permettono di calcolare $|\epsilon_f| \leq |\epsilon_a| + |\epsilon_d|$ ($|\sigma_f| \leq |\sigma_a| + |\sigma_d|$).

### 2.1.6 Problema diretto

Un problema classico sarà quello di, data $f$, un algoritmo risolutivo $f_a$ e una stima degli errori $d_{x_i}$, di stimare $\sigma_f$ per $P_0 \in D \subset \mathbb{R}^m$.

Ad esempio, prendiamo la funzione:

$$f : D \to \mathbb{R}, \quad f(x_1, x_2) = \frac{x_1}{x_2}, \quad D = [1, 3] \times [4, 5]$$

Diamo una stima dell'errore di arrotondamento delle variabili:

$$|\sigma_{x_i}| \leq \frac{1}{2} \cdot 10^{-2}$$

- Iniziamo con la stima dell'errore inerente. Si ha, dall'errore assoluto inerente della divisione:

  $$\sigma_d = \frac{1}{x_2}\sigma_{x_1} - \frac{x_1}{x_2^2}\sigma_{x_2}$$

  Quello che ci interessa è dare stime superiori, quindi prendiamo i due valori:

  $$\left|\frac{1}{x_2}\right| \approx \frac{1}{4}, \quad \left|\frac{x_1}{x_2^2}\right| \approx \frac{7}{16}$$

  Dai valori massimi che si possono ottenere sul dominio $D$, da cui:

  $$|\sigma_d| \leq \frac{1}{4} \cdot \frac{1}{2} \cdot 10^{-2} + \frac{7}{16} \cdot \frac{1}{2} \cdot 10^{-2} = \left(\frac{1}{8} + \frac{7}{32}\right) \cdot 10^{-2} = \frac{7}{32} \cdot 10^{-2}$$

- Vediamo quindi la stima dell'errore algoritmico. L'albero dell'algoritmo sarà banalmente:

  $$\begin{array}{ccc} & r_1 = \frac{x_1}{x_2} & \\ & \diagup \quad \diagdown & \\ x_1 & & x_2 \end{array}$$

  da cui posto $\sigma_{x_i} = 0$ resterà solo l'errore di arrotondamento assoluto $\sigma_1$:

  $$|\sigma_a| = \sigma_1 = U \cdot \max\left(\frac{x_1}{x_2}\right) = \frac{3}{4} \cdot \frac{1}{2} \cdot 10^{-2} = \frac{3}{8} \cdot 10^{-2}$$

Otteniamo quindi l'errore totale come la somma dell'errore algoritmico e dell'errore inerente, cioè:

$$|\sigma_f| = |\sigma_a| + |\sigma_f| = \left(\frac{7}{32} + \frac{3}{8}\right) \cdot 10^{-2} = \frac{19}{32} \cdot 10^{-2}$$

### 2.1.7 Problema inverso

Il problema inverso potrebbe essere quello di, dato $\tau > 0$, $f$ e un punto $P_0 \in \mathbb{R}^n$, determinare un algoritmo ed un valore di precisione macchina $U$ tali per cui $|s_f| < \tau$.

Prendiamo ad esempio il problema considerato prima, ma sul dominio:

$$D = [1, 2] \times [-2, -1]$$

e cerchiamo una precisione macchina $U$ tale per cui l'errore assoluto $\sigma_f$ è limitato a $\tau = 10^{-2}$, cioè:

$$|\sigma_f| \leq 10^{-2}$$

- Iniziamo di nuovo con la stima dell'errore inerente. Dagli stessi errori inerenti considerati prima si hanno le stime:

$$\left|\frac{1}{x_2}\right| \approx 1, \quad \left|\frac{x_1}{x_2^2}\right| \approx 2$$

Dai valori massimi che si possono ottenere sul dominio $D$, da cui:

$$|\sigma_d| \leq U + 2U = 3U$$

- Vediamo quindi la stima dell'errore algoritmico, che prendendo il valore massimo della funzione in maniera analoga a prima sarà:

$$|\sigma_a| = \sigma_1 = U \cdot \max\left(\frac{x_1}{x_2}\right) = 2U$$

L'errore totale sarà quindi: $|\sigma_f| = |\sigma_a| + |\sigma_f| = 3U + 2U = 5U$

Imponiamo quindi la diseguaglianza iniziale, cioè:

$$|\sigma_f| = 5U \leq 10^{-2} \implies U \leq \frac{1}{5} \cdot 10^{-2}$$

# 3 Lezione del 03-03-25

## 3.1 Riassunto sulla stima dell'errore

Riassumiamo quindi le regole viste per la stima dell'errore su funzioni razionali. Avevamo dato la definizione di errore **assoluto** $\sigma_f$ e errore **relativo** $\epsilon_f$, entrambi composti da due fattori denominati errore **algoritmico** e errore **inerente**, con pedici rispettivamente $a$ e $d$.

- Riguardo all'errore **inerente assoluto** avevamo preso su un dominio $D$ la stime:

$$|\sigma_d| \leq \sum_{j=1}^{m} A_j \cdot |\sigma_j|$$

con $|\sigma^j|$ **errore di arrotondamento** e $A_j$ **coefficiente di amplificazione**:

$$A_j = \max_{P \in D}\left(\frac{\partial f}{\partial x_j}(P)\right)$$

Per l'errore di arrotondamento avevamo visto potevamo prendere:

$$|\sigma_j| \leq U \cdot |x_j|$$

con $U$ precisione macchina.

