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 β 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 π 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 β. 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 β, 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 β 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	β	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 ∞, 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 α_{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 β^{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 β^{L-1} con distanza β^{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^π. I passaggi nell'ordine appena visto saranno:

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

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

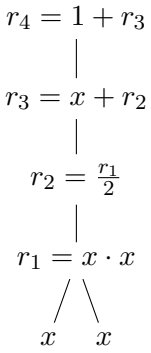
2. Si riporta la $\overline{f}(x)$ a $\overline{f}_a(x)$:

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

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 π 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 ϵ_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	σ_d	ϵ_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.

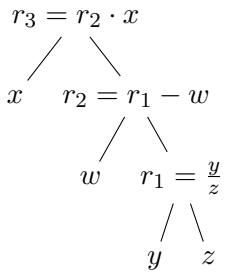
L'idea è di seguire l'errore generato dall'algoritmo sul grafo (o albero) che lo rappresenta sfruttando le relazioni per l'errore inerente nelle 4 operazioni aritmetiche. Prendiamo ad esempio la funzione:

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

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 ϵ_3, ϵ_2 e ϵ_1 saranno termini associati ad ogni risultato intermedio che rappresenteranno gli errori di troncamento dei risultati intermedi stessi, e ϵ_{r3}, ϵ_{r2} e ϵ_{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 ϵ_i legati ad ogni variabile (come avevamo detto sopra, gli operandi sono considerati come privi di errore di arrotondamento). Per la stima di ϵ_3, ϵ_2 e ϵ_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:

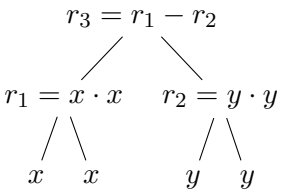
1. Si prendono i risultati intermedi:

$$r_1 = x \cdot x$$

$$r_2 = y \cdot y$$

$$r_3 = x - y$$

Da cui il grafo:



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

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

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

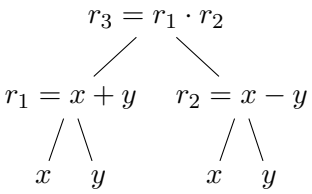
2. Un altro approccio è quello di prendere i risultati intermedi:

$$r_1 = x + y$$

$$r_2 = x - y$$

$$r_3 = r_1 \cdot r_2$$

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 ϵ_a e ϵ_d (σ_a e σ_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 σ_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:

  $$\left|\frac{1}{x_2}\right| \approx \frac{1}{4}, \quad \left|\frac{x_1}{x_2^2}\right| \approx \frac{3}{16}$$

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

  $$|\sigma_d| \leq \frac{1}{4} \cdot \frac{1}{2} \cdot 10^{-2} + \frac{3}{16} \cdot \frac{1}{2} \cdot 10^{-2} = \left(\frac{1}{8} + \frac{7}{32}\right) \cdot 10^{-2} = \frac{7}{32} \cdot 10^{-2}$$

- 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 σ_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 σ_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** σ_f e errore **relativo** ϵ_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:

$$|\epsilon_j| \leq U$$

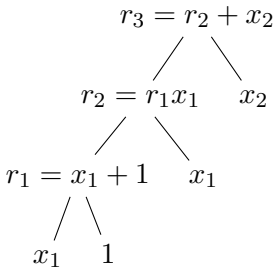
Vediamo come ultimo esempio il calcolo dell'errore relativo $|\epsilon_f|$ per la funzione:

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

- Iniziamo con il calcolo dell'errore inerente, senza fare assunzioni su ϵ_{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** σ_{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:

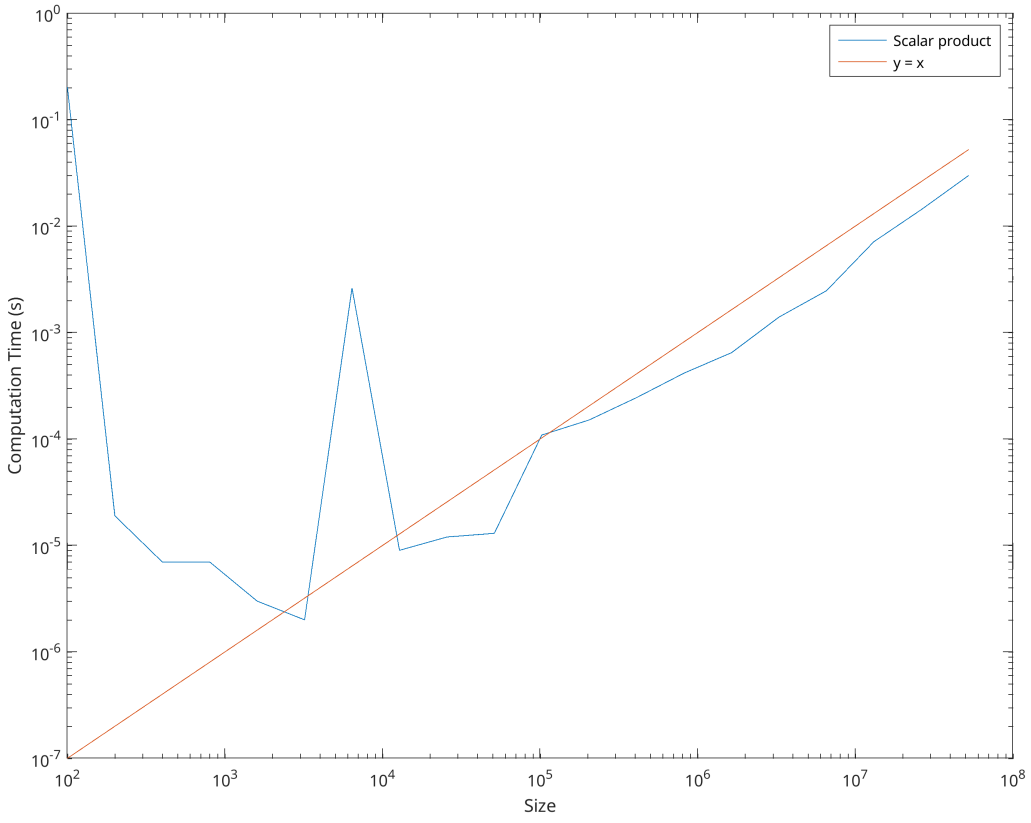
```
function n = time_scalar(num, base, mul)
    if nargin < 3
        mul = 100;
    end
    if nargin < 2
        base = 2;
```

```
    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.

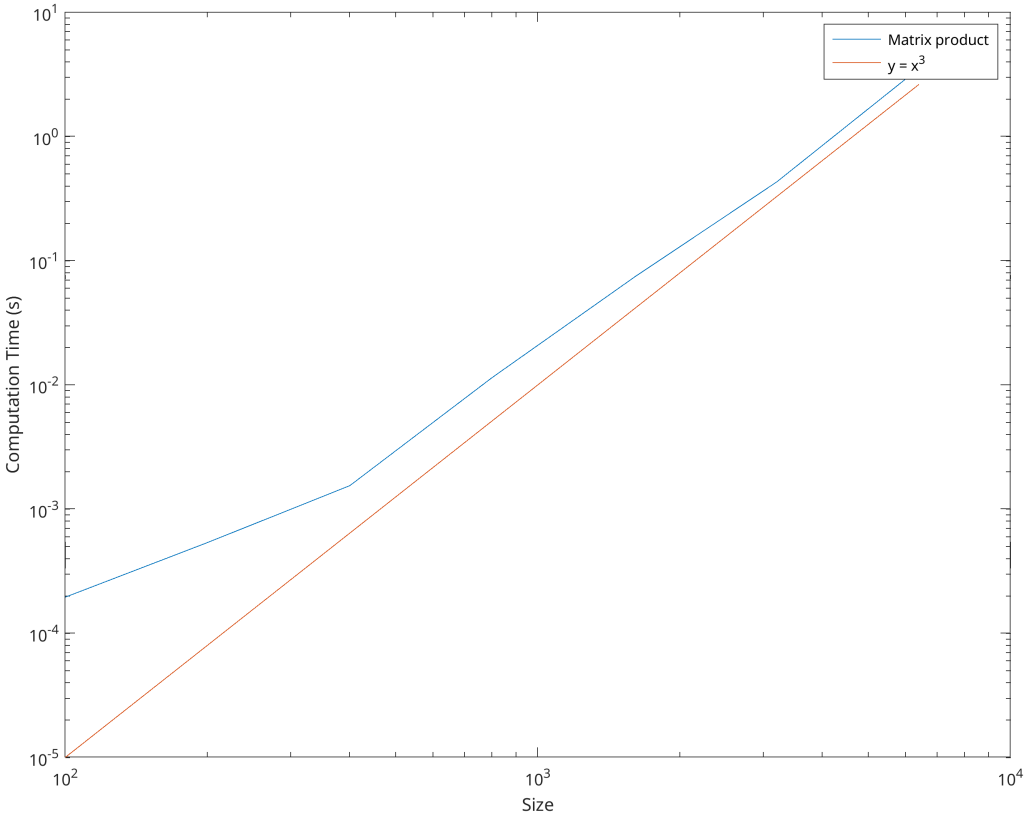
- Vediamo allora l'andamento del **prodotto fra matrici**. Lo script MATLAB è praticamente identico al precedente:

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

4.1 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 (le righe) di $C = A \times B$ sono combinazioni lineari delle colonne (delle righe) di A.

4.2 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:

$$\det(A) = \begin{cases} a_{11}, & n = 1 \\ a_{11}a_{22} - a_{12}a_{21}, & n = 2 \\ \sum_{j=1}^{n}(-1)^{i+j}\det(A_{ij}), & n > 2 \quad \text{(sviluppo di Laplace)} \end{cases}$$

determinante.

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

4.2.1 Proprietà del determinante

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

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

4.3 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.

4.3.1 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:

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

4.3.2 Teorema di Binet-Cauchy

Concludiamo enunciando 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.

La dimostrazione di questo teorema si basa sullo sviluppo dei cofattori della matrice $A \cdot B$, dove si possono riarrangiare i termini per ottenere gli sviluppi dei cofattori delle matrici A e B.

4.4 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 si guarda 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:

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

□

4.4.1 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:

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

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

4.5 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.

4.6 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 permutazione se A si ottiene dalla matrice di identità permutandone righe e colonne.

4.6.1 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.

4.7 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:

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

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

4.7.1 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 ∞ soluzioni, e l'insieme delle soluzioni forma uno spazio vettoriale affine di dimensione $n - \text{rank}(A)$;

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 $\text{rank}(A) = n$, la 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.

4.7.2 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 ($O(n^3)$).

5 Lezione del 10-03-25

5.1 Autovalori e autovettori

Continuiamo a parlare di matrici quadrate, e introduciamo gli **autovalori** e **autovettori**:

Definizione 5.1: Autovalori e autovettori destri

Data $A \in \mathbb{C}^{n \times n}$, $\lambda \in \mathbb{C}$ si dice autovalore se esiste un vettore v in $\mathbb{C}^n$, con $v \neq 0$, tale che:

$$Av = \lambda v$$

L'equazione $Av = \lambda v$ non rappresenta un sistema lineare, in quanto sia v che λ sono variabili (e fra l'altro moltiplicate fra loro).

In particolare, v si dice **autovettore destro** per A rispetto a λ. Similmente, si può definire w **autovettore sinistro**:

Definizione 5.2: Autovalori e autovettori sinistri

Data $A \in \mathbb{C}^{n \times n}$, $\lambda \in \mathbb{C}$ si dice autovalore se esiste un vettore w in $\mathbb{C}^n$, con $w \neq 0$, tale che:

$$w^H A = w^H \lambda$$

Solitamente ci basta trovare gli autovettori destri in quanto:

$$(w^H A)^H = A^H w$$

$$(\lambda w^H)^H = \overline{\lambda} w$$

cioè w è un autovettore destro per λ se e solo se w è autovettore destro per A^H rispetto all'autovalore $\overline{\lambda}$. In questo caso, se la matrice è *simmetrica* o *hermitiana*, gli autovalori destri e sinistri coincidono.

5.1.1 Caratterizzazione degli autovalori

Potremmo interrogarci sull'esistenza di questi oggetti. Poniamo allora:

$$Av = \lambda v \implies Av - \lambda v = 0 \implies (A - \lambda I)v = 0$$

Chiaramente il sistema è risolto per $v = 0$, ma abbiamo escluso a priori tale possibilità. Abbiamo allora un sistema $A - \lambda I$, che ha almeno una soluzione per Rouché-Capelli, che è 0 se $\det(A - \lambda I) \neq 0$, e un valore $\neq 0$ altrimenti. Vogliamo quindi impostare:

$$p(\lambda) = \det(A - \lambda I) = 0$$

detta **equazione caratteristica** (del *polinomio caratteristico* posto nullo). Gli autovalori non saranno altro che le soluzioni dell'equazione caratteristica.

L'esistenza di tali soluzioni è data dal fatto che $\deg(p(\lambda)) = n$, e quindi dal teorema fondamentale dell'algebra esistono $c_0, c_1, ..., c_n$ tali che:

$$p(\lambda) = c_0 + c_1\lambda + ... + c_n\lambda^n = c_n(\lambda - \lambda_1)...(\lambda - \lambda_n) \sim (-1)^n(\lambda - \lambda_1)...(\lambda - \lambda_n)$$

e dal punto di vista dei minori:

$$= (-1)^n\lambda^n + (-1)^{n-1}\sigma_1\lambda^{n-1} + ... + (-1)^1\sigma_{n-1}\lambda + (-1)^0\sigma_n$$

$$= (-1)^n\lambda^n + (-1)^{n-1}\sigma_1\lambda^{n-1} + ... - \sigma_{n-1}\lambda + \sigma_n$$

dove σ_j è la somma dei determinanti delle sottomatrici di A di ordine j. □

Alcuni di questi σ_j sono semplici da calcolare:

$$\sigma_1 = \sum_{i=1}^{n} a_{ii} = \mathrm{tr}(A) \quad \text{(traccia)}$$

$$\sigma_n = \prod_{j=1}^{m} \lambda_j = \det(A)$$

vediamo quest'ultima relazione: si ha che il determinante di una matrice è uguale al prodotto dei suoi autovalori, ergo:

$$\det(A) = 0 \Leftrightarrow \exists\lambda_j = 0$$

Riepilogando una matrice $\in \mathbb{C}^{n\times n}$ ha sempre esattamante n autovalori, contati con la loro molteplicità. In particolare:

Definizione 5.3: Molteplicità algebrica

Un autovalore $\overline{\lambda}$ ha molteplicità algebrica k ($\alpha(\overline{\lambda}) = k$) se k è la sua molteplicità μ come radice del polinomio caratteristico.

Chiaramente:

$$\sum_{\overline{\lambda}} \alpha(\overline{\lambda}) = n$$

5.1.2 Caratterizzazione degli autovettori

Potremmo chiederci quanti sono gli autovettori. Abbiamo allora che:

$$Av = \lambda v$$

e preso $\theta \in \mathbb{C}$ allora:

$$A(\theta v) = \theta Av = \theta\lambda v = \lambda(\theta v)$$

cioè anche θv è autovalettore per $\lambda^* = \theta\lambda$. Questo ci dice che gli autovettori non sono mai unici. Inoltre, possiamo dire che dati $Av_1 = \lambda v_1$ e $Av_2 = \lambda v_2$:

$$A(v_1 + v_2) = Av_1 + Av_2 = \lambda v_1 + \lambda v_2 = \lambda(v_1 + v_2)$$

cioè sono soddisfatte le proprietà di omogeneità e additività, e quindi l'insieme degli autovettori relativi a un certo autovalore λ forma uno spazio vettoriale (detto **autospazio** rispetto a λ).

Questo ci permette di definire un'altra molteplicità:

Definizione 5.4: Molteplicità geometrica

Dato $\lambda \in \mathbb{C}$ autovalore di $A \in \mathbb{C}^{n \times n}$ si dice molteplicità geometrica k ($\gamma(\lambda) = k$) la dimensione dell'autospazio associato a λ, ovvero:

$$\gamma(\lambda) = \dim\left(\ker(A - \lambda I)\right)$$

Per Rouché-Capelli, si ha che:

$$\gamma(\lambda) = n - \operatorname{rank}(A - \lambda I)$$

Valgono poi le relazioni fra molteplicità algebrica e molteplicità geometrica:

$$1 \leq \gamma(\lambda) \leq \alpha(\lambda) \leq n$$

Inoltre se si hanno $\lambda_i \neq \lambda_j$ autovalori distinti ($\forall i \neq j,\ i, j \in \{1, ..., n\}$), allora necessariamente:

$$\gamma(\lambda_i) = \alpha(\lambda_i) = 1, \quad \forall i = 1, ..., n$$

cioè se tutti gli autovalori di A sono distinti, le loro molteplicità, sia α che γ, devono essere uguali a 1.

5.1.3 Matrici simili

Possiamo quindi definire la **similarità** fra matrici:

Definizione 5.5: Matrici simili

Date $A, B \in \mathbb{C}^{n \times n}$, si dicono simili ($B \sim A$) se esiste $S \in \mathbb{C}^{n \times n}$ invertibile tale che:

$$B = S^{-1}AS$$

in questo caso vale anche l'inverso:

$$A = SBS^{-1}$$

Le matrici fra di loro simili hanno diverse proprietà in comune. Ad esempio, $A \sim B$ hanno gli stessi autovalori con le stesse molteplicità α e γ. Inoltre se v è autovettore per A associato a λ, allora $S^{-1}v$ è autovettore per B associato sempre a λ. Questo si dimostra da:

$$\det(B - \lambda I) = 0 = \det(S^{-1}AS - \lambda I) = \det(S^{-1}AS - \lambda S^{-1}S) = \det(S^{-1}(A - \lambda I)S)$$

e da Binet-Cauchy:

$$\det(S^{-1}(A - \lambda I)S) = \det(S^{-1})\det(A - \lambda I)\det(S) = \det(A - \lambda I)$$

in quanto S^{-1} e S sono costanti moltiplicative, e quindi gli autovalori λ di A e B coincidono. Inoltre:

$$B - \lambda I = 0 \Rightarrow S^{-1}AS - \lambda I = 0 \Rightarrow S^{-1}(A - \lambda I)S = 0 \Rightarrow A - \lambda I = 0$$

cioè si ha lo stesso spazio delle soluzioni.

Presa (λ, v) *autocoppia* (coppia autovalore-autovettore) per A abbiamo allora:

$$B(S^{-1}v) = S^{-1}AS(S^{-1}v) = S^{-1}Av = S^{-1}\lambda v = \lambda(S^{-1}v)$$

e quindi $S^{-1}v$ è autovettore per B associato a λ. □

5.1.4 Matrici diagonalizzabili

Definiamo le matrici diagonalizzabili come matrici simili ad una matrice diagonale, cioè:

Definizione 5.6: Matrice diagonalizzabile

Data $A \in \mathbb{C}^{n\times n}$, A si dice diagonalizzabile se $\exists S$ invertibile tale che:

$$S^{-1}AS = A_D$$

con $A_D = \text{diag}(\lambda_1, ..., \lambda_n)$ diagonale.

Osserviamo che se A è diagonalizzabile, la matrice diagonale (sopra, A_D) contiene sulla diagonale gli autovalori di A.

Vediamo poi che:

$$S^{-1}AS = A_D \implies AS = SA_D$$

risultato piuttosto triste, e posto $S = \begin{pmatrix} s_1 & ... & s_n \end{pmatrix}$:

$$AS = \begin{pmatrix} As_1 & ... & As_n \end{pmatrix} = \begin{pmatrix} \lambda_1 s_1 & ... & \lambda_n s_n \end{pmatrix}$$

da dove notiamo che da $As_i = \lambda_i s_i$, la matrice S ha per colonne gli autovettori di A (comodo per il calcolo della matrice S quando A è definita in base canonica).

Non tutte le matrici quadrate sono diagonalizzabili. La diagonalizzabilità dipende infatti dalla possibilità di scegliere n autovettori di A linearmente indpendenti, cioè di trovare una base di $\mathbb{C}^n$ di autovalori di A.

Da questo deriva:

Teorema 5.1: Diagonalizzabilità di matrici

$A \in \mathbb{C}^{n\times n}$ è diagonalizzabile se e solo se:

$$\alpha(\lambda) = \gamma(\lambda), \quad \forall\lambda$$

dove i λ sono gli autovalori di A.

Questo semplicemente significa che una matrice è diagonalizzabile se le molteplicità algebriche e geometriche di tutti gli autovalori coincidono.

Si ha quindi che, per A diagonalizzabile, con $x \in \mathbb{C}^n$ si ha che $\exists c_1, ..., c_n \in \mathbb{C}$ tali che:

$$x = \sum_{i=1}^{n} c_i v_j$$

sugli autovettori v_j.

L'utilità sta nel fatto che possiamo scegliere il caso particolare dove la base di autovettori è unitaria/ortogonale (unitaria in $\mathbb{C}$ e ortogonale in $\mathbb{R}$). Da questo deriva:

Teorema 5.2: Teorema spettrale

Se la matrice A è hermitiana, cioe $A = A^H$, allora:

1. A è sempre diagonalizzabile;
2. Gli autovalori di A sono reali;
3. Si può scegliere una base di autovettori unitaria/ortogonale, cioè tale che:

$$< v_i, v_j >= \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

o in modo equivalente si può trovare $V \in \mathbb{C}^{n \times n}$ tale che:

$$V^H A V = \text{diag}(\lambda_1, ..., \lambda_n)$$

e

$$V^H V = V V^H = I$$

Da questo teorema vengono le definizioni:

Definizione 5.7: Matrice definita positiva/negativa

Una matrice hermitiana viene detta definita positiva (definita negativa) quado gli autovalori λ_i sono tutti > 0 (< 0).

5.1.5 Proprietà di autovalori e autovettori

Vediamo come si comportano autovalori e autovettori sottoposti a diverse operazioni fra matrici.

1. **Somma per identità:** data (λ, v) autocoppia per $A \in \mathbb{C}^{n \times n}$, $\delta \in \mathbb{C}$, allora $A + \delta I$ ha come autocoppia $\lambda + \delta, v$. Questo semplicemente da:

$$(A + \delta I)v = Av + \delta v = \lambda v + \delta v = (\lambda + \delta)v$$

2. **Somma di potenze:** supponiamo $q(x) = \sum_{i=0}^{d} q_i x^i$. Si può allora definire:

$$q(A) = \sum_{i=0}^{d} q_i A^i$$

Allora con (λ, v) autocoppia di A, si avrà che $(q(\lambda), v)$ è autocoppia per $q(A)$.

Notiamo che la proprietà (1) è il caso $q(x) = x + \delta$.

3. **Inversa:** data (λ, v) autocoppia per A, si avrà che (λ^{-1}, v) è autocoppia per A^{-1}. Questo da:

$$Av = \lambda v \implies A^{-1}Av = \lambda A^{-1} v \implies v = \lambda A^{-1} v$$

da cui:

$$\frac{1}{\lambda} v = A^{-1} v$$

□

4. **Autovalori complessi coniugati:** se $A \in \mathbb{R}^{n\times n}$, allora $\lambda \in \mathbb{C}$ è autovalore di A se e solo se anche $\overline{\lambda}$ coniugato è autovalore di A. Equivalentemente, gli autovalori complessi arrivano sempre in coppie coniugate. Questo da:

$$p(\lambda) = 0 = (-1)^n\lambda^n + (-1)^{n-1}\sigma_1\lambda^{n-1} + ... - \sigma_{n-1}\lambda + \sigma_n$$

con $\lambda \in \mathbb{C} \setminus \mathbb{R}$, dovra essere che:

$$p(\lambda) = 0 = p(\lambda_n)$$

□

Definiamo infine il **raggio spettrale**, cioè il *massimo modulo* degli autovalori.

Definizione 5.8: Raggio spettrale

La quantità $\rho(A) = \max_{j=\{1,...,n\}} |\lambda_j|$ è detta raggio spettrale di A.

Questa definizione è utile sempre al calcolo di esponenti di matrici. Ad esempio, vale il teorema:

Teorema 5.3: Limiti e raggio spettrale

Per un matrice A con raggio spettrale $\rho(A)$, vale:

$$\lim_{x\to+\infty} A^x = 0 \Leftrightarrow \rho(A) < 1$$

6 Lezione del 14-03-25

6.1 Norme

Concludiamo il ripasso di algebra lineare parlando del concetto di **norma**.

6.1.1 Norme vettoriali

In molti casi è utile definire (quantificare) la "grandezza" di un vettore, prendendo il vettore 0 come nullo e ogni vettore come più grande di esso. Formalmente, quello che si fa è definire funzioni con proprietà specifiche dette **norme**.

Definizione 6.1: Norma

Una funzione $|\cdot| : \mathbb{C}^n \to \mathbb{R}^+$ è detta norma se verifica le 3 proprietà:

1. $|x| = 0 \Leftrightarrow x = 0$
2. $|\alpha x| = |\alpha| \cdot |x|, \quad \forall \alpha \in \mathbb{C}, \forall x \in \mathbb{C}^n$
3. $|x + y| \leq |x| + |y|, \quad \forall x, y \in \mathbb{C}^n$ (diseguaglianza del triangolo)

Possiamo dare alcuni esempi di norme:

- $|x|_1 = \sum_{i=1}^n |x_i|$ (norma di Manhattan)

- $|x|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$ (norma Euclidea)
- $|x|_p = (\sum_{i=1}^p |x_i|^p)^{\frac{1}{p}}$ (p-norma)
- $|x|_\infty = \max_{i=1,...,n} |x_i|$ (norma a infinito)

Notiamo che l'insieme dei vettori di norma 1 cambia geometria al variare della norma che scegliamo. Ad esempio, in $\mathbb{R}^2$, l'insieme dei vettori di norma 1 dà un quadrato ruotato di 45 gradi, dei vettori di norma 2 una circonferenza, e via via al crescere di p si approssima un quadrato unitario centrato sull'origine (che la norma a infinito effettivamente raggiunge).

Le norme sono equivalenti in quanto esiste il teorema:

Teorema 6.1: Norme equivalenti

Date due norme $|\cdot|_a$ e $|\cdot|_b$, si ha che $\exists \alpha, \beta \in \mathbb{R}^+$ tali che:

$$\alpha|x|_a \leq |x|_b \leq \beta|x|_a, \quad \forall x \in \mathbb{C}^n$$

Questa proprietà è sempre valida su spazi vettoriali finiti (non vale necessariamente lo stesso su spazi infiniti). Se vale il teorema 6.1, quindi, allora vale anche che:

$$\beta^{-1}|x|_b \leq |x|_a \leq \alpha^{-1}|x|_b, \quad \forall x \in \mathbb{C}^n$$

Per le norme più comuni ($|\cdot|_1$, $|\cdot|_2$, $|\cdot|_\infty$) valgono le diseguaglianze:

- $|x|_2 \leq |x|_1 \leq \sqrt{n}|x|_2$

  Dimostriamo entrambe le diseguaglianze.

  - $|x|_2 \leq |x|_1$

    Questo si ha da:

    $$\sqrt{\sum_{i=1}^n |x_i|^2} \leq \sum_{i=1}^n |x_i|$$

    prendendo i quadrati:

    $$\sum_{i=1}^n |x_i|^2 \leq \left(\sum_{i=1}^n |x_i|\right)^2 = \sum_{i=1}^n |x_i|^2 + 2\sum_{i<j}^n |x_i||x_j|$$

    che è chiaramente vero con $2\sum_{i<j}^n |x_i||x_j| > 0$.
  - $|x|_1 \leq \sqrt{n}|x|_2$

    Questo si nota prendendo:

    $$\sum_{i=1}^n |x_i| = (||x||, 1) = |x_1| \cdot 1 + ... + |x_n| \cdot 1$$

    dove $||x||$ rappresenta il modulo componente a componente di x. Varrà allora dalla diseguaglianza di Cauchy-Schwarz:

    $$(||x||, 1) \leq |\,||x||\,| \cdot |1| = \sqrt{\sum_{i=1}^n |x_i|^2} \cdot \sqrt{\sum_{i=1}^n |1|^2} = \sqrt{n}\sqrt{\sum_{i=1}^n |x_i|^2}$$

    che è la tesi. □

- $|x|_\infty \leq |x|_1 \leq n|x|_\infty$

  Questo risulta chiaro da:

$$\max_{i=1,\dots,n} |x_i| \leq \sum_{i=1}^{n} |x_i| \leq n \max_{i=1,\dots,n} |x_i|$$

□

- $|x|_\infty \leq |x|_2 \leq \sqrt{n}|x|_\infty$

  Anche questo risulta chiaro da:

$$\max_{i=1,\dots,n} |x_i| \leq \sqrt{\sum_{i=1}^{n} |x_i|^2} \leq \sqrt{n} \max_{i=1,\dots,n} |x_i|$$

  prendendo i quadrati:

$$\left(\max_{i=1,\dots,n} |x_i|\right)^2 \leq \sum_{i=1}^{n} |x_i|^2 \leq n \left(\max_{i=1,\dots,n} |x_i|\right)^2$$

□

6.1.2 Norme matriciali

Si possono definire norme anche per matrici.

Definizione 6.2: Norma matriciale

Una funzione $|\cdot| : \mathbb{C}^{m\times n} \to \mathbb{R}^+$ è detta norma matriciale se verifica le 3 proprietà:

1. $|A| = 0 \Leftrightarrow A = 0$
2. $|\alpha A| = |\alpha| \cdot |A|, \quad \forall \alpha \in \mathbb{C}, \forall A \in \mathbb{C}^{m\times n}$
3. $|A + B| \leq |A| + |B|, \quad \forall A, B \in \mathbb{C}^{m\times n}$
4. Se $m = n$ chiediamo inoltre, visto che è ben definito il prodotto matriciale:
   $|A \cdot B| \leq |A| \cdot |B|, \quad \forall A, B \in \mathbb{C}^{m\times n}$
   da qui in poi assumiamo solo questo caso.

Esiste un modo per trovare norme matriciali a partire da norme vettoriali.

Definizione 6.3: Norma indotta

Una norma matriciale $|\cdot|_m$ si dice indotta se esiste una norma vettoriale $|\cdot|_v$ tale che:

$$|A|_m = \sup_{|x|_v \neq 0} \frac{|Ax|_v}{|x|_v} = \sup_{|x|_v = 1} |Ax|_v$$

Intuitivamente, diciamo che una matrice è tanto più "grande" tanto più *allarga* i vettori a distanza unitaria.

Le norme matriciali indotte dalle norme vettoriali più comuni ($|\cdot|_1, |\cdot|_2, |\cdot|_\infty$) sono le seguenti:

- $|A|_1 = \max_{j=1,\dots,n} \sum_{i=1}^{n} |a_{ij}|$, cioè il massimo della somme delle colonne di A;
- $|A|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^{n} |a_{ij}|$, cioè il massimo delle somme delle righe di A;
- $|A|_2 = \sqrt{\rho(A^H A)}$, cioè la radice dell'autovalore di modulo massimo di $A^H A$. Notiamo che questo metodo è sensibilmente più difficile di calcolare la somma delle entrate delle matrice in riga o in colonna.

  Prendiamo $A^H A$ anzichè A in quanto:

  $$|A|_2 = \sup_{|x|_2=1} |Ax|_2$$

  e:

  $$|Ax|_2 = < Ax, Ax > = < x, A^H Ax >$$

  il resto della dimostrazione non viene dato in quanto si basa su un risultato che esula dal corso (*quoziente di Rayleigh*), ma resta il fatto che ci serve una forma del tipo $A^H A$ anziché la sola matrice A.

Un esempio di norma matriciale non indotta è la norma di Frobenius:

$$|A|_F = \sqrt{\sum_{ij}^{n} |\alpha_{ij}|^2}$$

che possiamo assumere come una grande norma $|\cdot|_2$ su tutta la matrice.

Un modo per verificare che questa norma non è effettivamente indotta è osservare che:

$$|I|_F = \sqrt{n}$$

mentre per ogni norma indotta vale che:

$$|I|_i = \sup_{|x|\neq 0} \frac{|Ix|}{|x|} = \sup_{|x|\neq 0} \frac{|x|}{|x|} = 1 \neq \sqrt{n} \quad \forall n > 1$$

□

Notiamo che questa verifica vale in una sola direzione: se la norma dell'identità valesse effettivamente 1, potremmo dire che questa proprietà non è violata, ma non che di conseguenza la norma è indotta, cioè:

$$|I|_m \neq 1 \implies |\cdot|_m \text{ non è indotta}$$

$$|I|_m = 1 \not\implies |\cdot|_m \text{ è indotta (può non esserlo)}$$

Diamo quindi la definizione di compatibilità:

Definizione 6.4: Compatibilità

Una norma matriciale $|\cdot|_m$ si dice compatibile con una norma vettoriale $|\cdot|_v$ se vale:

$$|Ax|_v \leq |A|_m \cdot |x|_v, \quad \forall A \in \mathbb{C}^{n\times n}, \quad \forall x \in \mathbb{C}^n$$

Osserviamo che le norme matriciali indotto sono ovviamente compatibili con le norme vettoriali che le inducono. Infatti posto $x \neq 0$:

$$\frac{|Ax|_v}{|x|_v} \leq |A|_m \implies |Ax|_v \leq |A|_m \cdot |x|_v$$

□

6.1.3 Norme e autovalori

Le diseguaglianze precedenti e le proprietà delle norme ci dicono che gli autovalori non possono essere ovunque sul piano complesso, ma in modulo devono essere più piccoli di $|A|_m$ se $|\cdot|_m$ è compatibile (a maggior ragione indotta) con una norma vettoriale. Infatti, con (x, λ) autocoppia per A, cioè:

$$Ax = \lambda x, \quad \lambda \in \mathbb{C}, \quad x \neq 0$$

si ha:

$$|\lambda x| = |\lambda| \cdot |x| = |Ax|_v \leq |A|_m \cdot |x|_v \implies |\lambda| \leq |A|_m$$

□

Questo significa che se si prende il "disco" centrato sull'origine con raggio uguale alla norma della matrice A, allora tutti gli autovalori di A devono trovarsi all'interno di tale disco, cioè:

$$\lambda \text{ autovalore} \Leftrightarrow \lambda \in \{z \in \mathbb{C} : |z| \leq |A|_m\}$$

In realtà l'assunzione $|\cdot|_m$ compatibile può essere rimossa, ovvero vale il seguente teorema:

Teorema 6.2: di Hirsch

Data $A \in \mathbb{C}^{n \times n}$ allora $\forall |\cdot|$ norma matriciale vale:

$$\rho(A) = \max\{|\lambda| \text{ autovalore}\} \leq |A|$$

Questo si dimostra prendendo (λ, x) autocoppia per A e considerando la matrice:

$$B = \begin{pmatrix} x & | & 0 \end{pmatrix}$$

per cui vale:

$$A \cdot B = \begin{pmatrix} Ax & A0 & \dots & A0 \end{pmatrix} = \begin{pmatrix} \lambda x & 0 & \dots & 0 \end{pmatrix} = \lambda \begin{pmatrix} x & 0 & \dots & 0 \end{pmatrix} = \lambda B$$

allora vale:

$$|\lambda||B| = |\lambda B| = |A \cdot B| \leq |A| \cdot |B|$$

dividendo per B ($B \neq 0$ da $x \neq 0$ autovettore):

$$\frac{|\lambda||B|}{|B|} \leq \frac{|A| \cdot |B|}{|B|} \implies |\lambda| \leq |A|$$

e quindi $\rho(A) \leq |A|$. □

Questa misura potrebbe non essere molto utile in quanto la norma $|A|$ potrebbe essere aribtrariamente più grande del raggio spettrale $\rho(A)$

6.1.4 Teoremi di Gershgorin

Esistono alcuni teoremi più forti riguardo alle norme degli autovalori, che prendono il nome di **teoremi di Gershgorin**.

Iniziamo dalla definizione di *cerchio* di Gershgorin:

Definizione 6.5: Cerchio di Gershgorin

Data $A \in \mathbb{C}^{n \times n}$ definiamo gli insiemi:

$$\mathcal{F}_i(A) = \{z \in \mathbb{C} : |z - a_{ii}| \leq \rho_i\}, \quad \rho_i = \sum_{j \neq i}^{n} |a_{ij}| \geq 0$$

cerchi di Gershgorin di A associati alla riga i.

Questi sono effettivamente i cerchi di raggio uguale alla somma degli elementi fuori dalla diagonale, con centro sull'asse dei reali il valore presente sulla diagonale, per ogni riga.

Possiamo quindi enunciare i teoremi di Gershgorin:

Teorema 6.3: Primo teorema di Gershgorin

Se $\lambda \in \mathbb{C}$ è autovalore vale che:

$$\lambda \in \bigcup_{i=1}^{n} \mathcal{F}_i(A)$$

cioè gli autovalori stanno nell'unione dei cerchi di Gershgorin. Per dimostrare questo risultato si prende l'autocoppia di A (λ, x) e si indica con k l'indice della componente di massimo modulo in x. Quindi $x \neq 0$ perchè x autovettore, e k è scelto tale che $|x_k| > |x_j|$, $\forall j = 1, ..., n$. Sappiamo quindi che vale $Ax = \lambda x$ dalla definizione di autovalore, e consideriamo la riga k nell'uguaglianza:

$$a_{k1}x_1 + a_{k2}x_2 + ... + a_{kn}x_n = \lambda x_k \Leftrightarrow \sum_{j=1}^{n} a_{kj}x_j = \lambda x_k \Leftrightarrow \sum_{j \neq k}^{n} a_{kj}x_j = \lambda x_k - a_{kk}x_k = (\lambda - a_{kk})x_k$$

$$\implies |\lambda - a_{kk}| \cdot |x_k| = \left| \sum_{j \neq k}^{n} a_{kj}x_j \right| \leq \sum_{j \neq k}^{n} |a_{kj}| \cdot |x_j|$$

Dato $|x_k| > 0$, si dividono da entrambi i lati per $|x_k|$:

$$\implies |\lambda - a_{kk}| \leq \sum_{j \neq k}^{n} |a_{kj}| \frac{x_j}{x_k} \leq \sum_{j \neq k}^{n} |a_{kj}| = \rho_k$$

Quindi:

$$|\lambda - a_{kk}| \leq \rho_k \implies \lambda \in \mathcal{F}_k(A)$$

e, dato che l'argomento utilizzato vale per un qualsiasi autovalore, abbiamo la tesi del teorema:

$$\lambda \in \bigcup_{i=1}^{n} \mathcal{F}_i(A)$$

□

Osserviamo che in realtà avremmo dimostrato che $\lambda \in \mathcal{F}_k(A)$ con k indice dell'elemento di massimo modulo nell'autovettore. Tuttavia, nella situazione usuale in cui non conosciamo né λ ne x, sapere k non è plausibile.

Definizione 6.6: Matrice a predominanza diagonale forte

Una matrice A si dice a predominanza diagonale forte se vale:

$$|a_{ii}| > \sum_{j \neq i}^{n} |a_{ij}| = \rho_i, \quad \forall i = 1, ..., n$$

Un **corollario** del teorema 6.3 è che se A è a predominanza diagonale forte, allora A è non singolare ($\det(A) \neq 0$).

Questo si dimostra dal fatto che tutti i cerchi di Gershgorin di A non comprendono l'origine, e quindi:

$$0 \notin \bigcup_{i=1}^{n} \mathcal{F}_i(A)$$

e 0 non può essere autovalore di A.

Teorema 6.4: Secondo teorema di Gershgorin

Se M_1 è unione di s cerchi di Gershgorin ed M_2 è unione di $n - s$ cerchi di Gershgorin e vale $M_1 \cap M_2 = \emptyset$, allora M_1 contiene esattamente s autovalori e M_2 ne contiene esattamente altri $n - s$.

Osserviamo che il primo teorema di Gershgorin (6.3) non ci dice che necessariamente c'è un autovalore per ogni cerchio $\mathcal{F}_i(A)$, cioè può succedere che $\mathcal{F}_i(A)$ non ne contiene nemmeno uno. Il secondo teorema di Gershgorin (6.4) invece ci dice che se abbiamo un cerchio isolato, allora questo contiene esattamente un autovalore.

Un **corollario** del secondo teorema di Gershgorin è che se A ha n cerchi disgiunti, allora A ha n autovalori distinti ed è diagonalizzabile.

6.1.5 Matrici reali e Gershgorin

Ci sono delle note da fare sulle **matrici reali**. Osserviamo infatti che se $A \in \mathbb{R}^{n \times n}$, se λ è autovalore di A allora lo è anche $\overline{\lambda}$. Quindi, se un cerchio di Gershgorin contiene un autovalore complesso, dovrà necessariamente contenere anche il suo coniugio. Ora, visto che un cerchio isolato contiene necessariamente un solo autovalore, sarà che un cerchio isolato in una matrice a entrate reali contiene necessariamente un autovalore reale, e quindi la matrice ha necessariamente un autovalore reale.

6.1.6 Matrici trasposte e Gershgorin

Dato che A e A^T hanno gli stessi autovalori (dallo stesso polinomio caratteristico), si può prendere come regione dove stanno gli autovalori l'intersezione:

$$\left[\bigcup_{i=1}^{n} \mathcal{F}_i(A)\right] \cap \left[\bigcup_{i=1}^{n} \mathcal{F}_i(A^T)\right]$$

dove notiamo bene che:

$$\left[\bigcup_{i=1}^{n} \mathcal{F}_i(A)\right] \cap \left[\bigcup_{i=1}^{n} \mathcal{F}_i(A^T)\right] \neq \bigcup_{i=1}^{n} \left(\mathcal{F}_i(A) \cap \mathcal{F}_i(A^T)\right)$$

infatti solitamente il termine a sinistra è ben più grande. Osserviamo poi che:

$$\mathcal{F}_i(A^T) = \{z \in \mathbb{C} : |z - a_{ii}| \leq \rho_i\}, \quad \rho_i = \sum_{j \neq i}^{n} |a_{ji}| \geq 0$$

cioè semplicemente si prendono le colonne anziché le righe.

6.1.7 Note sul calcolo dei dischi di Gershgorin

Vediamo come si può usare MATLAB per il calcolo e la visualizzazione dei dischi di Gershgorin di una matrice.

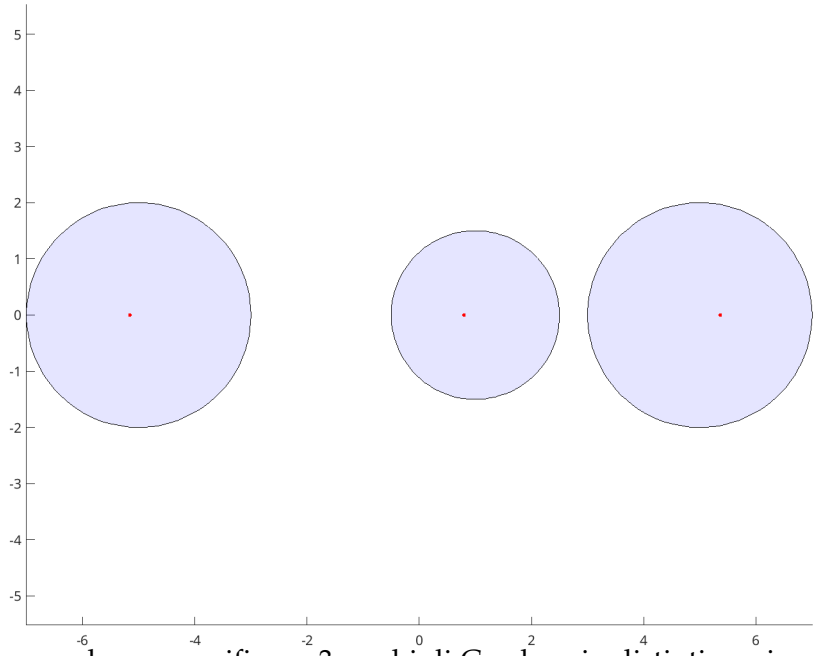
Potremmo pensare di scrivere una funzione per il calcolo e la rappresentazione grafica dei cerchi di Gershgorin di una matrice arbitraria (sia reale che complessa), ad esempio come:

```
function draw_gersh(A, draw_eigen)
    % draws a circle
    function circle(center, radius, col)
        t = linspace(0, 2*pi);
        patch(radius * cos(t) + real(center), ...
              radius * sin(t) + imag(center), col);
    end

    if nargin < 2
        draw_eigen = true;
    end

    n = size(A, 1);

    % graph setup
    close all;
    hold on;
    axis('equal');

    for k = 1:n
        center = A(k, k);
        radius = sum(abs(A(k, [1:k-1, k+1:end])));
        circle(center, radius, 'blue');
    end

    alpha(.1)

    if draw_eigen
        eigenvals = eig(A);
        plot(real(eigenvals), imag(eigenvals), 'red.');
    end

    hold off;
end
```

A questo punto potremo tracciare i cerchi di Gershgorin come:

```
>> A = [5, 1, 1; 1, -5, 1; 1, 0.5, 1];
>> draw_gersh(A)
```

da cui si ottiene il grafico:



Notiamo come, nel caso specifico, a 3 cerchi di Gershgorin distinti corrispondono 3 autovalori distinti, e quindi la matrice A è diagonalizzabile.

7 Lezione del 17-03-25

Avevamo visto i primi 2 teoremi di Gershgorin, cioè:

- **Primo teorema di Gershgorin:** ogni autovalore λ è contenuto in:

$$\lambda \in \bigcup_{i=1}^{n} \mathcal{F}_i(A)$$

definiti i *cerchi di Gershgorin*:

$$\mathcal{F}_i(A) = \left\{ z \in \mathbb{C}, \quad |z - a_{ii}| < \sum_{j \neq i} a_{ij} \right\}$$

- **Secondo teorema di Gershgorin:** un M_1 è unione di s cerchi e M_2 unione dei restanti $n - s$, allora: $M_1 \cap M_2 = \emptyset \implies M_1$ contiene s autovalori e M_2 i restanti $n - s$.

Vediamo quindi il terzo teorema di Gershgorin. Per fare ciò, introduciamo il concetto di **matrice irriducibile**, e ancor prima di **grafo** associato alla matrice.

7.1 Matrici irriducibili

Diamo la definizione:

Definizione 7.1: Grafo associato ad una matrice

Data una matrice $A \in \mathbb{C}^{n\times n}$ si dice grafo associato ad A, indicato con $G(A) = (V, E)$, il grafo che ha come vertici l'insieme $V = \{1, ..., n\}$ e come archi l'insieme E così definito:

$$(i, j) \in E \Leftrightarrow a_{ij} \neq 0$$

Solitamente si ignora la diagonale (notiamo che in caso di entrate $\neq 0$ risultano semplicemente in anelli aperti e chiusi sulllo stesso nodo).

A noi interesseranno grafi **fortemente connessi**:

Definizione 7.2: Grafo fortemente connesso

Un grafo si dice fortemente connesso se per ogni scelta di vertici $i, j \in 1, ..., n$ esiste un cammino *orientato* sul grafo che parte da i e arriva a j.

A questo punto possiamo definire le matrici **irriducibili**:

Definizione 7.3: Matrice irriducibile

Una matrice $A \in \mathbb{C}^{n\times n}$ si dice irriducibile se il grafo $G(A)$ è fortemente connesso.

Possiamo quindi enunciare il terzo teorema di Gershgorin:

Teorema 7.1: Terzo teorema di Gershgorin

Data una matrice $A \in \mathbb{C}^{n\times n}$ irriducibile si ha che, se λ è un autovalore tale che:

$$\lambda \in \partial\mathcal{F}_i(A)$$

dove con ∂ si indica il bordo, cioè esiste un indice, $\exists k \in \{1, ..., n\}$ tale che: $|\lambda - a_{kk}| = \sum_{j\neq k} a_{kj}$, allora dovrà essere che:

$$\lambda \in \bigcap_{i=1}^{n} \partial\left(\mathcal{F}_i(A)\right)$$

cioè per *tutti* gli indici, $\forall k \in \{1, ..., n\}$ vale che: $|\lambda - a_{kk}| = \sum_{j\neq k} a_{kj}$.

Questo significa che, nel caso di matrici irriducibili, un autovalore che sta sul bordo di un disco di Gershgorin dovrà necessariamente stare sul bordo di *tutti* i dischi di Gershgorin.

Avevamo visto la definizione di matrici a predominanza diagonale forte. Esistono anche matrici a predominanza diagonale *debole* (o semplicemente *diagonali deboli*):

Definizione 7.4: Matrice a predominanza diagonale debole

Una matrice A si dice a predominanza diagonale debole se vale:

$$|a_{ii}| \geq \sum_{j \neq i} |aij|$$

ed $\exists k \in \{1, ..., n\}$ tale per cui:

$$|a_{kk}| > \sum_{j \neq k} |akj|$$

Un **corollario** è che se A è a predominanza diagonale debole ed è irriducibile, allora A non è singolare.

Questo si dimostra direttamente dal fatto che il valore 0 appartiene ai cerchi per cui vale $|a_{ii}| \geq \sum_{j \neq i} |a_{ij}$, ma esiste almeno un cerchio per cui vale $|a_{kk}| > \sum_{j \neq k} |akj|$, cioè non contiene lo 0, e quindi per il terzo teorema di Gershgorin 0 non può essere autovalore (matrice non singolare). □

7.2 Matrici a blocchi

Talvolta è conveniente definire una matrice in termini di *sottomatrici*, o **blocchi**, al posto delle entrate scalari che la compongono, cioè si può scrivere:

$$A \in \mathbb{C}^{m \times n}, \quad A = \begin{pmatrix} A_{11} & ... & A_{1r} \\ ... & & ... \\ A_{s1} & ... & A_{sr} \end{pmatrix}$$

Si può sempre scrivere una matrice a blocchi, purché le sottomatrici abbiano dimensioni compatibili (sia le somme delle dimensioni devono corrispondere alle dimensioni effettive della matrice, sia le dimensioni delle matrici adiacenti devono corrispondere), cioè si deve avere:

$$A_{ij} \in \mathbb{C}^{m_i \times n_j}, \quad \sum_{i=1}^{s} m_i = m, \quad \sum_{j=1}^{r} n_j = n$$

Notiamo che per noi sarà il importante il caso di matrici quadrate poste come sottomatrici quadrate, cioè $m = n$, $s = r$ e $m_i = n_i \; \forall i = 1, ..., r \cdot s$.

Vediamo quindi la definizione di matrici **triangolari a blocchi**:

Definizione 7.5: Matrice triangolare a blocchi

Nelle condizioni di cui sopra (in particolare, matrici quadrate) una matrice si dice triangolare a blocchi se sono $\neq 0$ tutti i blocchi lungo la diagonale, e sopra o sotto la diagonale (dove rispettivamente si parla di matrice triangolare superiore o inferiore), cioè:

$$A \in \mathbb{C}^{m \times n}, \quad A = \begin{pmatrix} A_{11} & ... & A_{1r} \\ 0 & ... & ... \\ 0 & 0 & A_{sr} \end{pmatrix} \text{ (superiore) o } A = \begin{pmatrix} A_{11} & 0 & 0 \\ ... & ... & 0 \\ A_{s1} & ... & A_{sr} \end{pmatrix} \text{ (inferiore)}$$

Vediamo anche le matrici **diagonali a blocchi**:

Definizione 7.6: Matrice diagonale a blocchi

Nelle condizoni della definizione 7.5, una matrice si dice diagonale a blocchi se sono $\neq 0$ i soli blocchi lungo la diagonale, cioè:

$$A \in \mathbb{C}^{m \times n}, \quad A = \begin{pmatrix} A_{11} & 0 & 0 \\ 0 & ... & 0 \\ 0 & 0 & A_{sr} \end{pmatrix} \text{ (diagonale)}$$

7.2.1 Proprietà delle matrici a blocchi

Vediamo alcune proprietà delle matrici a blocchi:

1. Se A è triangolare a blocchi, allora l'insieme degli autovalori di A corrisponde all'unione degli insiemi di autovalori associati ai blocchi sulla diagonale A_{jj};

2. Dalla scorsa proprietà valgono, a cascata, tutte le proprietà che avevamo visto su autovalori e determinanti. Ad esempio il determinante sarà:

   $$\det(A) = \prod_{j=1}^{s} \det(A_{jj})$$

   ricordando che:

   $$\det(A_{jj}) = \prod_{i=1}^{\frac{n}{s}} \lambda_i$$

   con i λ_i autovalori di A_{jj};

3. Se A è diagonale, a blocchi, vale:

   $$A^{-1} = \begin{pmatrix} A_{11}^{-1} & 0 & 0 \\ 0 & ... & 0 \\ 0 & 0 & A_{sr}^{-1} \end{pmatrix}$$

4. Se si hanno 2 matrici a blcchi con lo stesso partizionamento, allora calcolare $A + B$ e $A \cdot B$ si può fare trattando i sottoblocchi come entrate scalari, cioè date:

   $$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

   con A_{11} corrispondente in dimensioni a B_{11} e via dicendo, si ha che:

   $$A + B = \begin{pmatrix} A_{11} + B_{11} & A_{12} + B_{12} \\ A_{21} + B_{21} & A_{22} + B_{22} \end{pmatrix}$$

   riguardo alla somma, e considerazioni simili (che non riportiamo) riguardo al prodotto.

7.3 Matrici riducibili

Vediamo ora una definizione che è effettivamente *duale* a quella di matrice irriducibile: quella di **matrice riducibile**:

Definizione 7.7: Matrice riducibile

Una matrice $A \in \mathbb{C}^{n\times n}$, $n \geq 2$, si dice riducibile se $\exists \Pi$ matrice di permutazione tale che:

$$\Pi A \Pi^T = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

a blocchi, con $A_{11} \in \mathbb{C}^{k\times k}$, $A_{22} \in \mathbb{C}^{(n-k)\times(n-k)}$, $k \in \{1, ..., n\}$.

Osserviamo che ci possono essere più di una matrice di permutazione Π che portano A in una forma triangolare a blocchi, con diversi valori di k.

Osserviamo poi che la forma triangolare superiore non è imperativa, cioè se $\exists \Pi_1$ tale che:

$$\Pi_1 A \Pi_1^T = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

triangolare superiore, allora vale anche che $\exists \Pi_2$ tale che:

$$\Pi_2 A \Pi_2^T = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix}$$

triangolare inferiore. Vediamo anzi che in verità queste due forme sono equivalenti.

7.3.1 Sistemi lineari con matrici riducibili

Supponiamo di voler risolvere $Ax = b$ con $A \in \mathbb{C}^n \times n$, con A riducibile, e conoscendo T. In questo caso conviene procedere sfruttando:

$$Ax = b \;\Leftrightarrow\; \Pi A x = \Pi b \;\Leftrightarrow\; \Pi A \Pi^T \Pi x = \Pi b \;\Leftrightarrow\; By = c$$

dove si è detto $B = \Pi A \Pi^T$, $c = \Pi b$, e $y = \Pi x$. $By = c$ sarà in forma:

$$\begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

con $c_1, y_1 \in \mathbb{C}^k$ e $c_2, y_2 \in \mathbb{C}^{n-k}$.

Una volta risolto $By = C$, si ritrova x mediante la relazione:

$$\Pi x = y \;\rightarrow\; x = \Pi^T y$$

Vediamo come risolvere effettivamente questo sistema. Scrivendo il sistema 2×2 a blocchi per esteso si ottiene:

$$\begin{cases} A_{11} y_1 + A_{12} y_2 = C_1 \\ A_{22} y_2 = c_2 \end{cases}$$

L'ultima equazione rappresenta un sistema $(n-k) \times (n-k)$, che risolviamo per trovare y_2 (notiamo non c'è dipendenza da y_1, dalla definizione come triangolare superiore a blocchi della matrice B). A questo punto si ottiene un altro sistema lineare:

$$A_{11} y_1 = c_1 - A_{12} y_2$$

con y_2 stavolta noto.

Il *guadagno* di questo metodo è che risolvere un sistema $n \times n$ costa generalmente $O(n^3)$. Effettuare il *preprocessing* reso disponibile dalla riducibilità della matrice ci permette di rendere la complessità uguale a:

$$O(k^3 + (n-k)^3 + k(n-k)) = O(k^3 + (n-k)^3)$$

dove $k(n-k)$ alla prima equazione rappresenta il costo della moltiplicazione matrice-vettore $A_{11}y_2$ nella seconda equazione di risoluzione del sistema a blocchi.

Se si hanno molti 0 da sfruttare, e quindi si può scegliere $k = \frac{n}{2}$, si ottiene un costo:

$$O\left(\left(\frac{n}{2}\right)^3 + \left(n - \frac{n}{2}\right)^3\right) = O\left(2\left(\frac{n}{2}\right)^3\right) = O\left(\frac{n^3}{4}\right)$$

cioè si guadagna di un fattore di 4 in termini di tempo di esecuzione.

Osserviamo inoltre che se i blocchi A_{11}, A_{22} sono a loro volta matrici riducibili, il processo si può applicare **ricorsivamente** sulle matrici della diagonale, in modo da ridurre ulteriormente la complessità del sistema (a patto di dover effettuare più passi di preprocessing).

8 Lezione del 21-03-25

8.1 Valutazione della riducibilità

Riprendiamo il discorso delle matrici riducibili, soffermandoci sul come capire quando una matrice è riducibile, e come ricavare, in caso affermativo, la marice di permutazione Π corrispondente.

Vale il seguente risultato (abbastanza banale guardando a quanto detto riguardo alle matrici irriducibili):

Teorema 8.1: Caratterizzazione di matrice riducibile

Una matrice A è riducibile quando non è irriducibile, cioè quando il suo grafo associato $G(A)$ non è fortemente connesso.

La dimostrazione avviene osservando innanzitutto questo **lemma**: se esiste una certa permutazione π degli elementi in riga, si può ricavare una matrice di permutazione Π e definire la matrice:

$$A' = \Pi A \Pi^T$$

Avremo allora che il grafo associato ad A' sarà lo stesso associato a $\Pi A \Pi^T$, solamente cambiando i nomi dei vertici, cioè:

$$G(A) \text{ fortemente connesso } \Leftrightarrow G(A') \text{ fortemente connesso}$$

La **dimostrazione** vera e propria dovrà quindi affermare che se una matrice è rducibile, allora il suo grafo non è fortemente connesso, cioè:

$$A \text{ riducibile } \Leftrightarrow G(A) \text{ non fortemente connesso}$$

$\Rightarrow$) Abbiamo che se A è riducibile allora $\exists \Pi$:

$$B = \Pi A \Pi^T = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

con $A_{11} \in \mathbb{C}^{k \times k}$ e $A_{22} \in \mathbb{C}^{(n-k)\times(n-k)}$. Che $G(B)$ non è fortemente connesso è chiaro dal blocco di 0 in basso a sinistra: significherà che non ci sono archi che collegano il blocco A_{22} al blocco A_{11}, cioè non esistono archi che vanno da $\{k+1, ..., n\}$ a $\{1, ..., n\}$.

$\Leftarrow$) Se $G(A)$ non è fortemente connesso allora $\exists (j, h)$ per cui da j non si raggiunge h. Dividamo allra $\{1, ..., n\}$ in due sottoinsiemi:

$$\begin{cases} \mathcal{P} = \{\text{vertici raggiungibili da } j\} \\ \mathcal{Q} = \{\text{vertici non raggiungibili da } j\} \end{cases}$$

con $\mathcal{P} \cup \mathcal{Q} = \{1, ..., n\}$ e $\mathcal{P} \cap \mathcal{Q} = \emptyset$. Non ci sarà quindi nessun arco che collega un elemento di $\mathcal{P}$ a un elemento di $\mathcal{Q}$. Se si considera una permutazione Π che manda in testa tutti gli elementi di $\mathcal{Q}$, si ritrova esattamente la forma della definizione 7.7, cioè quella di una matrice riducibile, notando $k = |\mathcal{Q}|$ e $n - k = \mathcal{P}$.

□

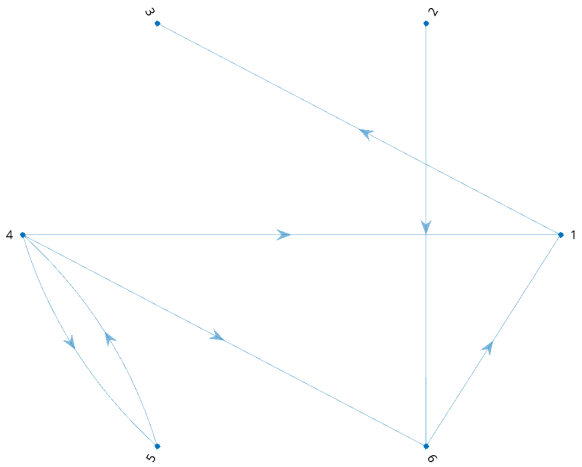
Prendiamo ad esempio la matrice:

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 3 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 2 & -1 & -2 \\ 0 & 0 & 0 & -1 & 2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Disegnamo il grafo $G(A)$ associato alla matrice A. In MATLAB, questo si potrà fare come:

```
>> A = [ 1 0 1 0 0 0; ... 1 0 0 0 0 1 ]
>> A(eye(size(A)) == 1) = 0 % rimuovi gli elementi sulla diagonale
>> G = digraph(A)
>> p = plot(G)
>> layout(p, "circle") % usa il layout circolare
```

che dalla A dell'esempio dà:



Vediamo quindi che le adiacenze fra nodi (pensando alla matrice, le dipendenze riga-colonna) sono:

Nodo	Raggiungibile	Non raggiungibile
1	1 3	2 4 5 6
2	1 2 3 6	4 5
3	3	1 2 4 5 6
4	1 3 4 5 6	2
5	1 3 4 5 6	2
6	1 3 6	2 4 5

Notiamo che il nodo 6 ci fornisce la partizione migliore: 3×3 e 3×3. Decidiamo quindi di prendere:

$$\mathcal{P} = \{1, 3, 6\}, \quad \mathcal{Q} = \{2, 4, 5\}$$

per cui la permutazione che manda $\mathcal{Q}$ in testa è:

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

Applicando $PiA\Pi^T$ si ottiene quindi:

$$\Pi A \Pi^T = \begin{pmatrix} A_{11} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} & A_{12} = \begin{pmatrix} 0 & 0 & 3 \\ 3 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix} \\ 0 & A_{21} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{pmatrix}$$

che è una forma più agile per la risoluzione della matrice originale A.

Nella sottodirectory `/code/matlab` si rende disponibile uno script `block_decomp.m` per la decomposizione in matrici a blocchi come nell'esempio. Un'esecuzione tipica dello script potrebbe avere l'aspetto:

```
p = block_decomp(A) % ottieni una permutazione
    Node       Reachable       Not reachable
    ----     -------------     -------------

     1       {[       1 3]}    {[  2 4 5 6]}
     2       {[  1 2 3 6]}     {[      4 5]}
     3       {[        3]}     {[1 2 4 5 6]}
     4       {[1 3 4 5 6]}     {[        2]}
     5       {[1 3 4 5 6]}     {[        2]}
     6       {[    1 3 6]}     {[    2 4 5]}

   Choose an index: 6 % chiesto dallo script
>> A(p, p) % permuta A
```

da cui si ottiene la stessa matrice a blocchi riportata sopra.

8.1.1 Riduzione iterata

Ricordiamo di poter iterare ricorsivamente il processo di riduzione, cioè di poter trovare per ogni blocco A_{ii} con $i \in \{1, 2\}$ un Π_i tale che:

$$\Pi_i A_{ii} \Pi_i^T = \begin{pmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ 0 & A_{22}^{(i)} \end{pmatrix}$$

così che valga:

$$\begin{pmatrix} \Pi_1 & 0 \\ 0 & \Pi_2 \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix} \begin{pmatrix} \Pi_1^T & 0 \\ 0 & \Pi_2^T \end{pmatrix} = \begin{pmatrix} \Pi_1 A_{11} \Pi_1^t & \Pi_1 A_{12} \Pi_2^T \\ 0 & \Pi_2 A_{22} \Pi_2^t \end{pmatrix}$$

$$= \begin{pmatrix} \begin{pmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & A_{22}^{(1)} \end{pmatrix} & * \\ 0 & \begin{pmatrix} A_{11}^{(2)} & A_{12}^{(2)} \\ 0 & A_{22}^{(2)} \end{pmatrix} \end{pmatrix}$$

e via dicendo, dove in $(*)$ comparrà qualcosa che al momento non ci interessa.

8.1.2 Problemi agli autovalori per riduzione

Notiamo che questo procedimento semplifica anche la risoluzione dei **problemi agli autovalori**: infatti iterando abbastanza, il problema si ridurrà a trovare i singoli autovalori di matrici sulla diagonale sempre più piccole, e quindi dal polinomio caratteristico di più facile risoluzione.

8.2 Sistemi lineari

Veniamo quindi alla trattazione dei sistemi lineari, che avevamo definito come forme $Ax = b$ con $A \in \mathbb{C}^{n \times n}, b \in \mathbb{C}^n$.

Studieremo 2 tipi di metodi risolutivi:

- **Metodi diretti:** esatti ma dispendiosi, se eseguiti in aritmetica esatta (cioè senza arrotondamenti) poterebbero in un numero n finito di passaggi alla soluzione esatta. Esempi di metodi diretti sono il **metodo di Cramer** (visto in 4.7.2) e l'**eliminazione di Gauss** (che vedremo fra poco);
- **Metodi iterativi:** meno accurati ma più efficienti computazionalmente, portano ad una successione $\{x_k\}_{k \in \mathbb{N}}$ di approssimazioni tali che $\lim_{k \to +\infty} x_k = x$ soluzione esatta. Notiamo però che, in generale, è impossibile trovare il valore esatto di x in un numero esatto di iterazioni. Per contropartita, risultano spesso molto più efficienti dei metodi diretti (esistono esempi di sistemi addirittura non risolvibili, nella pratica, con metodi diretti).

8.3 Metodi diretti per i sistemi lineari

Iniziamo a trattare i metodi diretti per la risoluzione dei sistemi lineari.

8.3.1 Sistemi triangolari

Diamo la definizione parallela a quella di matrice triangolare:

Definizione 8.1: Sistema triangolare

Si dice sistema triangolare un sistema della forma $Ux = c$ con U matrice triangolare.

Per un **sistema triangolare superiore**, si avrà la forma:

$$\begin{cases} u_{11}x_1 + u_{12}x_2 + ... + u_{1,n-1}x_{n-1} + u_{1n}x_n = c_1 \\ u_{22}x_2 + ... + u_{2,n-1}x_{n-1} + u_{2n}x_n = c_2 \\ \vdots \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n = c_{n-1} \\ u_{nn}x_n = c_n \end{cases}$$

Il **metodo risolutivo** sarà allora la *sostituzione all'indietro*, definita ricorsivamente come:

$$\begin{cases} x_n = \frac{c_n}{u_{nn}} \\ x_i = \frac{c_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}} \end{cases}$$

che equivale all'algoritmo, in MATLAB:

```
function x = bck_subst(U, b)
    n = height(U);
    x = zeros(n, 1);

    for i = n:-1:1
        p = b(i);
        for j = (i + 1):n
            p = p - U(i, j) * x(j);
        end
        x(i) = p / U(i, i);
    end
end
```

Riguardo alla complessità, si potra dire che al passo i si eseguono $n - i + n - i + 2 = 2(n - 1) + 2$ passaggi, cioè 2 per la divisione per u_{jj} e la somma fra c_i e il termine accumulato a destra, $n - i$ per i prodotti nella sommatoria e di nuovo $n - i$ per la sommatoria stessa. Sarà allora che:

$$\sum_{i=1}^{n} (2(n - i) + 2) \sim O(n^2)$$

cioè si ha complessità quadratica.

Osserviamo poi che per **sistemi triangolari inferiori** la situazione è uguale, cioè si risolve la prima equazione, si sostituisce il risultato nella seconda, e via dicendo:

$$\begin{cases} x_1 = \frac{c_1}{u_{11}} \\ x_i = \frac{c_i - \sum_{j=1}^{i-1} u_{ij}x_j}{u_{ii}} \end{cases}$$

che equivale all'algoritmo, in MATLAB:

```
function x = fwd_subst(L, b)
    n = height(L);
    x = zeros(n, 1);

    for i = 1:n
        p = b(i);
        for j = 1:(i -1)
            p = p - L(i, j) * x(j);
        end
        x(i) = p / L(i, i);
    end
end
```

Il metodo ottenuto, speculare al quello di sostituzione all'indietro, viene detto *sostituzione in avanti*, di costo identico ($O(n^2)$).

8.3.2 Metodo di eliminazione di Gauss

L'idea del metodo di eliminazione di Gauss è quella di partire da un sistema $Ax = b$, trasformarlo in un sistema equivalente $Ux = c$ (quindi triangolare superiore), ed applicare la sostituzione all'indietro.

Per arrivare alla forma $Ux = c$ si sostituiscono le equazioni del sistema con loro combinazioni lineari scelte in modo da annullare gli elementi inferiori alla diagonale.

L'idea è quella di eliminare, per ogni elemento i-esimo sulla diagonale a partire da quello in alto a destra, gli $n - i$ elementi che stanno al di sotto, cioè:

Algoritmo 1 Eliminazione di Gauss

Input: un sistema lineare qualsiasi $Ax = b$
Output: un sistema lineare triangolare superiore $Ux = c$
for $i = 1$ to n **do**
  for $j = i$ to n **do**
    Calcola il **moltiplicatore** $l_{ji} = \frac{a_{ji}^{(i-1)}}{a_{ii}^{(i-1)}}$
    Aggiungi alla riga j la riga i moltiplicata per l_{ij}
  end for
end for

8.3.3 Implementazione MATLAB del metodo di eliminazione Gauss

Una semplice implementazione in MATLAB del suddetto algoritmo può essere la seguente:

```
function [A, b] = gauss_decomp(A, b)
    n = height(A);

    for i = 1:n % i itera sulle diagonali
        den = A(i, i);

        for j = (i + 1):n % j itera sulle righe
            mul = A(j, i) / den; % moltiplicatore

            A(j, :) = A(j, :) - A(i, :) * mul;
            b(j) = b(j) - b(i) * mul;
        end
    end
```

```
14 end
```

da cui si potrà ottenere una riduzione di Gauss semplicemente come:

```
1  >> [U, c] = gauss_decomp(A, b)
```

Dal punto di vista della complessità, la riduzione in forma triangolare costa $O(\frac{2}{3}n^3)$, e chiaramente domina sul termine $O(n^2)$ della risoluzione di $Ux = c$ con la sostituzione all'indietro.

Facciamo una nota sulla fattibiltà della riduzione di Gauss, definendo:

Definizione 8.2: Moltiplicatori di Gauss

I termini $l_{ji} = \frac{a_{ji}^{(i-1)}}{a_{ii}^{(i-1)}}$ vengono detti moltiplicatori.

Chiaramente, per poter eseguire l'eliminazione di Gauss serve che $a_{jjj-1} \neq 0 \; \forall j = 1, ..., n-1$. Inoltre, i casi $a_{jj}^{j-1} \approx 0$ possono causare problemi di instabilità numerica. Vedremo in seguito metodi per ovviare a questo problema.

8.3.4 Fattorizzazione LU

Il primo passo dell'algoritmo di Gauss si può vedere come equivalente a moltiplicare l'equazione $Ax = b$ a sinistra per una particolare matrice H_1 triangolare inferiore:

$$H_1 = \begin{pmatrix} 1 & ... & ... & 0 \\ -l_{21} & 1 & ... & 0 \\ ... & & & \\ -l_{n1} & ... & ... & 1 \end{pmatrix}$$

con la diagonale a 1 e i moltiplicatori sulla prima colonna, così che $H_1 A$ risulti esattamente quello che volevamo per Gauss, cioè la combinazione lineare di ogni riga j con l'prima riga moltiplicata per il moltiplicatore l_{j1} (con $j > 2$).

Possiamo generalizzare questo processo a una serie di matrici H_i, per ogni elemento sulla diagonale, con la diagonale a 1 e i moltiplicatori corrispondenti a i sulla i-esima colonna:

$$H_i = \begin{pmatrix} 1 & ... & ... & 0 \\ 0 & 1 & ... & 0 \\ ... & -l_{ji} & ... & ... \\ 0 & -l_{ni} & ... & 1 \end{pmatrix}$$

Così, ancora una volta, AH_i risulterà quello che volevamo per Gauss, cioè la combinazione lineare di ogni riga j con l'i-esima riga moltiplicata per il moltiplicatore l_{ji}.

Varrà allora he il metodo di Gauss sarà equivalente a considerare:

$$H_{n-1} H_{n-2} ... H_1 Ax = H_{n-1} H_{n-2} ... H_1 b$$

e potremo quindi dire:

$$H_{n-1} H_{n-2} ... H_1 A = U, \quad L = H_1^{-1} ... H_{n-1}^1$$

da cui:

$$A = LU$$

Semplifichiamo i calcoli notando alcune proprietà delle matrici H_j:

1. Hanno l'inversa facile, in quanto basta invertire i segni:

$$H_i^{-1} = \begin{pmatrix} 1 & \dots & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & l_{ji} & \dots & \dots \\ 0 & l_{ni} & \dots & 1 \end{pmatrix}$$

2. Sono facili da moltiplicare, in quanto si può dire:

$$H_{i_1} \cdot H_{i_2} = \begin{pmatrix} 1 & \dots & \dots & 0 \\ -l_{2i_1} & 1 & \dots & 0 \\ \dots & -l_{ji_2} & \dots & \dots \\ -l_{ni_i} & -l_{ni_2} & \dots & 1 \end{pmatrix}$$

   e:

$$H_{i_1}^{-1} \cdot H_{i_2}^{-1} = \begin{pmatrix} 1 & \dots & \dots & 0 \\ l_{2i_1} & 1 & \dots & 0 \\ \dots & l_{ji_2} & \dots & \dots \\ l_{ni_i} & l_{ni_2} & \dots & 1 \end{pmatrix}$$

   cioè semplicemente si somma sotto la diagonale.

Questo significa che una volta svolta la prima parte dell'eliminazione di Gauss si è gia calcolata la fattorizzazione LU come la matrice dei moltiplicatori:

$$H_{i_1}^{-1} \cdot H_{i_2}^{-1} = \begin{pmatrix} 1 & \dots & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \dots & l_{3,2} & \dots & \\ l_{n1} & l_{n2} & \dots & 1 \end{pmatrix}$$

8.3.5 Implementazione MATLAB della riduzione LU

Modifichiamo il codice MATLAB della scorsa sezione per calcolare, oltre all'eliminazione di Gauss (cioè la matrice U), la matrice dei moltiplicatori L:

```
function [A, b, L] = gauss_decomp(A, b)
    n = height(A);

    L = eye(n); % prepara L

    for i = 1:n % i itera sulle diagonali
        den = A(i, i);

        for j = (i + 1):n % j itera sulle righe
            mul = A(j, i) / den; % moltiplicatore
            L(j, i) = mul;

            A(j, :) = A(j, :) - A(i, :) * mul;
            b(j) = b(j) - b(i) * mul;
        end
    end
end
```

A questo punto per calcolare la fattorizzazione LU di una matrice basterà eseguire:

```
>> [U, ~, L] = gauss_decomp(A, b)
>> L * U % idealmente dara' A
```

Si osserva quindi che se già si conoscono L ed U (magari di una matrice che dovremo usare spesso) risolvere $Ax = b$ costa $O(n^2)$, in quanto basta dire:

$$Ax = b \Rightarrow LUx = b \implies x = U^{-1}L^{-1}b$$

dove basta risolvere a cascata:

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

Questi sono due sistemi triangolari, uno **inferiore** (risolvibile per *sostituzione in avanti*), l'altro **superiore** (risolvibile per *sostituzione all'indietro*), da cui l'andamento complessivo $O(n^2)$.

In MATLAB, la soluzione si può quindi avere usando le funzioni definite finora:

```
>> [U, ~, L] = gauss_decomp(A, b)
>> y = fwd_subst(L, b)
>> x = bck_subst(U, y) % x e' la soluzione del sistema
```

Notiamo che questo vale se L ed U sono note (o come nell'esempio vengono calcolate), quindi tolto il prezzo dato dal doverle calcolare (come avevamo notato, conviene per matrici che magari dobbiamo usare spesso).

8.3.6 Metodo di Gauss per variabili matriciali

Se si vuole risolvere $AX = B$ con X e B matrici di vettori colonna di s colonne:

$$X = \begin{pmatrix} x_1 & \dots & x_s \end{pmatrix}, \quad B = \begin{pmatrix} b_1 & \dots & b_s \end{pmatrix}$$

cioè se si vogliono risolvere s sistemi lineari con la stessa matrice A:

$$Ax_1 = b_1, \quad \dots, \quad Ax_s = b_s$$

si può modificare l'algoritmo di Gauss, effettuando le mosse di Gauss sulla matrice aumentata $\left(A \,|\, B\right)$. Alla fine troveremo una matrice $\left(U \,|\, B^{(n-1)}\right)$, dove gli apici $(n-1)$ rappresentano che è la B che si ottiene all'$n-1$-esimo passaggio, che risolverà i sistemi triangolari superiori:

$$Ux_1 = b_1^{(n-1)}, \quad \dots, \quad Ux_s = b_s^{(n-1)}$$

Vediamo un esempio numerico. Partiamo dalle matrici A e B:

$$A = \begin{pmatrix} -2 & 1 & 1 \\ -1 & 3 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} -2 & 0 \\ -3 & 0 \\ 2 & 1 \end{pmatrix}$$

Avremo che le due colonne di B rappresentano due vettori "termini noti" diversi, e che quindi la soluzione X di $AX = B$ rappresenterà in verità due vettori soluzione:

$$X = \begin{pmatrix} x_1^{(1)} & x_1^{(2)} \\ x_2^{(1)} & x_2^{(2)} \\ x_3^{(1)} & x_3^{(2)} \end{pmatrix}$$

Costruiamo quindi la matrice AB ed applichiamo il metodo di riduzione di Gauss:

$$AB = \left(\begin{array}{ccc|cc} -2 & 1 & 1 & -2 & 0 \\ -1 & 3 & 1 & -3 & 0 \\ 1 & 1 & 2 & 2 & 1 \end{array}\right) \xrightarrow{\text{Gauss}} \left(\begin{array}{ccc|cc} -2 & 1 & 1 & -2 & 0 \\ 0 & \frac{5}{2} & \frac{1}{2} & -2 & 0 \\ 0 & 0 & \frac{11}{5} & \frac{11}{5} & 1 \end{array}\right) = UB^{(n-1)}$$

Avremo quindi due sistemi dati dalla forma $UX = B^{(n-1)}$:

$$x^{(1)}: \begin{cases} -2x_1^{(1)} + x_2^{(1)} + x_3^{(1)} = -2 \\ \frac{5}{2}x_2^{(1)} + \frac{1}{2}x_3^{(1)} = -2 \\ \frac{11}{5}x_3^{(1)} = \frac{11}{5} \end{cases} \qquad x^{(2)}: \begin{cases} -2x_1^{(2)} + x_2^{(2)} + x_3^{(2)} = 0 \\ \frac{5}{2}x_2^{(2)} + \frac{1}{2}x_3^{(2)} = 0 \\ \frac{11}{5}x_3^{(2)} = 1 \end{cases}$$

da cui le soluzioni:

$$x^{(1)} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \quad x^{(2)} = \begin{pmatrix} \frac{2}{11} \\ -\frac{1}{11} \\ \frac{5}{11} \end{pmatrix}$$

e quindi la matrice incognita X:

$$X = \begin{pmatrix} 1 & \frac{2}{11} \\ -1 & -\frac{1}{11} \\ 1 & \frac{5}{11} \end{pmatrix}$$

8.3.7 Implementazione MATLAB del metodo di Gauss

Implementiamo un risolutore per sistemi lineari, per comprendere a pieno questo risultato. Vogliamo calcolare X come la soluzione del sistema $AX = B$, e capiamo quindi che quello che cerchiamo sono i vettori colonna $x_1, ..., x_n$ tali per cui:

$$Ax_1 = B_1, \quad ..., \quad Ax_n = b_n$$

con b_i l'i-esimo vettore colonna di B. Sfruttiamo allora l'eliminazione di Gauss per ricavare due matrici, U e $B^{(n-1)}$, tali che $Ux = B^{(n-1)}$, con il vantaggio che U è triangolare superiore e quindi risolvibile per sostituzione all'indietro. Possiamo fare questo in MATLAB come:

```
AI = [A, B]
UB1 = gauss_decomp(AB) % modificando gauss_decomp per un solo argomento
U = UB1(1:n, 1:n)
B1 = UB1(1:n, (n + 1):(n + s)
```

A questo punto potremo trovare le colonne x_i come:

```
>> x1 = bck_subst(U, B1(1:n, 1))
>> x2 = bck_subst(U, B1(1:n, 2))
...
```

e infine concatenare le colonne come:

```
>> X = [x1, ..., xn]
```

che realizzato in un unico script risulta:

```
function X = gauss_solve(A, B)
    n = height(A);
    s = width(B);

    AB = [A, B];

    UB1 = gauss_decomp(AB);
    U = UB1(:, 1:n);
    B1 = UB1(:, (n + 1):(n + s));

    X = zeros(n, s);

```

```
    for i = 1:s
        X(:, i) = bck_subst(U, B1(:, i));
    end
end
```

8.3.8 Metodo di Gauss per il calcolo dell'inversa

Un caso particolare è il **calcolo dell'inversa** con l'algoritmo di **Gauss-Jordan**. Infatti, scegliendo:

$$AX = I$$

con $n = s$, le colonne in X diventeranno l'inversa di A (basti vedere che $AA^{-1} = I$ per definizione).

Facciamo un esempio numerico. Prendiamo la stessa matrice di prima:

$$A = \begin{pmatrix} -2 & 1 & 1 \\ -1 & 3 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

e calcoliamone l'inversa come soluzione a $AX = I$. Impostiamo quindi la matrice AI e riduciamola:

$$AI = \left(\begin{array}{ccc|ccc} -2 & 1 & 1 & 1 & 0 & 0 \\ -1 & 3 & 1 & 0 & 1 & 0 \\ 1 & 1 & 2 & 0 & 0 & 1 \end{array}\right) \xrightarrow{\text{Gauss}} \left(\begin{array}{ccc|ccc} -2 & 1 & 1 & 1 & 0 & 0 \\ 0 & \frac{5}{2} & \frac{1}{2} & -\frac{1}{2} & 1 & 0 \\ 0 & 0 & \frac{11}{5} & \frac{4}{5} & -\frac{3}{5} & 1 \end{array}\right) = UB^{(n-1)}$$

da cui i 3 sistemi lineari:

$$x^{(1)} : \begin{cases} -2x_1^{(1)} + x_2^{(1)} + x_3^{(1)} = 1 \\ \frac{5}{2}x_2^{(1)} + \frac{1}{2}x_3^{(1)} = -\frac{1}{2} \\ \frac{11}{5}x_3^{(1)} = \frac{4}{5} \end{cases} \qquad x^{(2)} : \begin{cases} -2x_1^{(2)} + x_2^{(2)} + x_3^{(2)} = 0 \\ \frac{5}{2}x_2^{(2)} + \frac{1}{2}x_3^{(2)} = 1 \\ \frac{11}{5}x_3^{(2)} = -\frac{3}{5} \end{cases}$$

$$x^{(3)} : \begin{cases} -2x_1^{(3)} + x_2^{(3)} + x_3^{(3)} = 0 \\ \frac{5}{2}x_2^{(3)} + \frac{1}{2}x_3^{(3)} = 0 \\ \frac{11}{5}x_3^{(3)} = 1 \end{cases}$$

che hanno come soluzione:

$$X = \begin{pmatrix} x^{(1)} & x^{(2)} & x^{(3)} \end{pmatrix} = A^{-1} = \begin{pmatrix} -\frac{5}{11} & \frac{1}{11} & \frac{2}{11} \\ -\frac{3}{11} & \frac{5}{11} & -\frac{1}{11} \\ \frac{4}{11} & -\frac{3}{11} & \frac{5}{11} \end{pmatrix}$$

8.3.9 Implementazione MATLAB del metodo di Gauss per il calcolo dell'inversa

Facciamo un ultimo esempio MATLAB di questo procedimento. Vogliamo calcolare A^{-1} come la soluzione del sistema $AX = I$, e capiamo quindi che quello che cerchiamo sono i vettori colonna $i_1, ..., i_n$ tali per cui:

$$Ai_1 = I_1, \quad ..., \quad Ai_n = I_n$$

con I_i il vettore di zeri con un 1 all'i-esima riga. Anche allora vogliamo sfruttare l'eliminazione di Gauss per ricavare due matrici, U e $B^{(n-1)}$, tali che $Ux = B^{(n-1)}$, con il vantaggio che U è triangolare superiore e quindi risolvibile per sostituzione all'indietro.

Questo è esattamente quello che ci permette di fare la funzione `gauss_solve()` definita prima, fissando il secondo argomento all'identità I (in matlab `eye(n)`):

```
function A_inv = gauss_inv(A)
    n = height(A);
    A_inv = gauss_solve(A, eye(n));
end
```

9 Lezione del 24-03-25

9.1 Pivoting

L'algoritmo di eliminazione di Gauss che abbiamo definito alla scorsa lezione ha un punto di fallimento nel caso uno degli elementi $a_{ii}^{(i-1)}$ sia $= 0$, o comunque ≈ 0, in quanto vorremmo a quel punto calcolare un moltiplicatore $l_{ji} = \frac{a_{ji}}{a_{ii}} \to$ non ben definito.

In tal caso si può modificare l'algoritmo sfruttando una matrice di permutazione che porti un elemento diverso da zero nella stessa posizione di a_{ii}. Vorremo quindi cercare un indice h tal per cui $a_{hi}^{(i-1)}$ sia di modulo massimo nella sua colonna al di sotto di i, cioè:

$$a_{hi}^{(i-1)} \geq \max_{j=i,\dots,n} |a_{ji}^{(i-1)}|$$

e scambiare la riga i con la riga h. Infatti, se $\det(A) \neq 0$, allora necessariamente esiste un $a_{ji}^{(i-1)} \neq 0$ (altrimenti si ha uno 0 obbligato sulla diagonale, che con la matrice triangolare a blocchi dà $\det(A^{(i-1)}) = 0$). □

Un altra conseguenza di questo approccio è che tutti i moltiplicatori l_{ji} diventeranno ≤ 1. L'algoritmo di Gauss con questa modifica si chiama **eliminazione di Gauss con pivoting parziale** (*parziale* perché ne esistono versioni più sofisticate, che non vedremo).

Osserviamo che ogni scambio di righe equivale a moltiplicare a sinistra per una matrice di permutazione Π_i. Quindi il metodo di Gauss con pivoting può essere rappresentato come:

Algoritmo 2 Eliminazione di Gauss con pivoting parziale

Input: un sistema lineare qualsiasi $Ax = b$
Output: un sistema lineare triangolare superiore $Ux = c$
for $i = 1$ to n **do**
  Trova la matrice Π_i che porta l'elemento di modulo massimo in testa
  $A \leftarrow \Pi_i A$
  for $j = i$ to n **do**
    Calcola il **moltiplicatore** $l_{ji} = \frac{a_{ji}^{(i-1)}}{a_{ii}^{(i-1)}}$
    Aggiungi alla riga j la riga i moltiplicata per l_{ij}
  end for
end for

Vediamo un esempio pratico dell'algoritmo prima di procedere all'implementazione MATLAB. Prendiamo la matrice A e il vettore b:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Nel ridurre la matrice aumentata Ab:

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 2 \\ 7 & 8 & 0 & 3 \end{array}\right)$$

ci accorgiamo che alla prima colonna l'entrata di modulo massimo è 7, di indice 3. Si permutano quindi la prima e la terza riga:

$$\xrightarrow{\Pi_1} \left(\begin{array}{ccc|c} 7 & 8 & 0 & 3 \\ 4 & 5 & 6 & 2 \\ 1 & 2 & 3 & 1 \end{array}\right) \xrightarrow{H_1\Pi_1} \left(\begin{array}{ccc|c} 7 & 8 & 0 & 3 \\ 0 & \frac{3}{7} & 6 & \frac{2}{7} \\ 0 & \frac{6}{7} & 3 & \frac{4}{7} \end{array}\right)$$

nuovamente, l'entrata di modulo massimo è all'indice 3. Si permutano quindi la seconda e la terza riga:

$$\xrightarrow{\Pi_2 H_1 \Pi_1} \left(\begin{array}{ccc|c} 7 & 8 & 0 & 3 \\ 0 & \frac{6}{7} & 3 & \frac{4}{7} \\ 0 & \frac{3}{7} & 6 & \frac{2}{7} \end{array}\right) \xrightarrow{H_2\Pi_2 H_1 \Pi_1} \left(\begin{array}{ccc|c} 7 & 8 & 0 & 3 \\ 0 & \frac{6}{7} & 3 & \frac{4}{7} \\ 0 & 0 & \frac{9}{2} & 0 \end{array}\right)$$

9.1.1 Implementazione MATLAB del metodo di eliminazione di Gauss con pivoting

Modifichiamo quindi la funzione gauss_decomp() per introdurre il meccanismo di pivoting appena visto:

```
function [A, b] = gauss_decomp(A, b)
    n = height(A);

    if nargin < 2
        b = zeros(n, 1);
    end

    for i = 1:n % i itera sulle diagonali
        % qui fai il pivot
        max_abs = max(abs(A(i:n, i)));
        h = find(abs(A(i:n, i)) == max_abs, 1);
        h = h + i - 1; % max abs si conta da i in poi

        A([i, h], :) = A([h, i], :); % permuta A
        b([i, h]) = b([h, i]); % permuta b

        den = A(i, i);

        for j = (i + 1):n % j itera sulle righe
            mul = A(j, i) / den; % moltiplicatore
            L(j, i) = mul;

            A(j, :) = A(j, :) - A(i, :) * mul;
            b(j) = b(j) - b(i) * mul;
        end
    end
end
```

9.1.2 Fattorizzazione LU con pivoting

Vediamo come ricavare una fattorizzazione LU dal metodo di Gauss modificato con il pivoting. Si ha quindi che la matrice U si evolve come:

$$A \to \Pi_1 A \to H_1\Pi_1 A \to ... \to H_{n-1}\Pi_{n-1}...H_1\Pi_1 A = U$$

mentre per la L dovremo notare che:

$$LU = \Pi A$$

dove la matrice Π rappresenta tutte le permutazioni fatte sulle righe di A.

Si ha quindi che:

- U è la matrice triangolare superiore trovata alla fine del metodo di Gauss con pivoting;
- L è la matrice dei moltiplicatori, a cui però si devono applicare gli scambi delle righe, come segue: se al passo i applico la matrice Π_i, devo applicare lo stesso cambio nelle prime $i-1$ colonne di L, sotto la diagonale.

Vediamo un esempio numerico spieghi il processo di formazione della matrice L e della matrice di permutazione Π. Presa la stessa matrice dell'esempio precedente:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}$$

abbiamo che le permutazioni sono, in sequenza:

$$\Pi_1 : \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \to \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}, \quad \Pi_2 : \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} \to \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}$$

o, in forma matriciale:

$$\Pi_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad \Pi_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Calcoliamo quindi L. Π_1 è irrilevante al calcolo di L, quindi la ignoriamo. Vediamo che i primi due moltiplicatori sono $l_{21} = \frac{4}{7}$ e $l_{31} = \frac{1}{7}$, da cui si imposta $L^{(1)}$:

$$L^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{4}{7} & 1 & 0 \\ \frac{1}{7} & 0 & 1 \end{pmatrix}$$

Notiamo quindi che dalla Π_2 dobbiamo scambiare gli elementi sotto la diagonale della prima colonna, quindi:

$$L^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{7} & 1 & 0 \\ \frac{4}{7} & 0 & 1 \end{pmatrix}$$

Infine, l'ultimo moltiplicatore $l_{32} = \frac{1}{2}$ non ha ambiguità:

$$L^{(2)} = L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{7} & 1 & 0 \\ \frac{4}{7} & \frac{1}{2} & 1 \end{pmatrix}$$

Il calcolo di Π deriva invece direttamente studiando la permutazione complessiva data dalle $\Pi_1, ..., \Pi_{n-1}$, in questo caso:

$$\Pi : \begin{pmatrix}1\\2\\3\end{pmatrix} \xrightarrow{\Pi_1} \begin{pmatrix}3\\2\\1\end{pmatrix} \xrightarrow{\Pi_2} \begin{pmatrix}3\\1\\2\end{pmatrix}$$

da cui:

$$\Pi = \begin{pmatrix}0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{pmatrix}$$

Con brevi calcoli si verifica che:

$$LU = \Pi A$$

9.1.3 Implementazione MATLAB completa del metodo di eliminazione di Gauss con pivoting

Vediamo quindi l'implementazione completa, che calcola anche la matrice L e la matrice Π. Notiamo inoltre l'argomento condizionale b, che viene ignorato se non fornito (abbiamo constatato che spesso è così).