- Riguardo all'**errore inerente relativo** avevamo invece preso:

$$|\epsilon_d| \leq \sum_{j=1}^{m} \overline{A}_j \cdot |\epsilon_j|$$

con $|\epsilon_j|$ **errore di arrotondamento relativo** e $\overline{\sigma_j}$ **coefficiente di amplificazione relativo**:

$$\overline{A_j} = \max_{P \in D} \left( \frac{x_j \cdot \frac{\partial f}{\partial x_j}(P)}{f(P)} \right)$$

Per l'errore di arrotondamento relativo potevamo quindi prendere:

$$|\epsilon_j| \leq U$$

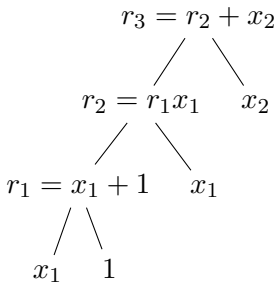
Vediamo come ultimo esempio il calcolo dell'errore relativo $|\epsilon_f|$ per la funzione:

$$f(x_1, x_2) = (x_1 + 1)x_1 + x_2$$

- Iniziamo con il calcolo dell'errore inerente, senza fare assunzioni su $\epsilon_{x_i}$:

$$|\epsilon_d| \leq \frac{x_1 \cdot \frac{\partial f}{\partial x_1}(x_1, x_2)}{f(x_1, x_2)} \epsilon_{x_1} + \frac{x_2 \cdot \frac{\partial f}{\partial x_2}(x_1, x_2)}{f(x_1, x_2)} \epsilon_{x_2} = \frac{x_1(2x_1 + 1)}{(x_1 + 1)x_1 + x_2} \epsilon_{x_1} + \frac{x_2}{(x_1 + 1)x_1 + x_2} \epsilon_{x_1}$$

- Calcoliamo poi l'errore algoritmico dell'algoritmo:




da cui:

$$|\epsilon_a| \leq \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} \epsilon_{r_2} + \frac{x_2}{(x_1 + 1)x_1 + x_2} \epsilon_{x_2} = \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} (\epsilon_2 + \epsilon_{r_1} + \epsilon_{x_1})$$

$$= \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} (\epsilon_2 + \epsilon_1 + \frac{x_1}{x_1 + 1} \epsilon_{x_1} + \frac{1}{x_1 + 1} \cdot 0) = \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} (\epsilon_2 + \epsilon_1)$$

Abbiamo quindi l'errore complessivo:

$$|\epsilon_f| \leq |\epsilon_a| + |\epsilon_d| = \epsilon_3 + \frac{(x_1 + 1)x_1}{(x_1 + 1)x_1 + x_2} (\epsilon_2 + \epsilon_1) + \frac{x_1(2x_1 + 1)}{(x_1 + 1)x_1 + x_2} \epsilon_{x_1} + \frac{x_2}{(x_1 + 1)x_1 + x_2} \epsilon_{x_1}$$

## 3.2 Errori di funzioni non razionali

Abbiamo finora trascurato il caso di funzioni non razionali. Prendiamo ad esempio di voler calcolare l'errore su funzioni come $e^{\cos(x+y)}$. In questo caso sarà necessario usare un approssimazione razionale di $f$ che chiamiamo $\overline{f}$, che poi porteremo a $\overline{f}_a$ che usa operazioni macchina detta **algoritmo**. In questo caso l'errore sarà dato dall'*errore inerente*, dall'*errore algortimico* e dall'**errore analitico** $\sigma_{an}$ della funzione, cioè potremo dire:

$$\overline{f}_a(P_1) - f(P_0) = \overline{f}_a(P_1) - \overline{f}(P_1) + \overline{f}(P_1) - f(P_1) + f(P_1) - f(P_0) = \sigma_a + \sigma_{an} + \sigma_d$$

L'errore inerente sarà calcolato sulla $f$ originale, mentre l'errore analitico sarà calcolato con la nuova $\overline{f}$, e in particolare dipenderà dall'approssimazione razionale che usiamo.

Vediamo per adesso approssimazioni polinomiali attraverso la **formula di Taylor**. Nel caso scalare si ha:

**Teorema 3.1: Formula di Taylor**

Data $f : \mathbb{R} \to \mathbb{R}$, $f \in C^1$, allora dato $x_0 \in \mathbb{R}$ si ha:

$$f(x) = \sum_{n=0}^{k} \frac{f^{(n)}(x_0)}{n!} \cdot (x - x_0)^n + \frac{f^{(k+1)}(\eta)}{(k+1)!}(x - x_0)^{k+1}$$

Dove:

$$\epsilon_l = \frac{f^{(k+1)}(\eta)}{(k+1)!}(x - x_0)^{k+1}$$

rappresenta l'**errore di Lagrange** al $k$-esimo grado, con $\eta \in [x_0, x]$ il punto di massimo della $k+1$-esima derivata di $f$.

In questo caso:

$$\overline{f}(x) = \sum_{n=0}^{k} \frac{f^{(n)}(x_0)}{n!} \cdot (x - x_0)^n$$

cioè la serie di Taylor troncata al $k$-esimo grado sarà una buona approssimazione per $f$, e l'errore analitico sarà dato da:

$$\sigma_{an} = R(x) = f(x) - T(x, k, x_0)$$

con $R(x)$ il resto fra lo sviluppo di Taylor troncato $T(x, k)$ e la funzione stessa $f(x)$. In questo caso fissato $k$ si potrà dare una stima di errore direttamente dall'errore di Lagrange, cioè:

$$R(x) = f(x) - T(x, k, x_0) = \frac{f^{(k+1)}(\eta)}{(k+1)!}(x - x_0)^{k+1}$$

e più in particolare, sfruttando il limite superiore dato da:

$$R(x) \leq \epsilon_l = \frac{\max_{[x_0,x]}\left(f^{(k+1)}\right)}{(k+1)!}(x - x_0)^{k+1} = \frac{M}{(k+1)!}(x - x_0)^{k+1}$$

### 3.2.1 Approssimazione dell'esponenziale

Prendiamo di volere calcolare l'errore dato dall'approssimazione dell'esponenziale al $k$-esimo grado su un intervallo comprensivo di $x_0 = 0$, che chiamiamo $D = [-l, u]$. Avremo quindi l'approssimazione:

$$e^x = T(x, k, 0) = \sum_{n=0}^{k} \frac{(x - x_0)^n}{n!}$$

e la funzione resto:

$$R(x) = e^x - T(x, k, 0) = \frac{f^{(k+1)}(\eta)}{(k+1)!}x^{k+1}$$

prendiamo quindi la stima superiore di $f^{(k+1)}(\eta)$ su $\eta \in D$ (il caso peggiore sarà quello in cui valutiamo $x$ sull'estremo superiore $u$):

$$f^{(k+1)}(\eta) \leq \max_{\eta \in [0,u]} f^{(k+1)}(\eta) = e^u$$

da cui:

$$R(x) \leq \frac{e^u x^{k+1}}{(k+1)!}$$

Stimiamo allora l'errore analitico relativo come:

$$|\epsilon_{an}| = \frac{R(x)}{f(x)} \leq \frac{\max_{[-l,u]} \left| \frac{e^u x^{k+1}}{(k+1)!} \right|}{\min_{[-l,u]} e^x} = \frac{e^{u-l} u^{k+1}}{(k+1)!}$$

### 3.2.2 Note sull'implementazione dell'esponenziale

Vediamo alcuni dettagli sull'implementazione pratica di un'approssimazione dell'esponenziale nel linguaggio MATLAB/Octave. Il primo modo che potrebbe venire in mente prevede di mantenere un numeratore e un denominatore:

```
function val = myexp1(x, n)
    num = 1;
    den = 1;

    val = 1;

    for i = 1:n
      num = num * x;
      den = den * i;

      val = val + num / den;
    end
end
```

Notiamo però che questo approccio presenta notevole instabilità numerica con grandi valori di $k$, ad esempio si veda:

```
>> myexp(20, 500)
ans =
   NaN
```

Questo deriva dal fatto che per grandi valori di $k$ si ottiene eventualmente `num = Inf` e `den = Inf`, da cui il `NaN`. Un'approccio migliore si ha quindi mantenendo direttamente il termine e accumulandolo:

```
function acc = myexp2(x, n)
    term = 1;
    acc = 1;

    for i = 1:n
        term = term * x / i;
        acc = acc + term;
    end
end
```

Notiamo però che in questo caso si hanno problemi di cancellazione per argomenti negativi, ad esempio si veda:

```
>> myexp(-30, 500)
ans =
  -3.0668e-05
```

In questo caso si ha più accuratezza imponendo argomenti positivi, ad esempio sfruttando:

$$e^{-x} = \frac{1}{e^x}$$

Si veda ad esempio:

```
>> 1/myexp(30, 500)
ans =
   9.3576e-14
```

Modifichiamo quindi la funzione per rilevare automaticamente esponenti negativi e adottare la forma corretta:

```
function acc = myexp3(x, n)
    neg = false;
    if x < 0
        neg = true;
        x = -x;
    end

    term = 1;
    acc = 1;

    for i = 1:n
        term = term * x / i;
        acc = acc + term;
    end

    if neg
        acc = 1 / acc;
    end
end
```

Provando questa funzione su un range di interi da $-30$ a $30$, con la serie di Taylor troncata a $500$, dà un errore massimo rispetto al valore reale di circa $10^{-16} \sim 10^{-17}$.

Veniamo quindi a fare alcuni richiami di algebra lineare. Nella maggior parte dei casi che ci interessano vorremo trattare non di scalari, ma di quantità vettoriali, ad esempio $x \in \mathbb{C}^n$, con $n > 1$.

## 3.3 Matrici complesse

Ci saranno utili le matrici perchè rappresentano direttamente tutte le **funzioni lineari**. Ad esempio, posta $f : \mathbb{C}^n \to \mathbb{C}^m$ tale che:

- $f(x + y) = f(x) + f(y)$ (addittività);
- $f(\lambda x) = \lambda f(x)$ (omogeneità)

detta *funzione lineare* allora $\exists! A \in \mathbb{C}^{m \times n}$ tale che $f(x) = Ax, \forall x \in \mathbb{C}^n$.

Nel corso useremo sia matrici in $\mathbb{R}$ che matrici in $\mathbb{C}$, dove l'appartenenza a ciascuno di questi campi dipende dalla appartenenza di essi delle **entrate** $A_{ij}$ della matrice. In ogni caso, una matrice reale non sarà che un caso particolare delle matrici complesse.