```
function [A, b, L, P] = gauss_decomp(A, b)
    n = height(A);

    if nargin < 2
        b = zeros(n, 1);
    end

    L = eye(n); % prepara L
    P = eye(n); % prepara P

    for i = 1:n % i itera sulle diagonali
        % qui fai il pivot
        max_abs = max(abs(A(i:n, i)));
        h = find(abs(A(i:n, i)) == max_abs, 1);
        h = h + i - 1; % max abs si conta da i in poi

        clA([i, h], :) = A([h, i], :); % permuta A
        b([i, h]) = b([h, i]); % permuta b
        if i > 1
            L([i, h], 1:(i - 1)) = L([h, i], 1:(i - 1)); % permuta L
        end
        P([i, h], :) = P([h, i], :); % permuta P

        den = A(i, i);

        for j = (i + 1):n % j itera sulle righe
            mul = A(j, i) / den; % moltiplicatore
            L(j, i) = mul;

            A(j, :) = A(j, :) - A(i, :) * mul;
            b(j) = b(j) - b(i) * mul;
        end
    end
end
```

9.1.4 Determinante con pivoting

Possiamo sfruttare la matrice Π per il calcolo del determinante. Si ha infatti dal teorema di Binet-Cauchy (4.1) che:

$$\det(\Pi)\det(A) = \det(\Pi A) = \det(LU) = \det(L)\det(U)$$

e quindi:

$$\det(A) = \det(\Pi)^{-1}\det(L)\det(U)$$

dove $\det(L) = 1$ (triangolare inferiore con diagonale di 1). Si nota poi che $\det(\Pi)^{-1}$ è $(-1)^s$ è il numero di pivot che effettuiamo. A questo punto $\det(U)$ è semplicemente il prodotto degli elementi sulla diagonale (triangolare superiore), cioè:

$$\prod_{i=1}^{n} a_{ii}^{(i-1)} = \prod_{i=1}^{n} u_{ii}$$

e quindi:

$$\det(A) = (-1)^s \prod_{i=1}^{n} a_{ii}^{(i-1)} = (-1)^s \prod_{i=1}^{n} u_{ii}$$

Si può quindi usare il metodo di Gauss, ancora una volta, per il calcolo del determinante di una matrice, con costo pari al costo dell'eliminazione di Gauss ($O(\frac{2}{3}n^3)$), molto meglio dello sviluppo di Laplace! ($O(n!)$).

9.1.5 Implementazione MATLAB del metodo di Gauss per il determinante

In MATLAB si può calcolare il prodotto delle diagonali come `prod(diag(A))` e il segno di una permutazione come `det(P)` (anche se sicuramente esistono approcci più efficienti). Si può quindi realizzare uno script simile al seguente per il calcolo del determinante sfuttando `gauss_decomp()` con permutazioni:

```
function d = gauss_det(A)
    function s = perm_sign(P)
        s = det(P); % ci sono modi piu' efficienti, vale l'esempio
    end

    [U, ~, ~, P] = gauss_decomp(A);
    d = prod(diag(U)) * perm_sign(P);
end
```

9.2 Condizionamento di un sistema lineare

Date $A \in \mathbb{C}^{n\times n}$ e $b \in \mathbb{C}^n$, supponiamo di voler trovare $Ax = b$ ma a causa di errori nei dati o errori di arrotondamento troviamo (con un qualunque metodo) un vettore perturbato $x + \delta x \in \mathbb{C}^n$ che risolve un sistema lineare di per sé perturbato:

$$(A + \delta A)(x + \delta x) = (b + \delta b)$$

con δA e δB perturbazioni "piccole" della matrice A e del vettore b, quindi $A+\delta A \in \mathbb{C}^{n\times n}$ e $b + \delta b \in \mathbb{C}^n$

La domanda è, se δA e δb sono piccole, posso concludere che anche δx è relativamente piccolo? Si scopre che la risposta a questa domanda è generalmente no. Prendiamo ad esempio il sistema 2×2:

$$\begin{pmatrix} 1 & -1 \\ 1 & 1.000001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

da cui x esatto è $\begin{pmatrix}1 & 0\end{pmatrix}$. Perturbando b a $\begin{pmatrix}0.999999 & 1\end{pmatrix}$, si ha x perturbato a $\begin{pmatrix}-10^{-6} & -1\end{pmatrix}$, che è chiaramente un cambiamento drastico. Viene da sé che agendo sulla matrice A potremo ottenere effetti anche più drammatici.

9.2.1 Condizionamento in δb

Riprendiamo quindi la definizione di errore relativo:

$$\epsilon = \frac{|\delta x|}{|x|}$$

Pirma di valutare questo errore, diamo la definizione di **numero di condizionamento**:

Definizione 9.1: Numero di condizionamento

Chiamiamo numero di condizionamento di una matrice A, data una certa norma $|\cdot|$, il valore:

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

Si potrà allora dare, rispetto alla sola deviazione in b (δb), il seguente risultato:

Teorema 9.1: Condizionamento in δb

Se $\delta A = 0$, si ha:

$$\frac{|\delta x|}{|x|} \leq \mu(A) \cdot \frac{|\delta b|}{|b|}$$

Assumendo $\delta A = 0$ con $\det(A) \neq 0$, come nell'esempio precedente, e quindi perturbazioni solo del termine noto, si ha:

$$A(x + \delta x) = (b + \delta b) \iff A\delta x = \delta b$$

visto che $Ax = b$. Passando alle norme, si ha che:

$$|\delta x| \leq |A^{-1}| \cdot |\delta b|$$

e inoltre:

$$|Ax| = |b| \implies |A||x| \geq |b| \implies |x| \geq \frac{|b|}{|A|}$$

quindi:

$$\frac{|\delta x|}{|x|} \leq \frac{|A^{-1}| \cdot |\delta b| \cdot |A|}{|b|} = \frac{|\delta b|}{|b|} \cdot |A| \cdot |A^{-1}|$$

dove ci interessa il valore $|A| \cdot |A^{-1}|$, cioè il numero di condizionamento $\mu(A)$, l'unico che non dipende dall'errore assoluto δb. □

Abbiamo quindi che se $\mu(A) >> 1$, allora l'errore relativo può essere molto più grande dell'errore relativo dei dati e il problema si dice *mal condizionato*.

10 Lezione del 28-03-25

Riprendiamo il discorso sull'errore inerente dei sistemi lineari.

10.0.1 Condizionamento in $\delta \mathbf{A}$

Avevamo preso delle perturbazioni sulle matrici A e b (dovute a vari effetti reali, quali errori di arrotondamento, di misura, ecc...) nella forma:

$$(A + \delta A)(x + \delta x) = (b + \delta b)$$

e volevamo capire quanto può essere grande l'errore relativo $\frac{|\delta x|}{|x|}$, da:

$$\frac{\text{sol. perturbata} - \text{sol. reale}}{\text{sol. reale}} = \frac{|x + \delta x - x|}{|x|} = \frac{|\delta x|}{|x|}$$

Nel caso di $\delta A = 0$, abbiamo visto di poter maggiorare tale quantità come:

$$\frac{|\delta x|}{|x|} \leq \mu(A) \cdot \frac{|\delta b|}{|b|}$$

con $\mu(A) = |A| \cdot |A^{-1}|$ *numero di condizionamento* (definizione 9.1).

Riguardo a $\mu(A)$, si ha che è ≥ 1, cioè chiaramente non si può ridurre l'errore oltre la perfezione, e se $\mu(A) \approx 10^k$, k è il numero di cifre significative che si *perdono* nel risultato $x + \delta x$.

Possiamo quindi reintrodurre il termine δA ed enunciare il seguente teorema:

Teorema 10.1: Condizionamento in $\delta \mathbf{A}$

Se $|\delta A| \cdot |A^{-1}| < 1$ allora si ha:

$$\frac{|\delta x|}{|x|} \leq \frac{\mu(A)}{1 - \mu(A) \cdot \frac{|\delta A|}{|A|}} \cdot \left(\frac{|\delta A|}{|A|} + \frac{|\delta b|}{|b|} \right)$$

dove osserviamo che se $\delta A = 0$ si ottiene la stessa diseguaglianza che abbiamo dimostrato col teorema 9.1.

La dimostrazione del teorema non è immediata. Ci arriviamo in 3 iterazioni successive, restringendo man mano le approssimazioni fatte.

1. Una prima stima si potrebbe avere calcolando direttamente:

   $$(A + \delta A)(x + \delta x) = (b + \delta b) \implies Ax + A\delta x + \delta A x + \delta A \delta x = b + \delta b$$

   $Ax = b$ dalle ipotesi, mentre il termine $\delta A \delta b$, il prodotto di due errori, è trascurabile, quindi:

   $$A\delta x + \delta A x \approx \delta b \implies \delta x \approx A^{-1} (\delta b - \delta A x)$$

   Passando alle norme si ha:

   $$|\delta x| \leq |A^{-1}| \left(|\delta b| + |\delta A||x| \right) \tag{1}$$

   da cui dividendo per $|x| \geq \frac{|A|}{|b|}$ e moltiplicando e dividendo il termine a destra per $|A|$, si ha:

   $$\frac{|\delta x|}{|x|} \leq |A^{-1}| \left(\frac{|\delta b|}{|b|} |A| + |\delta A| \frac{|A|}{|A|} \right) = |A^{-1}||A| \left(\frac{|\delta b|}{|b|} + \frac{|\delta A|}{|A|} \right) = \mu(A) \left(\frac{|\delta b|}{|b|} + \frac{|\delta A|}{|A|} \right)$$

   Da cui otteniamo essenzialmente la forma del teorema.

2. Per capire il denominatore del numero di condizionamento, che migliora il maggiorante nei casi dove A è quasi singolare, adottiamo un procedimento diverso che evidenzia l'errore fatto sull'inversa $(A+\delta A)^{-1}$ (nel calcolo precedente, starebbe nel termine $\delta A \delta x$ che abbiamo trascurato). Avremo quindi:

$$(A+\delta A)(x+\delta x) = (b+\delta b) \implies (x+\delta x) = (A+\delta A)^{-1}(b+\delta b)$$

$(A+\delta A)^{-1}$ risulta di difficile approssimazione. Riscriviamolo come:

$$(A+\delta A)^{-1} = (I + A^{-1}\delta A)^{-1}A^{-1}$$

e consideriamo la serie di Von Neumann:

$$(I-M)^{-1} = I + M + M^2 + ...$$

troncando al primo termine, si ha che possiamo riscrivere la prima inversa come:

$$(I + A^{-1}\delta A)^{-1} \approx I - A^{-1}\delta A$$

e quindi:

$$(A+\delta A)^{-1} \approx (I - A^{-1}\delta A)A^{-1} = A^{-1} - A^{-1}\delta A A^{-1} \tag{2}$$

Possiamo convincerci che l'approssimazione è valida sostituendo nella formula per $x+\delta x$ e trovando:

$$x+\delta x = (A^{-1} - A^{-1}(A+A)1 = (I+A1A)1A1taAA^{-1})(b+\delta b) = A^{-1}b + A^{-1}\delta b - A^{-1}\delta A A^{-1}b - A^{-1}\delta A A^{-}$$

$x = A^{-1}b$ dalle ipotesi, mentre il termine $A^{-1}\delta A A^{-1}\delta b$ contiene il prodotto di due errori, ed è trascurabile, quindi:

$$\delta x \approx A^{-1}\delta b - A^{-1}\delta A A^{-1}b = A^{-1}\delta b - A^{-1}\delta A x = A^{-1}(\delta b - \delta A x)$$

che è esattamente la forma della (1) che avevamo al primo metodo.

3. Riprendiamo la forma in (2) dell'inversa:

$$(A+\delta A)^{-1} \approx (I - A^{-1}\delta A)A^{-1}$$

per introdurre il denominatore che vediamo nell'enunciato del teorema. Passando alle norme, si avrà:

$$|I - A^{-1}\delta A||A^{-1}| \leq \frac{|A^{-1}|}{1 - |A^{-1}\delta A|}$$

sempre dalla serie di Nemeann.w Prendendo questa come la norma dell'inversa per il passaggio alle norme della (1), si ha:

$$\frac{|\delta x|}{|x|} \leq \frac{|A^{-1}|}{1-|A^{-1}\delta A|}\left(\frac{|\delta b|}{|b|}|A| + |\delta A|\frac{|A|}{|A|}\right) \leq \frac{|A^{-1}||A|}{1-|A^{-1}||\delta A|}\left(\frac{|\delta b|}{|b|} + \frac{|\delta A|}{|A|}\right)$$

dove l'$|A^{-1}||\delta A|$ al denominatore si può scrivere come $\mu(A)\frac{|\delta A|}{|A|}$, in quanto:

$$|A^{-1}||\delta A| = \mu(A)\cdot\frac{|\delta A|}{|A|} = |A||A^{-1|}\cdot\frac{|\delta A|}{|A|}$$

da cui la forma del teorema. □

10.0.2 Stima di mu

In genere è abbastanza costoso calcolare il numero di condizionamento, in quanto bisogna calcolare un inversa e quindi la sua norma. Quello che si può fare è cercarne una stima.

Ad esempio, se A è hermitiana ($A = A^H$) e si considera la norma euclidea $|\cdot|_2$, si ha che:

$$|A|_2 \cdot |A^{-1}|_2 = \rho(A) \cdot \rho(A^{-1}) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$$

cioè prendiamo il rapporto fra l'autovalore più grande e l'autovalore più piccolo di A, per cui si possono usare i metodi per gli autovalori (che vedremo verso la fine del corso).

10.0.3 Stime a posteriori

Supponiamo di aver calcolato $\tilde{x} \in \mathbb{C}^n$ con un qualunque metodo di approssimazione di x, prese A e b per buone. Per valutare se $\tilde{x}$ è una *buona* approssimazione basta guardare al **vettore residuo**, cioè:

$$r = b - A\tilde{x}$$

Potremmo chiederci se, con $|r|$ piccolo, si hanno anche $|x - \tilde{x}|$ piccoli. Sottraiamo allora $Ax = b$ da r:

$$A(x - \tilde{x}) = r \implies x - \tilde{x} = A^{-1}r$$

e quindi vale la diseguaglianza:

$$|x - \tilde{x}| \leq |A^{-1}||r|$$

Usando:

$$|x| \geq \frac{|b|}{|A|}$$

si otterrà allora che:

$$\frac{|x - \tilde{x}|}{|x|} \leq \frac{|A||A^{-1}||r|}{|b|} = \mu(A) \cdot \frac{|r|}{|b|}$$

Allora, in problemi ben condizionati, avremo che $\frac{|x-\tilde{x}|}{|x|}$ errore relativo e $|r|$ sono comparabili, mentre in problemi con condizionamento $\mu(A) >> 1$ si potrebbe avere:

$$\frac{\frac{|x-\tilde{x}|}{|x|}}{|r|} \approx \mu(A)$$

10.0.4 Tecniche per l'approssimazione delle inverse

Come nota conclusiva della sezione sull'errore nei sistemi lineari, notiamo che ogni volta che c'è bisogno di risolvere una forma del tipo:

$$Ax = b \implies x = A^{-1}b$$

il metodo naive sarebbe quello di calcolare A^{-1} e moltiplicare per b. Vediamo se si può fare di meglio.

Preso $n = 1$, la forma sarà quella di una semplice equazione lineare:

$$ax = b. \implies x = \frac{b}{a}$$

cioè basta fare una sola divisione, mentre l'approccio naive risulterebbe nel:

1. Calcolare il reciproco a^{-1};
2. Moltiplicare il reciproco per $b \implies x = a^{-1} \cdot b$.

che chiaramente risulta in più passaggi, e quindi più approssimazioni intermedie e in definitiva maggiore errore.

Nel caso matriciale il metodo di divisione equivale a fare una divisione matrice-vettore (che in MATLAB si effettua come `A \ b`), di complessità $O(\frac{2}{3}n^3)$. Di contro, il metodo naive (che in MATLAB si effettua come `inv(A) * b`) avrà complessità intorno ad $O(\frac{8}{3}n^3)$, in quanto calcola la fattorizzazione LU e risolve 2n sistemi triangolari.

Inoltre, l'approccio naive è anche meno accurato, in quanto se il numero di condizionamento $\mu(A)$ è alto, l'errore di approssimazione nei passaggi intermedi potrebbe accumularsi molto di più che rispetto all'approccio della semplice divisione matrice-vettore (tanto che la stessa documentazione di MATLAB suggerisce di evitarlo).

10.1 Metodi iterativi per i sistemi lineari

Veniamo quindi a trattare i metodi iterativi per la risoluzione dei sistemi lineari.

L'idea è quella di approssimare la soluzione di un sistema lineare $Ax = b$ generando una successione di vettori $\{x^{(k)}\}_{k \in \mathbb{N}}$ tali che:

$$\lim_{k \to +\infty} x^{(k)} = x$$

La motivazione è chiaramente quella di eludere l'alta complessità di $\sim O(n^3)$ che ha la risoluzione con metodi diretti. Altra motivazione potrebbe essere quella di non conoscere direttamente A, ma solo l'applicazione:

$$v \to A \cdot v$$

(si pensi, anche se rappresenta un caso *non* lineare, ai metodi di discesa a gradiente che ottimizzano funzioni non immediatamente calcolabili o anche solo esprimibili).

Abbiamo quindi che un buon metodo iterativo dovrà:

1. Costare meno di $O(n^3)$ ad ogni passaggio (altrimenti sarebbe inutile rispetto ai metodi diretti), e quindi richiedere solo prodotti di matrici con vettori, o risoluzione di sistemi lineari favorevoli (triangolari, diagonali, ecc...);
2. Data una certa **accuratezza** posta come obiettivo, impiegare un numero ragionevole di iterazioni per raggiungerla.

10.1.1 Metodi di punto fisso

L'idea è quella di partire da $Ax - b = 0$ e di riscrivere come un'equazione equivalente in forma:

$$x = Hx + c, \quad H \in \mathbb{C}^{n \times m},\ c \in \mathbb{C}^n$$

Facciamo alcuni esempi:

1. Si sceglie una matrice $G \in \mathbb{C}^{n \times n}$ invertibile e si considera:

   $$x = x - G(Ax - b) = (I - GA)x + Gb$$

   cioè:

   $$H = I - GA, \quad c = Gb$$

2. Si scompone A come:
$$A = A_1 + A_2$$
per cui:
$$Ax = b \Leftrightarrow (A_1 + A_2)c = b \Leftrightarrow A_1 x = -A_2 x + b \Leftrightarrow x = -A_1^{-1} A_2 x + A_1^{-1} b$$
cioè ancora:
$$H = -A_1^{-1} A_2, \quad c = A_1^{-1} b$$

Una volta trovata un'equazione di punto fisso $x = Hx + c$, quindi, si considera il seguente metodo iterativo:

$$\begin{cases} x^{(0)} \text{ dato} \\ x^{(k+1)} = Hx^{(k)} + c, \quad k = 1, 2, 3, ... \end{cases}$$

partendo da $x^{(0)}$ come dato a priori (sarà la soluzione che vorremo raffinare).

Vediamo che infatti:

Definizione 10.1: Matrice di iterazione

La matrice H di un equazione di punto fisso $x = Hx + c$ viene detta matrice di iterazione.

Per avere una verifica della validità dei metodi di punto fisso, enunciamo il seguente teorema:

Teorema 10.2: Validità dei metodi di punto fisso

Il metodo di punto fisso dato da:

$$\begin{cases} x^{(0)} \text{ dato} \\ x^{(n+1)} = Hx^{(k)} + c, \quad k = 1, 2, 3, ... \end{cases}$$

converge $\forall x^{(0)} \in \mathbb{C}^n$ se e solo se $\rho(H) < 1$.

Questo si dimostra prendendo l'equazione di punto fisso $x = Hx + c$, soddisfatta da x soluzione esatta, per cui:

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

Possiamo chiamare $e^{(k)} = x^{(k)} - x$, cioè l'errore al passo k, e quindi l'errore al passo $k + 1$ sarà:
$$e^{(k+1)} = He^{(k)} = H^k e^{(0)}$$

Basterà allora prendere il limite:

$$\lim_{k \to +\infty} e^{(k+1)} = \lim_{k \to +\infty} H^k e^{(0)}$$

Perchè questo tenda a 0, basterà imporre $\rho(H) < 1$, in quanto in tal caso:

$$\lim_{k \to +\infty} H^k e^{(0)} = 0$$

qualsiasi sia l'errore iniziale $e^{(0)}$ (e quindi la scelta di soluzione iniziale $x^{(0)}$). □

In particolare, possiamo ricordare che se $|H| < 1 \implies \rho(H) < 1$, quindi può risultare verificare questa condizione, che è sì più stretta ma anche più facile da verificare. Ricordiamo che non vale assolutamente il contrario, cioè $|H| > 1 \not\implies \rho(H) > 1$.

Di contro, vale anche la condizione $|\det(H)| \geq 1 \implies \rho(H) \geq 1$, che come prima non si inverte in $|\det(H)| < 1 \not\implies \rho(H) < 1$.

10.1.2 Velocità di convergenza

Guardando alla dimostrazione del teorema 10.2, possiamo osservare che il raggio spettrale $\rho(H)$ ci dà un informazione anche riguardo alla **velocità di convergenza** del metodo di punto fisso.

Infatti, si avrà che vale:

$$\frac{|e^{(k)}|}{|e^{(0)}|} \leq |H^k|$$

Dall'algebra lineare, si ha che quando k è abbastanza grande, $H^k \approx \rho(H)^k$, almeno per norme indotte, in quanto:

$$\lim_{k \to +\infty} \sqrt[k]{|H^k|} = \rho(H)$$

Questo ci dice che se si hanno 2 metodi di punto fisso con matrici di iterazione H_1 e H_2 tali che:

$$\rho(H_1) < \rho(H_2) < 1$$

allora il primo metodo converge più velocemente del secondo, è dall'ulteriore ipotesi < 1, entrambi convergono.

Inoltre, si può stimare il numero di iterazioni k necessarie a raggiungere un certo valore dell'errore:

$$\frac{|e^{(k)}|}{|e^{(0)}|} = \frac{|x^{(k)} - x|}{|e^{(0)}|} \leq \delta$$

infatti, in tal caso basterà imporre:

$$\rho(H)^k \leq \delta \implies k \geq \frac{\log(\delta)}{\log(\rho(H))}$$

10.1.3 Criteri di stop

Molto spesso non è pratico decidere un errore e fare le k iterazioni che dovrebbero portare a tale errore (in quanto non è scontato conoscere $\rho(H)$), ma bensì si preferisce definire un **criterio di stop** per la terminazione dell'algoritmo raggiunte determinate condizioni.

Queste condizioni possono essere:

1. Si può restringere il residuo:
$$\frac{|r^{(k)}|}{|b|} < \delta$$
2. Si può restringere direttamente la variazione di errore:
$$|x^{(k+1)} - x^{(k)}| < \delta$$

10.2 Metodi di Jacobi e Gauss-Seidel

Vediamo i metodi più famosi di questo tipo, detti metodi di **Jacobi** e **Gauss-Seidel**.

L'idea è di scomporre la matrice A come:

$$A = D - E - F =$$

$$\begin{pmatrix} d_1 & 0 & ... & 0 \\ 0 & d_2 & ... & 0 \\ 0 & ... & d_{n-1} & 0 \\ 0 & ... & 0 & d_n \end{pmatrix} - \begin{pmatrix} 0 & 0 & ... & 0 \\ -e_{21} & 0 & ... & 0 \\ -e_{n-1,1} & ... & 0 & 0 \\ -e_{n1} & ... & -e_{n,n-1} & 0 \end{pmatrix} - \begin{pmatrix} 0 & -f_{12} & ... & -f_{1n} \\ 0 & 0 & ... & -f_{2n} \\ 0 & ... & 0 & -f_{n-1,n} \\ 0 & ... & 0 & 0 \end{pmatrix}$$

con D diagonale, E l'opposta della triangolare inferiore a diagonale nulla e F l'opposta della triangolare superiore a diagonale nulla. Varrà quindi:

$$Ax = b \Leftrightarrow (D - E - F)x = b$$

10.2.1 Metodo di Jacobi

Il metodo di Jacobi consiste nel riscrivere quanto trovato come:

$$Dx = (E + F)x + b \implies x = D^{-1}(E + F)x + D^{-1}b$$

cioè trovare un equazione di punto fisso con:

$$H = D^{-1}(E + F), \quad c = D^{-1}b$$

e quindi applicare l'algoritmo:

$$\begin{cases} x^{(0)} \text{ dato} \\ x^{(k+1)} = D^{-1}(E + F)x^{(k)} + D^{-1}b = D^{-1}((E + F)x^{(k)} + b) \end{cases}$$

Osserviamo che ad ogni iterazione si calcola un prodotto matrice-vettore e si risolve un sistema diagonale ($O(n)$), in quanto la D si inverte facilmente (è diagonale):

$$H_J = D^{-1}(E + F) = \begin{pmatrix} \frac{1}{d_1} & 0 & ... & 0 \\ 0 & \frac{1}{d_2} & ... & 0 \\ 0 & ... & \frac{1}{d_{n-1}} & 0 \\ 0 & ... & 0 & \frac{1}{d_n} \end{pmatrix} \begin{pmatrix} 0 & -f_{12} & ... & -f_{1n} \\ -e_{21} & 0 & ... & -f_{2n} \\ -e_{n-1,1} & ... & 0 & -f_{n-1,n} \\ -e_{n1} & ... & -e_{n,n-1} & 0 \end{pmatrix}$$

per cui:

$$(H_J)_{ij} = \begin{cases} -\frac{a_{ij}}{a_{ii}}, & i \neq j \\ 0 & i = j \end{cases}$$

Le matrici che descriveranno il passo di Jacobi saranno allora:

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

osservando che per calcolare $x_i^{(k+1)}$ serviranno *tutte* le componenti di $x^{(k)}$, cioè in codice bisogna mantenere due vettori, in quanto non si può sovrascrivere $x^{(k)}$ prima di aver finito di calcolare $x^{(k+1)}$.

10.2.2 Implementazione MATLAB del metodo di Jacobi

In MATLAB, una possibile implementazione che sfrutta le funzionalità di calcolo matriciale può essere la seguente:

```
function x_old = jacobi(A, b, k, x_old)
    n = height(A);

    if nargin < 4
        % stima iniziale
        x_old = zeros(n, 1);
    end

    % imposta le matrici D E F
    D = diag(A);
    EF = -A + diag(D); % D e' un vettore

    % imposta c
    c = b ./ D;

    for j = 1:k
        x_new = (EF * x_old) ./ D + c;
        x_old = x_new;

        % stampa informazioni
        fprintf("it = %d, res = %.3f\n", j, norm(b - A * x_old));
        disp(x_old);
    end
end
```

10.2.3 Implementazione C++ del metodo di Jacobi

Possiamo sfruttare la forma scalare del metodo di Jacobi per realizzarne un'implementazione direttamente in C++. Definiamo inizialmente alcune funzioni helper in un header, `mat.h`, che useremo anche nell'esempio successivo a questo:

```
#ifndef MAT_H
#define MAT_H

#include <iostream>

inline double** get_matrix(unsigned n) {
  double** A = new double*[n];

  for(int r = 0; r < n; r++) {
    A[r] = new double[n];

    for(int c = 0; c < n; c++) {
      double e;
      std::cout << "Inserisci l'entrata " << r + 1 << ", " << c + 1 << "
    di A" << std::endl;
      std::cin >> e;

      A[r][c] = e;
    }
  }

  return A;
}

```

```
24 inline double* get_vector(unsigned n) {
25   double* b = new double[n];
26 
27   for(int r = 0; r < n; r++) {
28     double e;
29     std::cout << "Inserisci l'entrata " << r + 1 << " di b" << std::endl;
30     std::cin >> e;
31 
32     b[r] = e;
33   }
34 
35   return b;
36 }
37 
38 inline void print_matrix(double** A, unsigned n) {
39   for(int r = 0; r < n; r++) {
40     for(int c = 0; c < n; c++) {
41       std::cout << A[r][c] << "\t";
42     }
43     std::cout << std::endl;
44   }
45 }
46 
47 inline void print_vector(double* b, unsigned n) {
48   for(int r = 0; r < n; r++) {
49     std::cout << b[r] << std::endl;
50   }
51 }
52 
53 inline void init_env(unsigned &n, unsigned &k, double** &A, double* &b) {
54   // prendi dati
55   std::cout << "Inserisci la dimensione n" << std::endl;
56   std::cin >> n;
57   std::cout << std::endl;
58 
59   std::cout << "Inserisci il numero di iterazioni k" << std::endl;
60   std::cin >> k;
61   std::cout << std::endl;
62 
63   std::cout << "Inserzione della matrice A" << std::endl;
64   A = get_matrix(n);
65   std::cout << std::endl;
66 
67   std::cout << "Inserzione del vettore b" << std::endl;
68   b = get_vector(n);
69   std::cout << std::endl;
70 
71   // conferma dati
72   std::cout << "La matrice A e':" << std::endl;
73   print_matrix(A, n);
74   std::cout << std::endl;
75 
76   std::cout << "Il vettore b e':" << std::endl;
77   print_vector(b, n);
78   std::cout << std::endl;
79 }
80 
81 inline void clean_env(unsigned n, double** A, double* b) {
82   // ripulisci
83   for(int r = 0; r < n; r++) {
```

```
    delete[] A[r];
  }
  delete[] A;

  delete[] b;
}

inline void vec_copy(unsigned n, double* v1, double* v2) {
  for(int r = 0; r < n; r++) {
    v2[r] = v1[r];
  }
}

#endif
```

E definiamo quindi l'algoritmo come segue:

```
double* jacobi(unsigned n, unsigned k, double** A, double* b) {
  // inizializza vettori
  double* v1, *v2;
  v1 = new double[n];
  v2 = new double[n];

  for(int r = 0; r < n; r++) {
    v1[r] = 0;
  }

  for(int i = 0; i < k; i++) {
    std::cout << "Iterazione " << i + 1 << std::endl;

    std::cout << "Il vettore vecchio e':" << std::endl;
    print_vector(v1, n);

    for(int r = 0; r < n; r++) {
      v2[r] = b[r];

      for(int c = 0; c < n; c++) {
        if(r == c) continue;

        v2[r] -= A[r][c] * v1[c];
      }

      v2[r] /= A[r][r];
    }

    vec_copy(n, v2, v1);

    std::cout << "Il vettore nuovo e':" << std::endl;
    print_vector(v1, n);
    std::cout << std::endl;
  }

  // ripulisci
  delete v2;

  return v1;
}
```

A questo punto un programma dovrà semplicemente usare le funzioni per il caricamento dell'ambiente di `mat.h` e chiamare la funzione `jacobi()`:

```
int main() {
```

```
  // inizializza ambiente
  unsigned n, k; double** A; double* b;
  init_env(n, k, A, b);

  double* x = jacobi(n, k, A, b);

  std::cout << "Il risultato finale e':" << std::endl;
  print_vector(x, n);
  std::cout << std::endl;

  delete[] x;

  // ripulisci
  clean_env(n, A, b);
  return 0;
}
```

10.2.4 Metodo di Gauss-Seidel

Il metodo di Gauss-Seidel consiste nel riscrivere quanto trovato come:

$$(D-E)x = Fx + b \implies x = (D-E)^{-1}Fx + (D-E)^{-1}b$$

cioè trovare un equazione di punto fisso con:

$$H = (D-E)^{-1}F, \quad c = (D-E)^{-1}b$$

e quindi applicare l'algoritmo:

$$\begin{cases} x^{(0)} \text{ dato} \\ x^{(k+1)} = (D-E)^{-1}Fx^{(k)} + (D-E)^{-1}b \end{cases}$$

Avremo che la matrice di iterazione è:

$$H_{GS} = (D-E)^{-1}F$$

di cui osserviamo che la prima colonna è il vettore di zeri in quanto F ha anch'essa la prima colonna al vettore di zeri.

In ogni caso, notiamo che non è necessario esplicitare H_{GS}, ma ad ogni iterazione basta calcolare il prodotto matrice-vettore:

$$x^{(k)} \to Fx^{(k)}$$

e risolvere col metodo di sostituzione all'indietro il sistema lineare:

$$(D-E)y = Fx^{(k)}$$

Stesso discorso per c, che si trova sempre risolvendo una sola volta, col metodo di sostituzione all'indietro, il sistema lineare:

$$(D-E)c = b$$

10.2.5 Implementazione MATLAB del metodo di Gauss-Seidel

In MATLAB, una possibile implementazione che sfrutta le funzionalità di calcolo matriciale (in particolare la funzione per la sostituzione all'indietro `bck_subst()` che abbiamo già implementato) può essere la seguente:

```
function x_old = seidel(A, b, k, x_old)
    n = height(A);

    if nargin < 4
        % stima iniziale
        x_old = zeros(n, 1);
    end

    % imposta le matrici D E F
    DE = tril(A);
    F = -A + DE;

    % imposta c
    c = fwd_subst(DE, b);

    for j = 1:k
        x_old = fwd_subst(DE, F * x_old) + c;
        x_new = x_old;

        % stampa informazioni
        fprintf("it = %d, res = %.3f\n", j, norm(b - A * x_old));
        disp(x_old);
    end
end
```

10.2.6 Ottimizzazione del metodo di Gauss-Seidel

L'implementazione vista adesso del metodo di Gauss-Seidel non è la più efficiente. Vediamo infatti che, preso:

$$x^{(k+1)} = (D-E)^{-1}Fx^{(k)} + (D-E)^{-1}b$$

e moltiplicando per $D^{-1}(D-E)$ si ha:

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

Esplicitare una dipendenza di $x^{(k+1)}$ da sé stessa potrebbe sembrare poco intuitivo, ma è invece conveniente in quanto ogni elemento di $x^{(k+1)}$ dipenderà dai soli elementi precedenti, cioè si avrà:

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

In questo modo si hanno due vantaggi:

1. Non occorre risolvere sistemi triangolari;
2. Per calcolare $x_i^{(k+1)}$ non si ha bisogno di x_h^k per $h < i$, e quindi si possono sovrascrivere le entrate di $x^{(k)}$, e mantenere un unico vettore.

10.2.7 Implementazione C++ del metodo di Gauss-Seidel

Vediamo come si può usare quest'ultima versione (scalare) del metodo di Gauss-Seidel per realizzarne un'implementazione C++, sfruttando la stessa libreria `mat.h` dell'esempio 10.2.3:

```
double* jacobi(unsigned n, unsigned k, double** A, double* b) {
  // inizializza vettori
  double* v1;
  v1 = new double[n];

  for(int r = 0; r < n; r++) {
    v1[r] = 0;
  }

  for(int i = 0; i < k; i++) {
    std::cout << "Iterazione " << i + 1 << std::endl;

    std::cout << "Il vettore vecchio e':" << std::endl;
    print_vector(v1, n);

    for(int r = 0; r < n; r++) {
      v1[r] = b[r];

      for(int c = 0; c < n; c++) {
        if(r == c) continue;

        v1[r] -= A[r][c] * v1[c];
      }

      v1[r] /= A[r][r];
    }

    std::cout << "Il vettore nuovo e':" << std::endl;
    print_vector(v1, n);
    std::cout << std::endl;
  }

  return v1;
}
```

Come vediamo, la proprietà dimostrata comporta modifiche minime al codice (la semplice rimozione del vettore `v2`, con prodotti scalari che vengono fatti tutti su `v1`).

La realizzazione di un programma che esegue questo algoritmo è ancora analoga all'esempio 10.2.3 (implementazioni complete si trovano in `../matlab/code`).

10.2.8 Jacobi/Gauss-Seidel e matrici di permutazione

Un'ultima osservazione è che sia Jacobi che Gauss-Seidel hanno come condizione che $a_{ii} \neq 0, \forall i$, in quanto altrimenti D diagonale non sarebbe invertibile. Se questa condizione non è soddisfatta, si possono fare delle trasformazioni "indolori" sul sistema per ritrovare la diagonale non nulla (applicare matrici di permutazione e applicare il metodo sul sistema permutato, per trovare poi una soluzione che al limite andrà *de*-permutata).

Ad esempio, si può pensare di applicare Jacobi a:

$$\Pi A x = \Pi b$$

per trovare una qualche permutazione Πx di x.

11 Lezione del 31-03-25

Riprendiamo la trattazione dei metodi iterativi per i sistemi lineari.

11.0.1 Valutazione della convergenza

Avevamo visto che affinché un metodo di punto fisso sia convergente è necessario che la sua matrice di iterazione H in:

$$x = Hx + c$$

sia convergente, cioè abbia norma spettrale $\rho(H) < 1$.

Vediamo un esempio di valutazione di tale matrice sia per il metodo di Jacobi e di Gauss-Siedel applicato ad una certa matrice A, svolgendo esplicitamente tutti i calcoli:

$$A = \begin{pmatrix} 1 & \frac{2}{3} & \frac{4}{9} \\ \frac{2}{3} & 1 & \frac{2}{3} \\ \frac{4}{9} & \frac{2}{3} & 1 \end{pmatrix}$$

- **Jacobi:** avremo la matrice di convergenza:

  $$x = \underbrace{D^{-1}(E+F)}_{H_J} x + D^{-1}b$$

  da cui:

  $$H_J = D^{-1}(E+F) = I \cdot \begin{pmatrix} 0 & \frac{2}{3} & \frac{4}{9} \\ \frac{2}{3} & 0 & \frac{2}{3} \\ \frac{4}{9} & \frac{2}{3} & 0 \end{pmatrix}$$

  Calcoliamo quindi gli autovalori attraverso il polinomio caratteristico:

  $$p(\lambda) = \det\left(\lambda I - D^{-1}(E+F)\right) = \det \begin{pmatrix} \lambda & \frac{2}{3} & \frac{4}{9} \\ \frac{2}{3} & \lambda & \frac{2}{3} \\ \frac{4}{9} & \frac{2}{3} & \lambda \end{pmatrix} = \lambda^3 + \frac{16}{81} \cdot 2 - \frac{16}{81}\lambda - 2 \cdot \frac{4}{9}\lambda$$

  per la regola di Sarrus, quindi:

  $$p(\lambda) = \lambda^3 - \frac{88}{81}\lambda + \frac{32}{81}$$

  Accorgiamoci che $\lambda_1 = \frac{4}{9}$ è radice del polinomio, e quindi dividiamo:

  $$\frac{\lambda^3 - \frac{88}{81}\lambda + \frac{32}{81}}{\lambda - \frac{4}{9}} = \lambda^2 + \frac{4}{9}\lambda - \frac{8}{9}$$

  per cui:

  $$p(\lambda) = \left(\lambda - \frac{4}{9}\right)\left(\lambda^2 + \frac{4}{9}\lambda - \frac{8}{9}\right)$$

  Risolviamo quindi la quadratica:

  $$\lambda_{2,3} = \frac{-\frac{4}{9} \pm \sqrt{\frac{16}{81} + \frac{32}{9}}}{2} = -\frac{2}{9}\left(1 \mp \sqrt{19}\right)$$

  Delle 3 radici trovate si ha che $\lambda_3 = -\frac{2}{9}\left(1 + \sqrt{19}\right) \approx -1.19087$ è in modulo ≥ 0, ergo la condizione di convergenza non è rispettata, e il metodo di Jacobi potrebbe non convergere.

- **Gauss-Seidel:** avremo la matrice di convergenza:

$$x = \underbrace{(D-E)^{-1}F}_{H_{GS}}x + (D-E)^{-1}b$$

da cui:

$$H_{GS} = (D-E)^{-1}F = \begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{4}{9} & \frac{2}{3} & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 & -\frac{2}{3} & -\frac{4}{9} \\ 0 & 0 & -\frac{2}{3} \\ 0 & 0 & 0 \end{pmatrix}$$

che sfruttando quanto avevamo detto sui prodotti per inverse:

$$= \left((D-E)^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad (D-E)^{-1} \begin{pmatrix} -\frac{2}{3} \\ 0 \\ 0 \end{pmatrix} \quad (D-E)^{-1} \begin{pmatrix} -\frac{4}{9} \\ -\frac{2}{3} \\ 0 \end{pmatrix} \right)$$

cioè dobbiamo risolvere 3 sistemi lineari (di cui uno banale) usando il metodo di sostituzione all'indietro. Avremo quindi:

  - Il sistema:

$$\begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{4}{9} & \frac{2}{3} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\frac{2}{3} \\ 0 \\ 0 \end{pmatrix}$$

    che dà:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\frac{2}{3} \\ \frac{4}{9} \\ 0 \end{pmatrix}$$

  - Il sistema:

$$\begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{4}{9} & \frac{2}{3} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\frac{4}{9} \\ -\frac{2}{3} \\ 0 \end{pmatrix}$$

    che dà:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\frac{4}{9} \\ -\frac{10}{27} \\ \frac{4}{9} \end{pmatrix}$$

da cui:

$$H_{GS} = \begin{pmatrix} 0 & -\frac{2}{3} & -\frac{4}{9} \\ 0 & \frac{4}{9} & -\frac{10}{27} \\ 0 & 0 & \frac{4}{9} \end{pmatrix}$$

da cui $\lambda_1 = 0$ e $\lambda_2 = \frac{4}{9}$ con molteplicità $\mu_2 = 2$, quindi tutti gli autovalori a modulo < 1 e la condizione di convergenza è rispettata.

11.0.2 Criteri di convergenza di matrici di iterazione

Cerchiamo un modo più generale per fare ciò che abbiamo fatto nell'ultimo esempio. Vediamo che vale il seguente teorema:

Teorema 11.1: Criteri di convergenza di matrici di iterazione

Se $A \in \mathbb{C}^{n \times n}$ rispetta una delle seguenti condizioni:

1. A è ha predominanza diagonale forte;
2. A è irriducibile e a predominanza diagonale debole;

allora sia il metodo di Jacobi che il metodo di Gauss-Seidel danno iterazioni convergenti applicati ad un sistema $Ax = b$.

1. La dimostrazione del teorema comincia dall'imporre che $a_{ii} \neq 0$, $\forall i = 1, ..., n$. Avevamo visto che questa condizione può essere forzata scambiando A con una qualche permutazione ΠA.

   Abbiamo allora che la condizione è ovvia nel caso di predominanza diagonale forte (1), mentre nel caso di matrice irriducibile a predominanza diagonale debole (2), se esistesse un $a_{ii} = 0$ per qualche i allora applicando la predominanza diagonale debole si avrebbe $a_{ij} = 0$ ($0 \geq \sum_{i \neq j}^{n} a_{ij}$), che sarebbe assurdo (la matrice non può essere singolare). Notiamo che in questo caso nemmeno una matrice di permutazione Π potrebbe portarci a diagonale non negativa.

2. Dimostriamo la convergenza per i due metodi:

   - **Jacobi:** si ha che:

   $$H_J = D^{-1}(E + F) = \begin{pmatrix} 0 & ... & -\frac{a_{ij}}{a_{ii}} \\ ... & 0 & ... \\ -\frac{a_{ij}}{a_{ii}} & ... & 0 \end{pmatrix}$$

   e allora la norma a infinito di H_J sarà:

   $$|H_J|_\infty = \max_{i=1,...,n} \sum_{i \neq j}^{n} \left| \frac{a_{ij}}{a_{ii}} \right|$$

   Se A è a predominanza diagonale forte (1), allora $|a_{ii}| > \sum_{j \neq i}^{n} |a_{ij}|$, e quindi $|H_J|_\infty < 1$ e $\rho(H_j) < 1$.
   Se A è invece irriducibile a predominanza diagonale debole (2), sarà che nuovamente $|H_J|_\infty \leq 1$, quindi gli autovalori di A sono $|\lambda| \leq 1$, ma non esiste $|\lambda| = 1$ in quanto starebbe sul bordo di un cerchio di Gershgorin, quindi di tutti, ma esiste almeno un cerchio $\mathcal{F}_i(A)$ con raggio $r < 1$.

   - **Gauss-Seidel:** supponiamo che esista λ^* autovalore di H_{GS}:

   $$H_{GS} = (D - E)^{-1} F$$

   tale che $|\lambda^*| \geq 1$. Allora dovrà valere:

   $$0 = \det(\lambda^* I - H_{GS}) = \det\left(\lambda^* I - (D - E)^{-1} F\right) = \det\left((D - E)^{-1} \left(\lambda (D - E) - F\right)\right)$$

   che per Binet-Cauchy dà:

   $$0 = \det\left((D - E)^{-1}\right) \cdot \det\left(\lambda^* (D - E) - F\right)$$

Il primo termine è chiaramente $\neq 0$, e quindi si considera:

$$0 = \det\left(\lambda^*(D-E)-F\right) = (\lambda^*)^n \det\left(D-E-(\lambda^*)^{-1}F\right)$$

cioè:

$$0 = \det\left(D-E-(\lambda^*)^{-1}F\right)$$

La matrice che troviamo è quella dove gli elementi sopra la diagonale hanno modulo $\leq$ ai corrispondenti nella matrice A. Se A è dominante diagonale forte o dominante diagonale debole, quindi, conserva tale proprietà, e conserva anche l'irriducibilità. Quindi, necessariamente, è non singolare (altrimenti anche A sarebbe singolare). Allora, visto che per definizione H_{GS} deve essere singolare (prima colonna nulla), l'ipotesi $|\lambda^*| \geq 1$ è assurda, cioè deve essere $|\lambda^*| < 1$.

□

11.1 Sistemi rettangolari

Vediamo quindi come risolvere sistemi **rettangolari**, cioè del tipo $Ax = b$ con $A \in \mathbb{C}^{m\times n}$ rettangolare. In questo caso si avranno m equazioni in n incognite, quindi con $x \in \mathbb{C}^n$ e $b \in \mathbb{C}^m$. Supponiamo poi che A sia di rango massimo, ovvero $\text{rank}(A) = \min(m,n)$.

Scopriamo che anche in questo scenario si può applicare l'eliminazione di Gauss, e nel caso $m < n$, si può immaginare di rendere il sistema quadrato aggiungendo $n-m$ equazioni fittizie della forma $0 = 0$, lasciando $n-m$ parametri liberi:

$$\begin{pmatrix} 1 & ... & ... & ... \\ 0 & 1 & ... & ... \\ 0 & ... & 1 & ... \\ 0 & ... & ... & 0 \\ \vdots & & & \vdots \\ 0 & ... & ... & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_m \\ x_{m+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

Diciamo che questo tipo di sistemi si dicono **sottodeterminati**.

Caso più interessante sarà quello dei sistemi **sovradeterminati**, cioè dove $n < m$ e si hanno più equazioni che incognite. In questo caso l'applicazione dell'eliminazione di Gauss porta ad n equazioni determinate, e $m-n$ equazioni della forma $0 = ?$ dove $?$ sono i termini noti che si formano nelle ultime $m-n$ righe:

$$\begin{pmatrix} 1 & ... & ... \\ 0 & 1 & ... \\ 0 & ... & 1 \\ 0 & ... & 0 \\ \vdots & & \vdots \\ 0 & ... & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \\ b_{n+1} = ?_1 \\ \vdots \\ b_m = ?_{m-n+1} \end{pmatrix}$$

Se esiste almeno un termine noto $? \neq 0$, allora il sistema è impossibile (ed è questo il caso tipico). Questa affermazione è essenzialmente il teorema di Rouchè-Capelli, dove si impone:

$$\text{rank}(A|b) = \text{rank}(A)$$

Il problema che ci poniamo in questi casi è quello di minimizzare il residuo, cioè posto:

$$x - Ab = r(x)$$

cercare di risolvere il problema di ottimizzazione:

$$r^* = \min_{x \in \mathbb{C}^n} |b - Ax|_2 = \min_{x \in \mathbb{C}^n} |r(x)|_2$$

Questo tipo di problema viene detto **problema dei minimi quadrati**. Osserviamo che il problema è equivalente a risolvere:

$$(r^*)^2 = \min_{x \in \mathbb{C}^n} |b - Ax|_2^2 = \min_{x \in \mathbb{C}^n} |r(x)|_2^2$$

Cioè ci possiamo liberare della radice della norma quadratica.

Potremo innanzitutto chiederci come trovare il punto x^* che dà r^*, o in principio se questo problema ammette soluzione. Facciamo allora un esempio del caso $x \in \mathbb{R}^n$ Considereremo la funzione:

$$\psi(x) = |b - Ax|_2^2 = (b - Ax)^T (b - Ax) : \mathbb{R}^n \to \mathbb{R}$$

questa funzione è convessa, e quindi il punto x^* è quello che soddisfa $\nabla \psi(x) = 0$ (gradiente nullo). Vediamo allora le derivate parziali, osservando che:

$$|b - Ax|_2^2 = |r(x)|_2^2 = \sum_{i=1}^{m} r_i(x)^2$$

e:

$$r_i(x)^2 = \left(b_i - \sum_j^n a_{ij} x_j \right)^2$$

Calcoliamo quindi la derivata parziale vera e propria rispetto alla s-esima componente:

$$\frac{\partial}{\partial x_s} \left(r_i(x)^2 \right) = 2 \left(b_i - \sum_{j=1}^{n} a_{ij} x_j \right) (-a_{is})$$

da cui:

$$\frac{\partial}{\partial x_s} (\psi(x)) = \sum_{i=1}^{n} \frac{\partial}{\partial x_s} \left(r_i(x)^2 \right) = 2 \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{is} a_{ij} x_j - a_{is} b_i \right)$$

$$= 2 \left(\sum_{i=1}^{m} \sum_{j=1}^{n} a_{is} a_{ij} x_j - \sum_{i=1}^{n} a_{is} b_i \right) = 2 \left(\sum_{i=1}^{m} a_{is} \sum_{j=1}^{n} a_{ij} x_j - \sum_{i=1}^{m} a_{is} b_i \right)$$

$$= 2 \left(\left(A^T A x \right)_s - \left(A^T b \right)_s \right) = 2 \left(A^T A x - A^T b \right)_s$$

dove l's al pedice indica che si prende la s-esima riga.

Si ha quindi che:

$$\nabla \psi(x) = 0 \Leftrightarrow A^T A x - A^T b = 0 \Leftrightarrow A^T A x = A^T b$$

e quindi il sistema da risolvere per i punti di minimo di $\psi(x)$, cioè le soluzioni del problema dei minimi quadrati, è:

$$A^T A x = A^T b$$

detto **sistema delle equazioni normali**.

12 Lezione del 04-04-25

Riprendiamo il discorso dei problemi ai minimi quadrati.

12.1 Esistenza delle soluzioni al sistema delle equazioni normali

Avevamo detto che di sistemi $Ax = b$ con $A \in \mathbb{C}^{m \times n}$, $m > n$, cioè *sovradeterminati*, che se la soluzione di $Ax = b$ non esiste, ergo $\text{rank}(A|b) > \text{rank}(A)$, potrebbe essere interessante cercare di risolvere il problema di ottimizzazione:

$$\min_{x \in \mathbb{C}^n} |Ax - b|_2 = \min_{x \in \mathbb{C}^n} |b - Ax|_2$$

dove il punto p_1 di minimo è lo stesso di quello del problema:

$$\min_{x \in \mathbb{C}^n} |Ax - b|_2^2$$

Avevamo quindi definito la funzione ϕ, assunta per semplicità $\psi : \mathbb{R}^n \to [0, +\infty)$:

$$\psi(x) = |b - Ax|_2^2 = |r(x)|_2^2 = \sum_{i=1}^{n} r_i(x)^2$$

A questo punto il minimo sarà semplicemente nel punto:

$$\begin{pmatrix} \frac{\partial \psi}{\partial x_1}(x) \\ \vdots \\ \frac{\partial \psi}{\partial x_n}(x) \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

da cui avevamo visto ricavamo il sistema **lineare** (in quanto ψ è quadratica e quindi deriva ad una lineare):

$$A^T A x = A^T b$$

Non lo abbiamo dimostrato formalmente, ma possiamo dire che lo stesso vale nel campo complesso, cioè con $x \in \mathbb{C}^n$:

$$A^H A x = A^H b$$

prese le hermitiane invece delle trasposte.

Vediamo quindi che vale il seguente teorema:

Teorema 12.1: Esistenza delle soluzioni dei problemi ai minimi quadrati

Il sistema $A^T A x = A^T b$ ammette soluzione $\forall A \in \mathbb{R}^{m \times n}$ e $\forall b \in \mathbb{C}^m$ (1). Inoltre, la soluzione è unica se e solo se $\text{rank}(A) = n$, $(m > n)$ (2).

Dimostriamo le due tesi in ordine.

1. La prima tesi si dimostra verificando che $A^T b \in I_m(A^T A)$ con I_m *immagine* o **spazio delle colonne** di $A^T A$, cioè verificando che è rispettata la condizione:

   $$\text{rank}(A^T A | A^T b) = \text{rank}(A^T A)$$

   di Rouché-Capelli, per cui esiste una soluzione. Abbiamo quindi:

   $$I_m(A^T A) = \{y \in \mathbb{C}^n \ : \ y = A^T A z, \quad z \in \mathbb{C}^n\}$$

Il caso buono è $A^T b \in I_m(A)$, cioè:

$$I_m(A) = \{y \in \mathbb{C}^n \ : \ y = Az, \quad z \in \mathbb{C}^n\}$$

così che:

$$A^T b = A^T A z \in I_m(A)$$

Questo però non è sempre vero. Possiamo allora dire che uno spazio vettoriale è sempre dato dalla somma dello spazio delle colonne di una trasformazione A e il corrispettivo spazio perpendicolare:

$$\mathbb{C}^m = I_m(A) \oplus I_m(A)^\perp$$

con lo spazio perpendicolare allo spazio delle colonne definito come:

$$I_m(A)^\perp = \{y : y^H z = 0, \quad \forall z \in I_m(A)\}$$

Potremo allora riscrivere b come la somma delle componenti *parallela* e perpendicolare:

$$b \in \mathbb{C}^m \implies b = b_1 + b_2, \quad \begin{cases} b_1 \in I_m(A) \\ b_2 \in I_m(A)^\perp \end{cases}$$

Prendiamo allora $A^T b$ come:

$$A^T b = A^T b_1 + A^T b_2$$

Si ha che $A^T b_1 \in I_m(A^T A)$ da quanto avevamo detto prima, mentre riguardo ad $A^T b_2$ si può dire:

$$A^T b_2 = \begin{pmatrix} a_1^T b_2 \\ \vdots \\ a_m^T b_2 \end{pmatrix} = 0$$

perchè $b_2 \in I_m(A)^\perp$ e le colonne di A stanno in $I_m(A)$, quindi la condizione di Rouché-Capelli è soddisfatta e la soluzione esiste. □

2. Per l'unicità, vogliamo verificare che $\det(A^T A) \neq 0$ soddisfatta la proprietà $\text{rank}(A) = n$. Abbiamo allora che dal teorema di Binet-Cauchy *generalizzato*:

$$\det(A^T A) = \sum_{[j]} \det(A^T_{[j]}) \cdot \det(A_{[j]}) = \sum_{[j]} |\det(A_{[j]})|^2$$

dove la sommatoria varia su tutte le possibili scelte di n indici in $\{1, ..., m\}$. Vogliamo allora mostrare che $\det(A^T A) \geq 0$, cioè che $\exists$ almeno una sottomatrice $A_{[j]}$ tale che $\det(A_{[j]}) \neq 0$. Questa esisterà sempre se $\text{rank}(A) = n$, e quindi anche l'ultima tesi del teorema è soddisfatta. □

Vediamo due esempi pratici per osservare il teorema in azione.

1. Prendiamo il sistema:

$$\begin{cases} 2x_1 - x_2 = 1 \\ -x_1 + x_2 = 0 \\ x_1 + 2x_2 = 1 \end{cases}$$

con $m = 3$ equazioni in $n = 2$ incognite. Avremo che le matrici A e b, e l'incognita x, saranno:

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 1 \\ 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

per il sistema $Ax = b$. Vediamo che il rango di A è 2, quindi dal teorema appena dimostrato la soluzione sarà unica. Calcoliamo allora le matrici A^TA e A^tb per il sistema $A^TAx = A^Tb$:

$$A^TA = \begin{pmatrix} 6 & -1 \\ -1 & 6 \end{pmatrix}, \quad A^Tb = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

Avremo quindi il sistema:

$$\begin{cases} 6x_1 - x_2 = 3 \\ -x_1 + 6x_2 = 1 \end{cases}$$

da cui:

$$x^* = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \frac{19}{35} \\ \frac{9}{35} \end{pmatrix}$$

Che è la soluzione ottima. Possiamo quindi calcolare la norma di residuo:

$$r = b - Ax^* = b - A \cdot \begin{pmatrix} \frac{19}{35} \\ \frac{9}{35} \end{pmatrix} = \begin{pmatrix} \frac{6}{35} \\ \frac{10}{35} \\ -\frac{2}{35} \end{pmatrix}$$

da cui ottieniamo uno scarto di ≈ 0.3381 dal valore ideale.

2. Un problema simmetrico al precedente potrebbe essere quello di trovare per quali parametri una matrice parametrizzata rispetta il teorema, cioè ammette una soluzione unica. Prendiamo ad esempio il sistema:

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & 1 \\ 0 & 0 & \alpha \\ \beta & 0 & 0 \end{pmatrix}}_{A} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}}_{b}$$

con $\alpha, \beta \in \mathbb{R}$. Iniziamo imponendo l'antitesi del teorema cioè:

$$\text{rank}(A) < n \;\Leftrightarrow\; \exists A_{[j]} \in \mathbb{R}^{3\times 3} \text{ non invertibile}$$

con $A_{[j]}$ le sottomatrici quadrate 3×3 prese dal teorema di Binet-Cauchy. Consideriamo allora tutte le possibili $A_{[j]}$ coi loro determinanti:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 1 & \alpha & 1 \\ \beta & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & \alpha \\ \beta & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & \alpha & 1 \\ 0 & 0 & \alpha \\ \beta & 0 & 0 \end{pmatrix}$$
$$\downarrow \qquad \downarrow \qquad \downarrow \qquad \downarrow$$
$$\alpha(\alpha - 1) \qquad \beta(1 - \alpha) \qquad \alpha\beta \qquad \alpha^2\beta$$

da cui avere tutti determinanti nulli singifica soddisfare il sistema (non lineare):

$$\begin{cases} \alpha(\alpha - 1) = 0 \\ \beta(1 - \alpha) = 0 \\ \alpha\beta = 0 \\ \alpha^2\beta = 0 \end{cases}$$

con soluzioni $(\alpha, \beta) = (0, 0)$ o $(\alpha, \beta) = (1, 0)$. Si ha quindi che per queste due combinazioni di valori di α e β il sistema di partenza non ammette soluzione ottima unica.

12.1.1 Considerazioni di complessità del sistema delle equazioni normali

Valutiamo la complessità del sistema visto finora. Dobbiamo:

1. Calcolare $A^H A$, che ha complessità $O(mn^2)$ (da una $(n \times m)(m \times n)$);
2. Calcolare $A^H b$, che ha complessità $O(mn)$ (da una $(n \times m)(m \times 1)$);
3. Risolvere il sistema $A^H Ax = A^H b$, che ha complessità $O(mn^2 + n^3) \sim O(n^3)$.

Vediamo che il problema non è tanto la complessità, ma il fatto che $A^T A$ ha spesso un numero di condizionamento piuttosto alto, ad esempio nel caso $m = n$ si ha:

$$\mu(A^T A) = \mu(A)^2$$

Se $\mu(A^T A)$ ($< 10^k$ con k moderato), questo è fastidioso in quanto abbatte sostanzialmente la precisione. Conviene quindi applicare un metodo alternativo.

12.2 Metodo QR per problemi ai minimi quadrati

Vediamo quindi un metodo basato sulla fattorizzazione QR.

12.2.1 Fattorizzazione QR

La **fattorizzazione QR** è un tipo di fattorizzazione matriciale, come lo era la *fattorizzazione LU* che abbiamo già visto (8.3.4).

Diamone quindi la definizione:

Definizione 12.1: Fattorizzazione LU

Data $A \in \mathbb{C}^{m \times n}$, con $m \geq n$, una fattorizzazione QR di A è una coppia di matrici (Q, R) tali che: $A = QR$, con $Q \in \mathbb{C}^{m \times m}$ unitaria ($Q^H Q = QQ^H = I$), e $R \in \mathbb{C}^{m \times m}$ della forma triangolare superiore rettangolare (con un certo numero di righe a zero sotto il triangolo), cioè esprimibile come:

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

con $R_1 \in \mathbb{C}^{n \times n}$ triangolare superiore.

Varrà allora il teorema:

Teorema 12.2: Esistenza della fattorizzazione QR

Per ogni matrice $\exists$ sempre una fattorizzazione QR, ma non è mai unica.

Questo viene dal fatto che data $A = QR$, si potrebbe prendere $D \in \mathbb{R}^{m \times m}$ diagonale con $|d_j| = 1$ per $j = 1, ..., m$, e dire:

$$A = Q_2 R_2, \quad \begin{cases} Q_2 = QD \\ R_2 = D^{-1} R \end{cases}$$

in quanto:

$$A = QR = \underbrace{QD}_{Q_2} \cdot \underbrace{D^{-1} R}_{R_2}$$

Una proprietà di rilievo è che la norma 2 e la norma di Frobenius sono invarianti per moltiplicazioni con matrici unitarie, ovvero:

$$|A| = |QA| = |AQ| = |Q_1 A Q_2|$$

con norma $|\cdot|$ 2 o di Frobenius.

Dimostriamo per le due norme:

- **Norma 2:** questo si ha prendendo la definizione di norma 2:

  $$|A|_2 = \sqrt{\rho(A^H A)}$$

  Allora la norma di AQ, ad esempio, sarà:

  $$|AQ|_2 = \sqrt{\rho((AQ)^H AQ)} = \sqrt{\rho(Q^H A^H AQ)}$$

  $Q^H A^H AQ$ è simile a $A^H A$ in quanto dalla definizione di unarietà di Q, si può intendere $Q^H = Q^{-1}$ e quindi prendere la forma come un applicazione di matrice di trasformazione, da cui la tesi per la norma 2. □

- **Norma di Frobenius:** questo si ha prendendo la definizione di norma di Frobenius:

  $$|A|_F = \sqrt{\text{trace}(A^H A)}$$

  Allora la norma di AQ, ad esempio, sarà:

  $$|AQ|_F = \sqrt{\text{trace}(Q^H A^H AQ)}$$

  da cui come sopra. □

Nei problemi ai minimi quadrati, la fattorizzazione QR è utile in quanto se $A = QR$:

$$\min_{x \in \mathbb{C}^n} |b - Ax|_2 = \min_{x \in \mathbb{C}^n} |b - QRx|_2 = \min_{x \in \mathbb{C}} |Q^H b - Rx|_2$$

moltiplicando per Q^H all'ultimo passaggio (che preserva la norma, come abbiamo appena dimostrato).

Chiamato $Q^H b = c$, osserviamo quindi di poter prendere:

$$\psi(x) = |cx - Rx|_2^2$$

che preso:

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

diventa:

$$\psi(x) = \left| \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} - \begin{pmatrix} R_1 \\ 0 \end{pmatrix} x \right|_2^2 = \left| \begin{pmatrix} c_1 - R_1 x \\ c_2 \end{pmatrix} \right|_2^2 = |c_1 - R_1 x|_2^2 + |c_2|_2^2$$

l'unico termine su cui abbiamo controllo è il primo, e possiamo quindi trascurare $|c_2|_2^2$. Prendiamo allora:

$$\psi'(x) = |c_1 - R_1 x|_2^2$$

che rappresenta un sistema lineare $n \times n$, dove bastera scegliere x tale che:

$$R_1 x = c$$

così da trovare 0 e ottenere il valore di ψ' (e quindi di ψ) minore possibile. Il vantaggio aggiunto è che R_1 è triangolare superiore, e quindi si può calcolare:

$$x = R_1^{-1} c$$

per sostituzione all'indietro.

Riassumendo, quindi, abbiamo ottenuto il metodo:

1. Si calcola la fattorizzazione $A = QR$;
2. Si calcola $Q^H b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$, che ha costo $O(nm)$ (non è $O(m^2 n)$, per via della struttura di Q);
3. Si risolve $R_1 x = c_1$, che è un sistema lineare triangolare superiore $n \times n$, quindi ha costo $O(n^2)$ per sostituzione all'indietro.

L'unica incognita è allora il costo della fattorizzazione, che anticipiamo è $O(mn^2)$.

12.2.2 Matrici di Householder

Una proprietà rilevante è che dato un vettore $v \in \mathbb{C}^m$ esiste sempre una matrice unitaria H_v tale che:

$$H_v \cdot v = c \cdot e_1, \quad c \in \mathbb{C}$$

per e_1 primo elemento della base canonica. Dato che la moltiplicazione per matrici unitarie non cambia la norma si ha:

$$|c| = |v|_2$$

Per creare tale matrice, scegliamo $\tilde{v}$:

$$\tilde{v} = v \pm |v|_2 \cdot e_1 = \begin{pmatrix} v_1 \pm |v|_2 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}$$

H_v dovrà quindi essere scelta come:

$$H_v = I - 2\frac{\tilde{v}\tilde{v}^H}{|\tilde{v}|_2^2}$$

dove il termine $\frac{\tilde{v}\tilde{v}^H}{|\tilde{v}|_2^2}$ rappresenta una proiezione sulla retta tracciata dal vettore $\tilde{v}$, e quindi H_v uguale alla *matrice di riflessione* sull'iperpiano ortogonale a $\tilde{v}$. Visto che abbiamo scelto $\tilde{v}$ perchè fosse effettivamente *"equidistante"* sia da v che da e_1, otteniamo una matrice che porta v esattamente sulla base e_1.

Le matrici costruite in questo modo vengono anche dette **Matrici di Householder** associate al vettore v.

Possiamo scrivere una funzione MATLAB, che ci tornerà utile a breve, per il calcolo della matrice di Householder relativa a un certo vettore:

```
function H = householder(v)
    H = eye(length(v));
    vt = v;

    if vt(1) < 0
        vt(1) = vt(1) - norm(v); % si sottrae
    else
        vt(1) = vt(1) + norm(v);
    end

    H = H - 2 * vt * vt' / norm(vt)^2;
end
```

Vediamo inoltre come si è scelto il segno della norma aggiunta a `vt` ($\tilde{v}$) in modo da ridurre gli errori di cancellazione.

12.2.3 Calcolo della fattorizzazione QR

Analogamente al metodo di Gauss, per la fattorizzazione QR si moltiplica a sinistra per matrici unitarie che *"sistemano"* gli elementi sotto la diagonale.

Sfruttando la proprietà delle matrici di Householder, agiamo come segue:

$$H_1 = H_{a_1} = \text{matrice di Householder associata ad } a_1\text{, prima colonna di } A$$

diciamo:

$$H_1 \begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix} = \begin{pmatrix} ? & \dots \\ 0 & \dots \\ \vdots & \dots \\ 0 & \dots \end{pmatrix}$$

a questo punto basterà prendere H_2 per *"sistemare"* gli elementi della sottomatrice di coda successiva:

$$H_2 = \left(\begin{array}{c|ccc} 1 & 0 & \dots & 0 \\ \hline 0 & & & \\ \vdots & & \tilde{H}_2 & \\ 0 & & & \end{array}\right)$$

e così via, introducendo di volta in volta più elementi della diagonale in testa. Si potrà quindi definire la matrice R come:

$$R = H_n \dots H_2 H_1 A$$

dove il prodotto delle inverse (quindi hermitiane) delle matrici unitarie H_i rappresenterà la matrice Q:

$$Q = H_1^H H_2^H ... H_n^H = H_1 H_2 ... H_n$$

visto che le matrici di Householder sono hermitiane, dato riportato solo adesso per tenere il lettore sulle spine.

12.2.4 Ottimizzazioni della fattorizzazione QR

Osserviamo che per avere costo $O(mn^2)$ (ed anche costo $O(mn)$ per $Q^H b$) si sfrutta il fatto che H_j è del tipo *identità + rango 1*, cioè:

$$H_j = I - \alpha\beta^T$$

dove gli α e β vengono presi qui a scopo di esempio, risulterebbero dalla forma della matrice di Householder cha abbiamo visto alla 12.2.2. La parte importante è che il prodotto per una matrice generica B assume la forma:

$$H_j \cdot B = B - \alpha \cdot \beta^T \cdot B$$

dove l'ultimo termine è il prodotto di una colonna per il prodotto di una riga per B. Quindi, nel caso B sia un vettore b, si hanno solo prodotti scalari di costo m, ergo:

$$H_j b \to O(m)$$

Se prendiamo il prodotto per tutte le matrici H_j, cioè per Q, abbiamo che questa cosa si ripete n volte e quindi il costo è $O(mn)$.

$$\implies H_n \cdot H_{n-1} \cdot ... \cdot H_1 b \to O(mn)$$

il costo $O(mn^2)$ a questo punto è immediato dal fatto che abbiamo semplicemente una dimensione n in più dalla matrice A.

12.2.5 Osservazioni ulteriori sulla fattorizzazione QR

Osserviamo poi che nel caso $m = n$ si può sfruttare la fattorizzazione QR per risolvere sistemi lineari, con costo $O(\frac{4}{3}n^3)$ per la fattorizzazione, il doppio rispetto a Gauss, ma solitamente più stabile.

Osserviamo infine che se $A \in \mathbb{R}^{m \times n}$ si può restare nei reali ed avere $Q \in \mathbb{R}^{m \times m}$, $Q^T Q = I$, ecc...

12.2.6 Implementazione MATLAB della decomposizione QR

Riportiamo l'implementazione MATLAB di questo algoritmo:

```
function [Q, A] = qr_decomp(A)
    n = width(A);
    m = height(A);
    Q = eye(m); % accumulatore

    for i = 1:n
        Hi = householder(A(i:end, i));
        Hi = blkdiag(eye(i - 1), Hi);
        A = Hi * A;
        Q = Q * Hi;
    end
end
```

Dove la funzione `blkdiag` ci permette di realizzare la forma a blocchi diagonale, con l'identità in testa, che la H_j assume ad ogni passaggio dell'algoritmo.

12.2.7 Implementazione MATLAB ottimizzata della decomposizione QR

Osserviamo quindi la versione della fattorizzazione QR, implementata in MATLAB, che effettua le ottimizzazioni riportate nella sezione 12.2.4:

```
function [Q, A] = opt_qr_decomp(A)
    function [a, b] = householder_vec(v)
        vt = v;

        if vt(1) < 0
            vt(1) = vt(1) - norm(v); % si sottrae
        else
            vt(1) = vt(1) + norm(v);
        end

        a = vt;
        b = 2 * vt / norm(vt)^2;
    end

    % applicazione a sinistra
    function A = rank_one_l(A, a, b, i)
        A(i:end, :) = A(i:end, :) ...
                        - a * b' * A(i:end, :);
    end

    % applicazione a destra
    function A = rank_one_r(A, a, b, i)
        A(:, i:end) = A(:, i:end) ...
                        - (A(:, i:end) * a) * b';
    end

    n = width(A);
    m = height(A);

    Q = eye(m);

    for i = 1:n
        [a, b] = householder_vec(A(i:end, i));
        A = rank_one_l(A, a, b, i);
        Q = rank_one_r(Q, a, b, i);
    end
end
```

dove la matrice di Householder viene mantenuta in due vettori, e si sono definite le funzioni `rank_one_l()` e `rank_one_r()` per l'aggiornamento di rango uno, rispettivamente a sinistra e a destra (che nient'altro sarebbe se non la moltiplicazione per la matrice di Householder).

12.2.8 Implementazione C++ della decomposizione QR

L'ultima versione ottimizzata della decomposizione QR si presta particolarmente bene all'implementazione in linguaggi di livello più basso.

Potremmo infatti avere la funzione, del tutto analoga alla scorsa implementazione MATLAB (se non per il fatto che si assume la matrice A sia quadrata):

```
void qr_decomp(double** A, double** Q, unsigned n) {
  // inizializza vettori a, b
  double* a, *b;
  a = new double[n];
  b = new double[n];
```

```
  for(unsigned i = 0; i < n; i++) {
    std::cout << "Passaggio " << i << std::endl;

    householder_vec(A, i, n, a, b);

    std::cout << "I vettori di Householder sono:" << std::endl;
    std::cout << "a:" << std::endl;
    print_vector(a, n - i);
    std::cout << "b:" << std::endl;
    print_vector(b, n - i);

    rank_one_l(A, i, n, a, b);
    rank_one_r(Q, i, n, a, b);

    std::cout << std::endl;
  }

  // ripulisci
  delete[] a;
  delete[] b;
}
```

dove la `householder_vec()` è anch'essa simile alla versione MATLAB:

```
void householder_vec(double** A, unsigned i, unsigned n, double* a, double
    * b) {
  double sqr_norm = 0;

  // prendi la i-esima colonna di A
  for(unsigned j = i; j < n; j++) {
    a[j - i] = A[j][i];
    sqr_norm += a[j - i] * a[j - i];
  }

  double norm = std::sqrt(sqr_norm);
  sqr_norm -= a[0] * a[0];

  if(a[0] < 0) {
    a[0] -= norm;
  } else {
    a[0] += norm;
  }

  sqr_norm += a[0] * a[0];

  for(unsigned j = 0; j < n - i; j++) {
    b[j] = 2 * a[j] / sqr_norm;
  }
}
```

se non per una probabile ottimizzazione che si potrebbe fare riguardo alle radici, che però ignoriamo.

A questo punto si implementano le `rank_one_l()` e `rank_one_r()` srotolando le moltiplicazioni, ed evidenziando quindi il costo delle applicazioni delle matrici di rango uno, che è effettivamente $O(mn^2)$:

```
// applicazione a sinistra
void rank_one_l(double** A, unsigned i, unsigned n, double* a, double* b)
    {
  // A(i:end, :) = A(i:end, :) - a * b' * A(i:end, :);
```

```
  for(unsigned c = 0; c < n; c++) {
    double sum = 0;

    for(unsigned r = i; r < n; r++) {
      sum += b[r - i] * A[r][c];
    }

    for(unsigned r = i; r < n; r++) {
      A[r][c] -= a[r - i] * sum;
    }
  }
}

// applicazione a destra
void rank_one_r(double** A, unsigned i, unsigned n, double* a, double* b)
    {
  // A(i:end, :) = A(i:end, :) - (A(i:end, :) * a) * b';
  for(unsigned r = 0; r < n; r++) {
    double sum = 0;

    for(unsigned c = i; c < n; c++) {
      sum += A[r][c] * a[c - i];
    }

    for(unsigned c = i; c < n; c++) {
      A[r][c] -= sum * b[c - i];
    }
  }
}
```

L'algoritmo è dunque completo, e realizza usando solo semplici cicli di complessità al massimo quadratica/cubica la decomposizione QR. Ricordiamo che il codice visto finora è disponibile nella directory `/code` degli appunti del corso, e in particolare queste implementazioni di livello più basso in C++ sono disponibili in `/code/cpp`.

12.2.9 Implementazione MATLAB di un risolutore ai minimi quadrati

Abbiamo quindi tutti gli elementi necessari per realizzare un semplice risolutore che implementi ciò che MATLAB fa già naturalmente quando gli si pone l'istruzione `A \ b` con il numero di righe di A maggiore del numero di colonne, cioè risolvere un sistema sovradeterminato usando i minimi quadrati.

Prendiamo quindi la serie di passaggi descritta in 12.2.1 ed implementiamo un algoritmo che li effettui:

```
function x = least_squares(A, b)
    [Q, R] = qr_decomp(A);
    c = Q' * b;

    % trova righe a zero
    tol = 1e-12;
    R_diag = abs(diag(R));
    r = find(R_diag > tol, 1, 'last');

    R_trunc = R(1:r, 1:r);
    c_trunc = c(1:r);

    % risolvi il sistema superiore
    x1 = bck_subst(R_trunc, c_trunc);

```

```
    % reinserisci gli zeri
    x = [x1; zeros(size(R,2) - r, 1)];
end
```

dove notiamo che dobbiamo troncare sia la matrice R che il vettore c ai soli elementi non nulli (cosa fatta controllando gli elementi non nulli della diagonale), in quanto la `bck_subst()` non si comporta bene altrimenti.

Potremo quindi usare l'algoritmo per risolvere problemi ai minimi quadrati come segue:

```
% dati A, b noti
x = least_squares(A, b);
% x minimizza lo scarto quadratico
```

13 Lezione del 07-04-25

13.1 Interpolazione di funzioni

Poniamo di avere una qualche funzione $f(x) : \mathbb{C} \to \mathbb{C}$ (in realtà spesso prenderemo senza ledere alla generalità della trattazione $f(x) : \mathbb{R} \to \mathbb{R}$) che non conosciamo esplicitamente ma di cui possiamo solo calcolare il valore $y_0, ..., y_k$ in determinati punti $x_0, ..., x_k$ non necessariamente equspaziati fra di loro. Chiamiamo questi punti **nodi**. Il problema sarà allora trovare un'*approssimazione* di $f(x)$ **interpolando** i punti dati.

- Un approccio valido quando si hanno pochi punti (k) è considerare un polinomio di grado k che passi per tutti questi, sperando che si raggiunga un'approssimazione sufficientemente precisa nei punti esterni ai nodi. Questo è l'approccio che vedremo della sezione 13.1.1 in poi;
- Un altro approccio, quando si hanno molti punti, è trovare una funzione particolarmente semplice (solitamente una retta) che minimizzi la distanza da questi. E' questo ad esempio il caso della *regressione lineare*. Questo è l'approccio dell'*approssimazione ai minimi quadrati* che vedremo in 15.2.

13.1.1 Interpolazione polinomiale

Vogliamo quindi trovare un polinomio $p(x)$ tale che:

$$p(x_i) = y_i, \quad \forall i \in \{0, ..., k\}$$

Quello che facciamo è porre il polinomio come:

$$p(x) = a_0 + a_1 x + ...a_k x^k$$

e impostare il sistema:

$$\begin{cases} a_0 + a_1 x_0 + ... + a_k x_0^k = y_0 \\ \vdots \\ a_0 + a_1 x_k + ... + a_k x_k^k = y_k \end{cases}$$

che è lineare e ha come incognita il vettore dei coefficienti del polinomio cercato.

Notiamo di aver scelto come base la base canonica dei polinomi:

$$B_{\text{canonica}} = \left\{1, x, x^2, ..., x^k\right\}$$

Avremmo potuto scegliere qualsiasi altra base dello spazio dei polinomi $\mathbb{R}[x]$, e vedremo infatti anche questo caso.

Riscrivendo quindi il sistema di cui sopra in forma matriciale si ha:

$$\underbrace{\begin{pmatrix} 1 & x_0 & \dots & x_0^k \\ \vdots & & & \vdots \\ 1 & x_k & \dots & x_k^k \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{pmatrix}}_{a} = \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_k \end{pmatrix}}_{y}$$

dove la matrice V viene detta **matrice di Vandermonde**.

Potrebe interessarci capire quando questo sistema ammette soluzione. Ad esempio, è impossibile che esista $i \neq j$ tali che $x_i = x_j$, in quanto questo renderebbe il sistema singolare (due righe uguali $\implies \det(V) = 0$). Questo caso non è desiderabile in quanto potrebbero esserci più soluzioni (se anche $y_i = y_j$) o nessuna (se $y_i \neq y_j$).

Possiamo allora enunciare il teorema:

Teorema 13.1: Determinante della matrice di Vandermonde

Il determinante della matrice di Vandermonde è:

$$\det(V) = \prod_{0 \leq i < j \leq k} (x_j - x_i)$$

Possiamo dare una dimostrazione immediata di questo risultato ricordando che si può aggiungere un multiplo scalare di una colonna ad un altra colonna senza cambiare il determinante. Prendiamo allora la matrice nella forma più completa:

$$V = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^k \\ 1 & x_1 & x_1^2 & \dots & x_1^k \\ 1 & x_2 & x_2^2 & \dots & x_2^k \\ \vdots & & & & \vdots \\ 1 & x_k & x_k^2 & \dots & x_k^k \end{pmatrix}$$

Sottraiamo quindi ad ogni colonna dopo la prima il prodotto della colonna precedente per x_0, cancellando effettivamente la prima riga:

$$\rightarrow \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & x_1 - x_0 & x_1^2 - x_1 x_0 & \dots & x_1^k - x_1^{k-1} x_0 \\ 1 & x_2 - x_0 & x_2^2 - x_2 x_0 & \dots & x_2^k - x_1^{k-1} x_0 \\ \vdots & & & & \vdots \\ 1 & x_k - x_0 & x_k^2 - x_k x_0 & \dots & x_k^k - x_1^{k-1} x_0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & x_1 - x_0 & x_1(x_1 - x_0) & \dots & x_1^{k-1}(x_1 - x_0) \\ 1 & x_2 - x_0 & x_2(x_2 - x_0) & \dots & x_2^{k-1}(x_2 - x_0) \\ \vdots & & & & \vdots \\ 1 & x_k - x_0 & x_k(x_k - x_0) & \dots & x_k^{k-1}(x_k - x_0) \end{pmatrix}$$

Continuiamo questo procedimento sottraendo stavolta dalla colonna 2 in poi, e moltiplicando per x_1:

$$\rightarrow \begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 1 & x_1 - x_0 & 0 & ... & 0 \\ 1 & x_2 - x_0 & x_2(x_2 - x_0) - x_1(x_2 - x_0) & ... & x_2^{k-1}(x_2 - x_0) - x_1(x_2 - x_0) \\ \vdots & & & & \vdots \\ 1 & x_k - x_0 & x_k(x_k - x_0) - x_1(x_k - x_0) & ... & x_k^{k-1}(x_k - x_0) - x_1(x_k - x_0) \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 1 & x_1 - x_0 & 0 & ... & 0 \\ 1 & x_2 - x_0 & (x_2 - x_1)(x_2 - x_0) & ... & (x_2^{k-1} - x_1)(x_2 - x_0) \\ \vdots & & & & \vdots \\ 1 & x_k - x_0 & (x_k - x_1)(x_k - x_0) & ... & (x_k^{k-1} - x_1)(x_k - x_0) \end{pmatrix}$$

Dovrebbe quindi risultare chiaro che ripetendo questa operazione k volte si ottiene una matrice triangolare inferiore, di cui ricordiamo il determinante si calcola moltiplicando gli elementi sulla diagonale, cioè si ottiene:

$$\det(V) = 1 \cdot (x_1 - x_0) \cdot (x_2 - x_1)(x_2 - x_0) \cdot ... = \prod_{0 \leq i < j \leq k} (x_j - x_i)$$

che è (gesticolando un pò) la tesi. □

Da questo deriva il **corollario** immediato che se $x_i \neq x_j \; \forall i \neq j$, allora $\det(V) \neq 0$, cioè V non è singolare.

Per trovare $p(x)$ basterà quindi risolvere il sistmea:

$$Va = y$$

e poniamo:

$$p(x) = \sum_{i=0}^{k} a_i x^i$$

da cui come avevamo detto si ottiene un polinomio con $\deg(p(x)) \leq k$ ($\leq$ perchè si potrebbe avere il coefficiente $a_k = 0$, e così via, cioè il grad è al più k).

Risolvere il sistema costa quindi, come avevamo visto, un qualcosa che è $\sim O(n^3)$, con particolare interesse al numero di condizionamento:

$$\mu(V) = |V| \cdot |V^{-1}|$$

che solitamente per la matrice di Vandermonde è particolarmente grande.

Ricordiamo che questo è un problema non da poco. Infatti, avevamo che il numero di condizionamento dava un limite alla variazione del resto del sistema in funzione della perturbazione delle soluzioni, cioè presa una $\tilde{x}$ perturbata da una x soluzione di $Ax = b$ si aveva:

$$|x - \tilde{x}| \leq \mu(A) \cdot |A\tilde{x} - b|$$

Ad esempio, preso il sistema:

$$\underbrace{\begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 1 \\ \epsilon \end{pmatrix}}_{b}$$

potremmo pensare di considerare la soluzione perturbata:

$$\tilde{x} = \begin{pmatrix} 1 \\ 1+\delta \end{pmatrix}$$

Avremo quindi l'*errore sulla soluzione*:

$$|x - \tilde{x}|_2 = |\delta|$$

e l'*errore sul resto*:

$$|A\tilde{x} - b|_2 = \left| \begin{pmatrix} 0 \\ \delta\epsilon \end{pmatrix} \right|_2 = |\delta| \cdot \epsilon$$

che chiaramente cresce al crescere di ϵ.

Questo ci è problematico in quanto l'*errore sulla soluzione* può essere considerato come l'**errore sui coefficienti** del polinomio, e l'*errore sul resto* può essere considerato come l'**errore di approssimazione** della funzione, cioè:

$$|A\tilde{x} - b| = |p(x_i) - y_i|$$

Vediamo che in verità la situazione è sotto controllo se:

$$\mu(A) \leq \frac{1}{u}$$

per u precisione macchina, in quanto l'errore dato dal cattivo condizionamento diventa comparabile con la precisione macchina, comunque piuttosto piccola. Un altro caso di interesse è quello in cui i punti x_i sono equispaziati sulla circonferenza unitaria sul piano di Argand-Gauss. Vediamo che anche qui l'errore è mantenuto sotto controllo, e questo sarà infatti il caso che considereremo per la *trasformata (discreta) veloce di Fourier*.

Un'idea potrebbe essere quindi quella di cambiare la base dei monomi in:

$$B' = \{\phi_0(x), \phi_1(x), ..., \phi_k(x)\}$$

e porre quindi:

$$p(x) = a_0\phi_0(x) + a_1\phi_1(x) + ...a_k\phi_k(x)$$

Il sistema alla sostituzione dei nodi continuerà ad essere lineare, in quanto:

$$p(x_i) = a_0\phi_0(x_i) + a_1\phi_1(x_i) + ... + a_k\phi_k(x_i), \quad \forall i \in \{0, ..., k\}$$

Potremo quindi impostare un altro sistema lineare:

$$\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & ... & \phi_k(x_0) \\ \vdots & & & \vdots \\ \phi_0(x_k) & \phi_1(x_k) & ... & \phi_k(x_k) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_k \end{pmatrix}$$

Abbiamo quindi definito la matrice di Vandermonde "generalizzata" a:

Definizione 13.1: Matrice di Vandermonde

La matrice di Vandermonde su una certa base polinomiale $\{\phi_0(x), ..., \phi_k(x)\}$ è definita come:

$$V_{ij} = \phi_j(x_i), \quad \forall i, j \in \{0, 1, ..., k\}$$

Finora avevamo preso il caso della base canonica, cioè $\{1, x, ..., x^k\}$, ma nessuno ci nega di prendere altre basi. Basi che ci saranno di particolare interesse, vedremo, sono la base di **Lagrange** e la base di **Newton**.

13.1.2 Base di Lagrange

Presi sempre i nostri nodi di interpolazione, decidiamo di scegliere la base guardando proprio tali nodi (anziché prenderla a priori, come avevamo fatto per Vandermonde).

Chiamiamo i polinomi di base $l_0(x), ..., l_k(x)$, con:

$$l_i(x) = (x - x_0)...(x - x_{i-1})(x - x_{i+1})...(x - x_k) = \prod_{j \neq i}^{n} (x - x_j)$$

cioè il prodotto dei fattori che si annullano in x_j per tutti gli $j \neq i$. Spesso poi decidiamo di normalizzare i polinomi di base, indicandoli come:

$$l_i(x) = \frac{(x - x_0)...(x - x_{i-1})(x - x_{i+1})...(x - x_k)}{(x_i - x_0)...(x_i - x_{i-1})(x_i - x_{i+1})...(x_i - x_k)} = \prod_{j \neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Otterremo quindi la base, diversa da quella canonica:

$$B_{\text{Lagrange}} = \{l_0(x), l_1(x), ..., l_k(x)\}$$

Proviamo quindi ad imporre le condizione, con:

$$p(x) = a_0 l_0(x) + a_1 l_1(x) + ... + a_k l_k(x)$$

valutando il polinomio in un punto x_i:

$$p(x_i) = y_i = a_0 l_0(x_i) + ... + a_{i-1} l_{i-1}(x_i) + a_i l_i(x_i) + a_{i+1} l_{i+1}(x_i) + ... + a_k l_k(x_i)$$

dove vediamo che, normalizzando, si conserva solo il termine in a_i, da cui immediatamente:

$$a_i = y_i$$

e:

$$p(x) = y_0 l_0(x) + y_1 l_1(x) + ... + y_k l_k(x)$$

Possiamo quindi costruire il polinomio interpolante senza fare nessun conto, semplicemente prendendo la combinazione lineare a coefficienti y_i della base di lagrange l_i costruita sui nodi.

Vediamo quindi la forma della matrice di Vandermonde:

$$V_l = \begin{pmatrix} l_0(x_0) & l_1(x_0) & ... & l_k(x_0) \\ \vdots & & & \vdots \\ l_0(x_k) & l_1(x_l) & ... & l_k(x_k) \end{pmatrix} = \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & 0 & 1 \end{pmatrix}$$

da cui abbiamo la conferma: si ricava direttamente l'identità e quindi $Va = y$ diventa $Ia = y \implies a = y$.

13.1.3 Esempio: inteprolazione di Lagrange

A scopo di esempio, vediamo l'interpolazione di Lagrange della funzione:

$$f(x) = \cos\left(\frac{x}{3}\right) \sin(2x)$$

con nodi equispaziati presi da -6 a 6, con intervallo 1.

Calcolando la funzione attraverso un software al computer, si otterrà la tabella di nodi:

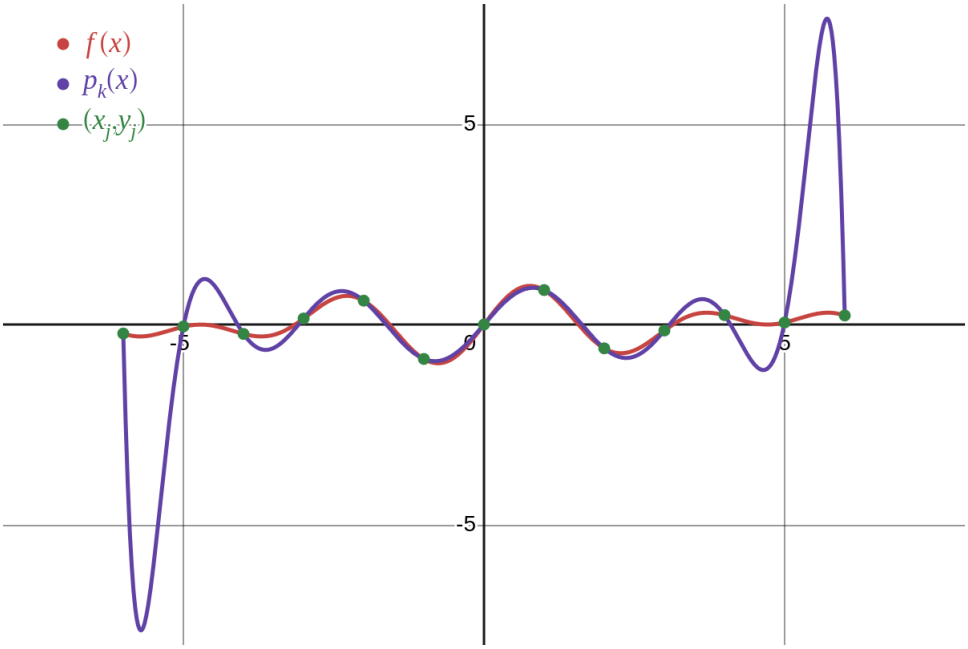
x_j	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
y_j	-0.22	-0.05	-0.23	0.15	0.59	-0.86	0	0.86	-0.59	-0.15	0.23	0.05	0.22

Da cui il polinomio:

$$p_k(x) = -0.22l_0(x) - 0.05l_1(x) - 0.23l_2(x) + 0.15l_3(x) + 0.59l_4(x) - 0.86l_5(x) + 0.86l_6(x)$$

$$-0.59l_7(x) - 0.15l_8(x) + 0.23l_9(x) + 0.05l_{10}(x)0.22l_{11}(x)$$

Vediamo quindi un grafico che mostra tale polinomio, sovraimposto alla funzione originale e ai nodi campionati:



Come vediamo, almeno nei punti intermedi l'interpolazione è piuttosto vicina, mentre ai punti estremi si notano degli artefatti non indifferente (principalmente dati dal comportamento oscillatorio della $f(x)$).

13.1.4 Base di Newton

Vediamo quindi la base di Newton, che costruiamo come:

$$\begin{cases} n_0(x) = 1 \\ n_1(x) = x - x_0 \\ n_2(x) = (x - x_0)(x - x_1) \\ \vdots \\ n_k(x) = (x - x_0)...(x - x_{k-1}) \end{cases}$$

cioè ogni n_i dipende solo dai $\{x_0, ..., x_{i-1}\}$ precedenti. Questo rende più facile l'aggiornamento se si aggiunge un nuovo nodo in x_{k+1}, per cui basterà dire:

$$n_{k+1}(x) = (x - x_0)...(x - x_k) = n_k(x)(x - x_k)$$

Otterremo quindi la base:

$$B_{\text{Newton}} = \{n_0(x) = 1, n_1(x), ..., n_k(x)\}$$

e la matrice di Vandermonde:

$$V_n = \begin{pmatrix} n_0(x_0) & n_1(x_0) & ... & n_k(x_0) \\ \vdots & & & \vdots \\ n_0(x_k) & n_1(x_k) & ... & n_k(x_k) \end{pmatrix}$$

con:

$$p(x) = a_0 n_0(x) + a_1 n_1(x) + ... + a_k n_k(x)$$

e nel nodo:

$$p(x_i) = y_i \Leftrightarrow Va = y$$

dallo svolgimento della matrice di Vandermonde si ha che la prima colonna è tutta di 1, e il resto della matrice è triangolare inferiore:

$$V_n = \begin{pmatrix} 1 & 0 & ... & 0 \\ 1 & ? & ... & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & ? & ... & ? \end{pmatrix}$$

e quindi ogni a_i dipende solo dagli $\{y_0, ...y_i\}$.

Vedremo la struttura specifica degli a_i che formano il polinomio inerpolante alla prossima lezione.

14 Lezione del 11-04-25

Riprendiamo il discorso dell'interpolazione polinomiale nella prospsettiva di arrivare all'interpolazione di Newton, a cui abbiamo accennato alla scorsa lezione.

Avevamo detto di avere $(x_0, y_0), (x_1, y_1), ..., (x_k, y_k)$, cioè $k+1$ punti **distinti**, cioè con $x_i \neq x_j, \forall i \neq j$, tali per cui volevamo interpolarli con un polinomio di grado k. Questo esisteva ed era unico appunto sotto l'ipotesi di punti distinti.

Avevamo qiundi visto le possibili base per la realizzazione di tale polinomio, di *Vandermonde* e di *Lagrange*.

14.1 Differenze divise

Avevamo poi introdotto la base di **Newton**, che definiva il polinomio del tipo:

$$p_k(x) = \sum_{j=0}^{k} a_j n_j(x) = \sum_{j=0}^{k} f[x_0, ..., x_j] n_j(x)$$

con:

$$\begin{cases} n_0(x) = 1 \\ n_1(x) = x - x_0 \\ n_2(x) = (x - x_0)(x - x_1) \\ \vdots \\ n_k(x) = (x - x_0)...(x - x_{k-1}) \end{cases}$$

Per la precisione, eravamo rimasti in attesa di definire i coefficienti a_j del polinomio interpolante.

Cerchiamo di trovarli sfruttando la matrice di Vandermonde generalizzata (definizione 13.1):

$$V = \begin{pmatrix} n_0(x_0) & n_1(x_0) & n_2(x_0) & ... & n_k(x_0) \\ n_0(x_1) & n_1(x_1) & n_2(x_1) & ... & n_k(x_1) \\ n_0(x_2) & n_1(x_2) & n_2(x_2) & ... & n_k(x_2) \\ \vdots & & & & \vdots \\ n_0(x_k) & n_1(x_k) & n_2(x_k) & ... & n_k(x_k) \end{pmatrix}$$

Applicando la definizione di base di Newton, si ottiene una forma del tipo:

$$= \begin{pmatrix} 1 & 0 & & & ...0 \\ 1 & x_1 - x_0 & 0 & & ...0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & 0 & ...0 \\ \vdots & & & & \vdots \\ 1 & x_k - x_0 & (x_k - x_0)(x_k - x_1) & ... & (x_k - x_0)(x_k - x_1)...(x_k - x_{k-1}) \end{pmatrix}$$

cioè triangolare inferiore, e quindi risolvibile per sostituzione in avanti, imponendo:

$$Va = y \Leftrightarrow V \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}$$

Svolgendo un paio di passaggi sella sostituzione in avanti, si ha:

$$\begin{cases} a_0 = y_0 \\ y_0 + a_1(x_1 - x_0) = y_1 \implies a_1 = \frac{y_1 - y_0}{x_1 - x_0} \\ y_0 + (x_2 - x_0)\frac{y_1 - y_0}{x_1 - x_0} + (x_2 - x_0)(x_2 - x_1)a_2 = y_2 \implies a_2 = \frac{\frac{y_2 - y_0}{x_2 - x_0} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_1} \end{cases}$$

Cioè si nota una forma comune per i coefficienti a_j

Questi coefficienti, che indichiamo con $a_j = f[x_0, ..., x_j] \ \forall j$ si definiscono quindi per *ricorsione*, e prendono il nome **differenze divise**. Definiamo quindi la formula di ricorrenza:

Definizione 14.1: Differenza divisa

Data una funzione $f : \mathbb{R} \to \mathbb{R}$, e dati $x_0, ..., x_{k-1} \in \mathbb{R}$, con $x_i \neq x_j \, \forall i \neq j$, vogliamo una funzione $f[x]$ che rispetti:

$$\begin{cases} f[x] = f(x), & k = 0 \\ f[x_0, x] = \frac{f(x)-f(x_0)}{x-x_0}, & k = 1 \\ f[x_0, x_1, x] = \frac{\frac{f(x)-f(x_0)}{x-x_0} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x-x_1}, & k = 2 \\ f[x_0, ..., x_{k-1}, x] = \frac{f[x_0,...,x]-f[x_0,...,x_{k-1}]}{x-x_{k-1}}, & \text{altrimenti} \end{cases}$$

Chiamiamo questa funzione differenza divisa e la indichiamo come $f[x_0, ..., x]$.

Le differenze divise ricordano la forma del *rapporto incrementale*, e anzi corrispondono esattamente a questo per $k = 1$, mentre per $k > 1$ si può pensare ad un'approssimazione *"grezza"*, o meglio *finita*, delle derivate di ordine k, cioè:

$$\begin{cases} k = 0 : & f[x] = f(x) \\ k = 1 : & f[x_0, x] = \frac{f(x)-f(x_0)}{x-x_0} \approx \frac{df}{dx}(x_0) \\ k = ? : & f[x_0, ...x_{k-1}, x] = \frac{f[x_0,...,x]-f[x_0,...,x_{k-1}]}{x-x_{k-1}} \approx \frac{d^k f}{dx^k}(x_0) \end{cases}$$

14.1.1 Proprietà delle differenze divise

Possiamo trovare le seguenti proprietà delle differenze divise:

1. $\forall P$ permutazione $\{i_0, ..., i_k\}$ di $\{0, ..., k\}$ vale:

   $$f[x_0, x_1, ..., x_k] = f[x_{i_0}, x_{i_1}, ..., x_{i_k}]$$

   cioè una permutazione dei $0, ..., k$ non cambia la differenza divisa;

2. $f[x_0, ..., x_{k-1}, x]$ è una funzione non definita in $x_0, ..., x_{k-1}$. Tuttavia, se $f \in C^1(I)$ con I contenente $x_0, ..., x_k$, allora è prolungabile per continuità su tale intervallo;

3. Se f è non solo C^1, ma $f \in C^k(I)$, allora esiste ϵ tale che:

   $$\epsilon \in \left[\min_{0 \leq i \leq k} x_i, \max_{0 \leq i \leq k} x_i \right]$$

   tale che:

   $$f[x_0, ..., x_k] = \frac{f^{(k)}(\epsilon)}{k!}$$

   Questa è una conseguenza del teorema di Lagrange.

14.1.2 Calcolo delle differenze divise

Esiste un metodo piuttosto schematico per il calcolo delle differenze divise. Dati $k + 1$ punti di interpolazione, si possono calcolare le differenze divise fino all'ordine k secondo questo schema (posto $k = 3$), detto **quadro delle differenze divise**:

x	$f(x)$	DD1	DD2	DD3
x_0	$f(x_0)$	//	//	//
x_1	$f(x_1)$	$f[x_0, x_1]$	//	//
x_2	$f(x_2)$	$f[x_0, x_2]$	$f[x_0, x_1, x_2]$	//
x_3	$f(x_3)$	$f[x_0, x_3]$	$f[x_0, x_2, x_3]$	$f[x_0, x_1, x_2, x_3]$

Cioè a ogni passaggio completiamo una riga, da sinistra verso destra, con le differenze divise note a quel passaggio. Notiamo che in verità le differenze divise che si trovano sulla diagonale sono le sole che compaiono effettivamente nel polinomio di Newton.

Supponendo di scrivere gli elementi della tabella come le entrate di una matrice $A = aij$ triangolare inferiore, per calcolare una sua entrata a_{ij} la formula da applicare è la seguente:

$$a_{ij} = \frac{a_{i,j-1} - a_{j-1,j-1}}{x_i - x}$$

Poniamo infatti A:

$$A = \begin{pmatrix} f(x_0) & & ... & 0 \\ f(x_1) & f[x_0, x_1] & & \\ \vdots & f[x_0, x_2] & f[x_0, x_1, x_2] & \vdots \\ \vdots & \vdots & \vdots & \\ f(x_k) & f[x_0, x_k] & f[x_0, x_1, x_k] & ... \quad f[x_0, x_1, ..., x_k] \end{pmatrix}$$

14.1.3 Esempio: quadro delle differenze divise

Vediamo un caso pratico di quadro risolto:

x	$f(x)$	DD1	DD2	DD3
0	3	//	//	//
1	8	5	//	//
2	15	6	1	//
3	17	11	2	$\frac{1}{2}$

A questo punto il polinomio sarà:

$$p_3(x)$$

$$= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2)$$

$$= 3 + 5(x - 0) + 1(x - 0)(x - 1) + \frac{1}{2}(x - 0)(x - 1)(x - 2) = \frac{1}{2}x^3 - \frac{1}{2}x^2 + 5x + 3$$

Osserviamo che, nel caso in cui gli elementi di una colonna delle differenze divise siano tutti uguali, gli elementi delle colonne successive sono tutti nulli, e in particolare il grado del polinomio di interpolazione p_k sarà p_i con i l'ultima colonna non nulla (cioè l'ordine dell'ultima differenza divisa non nulla).

14.1.4 Esempio: colonne nulle nel quadro

Vediamo ad esempio il quadro delle differenze divise dove troviamo la situazione:

x	$f(x)$	DD1	DD2	DD3	DD4
0	5	//	//	//	//
-1	3	2	//	//	//
2	3	-1	-1	//	//
-2	-9	7	-5	1	//
3	11	2	0	1	0

Come vediamo, alla colonna DD3 le differenze divise sono 1 e 1, ergo la differenza divisa in DD4 è nulla. Questo significherà che il polinomio interpolante, sebbene sono stati presi $k+1=5$ punti, da cui ci aspetteremo un polinomio di grado $k=4$, ha invece grado 3, cioè è:

$$p_2(x) = 5 + 2(x-0) - 1(x-0)(x+1) + 1(x-0)(x+1)(x-2) = x^3 - 2x^2 - x + 5$$

Osserviamo infine che data una tabella di valori $(x_0, y_0), ..., (x_k, y_k)$ da interpolare, possiamo riordinarli come vogliamo nel quadro delle differenze divise. Questo può essere utile a semplificare i calcoli.