Si danno poi per scontate le definizioni di matrici:

- *Quadrate* ($n = m$):
- *Rettangolari* ($n \neq m$);
- *Diagonali* (elementi nulli fuori dalla diagonale);
- *Triangolari superiori/inferiori* (elementi nulli sotto/sopra la diagonale)

### 3.3.1 Trasposta coniugata

Definiamo infine la **trasposta coniugata** di una certa matrice, generalizzata al campo complesso dalla **matrice hermitiana**:

**Definizione 3.1: Trasposta coniugata**

Data una matrice $A \in \mathbb{C}^{n \times m}$, la trasposta coniugata $A^T$ sarà:

$$(A^T)_{ij} = A_{ji}$$

e la matrice hermitiana $A^H$ sarà:

$$(A^H)_{ij} = \overline{A_{ji}}$$

## 3.4 Algebra vettoriale

Diamo alcune nozioni riguardo alle operazioni fra vettori.

### 3.4.1 Prodotto scalare

Diamo la definizione di prodotto scalare, generalizzato al campo complesso dal **prodotto hermitiano** (entrambi *prodotti interni*):

**Definizione 3.2: Prodotto interno**

Definiamo il prodotto interno fra due vettori $x, y \in \mathbb{C}^n$ come:

$$< x, y >= \sum_{j=1}^{n} x_j \overline{y_j}$$

dove $\overline{y_j}$ rappresenta il **coniugato** di $y_j$, che chiaramente in $\mathbb{R}$ si riduce a $y_j$ stesso e quindi:

$$< x, y >= \sum_{j=1}^{n} x_j y_j, \quad x, y \in \mathbb{R}^n$$

### 3.4.2 Indipendenza lineare

Diamo la definizione di indipendenza linare:

**Definizione 3.3:**

Un insieme di vettori $\{x_1, ..., x_s\}$ si dice linearemente indipendente se:

$$x_1 + ... + x_s = 0 \leftrightarrow x_1, ..., x_s = 0$$

Possiamo sfruttare l'indipendenza lineare per definire **basi**:

**Definizione 3.4: Base**

Se $s = n$ l'insieme $\{x_1, ..., x_s\}$ si dice **base** di $\mathbb{C}^n$.

Questo significa che $\forall y \in \mathbb{C}^m$, $\exists!\{c1, ..., c_s\}$ tali che $y = \sum_{j=1}^{n} c_j x_j$, cioè combinazioni lineari dei vettori di base individuano qualsiasi vettore nello spazio $\mathbb{C}^n$.

## 3.5 Operazioni fra matrici

Date due matrici $A$ e $B$ con lo stesso numero di righe e colonne si possono definire le operazioni:

- **Somma:** $A, B, C \in \mathbb{C}^{m \times n}$, $A + B = C$, $c_{ij} = a_{ij} + b_{ij}$;
- **Prodotto:** $A \in \mathbb{C}^{m \times n}, B \in \mathbb{C}^{n \times p}, C \in \mathbb{C}^{m \times p}$, $A \cdot B = C$, $c_{ij} = \sum_{h=1}^{n} a_{ih} b_{hj}$ sia in reali che in complessi.

### 3.5.1 Considerazioni computazionali sulle operazioni matriciali

Dal punto di vista computazionale, è immediato che il prodotto scalare ha complessità $O(n)$, la somma matriciale ha complessità $O(m \cdot n)$.

Per la moltiplicazione matriciale, poste:

$$A \in \mathbb{C}^{m \times n}, \quad B \in \mathbb{C}^{n \times p} \implies A \cdot B = C \in \mathbb{C}^{m \times p}$$

si ha che la complessità è $O(m \cdot n \cdot p)$.

L'ultimo risultato dipende dal fatto che la moltiplicazione richiede di effetuarre effettivamente $m \cdot p$ prodotti scalari, e $n$ è la lunghezza di uno di questi prodotti scalari, cio il numero di colonne di $A$ o di righe di $B$ (che sappiamo essere uguali perché la moltplicazione sia possibile in primo luogo).

Questo significa che per matrici quadrate si ha generalmente complessità $O(n^3)$ (anche se esistono algoritmi che si avvicinano al bound inferiore di $O(n^2)$).

# 4 Lezione del 07-03-25

Proseguiamo i richiami di algebra lineare.

### 4.0.1 Approfondimento sulle prestazioni delle operazioni matriciali

Usiamo MATLAB per studiare il tempo di computazione richiesto per effettuare alcune delle operazioni che abbiamo visto.

- Potremmo iniziare con il **prodotto scalare**. Avevamo detto che la complessità era $O(n)$. Scriviamo allora uno script per valutare, in scala logaritmica, il prodotto scalare fra vettori di dimensioni crescenti:

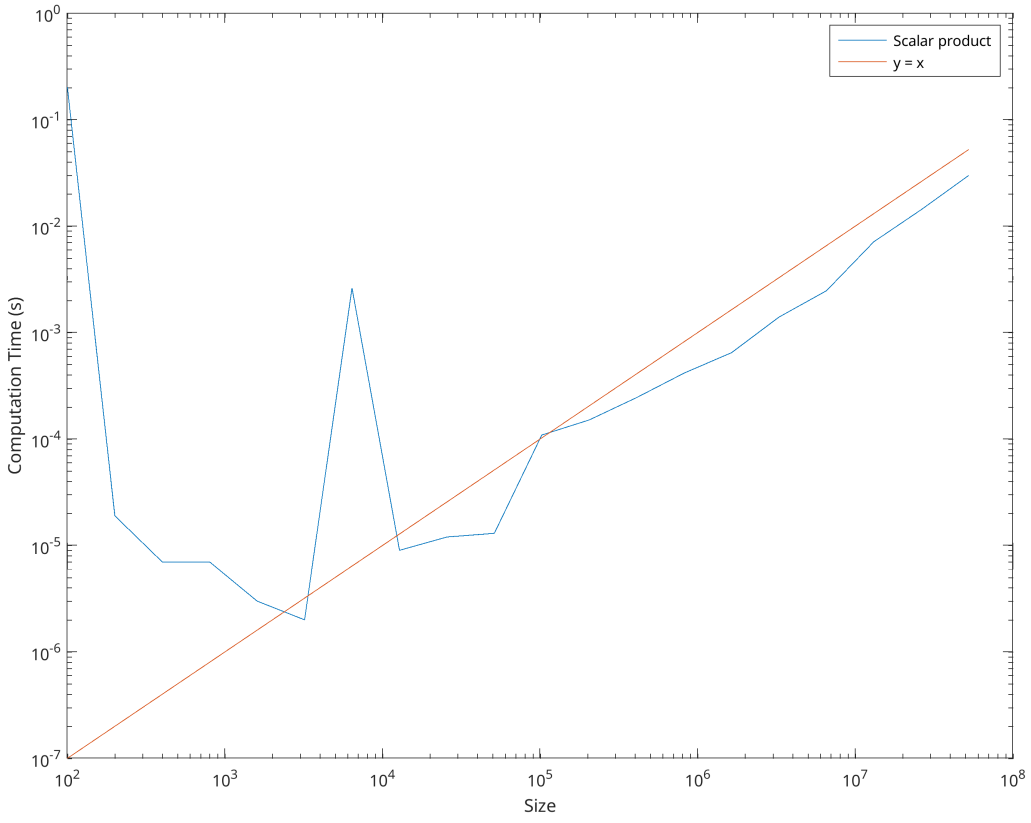
```
function n = time_scalar(num, base, mul)
    if nargin < 3
        mul = 100;
    end
    if nargin < 2
        base = 2;
```

```
    end

    n = zeros(num, 1);
    for i = 1:num
        n(i) = base^(i - 1) * mul;
    end

    times = zeros(num, 1);

    for i = 1:num
        A = randn(1, n(i));
        B = randn(n(i), 1);
        tic;
        C = A * B;
        times(i) = toc;
    end

    loglog(n, times);
    xlabel('Size');
    ylabel('Computation Time (s)');
end
```

Un esempio di esecuzione potrebbe allora essere:

```
>> n = time_scalar(20);
>> hold on;
>> loglog(n, n * 10^-9);
>> hold off;
```

dove si è cercato di fare il *fitting* della curva ottenuta con la funzione $y = x$ (dove la variabile è chiaramante $n$, e si è aggiunta una costante moltiplicativa $k$ per avvicinarci di più al grafico originale). Il grafico ottenuto è quindi:



Vediamo dal grafico che le prestazioni sono effettivamente vicine a quelle previste dalla complessità computazionale, se non per errori dati all'overhead di inizializzazione del timer `tic` e `toc` di MATLAB (lato sinistro del grafico) e della velocità generale molto elevata delle operazioni, che rende più visibile il rumore dato dallo scheduling della CPU e altri fattori di basso livello.

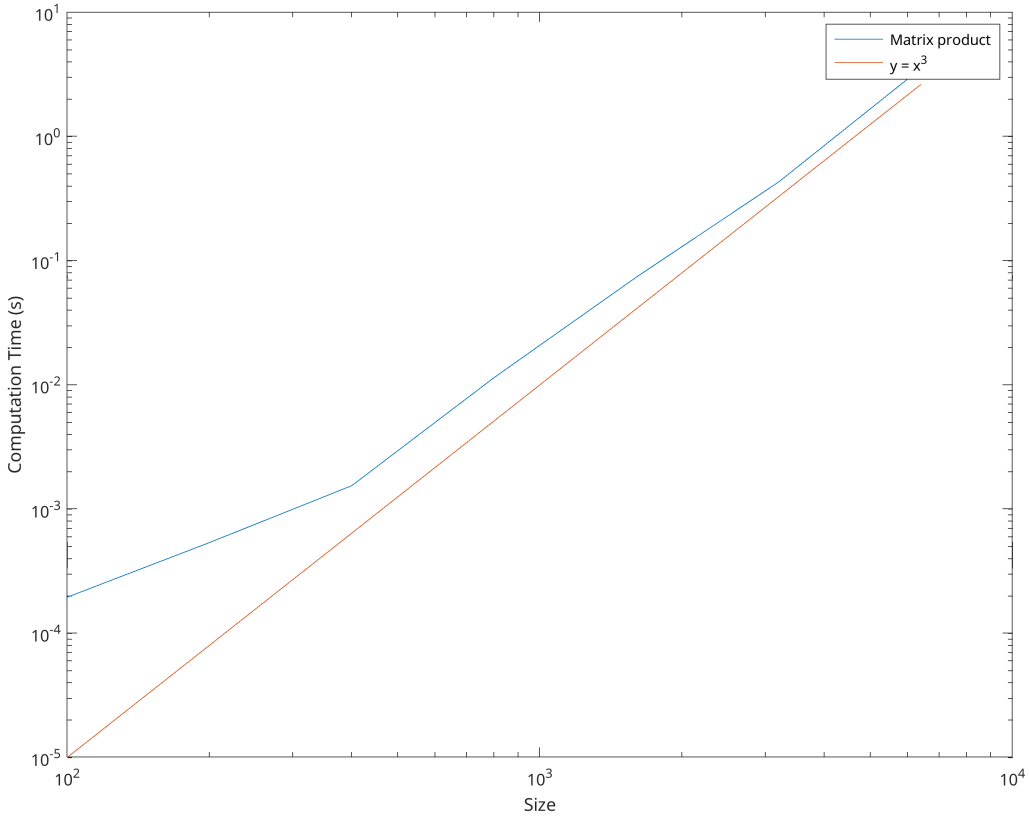
- Vediamo allora l'andamento del **prodotto fra matrici**. Lo script MATLAB è praticamente identico al precedente:

```
function n = time_matrix(num, base, mul)
    if nargin < 3
        mul = 100;
    end
    if nargin < 2
        base = 2;
    end

    n = zeros(num, 1);
    for i = 1:num
        n(i) = base^(i - 1) * mul;
    end

    times = zeros(num, 1);

    for i = 1:num
        A = randn(n(i));
        B = randn(n(i));
        tic;
        C = A * B;
        times(i) = toc;
    end

    loglog(n, times);
    xlabel('Size');
    ylabel('Computation Time (s)');
end
```

che eseguito come:

```
>> n = time_matrix(7);
>> hold on;
>> loglog(n, n.^3 * 10^-11);
>> hold off;
```

ci restituisce il grafico:




che notiamo è già più vicino del prodotto scalare (a causa del tempo di esecuzione maggiore richiesto, che "maschera" bene gli effetti a basso livello della CPU).

### 4.0.2 Proprietà del prodotto matriciale

Abbiamo che in genere il prodotto fra matrici non è **commutativo**, cioè:

$$AB \neq BA$$

di contro, vale l'**associativa**:

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

e la **distributiva**, separatamente ai due lati:

$$(A + B) \cdot C = A \cdot C + B \cdot C$$

$$C \cdot (A + B) = C \cdot A + C \cdot B$$

Inoltre, notiamo che quello delle matrici non è un *dominio integrale*:

$$A \cdot B = 0 \not\Longrightarrow A = 0 \vee B = 0$$

Vediamo ad esempio come sfruttare la proprietà associativa può permetterci di ottenere algoritmi più veloci. Supponiamo di voler calcolare:

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

con $A \in \mathbb{C}^{m \times n}$; $B \in \mathbb{C}^{n \times p}$ , $C \in \mathbb{C}^{p \times q}$.

L'uguaglianza ci darà due metodi:

- $(A \cdot B) \cdot C$:
  - $A \cdot B = P \in \mathbb{C}^{m \times p}, \quad O(m \cdot n \cdot p)$
  - $P \cdot C = R \in \mathbb{C}^{m \times q}, \quad O(m \cdot p \cdot q)$

  $\implies O(mnp + mpq) = O\left(mp(n+q)\right)$
- $A \cdot (B \cdot C)$:
  - $B \cdot C = P \in \mathbb{C}^{n \times q}, \quad O(n \cdot p \cdot q)$
  - $A \cdot P = R \in \mathbb{C}^{m \times q}, \quad O(m \cdot n \cdot q)$

  $\implies O(npq + mnq) = O\left(nq(p+m)\right)$

che sono quindi uguali per $m = n = p$, ma (anche radicalmente diversi) diversi al variare di questi parametri.

In generale, quindi, se si hanno matrici con poche righe o poche colonne, è opportuno cercare di mantenere questa properietà più a lungo possibile nei risultati intermedi.

Possiamo fare considerazioni simili a quelle fatte sull'associativa con la distributiva. Ad esempio, posta l'uguaglianza:

$$(A + B) \cdot C = AC + BC$$

con $A, B \in \mathbb{C}^{m \times n}$ e $C \in \mathbb{C}^{n \times p}$, abbiamo due metodi:

- $(A + B) \cdot C$:
  - $A + B = P \in \mathbb{C}^{m \times n}, \quad O(m \cdot n)$
  - $P \cdot C = R \in \mathbb{C}^{m \times p}, \quad O(m \cdot n \cdot p)$

  $\implies O(mn + mnp) = O\left(mn(1+p)\right)$
- $AC + BC$:
  - $A \cdot C = P_1 \in \mathbb{C}^{m \times p}, \quad O(m \cdot n \cdot p)$
  - $B \cdot C = P_2 \in \mathbb{C}^{m \times p}, \quad O(m \cdot n \cdot p)$

  $\implies O(mnp + mnp + mp) = O\left(mp(1+2n)\right)$

che sono sì asintoticamente uguali per $m = n = p$ ($O(n^3)$), ma non esattamente uguali in quanto da un lato si ha $O(n^2 + n^3)$ e dall'altro $O(n^2 + 2n^3)$. Questo risulta chiaramente dal fatto che col secondo metodo decidiamo di effettuare il prodotto matriciale (che è l'operazione più costosa) non una ma 2 volte.

Un altra proprietà del prodotto di matrici e che si può vedere il risultato riga per riga o colonna per colonna nei seguenti modi:

$$A = \begin{pmatrix} A_1 \\ ... \\ A_m \end{pmatrix}, \quad B = \begin{pmatrix} B_1 & ... & B_p \end{pmatrix}$$

$$\implies A \cdot B = \begin{pmatrix} A \cdot B_1 & ... & A \cdot B_p \end{pmatrix} = \begin{pmatrix} A_1 B \\ ... \\ A_m B \end{pmatrix}$$

Questo ci dice che le colonne (delle righe) di $C = A \times B$ sono combinazioni lineari delle colonne (delle righe) di $A$ (di $B$).

## 4.1 Determinante

Abbiamo visto la definizione di **determinante** per matrici quadrate:

**Definizione 4.1: Determinante**

Si definisce la funzione:

$$\det(A) : \mathbb{C}^{n\times n} \to \mathbb{C}$$

con:

$$\det(A) = \begin{cases} a_{11}, & n = 1 \\ a_1 1 a_2 2 - a_1 2 a_2 1, & n = 2 \\ \sum_{j=1}^{n} (-1)^{i+j} \det(A_{ij}), & n > 2 \quad \text{(sviluppo di Laplace)} \end{cases}$$

determinante.

Osserviamo che il calcolo del determinante attraverso lo sviluppo di Laplace ha complessità algoritmica $O(n!)$.

### 4.1.1 Proprietà del determinante

Sappiamo che $A$ invertibile $\Leftrightarrow \det(A)$ (cioè $A$ **singolare**). Inoltre valgono:

$$\det(A^T) = \det(A)$$

$$\det(A^H) = \overline{\det(A)}$$

## 4.2 Rango

Vediamo poi come ci può servire il determinante delle **sottomatrici**:

**Definizione 4.2: Sottomatrice**

Una sottomatrice di $A$ è una matrice ottenuta prendendo la restrizione di $A$ a un sottoinsieme di righe e di colonne, cioè data $A \in \mathbb{C}^{n\times n}$, $I, J \subseteq \{1, ..., n\}$ con $|I| = n_1$ e $|J| = n_2$, sarà allora:

$$A(I, J) \in \mathbb{C}^{n_1 \times n_2}$$

ottenuta incrociando le righe in $I$ con le colonne in $J$.

Definiamo quindi i **minori**:

**Definizione 4.3: Minore**

Si dice minore di ordine $k$, con $k \in \{1, ..., n\}$ il determinante di una sottomatrice quadrata $k \times k$.

E le sottomatrici **principali** e **principali di testa**:

**Definizione 4.4: Sottomatrice principale**

Una sottomatrice si dice principale se gli insiemi $I, J$ usati per estrarla sono $I = J$.

**Definizione 4.5: Sottomatrice principale di testa**

Una sottomatrice quadrata di ordine $k$ si dice sottomatrice principale di testa se $I = J = \{1, ..., k\}$.

Allo stesso modo si possono definire **matrici di coda** (basterà prendere indici da $k$ ad $n$).

Possiamo quindi definire il **rango** di matrice:

**Definizione 4.6: Rango**

Il rango $\text{rank}(A)$ di una matrice $A$ è definito come il massimo numero di colonne (o di righe) linearmente indipendenti, ed è uguale all'ordine massimo dei minori $\neq 0$ in $A$.

### 4.2.1 Proprietà del rango

Caso particolare del rango sarà chiaramente quello dove prendiamo come sottomatrice la matrice stessa: $\det(A) \neq 0 \Leftrightarrow \text{rank}(A) = n$, e quindi gli insiemi delle colonne e delle righe di $A$ sono tutte linearmente indipendenti.

Si ha poi che:

$$\text{rank}(A) = \dim(\text{Im}(A))$$

dove $\text{Im}(A)$ è la dimensione dell'**immagine** di $A$:

$$\text{Im}(A) = \{y \in \mathbb{C}^m : y = Ax,\ x \in \mathbb{C}^n\}$$

### 4.2.2 Teorema di Binet-Cauchy

Concludiamo dimostrando il teorema di Binet-Cauchy sul determinante rispetto al prodotto matriciale.

**Teorema 4.1: di Binet-Cauchy**

Prese due matrici, $A \in \mathbb{C}^{n\times n}$ e $B \in \mathbb{C}^{n\times n}$, e $C = A \cdot B$ di dimensioni uguali, sarà:

$$\det(C) = \det(A) \cdot \det(B)$$

Nel caso generale avremo $A \in \mathbb{C}^{n\times p}$ e $B \in \mathbb{C}^{p\times n}$, da cui:

$$\det(C) = \begin{cases} 0, & p \leq n \\ \sum_j A_{[j]} \cdot B_{[j]} \end{cases}$$

dove $A_{[j]}$ e $B_{[j]}$ sono i minori di ordine $n$ relativi ala stessa scelta di indici in $A$ e in $B$.

## 4.3 Matrice inversa

Diamo la definizione di **matrice inversa**:

**Definizione 4.7: Matrice inversa**

Data $A \in \mathbb{C}^{n\times n}$ tale che $\det(A) \neq 0$, si definisce $A^{-1} \in \mathbb{C}^{n\times n}$ tale che:

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

Se guardo ad $A$ come una funzione lineare $A : \mathbb{C}^n \to \mathbb{C}^n$, sarà che:

$$A : x \to Ax, \quad A^{-1} : Ax \to x$$

questo da:

$$\left(A^{-1} \circ A\right) = A^{-1}y = A^{-1}Ax = Ix = x$$

### 4.3.1 Proprietà della matrice inversa

Vediamo alcune proprietà di $A^{-1}$:

1. Ricordiamo di nuovo che $A^{-1}$ esiste se e solo se $\det(A) \neq 0$, cioè equivalentemente se $\text{rank}(A) = n$, o $A$ ha spazi riga e colonna linearmente indipendenti;
2. Vediamo poi il calcolo di $\det(A^{-1})$: da $\det(I) = 1$, e:

$$\det(A \cdot A^{-1}) = \det(A) \cdot \det(A^{-1}), \quad \text{(Binet)}$$

$$\implies \det(A^{-1}) = \frac{1}{\det(A)}$$

3. Vediamo poi che:

$$(A \cdot B)^{-1} = B^{-1}A^{-1}$$

   per matrici quadrate $A, B \in \mathbb{C}^{n\times n}$;

4. Riguardo alle trasposte e alle Hermitiane vale:

$$(A^T)^{-1} = (A^{-1})^T = A^{-T}$$

$$(A^H)^{-1} = (A^{-1})^H = A^{-H}$$

   e:

$$(AB)^T = B^T A^T$$

$$(AB)^H = B^H A^H$$

## 4.4 Matrici particolari

Definiamo alcune matrici particolari:

**Definizione 4.8: Matrici particolari**

Data $A \in \mathbb{C}^{n \times n}$, si dice di $A$ che è:

- **Hermitiana:** $A = A^H$;
- **Antiermitiana:** $A = -A^H$;
- **Unitaria** $A^H A = AA^H = I$, $A^{-1} = A^H$;
- **Normale** $A^H A = AA^H$;
- **Simmetrica** $A = A^T$;
- **Antisimmetrica** $A = -A^T$;
- **Ortogonale** $A^T A = AA^T = I$, $A^T = A^{-1}$

dove notiamo che **simmetrica** e **antisimmetrica** significano *hermitiana* e *antihermitiana* in $\mathbb{R}$, e **ortogonale** significa *unitaria* in $\mathbb{R}$.

Per di più vale:

$$\{\text{matrici simmetriche reali}\} \subseteq \{\text{matrici hermitiane}\} \subseteq \text{matrici normali}$$

e:

$$\{\text{matrici ortogonali}\} \subseteq \{\text{matrici unitarie}\}$$

dove notiamo che unitarie ed ortogonali hanno l'inversa "facile", nel senso che basta trasporre o trovare l'hermitiana.

## 4.5 Matrici di permutazione

Definiamo le **matrici di permutazione**:

**Definizione 4.9: Matrice di permutazione**

$A \in \mathbb{R}^{n \times n}$ si dice matrice di permutaione se $A$ si ottiene dalla matrice di identità permutandone righe e colonne.

### 4.5.1 Proprietà delle matrici di permutazione

Tutte le matrici di permutazione sono matrici ortogonali (questo si vede dalla loro unimodularità, o osservando che $P^T$ si ottiene dalla permutazione inversa).

Inoltre, vediamo che il prodotto di $A$ con una matrice di permutazione si limita a scambiare righe e colonne in accordo con la permutazione della matrice. In particolare, $A \cdot P$ dà la permutazione delle colonne, mentre $P \cdot A$ dà la permutazione delle righe.

## 4.6 Sistemi lineari

Abbiamo un sistema di $m$ equazioni lineari in $n$ incognite:

$$\begin{cases} a_{11}x_1 + ...a_{1n}x_n = b_1 \\ ... \\ a_{m1}x_1 + ...a_{mn}x_n = b_m \end{cases}$$

Ci è spesso utile riscrivere sistemi di questo tipo in forma matriciale:

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \in \mathbb{C}^{m \times n}, \quad x = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} \in \mathbb{C}^n, \quad b = \begin{pmatrix} b1 \\ \dots \\ b_m \end{pmatrix} \in \mathbb{C}^m$$

Il problema principe sarà quello, date $A$ e $b$, di trovare $x$.

### 4.6.1 Teorema di Rouché-Capelli

Richiamiamo il teorema:

**Teorema 4.2: di Rouché-Capelli**

$Ax = b$ ammette almeno una soluzione se $\text{rank}(A) = \text{rank}(A|b)$, con $(A|b) \in \mathbb{C}^{m \times (n+1)}$ la matrice ottenuta aumentando $A$ con $b$.
Per quanto riguarda l'unicità, supposto $m \geq n$ (sistema *sovradeterminato*):

- Se $\text{rank}(A) = n$ allora la soluzione è unica;
- Se $\text{rank}(A) < n$ allora ci sono $\infty$ soluzioni, e l'insieme delle soluzioni forma uno spazio vettoriale affine di dimensinoe $n - \text{rank}(A)$;

Nel caso il vettore $b = 0$, il sistema si dice **omogeneo**, e la soluzione nulla esiste sempre.
Abbiamo poi che:

**Definizione 4.10:**

L'insieme delle soluzioni di $Ax = 0$ si chiama Kernel (nucleo) della matrice:

$$\ker(A) = \{x \in \mathbb{R}^n : Ax = 0\}$$

notiamo che il kernel è un sottospazio vettoriale di dimensione $n - \text{rank}(A)$.
Osserviamo infine che nel caso $m = n$, se $\det(A) \neq 0$, cioè $\text{rank}(A) = n$, òa soluzione di $Ax = b$ è unica per ogni $b \in \mathbb{R}^m$ e si scrive:

$$x = A^{-1}b$$

Vedremo che in generale calcolare l'inversa non è conveniente rispetto ad altri metodi di approssimazione.

### 4.6.2 Regola di Cramer

Il vettore soluzione si può scrivere ache componente per componente come:

$$x_j = \frac{\det(A_j)}{\det(A)}$$

dove $A_j$ è la matrice ottenuta da $A$ sotituendo la colonna $j$ con $b$.
Questa via costa come calcolare $O(n)$ determinanti di matrici $n \times n$, ed è quindi poco conveniente ($O(n^3)$).