14.1.5 Esempio: riordinamento dei valori

Poniamo di avere i seguenti dati:

x	$f(x)$
0	2
1	1
α	4
-1	$3\alpha + 1$
3	11

con $\alpha \in \mathbb{R} \setminus \{0, 1, -1, 3\}$ e di chiederci quale deve essere il valore di α perché il polinomio interpolante abbia grado minimo.

Vediamo quindi di riordinare i dati in modo che i termini dipendenti da α vadano a finire in fondo alla tabella:

x	$f(x)$	DD1	DD2	DD3	DD4
0	2	//	//	//	//
1	1	-1	//	//	//
3	11	3	2	...	...
-1	$3\alpha + 1$	$1-3\alpha$	$\frac{3\alpha-2}{2}$	...	...
α	4	$\frac{2}{\alpha}$	$\frac{2+\alpha}{\alpha(\alpha-1)}$	...	...

Alla colonna DD1 non abbiamo speranza di soddisfare la condizione di colonna tutta uguale (ci sono -1 e 3), mentre alla colonna DD2 possiamo provare ad imporre:

$$\begin{cases} \frac{3\alpha-2}{2} = 2 \implies \alpha = 2 \\ \frac{2+\alpha}{\alpha(\alpha-1)} = 2 \implies \alpha = 2 \end{cases}$$

che quindi, per pura fortuna, corrisponde. Possiamo allora prendere $\alpha = 2$ e completare il quadro:

x	$f(x)$	DD1	DD2	DD3	DD4
0	2	//	//	//	//
1	1	-1	//	//	//
3	11	3	2	//	//
-1	$3\alpha+1$	$1-3\alpha$	2	0	//
α	4	$\frac{2}{\alpha}$	2	0	0

da cui il polinomio al solo grado 2:

$$p_2(x) = 2 - x + 2x(x-1) = 2x^2 - 3x + 2$$

14.1.6 Esempio: interpolazione di Newton

A scopo di esempio, vediamo l'interpolazione di Newton della funzione:

$$f(x) = \frac{x^3}{30} - \cos\left(\frac{3}{2}x\right)$$

Approssimata secondo la tabella di nodi:

x_j	-5	-4	-1.9	0	2.3	3.5	5
y_j	-4.51	-3.09	0.73	-1	1.36	0.92	3.82

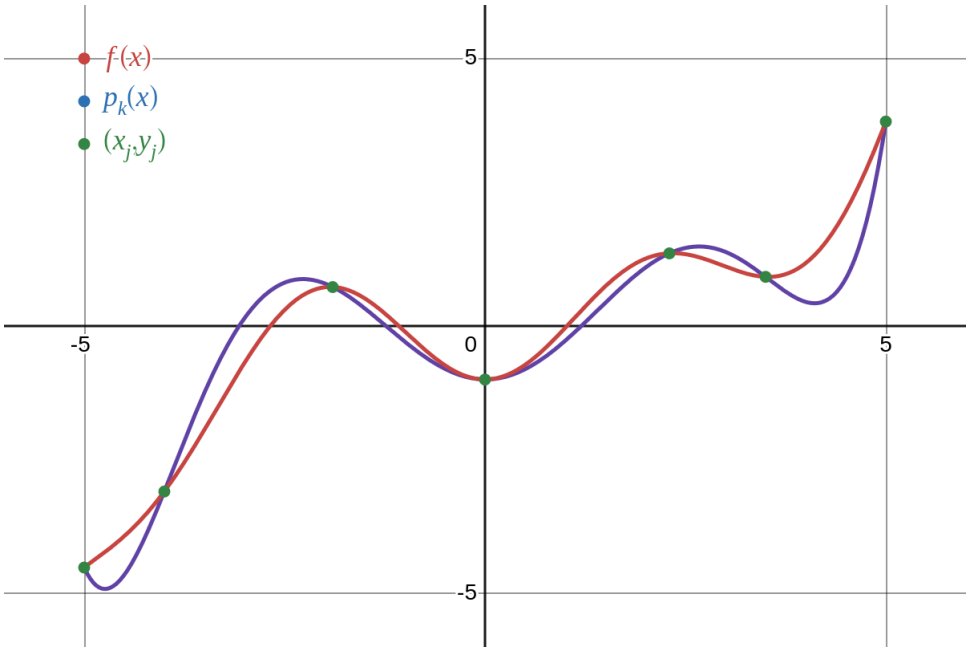
Il polinomio si trova quindi prendendo le basi di Newton $n_j(x)$ e calcolando le differenze divise, come abbiamo visto finora:

x	$f(x)$	DD1	DD2	DD3	DD4	DD5	DD6
-5	-4.51	//	//	//	//	//	//
-4	-3.09	1.42	//	//	//	//	//
-1.9	0.73	1.69	0.13	//	//	//	//
0	-1	0.7	-0.18	-0.16	//	//	//
2.3	1.36	0.8	-0.1	-0.05	0.05	//	//
3.5	0.92	0.64	-0.1	-0.04	0.03	-0.01	
5	3.82	0.83	-0.06	-0.03	0.03	~ 0	~ 0

Da cui il polinomio:

$$p_k(x) = -4.51n_0(x) + 1.42n_1(x) + 0.13n_2(x) - 0.16n_3(x) + 0.05n_4(x) - 0.01n_5(x)$$

Vediamo quindi un grafico che mostra tale polinomio, sovraimposto alla funzione originale e ai nodi campionati:



Il grafico è stato realizzato con Desmos (`https://www.desmos.com/calculator/tavjyzgpd0`), assieme allo scorso esempio sull'interpolazione di Lagrange (`https://www.desmos.com/calculator/w2tx5n7jtp`, esempio 13.1.3). Il calcolo al computer ci permette chiaramente di ottenere risultati più precisi, as esempio in questo caso a valutare il coefficiente della base di sesto grado (che svanisce approssimando alle prime 2 cifre decimali).

14.2 Errore nell'approssimazione polinomiale

Veniamo quindi alla valutazione dell'errore. Vale il seguente teorema:

Teorema 14.1: Errore dell'approssimazione polinomiale

Presi $k+1$ punti $x_0, ..., x_k$ con $x_i \neq x_j \, \forall i \neq j$. Sia $f : \mathbb{R} \to \mathbb{R}$ tale che $f \in C^{k+1}(I)$ con $I \subseteq \mathbb{R}$ contenente $x_0, ..., x_k$. Se $p_k(x)$ è il polinomio di interpolazione di grado al più k che interpola i punti $(x_0, f(x_0)), ..., (x_k, f(x_k)))$ allora riguardo all'errore vale:

$$f(x) - p_k(x) = \Pi(x) \frac{f^{(k+1)}(\xi)}{(k+1)!}$$

con:

$$\Pi(x) = \prod_{j=0}^{k} (x - x_j)$$

per un certo ξ tale che:

$$\xi \in \left[\min_{0 \leq i \leq k} x_i \cup x, \max_{0 \leq i \leq k} x_i \cup x \right]$$

Osserviamo quindi che conoscendo una stima di $|f^{k+1}|$ e di $|\Pi(x)|$ si può provare a stimare l'errore del polinomio di interpolazione quando si valuta $p_k(x)$ fuori dai nodi $\{x_j\}_{j=0}^k$.

14.3 Considerazioni ulteriori sull'approssimazione polinomiale

Vediamo quindi alcune nozioni che riguardano sia l'approssimazione di Newton che quella di Lagrange, che abbiamo visto rispettivamente in 13.1.4 e 14.1, e 13.1.2 (e che riguardano in verità anche il metodo di approssimazione di Vandermonde visto in 13.1.1).

14.3.1 Riassunto fra base di Newton e base di Lagrange

Facciamo quindi un riassunto confrontando la base di Newton e la base di Lagrange.

	Newton	**Lagrange**
Base	$n_j(x) = (x - x_0)...(x - x_{j-1})$	$l_j(x) = \prod_{i \neq j}^k \frac{x - x_i}{x_j - x_i}$
Coef.	$f[x_0, ...x_j]$	$f(x_j) = y_j$
	Con la base di Newton, se si è calcolato $p_k(x)$ e si vuole $p_{k+1}(x)$, basta calcolare m_{k+1} e $f[x_0, ..., x_k]$ da $f[x_0, ..., x_{k-1}]$	Con la base di Lagrange, per fare la stessa cosa bisogna ricalcolare tutti gli $e_j(x)$
	Il grado del polinomio si nota facilmente	Per trovare il grado del polinomio bisogna espandere nella base dei monomi

Viene da sé che i metodi di Vandermonde, Lagrange e Newton restituiscono tutti lo stesso polinomio (se non in forma algebricamente diversa), in quanto il polinomio interpolante di grado minimo è l'unico che passa per i punti (x_j, y_j) forniti.

14.3.2 Fenomeno di Runge

Vediamo un fenomeno che accumuna tutti i tipi di approssimazione polinomiale: presa una certa approssimazione di grado k di una funzione $f(x)$, quindi un polinomio $p(x)$ passante per $k + 1$ punti campionati da $f(x)$, si potrebbe pensare che aumentando il valore di k si otterrebbero approssimazioni via via più accurate. Questo è giusto per quanto riguarda i punti interni, ma riguardo agli estremi dell'intervallo di approssimazione notiamo che la funzione di errore $r(x) = p(x) - f(x)$ ha un errore costante se non addirittura crescente al crescere di k.

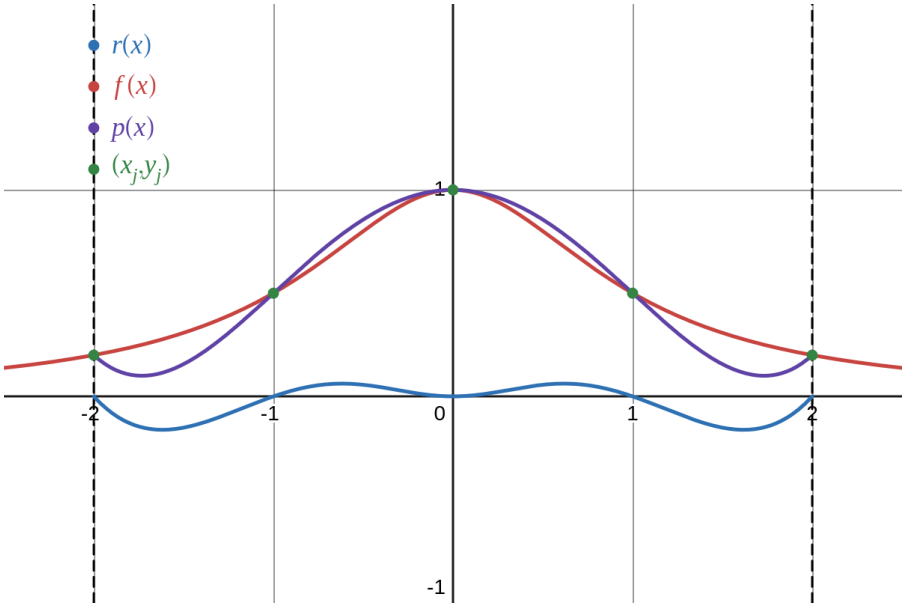
Prendiamo ad esempio la funzione:

$$f(x) = \frac{1}{1 + x^2} \quad \text{(versiera di Agnesi)}$$

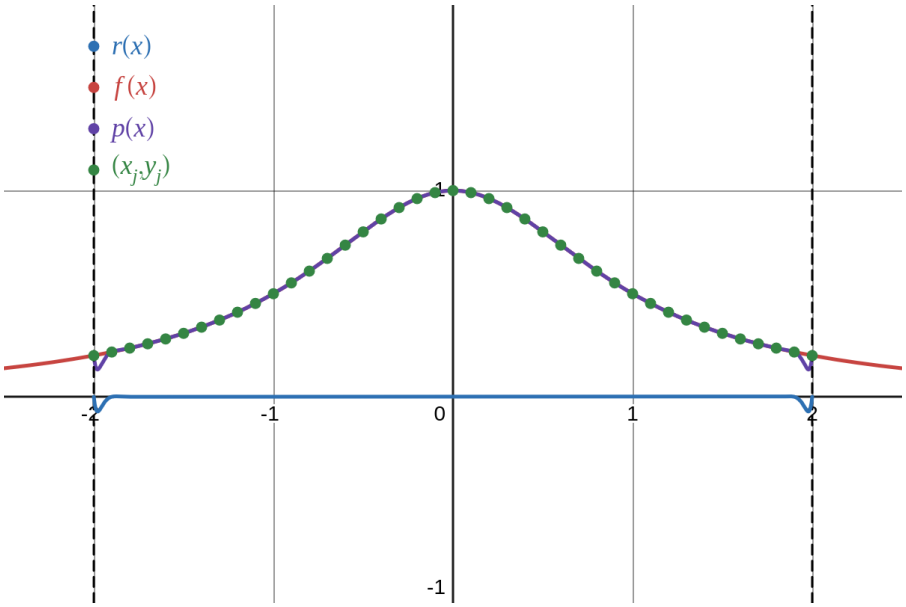
interpolata fra -2 e 2 ad intervalli regolari, quindi preso k interpolata sui nodi:

$$x_k = -2 + \frac{4}{k} i, \quad i = 0, 1, ..., k$$

Con $k = 4$, si ottiene il seguente polinomio interpolante:



Aumentando il grado a $k = 40$, quindi moltiplicando per 10, si ottiene invece il polinomio interpolante:



che è sì più accurato nei punti interni, ma ha comunque un certo errore in corrispondenza degli estremi $x = -2$ e $x = 2$.

Si può dimostrare che questo fenomeno permane al crescere di k, ergo aumentare il grado non corrisponde sempre a migliorare l'accuratezza dell'approssimazione.

14.3.3 Nodi di Chebysev

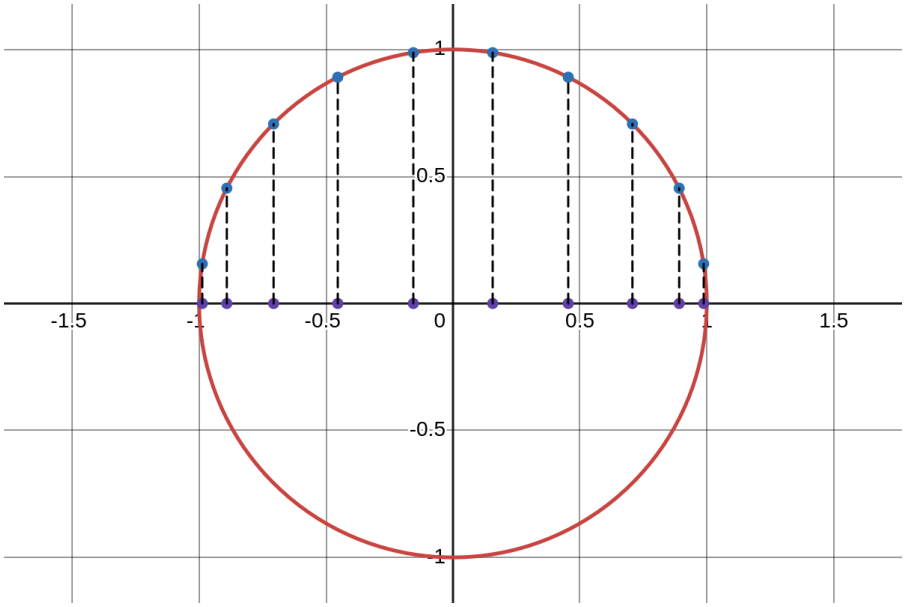
Un modo per migliorare l'accuratezza delle approssimazioni polinomiali, e sopratutto per evitare il fenomeno di Runge in prossimità degli estremi, può essere quello di cambiare i nodi presi, usando i cosiddetti **nodi di Chebyshev**.

Definizione 14.2: Nodi di Chebyshev

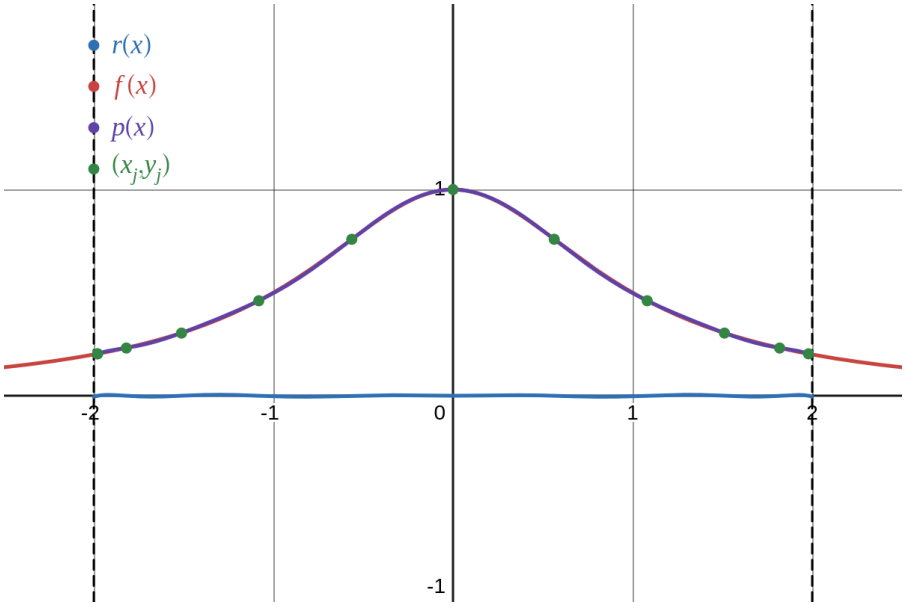
Si chiamano nodi di Chebysev unitari gli n punti x_k:

$$x_k = \cos\left(\frac{\left(K + \frac{1}{2}\right)\pi}{n_p}\right), \quad k = 0, ..., n-1$$

I nodi di Chebyshev corrispondono alle proiezioni sull'asse x di punti equispaziati sulla parte superiore di una circonfenferenza unitaria, cioè sul grafico:



Utilizzando i nodi di Chebyshev nel calcolo del polinomio interpolante, si usano risultati molto più accurati. Ad esempio, per l'approssimazione della stessa curva del paragrafo scorso, si ottiene che già con $k = 10$ il grafico ha la aspetto:



da dove notiamo quindi che l'errore, sopratutto vicino agli estremi, è significativamente ridotto.

14.4 Interpolazione osculatoria di Hermite

Vediamo un altro metodo per interpolare polinomi. Supponiamo di conoscere non soltanto ai nodi il valore della funzione, ma anche il valore della sua derivata. In questo caso potremo costruire un polinomio in cui le valutazioni e quelle derivata equivalongono rispettivamente ai valori di f e f'. Questo potrebbe essere utile nel caso in cui conosciamo sia la posizione che la velocità di un corpo in certi istanti e vogliamo approssimarne la traiettoria.

Definiamo quindi:

Definizione 14.3: Polinomio interpolante di Hermite

Dati $x_0, ..., x_k \in I \subseteq \mathbb{R}$ e $2k+2$ valori $\{f(x_0), ..., f(x_k)\}$ e $\{f'(x_0), ..., f'(x_k)\}$ di una certa funzione $f \in C^1(I)$, si dice polinomio interpolante di Hermite, per f coi nodi $x_0, ..., x_k$, il polinomio $H_{2k+1}(x)$ di grado al più $2k+1$ tale che:

$$H_{2k+1}(x_j) = f(x_j), \quad H'_{2k+1}(x_j) = f'(x_j), \quad \forall j = 0, ..., k$$

Varrà quindi il seguente teorema:

Teorema 14.2: Esistenza del polinomio interpolante di Hermite

Se $x_i \neq x_j \ \forall i \neq j$ allora esiste ed è unico il polinomio di Hermite H_{2k+1} per $f \in C^1(I)$.

La dimostrazione di unicità si ha ipotizzando, per assurdo, che esista un secondo polinomio di hermite S_{2k+1}. A questo punto si potrà definire la funzione di errore:

$$R(x) = H_{2k+1}(x) - S_{2k+1}(x)$$

Questà dovrà annullarsi fino al primo grado in $x_0, ..., x_k$, per cui dovrà essere divisiibile per:

$$Q(x) = (x - x_0)^2 ... (x - x_k)^2$$

Questo però significherebbe che $R(x)$, e quindi H_{2k+1} e S_{2k+1}, dovrebbero essere almeno di grado $2k+2$ (il grado di $Q(x)$), che viola le ipotesi. □

Vediamo allora come trovare il polinomio H_{2k+1} vero e proprio. Si potrebbe formulare un sistema lineare come è stato fatto nel caso della base canonica, che dava la matrice standard di Vandermonde.

In questo caso partiremo dall'ipotizzare riguardo a H_{2k+1}:

$$\deg(H_{2k+1}) = 2k+1, \quad H_{2k+1} = a_0 + a_1 x + a_2 x^2 + ... + a_{2k+1} x^{2k+1} = \sum_{i=0}^{2k+1} a_i x^i$$

e quindi imporre le condizioni:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + ... + a_{2k+1} x_0^{2k+1} = f(x_0) \\ \vdots \\ a_0 + a_1 x_k + a_2 x_k^2 + ... + a_{2k+1} x_k^{2k+1} = f(x_k) \\ a_1 + 2a_2 x_0 + ... + (2k+1) a_{2k+1} x_0^{2k} = f'(x_0) \\ \vdots \\ a_1 + 2a_2 x_k + ... + (2k+1) a_{2k+1} x_k^{2k} = f'(x_k) \end{cases}$$

da cui, volendo esprimere il sistema come $Va = y$, si ottiene la "matrice di Vandermonde":

$$V = \begin{pmatrix} 1 & x_0 & x_0^2 & ... & x_0^{2k+1} \\ \vdots & & & & \vdots \\ 1 & x_k & x_k^2 & ... & x_k^{2k+1} \\ 0 & 1 & 2x_0 & ... & (2k+1)x_0^{2k} \\ \vdots & & & & \vdots \\ 0 & 1 & 2x_k & ... & (2k+1)x_k^{2k} \end{pmatrix}$$

Spesso, però, questo procedimento non conviene in quanto si incontrano gli stessi problemi di malcondizionamento che avevamo incontrato anche nella matrice standard di Vandermonde.

Tipicamente allora si procede in maniera analoga all'interpolazione di Lagrange, ma generalizzata per ottenere le condizioni sulle derivate. Più precisamente, cerchiamo un polinomio della forma:

$$H_{2k+1}(x) = \sum_{j=0}^{k} h_{0j}(x) f(x_j) + \sum_{j=0}^{k} h_{1j}(x) f'(x_j)$$

Avremo che $f(x_j)$ e $f'(x_j)$ sono chiaramente i dati del problema, che farano da coefficienti, mentre $h_{0j}(x)$ e $h_{1j}(x)$ saranno polinomi di grado $2k+1$ tali che:

$$h_{0j}(x_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad , \quad h'_{0j}(x_i) = 0 \quad \forall\, 0 \leq i \leq k$$

$$h_{1j}(x_i) = 0, \quad h'_{1j}(x_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \qquad \forall\, 0 \leq i \leq k \tag{3}$$

che possiamo interpretare come le condizioni imposte alle basi di Lagrange generalizzata al caso con la derivata prima.

Se valgono queste proprietà, infatti, si ha che:

$$H_{2k+1}(x_i) = \sum_{j=0}^{k} h_{0j}(x_j) f(x_i) + \sum_{j=0}^{k} h_{1j}(x_i) f'(x_j) = f(x_j) + 0$$

e rispetto alla derivata prima:

$$H'_{2k+1}(x_i) = \sum_{j=0}^{k} h'_{0j}(x_i) f(x_j) + \sum_{j=0}^{k} h'_{1j} f'(x_j) = 0 + f'(x_i)$$

con in entrambi i casi, $\forall\, i = 0, ..., k$.

Osserviamo che come nel caso dell'interpolazione standard, stiamo rinunciando a cercare una rappresentazione di H_{2k+1} nella base dei monomi $\{1, x, ..., x^{2k+1}\}$ e cerchiamo invece un'espressione rispetto alla base:

$$B_{\text{Hermite}} = \{h_{01}(x), ..., h_{0k}(x), h_{10}(x), ..., h_{1k}(x)\}$$

in cui i coefficienti delle combinazioni lineari sono:

$$f(x_0), ..., f(x_k), f'(x_0), ..., f'(x_k)$$

Il problema si riconduce quindi a trovare $h_{0j}(x)$, $h_{1j}(x)\ \forall j$ che soddisfino le proprietà (3).

L'idea parte dall'*ansatz* di cercare h_{0j} e h_{1j} della forma:

$$h_{0j} = (Ax + B)e_j^2(x), \quad h_{1j} = (Cx + D)e_j^2(x)$$

con $e_j(x)$ j-esimo polinomio in base di Lagrange, cioè:

$$e_j(x) = \prod_{i \neq j}^{k} \frac{x - x_i}{x_j - x_i}$$

Imponiamo quindi le proprietà (3) per ricavare A, B, C, D di ogni h_{0j} e h_{1j}.

- $h_{0j}(x)$: avremo che la proprietà di grado zero è sempre rispettata finché:

  $$Ax_j + B = 1$$

  in quanto $e_j^2(x_j)$ vale 1 o 0 esattamente come richiesto. Riguardo al primo grado, avremo invece:

  $$h'_{0j}(x_j) = Ae_j^2(x_j) + 2(Ax_j + B)e_j(x_j)e'_j(x_j) = 0$$

  dove si possono eliminare i termini $Ax_j + B$ e $e_j(x_j)$, e quindi:

  $$A + 2e'_j(x_j) = 0 \implies A = -2e'_j(x_j)$$

  sostituendo in $Ax_j + B$ si trova allora:

  $$B = 1 + 2e'_j(x_j)x_j$$

  da cui la funzione finale:

  $$h_{0j} = \Big(1 - 2e'_j(x_j)(x - x_j)\Big)\, e_j^2(x)$$

- $h_{1j}(x)$: avremo che la proprietà di grado zero è sempre rispettata finché:

  $$Cx_j + D = 0$$

  in quanto $e_j^2(x_j)$ vale 1 soltanto in x_j, e lì vogliamo annullarlo. Riguardo al primo grado, avremo invece:

  $$h'_{1j}(x) = Ce_j^2(x) + 2(Cx + D)e_j(x)e'_j(x) = 1$$

dove si possono fare le stesse semplificazioni di sopra, e quindi:

$$h'_{1j} = Ce_j^2(x) \implies C = 1$$

sostituendo in $Cx_j + D$ si trova allora:

$$D = -x_j$$

da cui la funzione finale:

$$h_{1j}(x) = (x - x_j)e_j^2(x)$$

Abbiamo quindi ottenuto le due funzioni h_{0j} e h_{1j}:

$$\begin{cases} h_{0j}(x) = \Big(1 - 2e'_j(x_j)(x - x_j)\Big)\, e_j^2(x) \\ h_{1j}(x) = (x - x_j)e_j^2(x) \end{cases}$$

ossia il polinomio interpolante di Hermite avrà la forma:

$$H_{2k+1}(x) = \sum_{j=0}^{k} f(x_j)\left(1 - 2e'_j(x_j)(x - x_j)\right)e_j^2(x) + \sum_{j=0}^{k} f'(x_j)(x - x_j)e_j^2(x)$$

14.4.1 Esempio: raccordo di binari

Supponiamo di avere 2 binari e di volerli unire in maniera C^1, ovvero di voler trovare un polinomio $H_3(x)$ tale che:

$$\begin{cases} H_3(0) = 0 \\ H_3(d) = L \end{cases}, \quad \begin{cases} H'_3(0) = 0 \\ H'_3(d) = S \end{cases}$$

Il polinomio dovrà essere di grado 3 (le due condizioni alle derivate contano nel calcolo del grado). Dobbiamo calcolare, prima di tutto, le basi di Lagrange:

$$e_0(x) = \frac{x - d}{d}, \quad e_1(x) = \frac{x}{d}$$

e sostituire quindi nelle formule per $h_{0j}(x)$ e $h_{1j}(x)$:

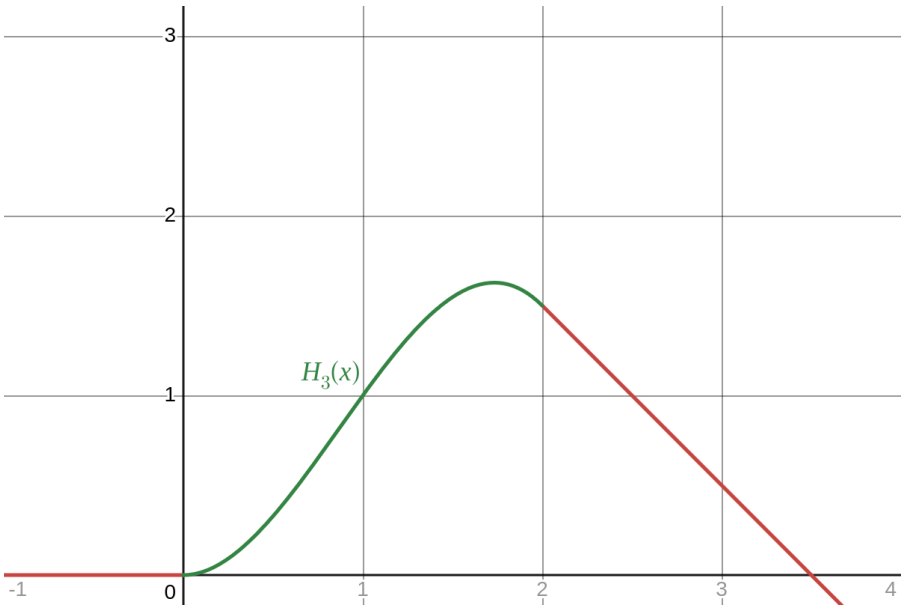
$$h_{00}(x) = \left(1 + 2\frac{x}{d}\right)\frac{(x - d)^2}{d^2}, \quad h_{01}(x) = \left(1 - 2\frac{x - d}{d}\right)\frac{x^2}{d^2}$$

$$h_{10}(x) = x\frac{(x - d)^2}{d^2}, \quad h_{11} = (x - d)\frac{x^2}{d^2}$$

per cui otterremo:

$$H_3(x) = 0 \cdot h_{00}(x) + L \cdot h_{01}(x) + 0 \cdot h_{10}(x) + S \cdot h_{11}(x) = L\left(1 - 2\frac{x - d}{d}\right)\frac{x^2}{d^2} + S(x - d)\frac{x^2}{d^2}$$

Vediamo un esempio grafico:



14.4.2 Errore nell'approssimazione di Hermite

Valutiamo quindi l'errore:

Teorema 14.3: Errore dell'approssimazione di Hermite

Data $f \in C^{2k+2}(I)$ allora:

$$f(x) - H_{2k+1}(x) = \Pi^2(x)\frac{f^{(2k+1)(\xi)}}{(2k+2)!}$$

con le stesse definizioni del 14.1.

15 Lezione del 14-04-25

Abbiamo finora presupposto di avere una serie di punti (x_j, y_j), con $j = 0, ..., k$, tratti da una funzione $f : [a, b] \to \mathbb{R}$, cioè definita su un certo intervallo $[a, b]$, con $x_j \in [a, b]$ e $y_j = f(x_j) \in \mathbb{R}$. Da questi punti volevamo ricavare un polinomio $p(x)$ di grado al più k tale che questo interpolasse i punti, cioè verificasse:

$$p(x_j) = y_j$$

Da questa configurazione avevamo visto più metodi per il calcolo effettivo di $p(x)$, e avevamo ipotizzato che il modo migliore per aumentarne l'accuratezza, cioè ridurre il massimo della funzione:

$$r(x) = f(x) - p(x), \quad x \in [a, b]$$

fosse aumentare k.

Questo però aveva diversi aspetti negativi:

- Avere k maggiore significa aumentare i punti campionati, cosa che potrebbe non essere sempre fattibile;
- Non è detto che k maggiore aumenti l'accuratezza: avevamo visto infatti il *fenomeno di Runge* agli estremi, per cui il limiter dell'errore all'aumentare di k non era zero, cioè:

$$\lim_{k \to +\infty} |r(x)| \neq 0$$

Avevamo visto un modo per migliorare questa situazione, che era usare nodi non equispaziati ma equispaziati su una circonferenza e proiettati sull'asse reale: questi erano i *nodi di Chebyshev*.

Abbiamo in generale, però, che aumentare il grado k non è la situazione ottimale, e si preferirebbe continuare con funzioni polonomiali di grado basso. Introduciamo per questo esatto motivo l'**interpolazione polinomiale a tratti**.

15.1 Interpolazione polinomiale a tratti

Decidiamo quindi di spezzare il dominio $[a, b]$ in sottointervalli di tipo $[x_{i-1}, x_i]$ e di usare in ogni sottointervallo polinomi di grado basso. In ogni sottointervalloconsideriamo quindi polinomi $p_i(x)$ di grado al più $s < k$ tali che $p_i(x)$ interpola f in x_{i-1} e x_i, cioè:

$$p_i(x) : [x_{i-1}, x_i] \to \mathbb{R}, \quad p_i(x_{i-1}) = y_{i-1}, \quad p_i(x_i) = y_i$$

Vediamo quindi alcuni casi particolari di questo tipo di interpolazione al variare di s, presa la tabella di punti:

	$\mathbf{x_j}$	$\mathbf{y_j}$
P_0	-13	5
P_1	-4	-3
P_2	1	2
P_3	13	-1

15.1.1 $\mathbf{s = 1}$: interpolazione lineare a tratti

Nel caso più semplice l'interpolazione lineare a tratti si riduce al definire una serie di rette che collegano ogni coppia di punti x_{i-1}, x_i, in forma:

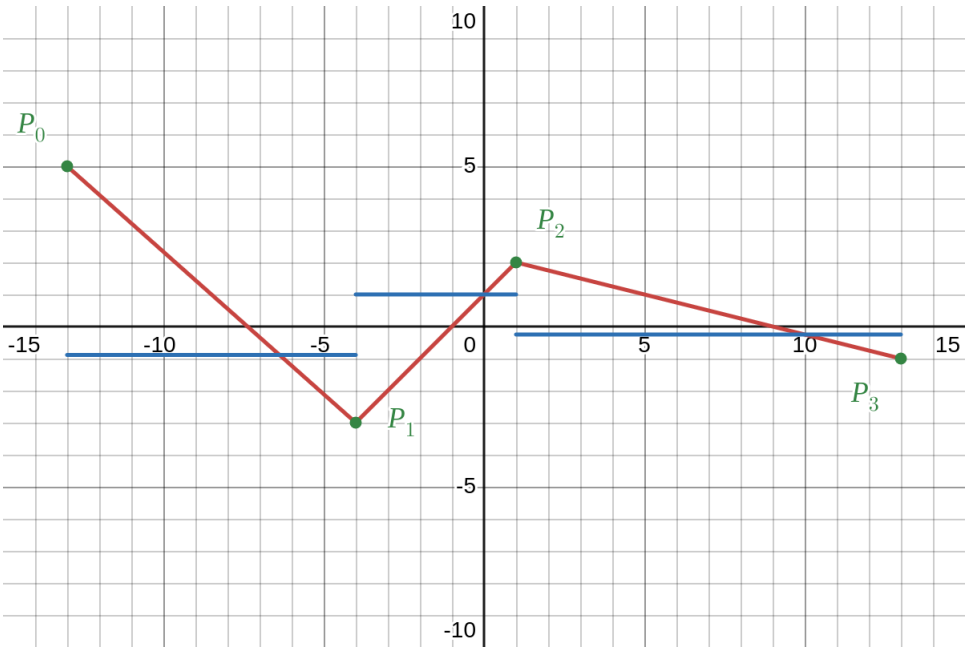
$$l_i(x) = m_i(x) + q_I, \quad i = 1, ..., k$$

In particolare, per i 4 punti che abbiamo preso vorremo definire:

$$l(x) = \begin{cases} l_1(x) = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0, & x_0 \leq x < x_1 \\ l_2(x) = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1, & x_1 \leq x < x_2 \\ l_3(x) = \frac{y_3 - y_2}{x_3 - x_2}(x - x_2) + y_2, & x_2 \leq x \leq x_3 \end{cases}$$

direttamente prendendo i fasci di rette in ogni punto x_0, x_1, x_2 (tutti tranne l'ultimo) e imponendo i coefficienti angolari.

Sul grafico, questo tipo di interpolazione (assieme alla derivata prima) avrà l'aspetto:



da dove si nota chiaramente che con questo approccio la continuità non è assicurata nemmeno al primo grado.

15.1.2 $s = 2$: spline quadratiche

Una soluzione più "liscia" si può avere sfruttando le cosiddette **spline quadratiche**:

Definizione 15.1: Spline quadratica

Dati $k+1$ punti (x_j, y_j) in un intervallo $[a, b]$ con $j = 0, ..., k$ si definisce spline quadratica ($s = 2$) relativa ai $k+1$ punti la funzione definita a tratti $S_2(x)$ che rispetta le condizioni:

1. $S_2(x)$ è un polinomio di grado 2 se ristretto ad un intervallo $[x_{i-1}, x_i]$;
2. $S_2(x_i) = y_i$ per ogni punto $i = 0, ..., k$;
3. $S_2(x) \in C^1([a, b])$.

In questi termini, l'interpolazione lineare a tratti vista finora rappresenta una sorta di interpolazione per *"spline lineari"*, mentre nella prossima sezione vedremo le *spline cubiche*. Vediamo quindi che le condizioni imposte equivalgono a dire:

$$\begin{cases} p_i(x_i) = y_i, & i = 1, ..., k \\ p_i(x_{i-1}) = y_{i-1}, & i = 1, ..., k \\ p_i'(x_i) + p_{i+1}'(x_i), & i = 1, ..., k-1 \end{cases}$$

cioè abbiamo complessivamente $3k - 1$ condizioni. Di contro, guardando alla prima condizione si ha che ogni tratto dell'interpolazione avrà forma:

$$q_i(x) = a_i x^2 + b_i x + c_i, \quad i = 1, ..., k$$

cioè $3k$ parametri complessivi.

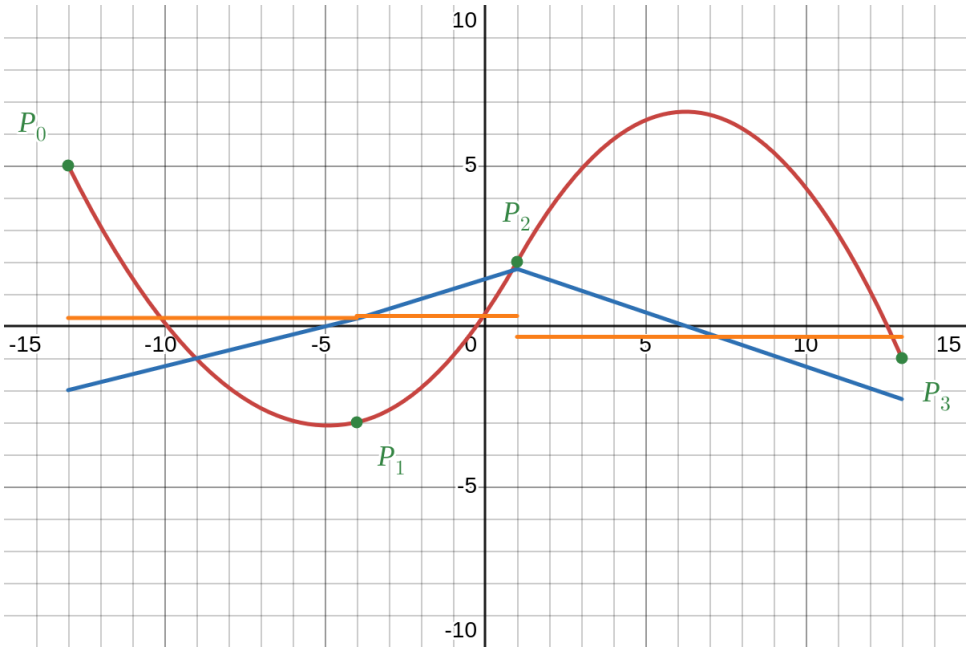
Visto che $3k$ parametri per $3k-1$ condizioni sono troppi, dobbiamo introdurre un'altra condizione: diciamo che la derivata in x_0 deve essere uguale ad un valore fisso d_0.

A questo punto basterà trovare a_i, b_i e c_i in funzione di x_{i-1}, x_i e la derivata in x_{i-1} (che chiamiamo d_{i-1}), che saranno:

$$\begin{cases} a_i = \frac{y_i - y_{i-1}}{(x_{i-1} - x_i)^2} + \frac{d_{i-1}}{x_{i-1} - x_i} \\ b_i = d_{i-1} - 2a_i x_{i-1} \\ c_i = y_i - a_i x_i^2 - b_i x_i \end{cases}$$

Calcolando tali valori per ognuno dei k tratti dell'interpolazione si trovano i parametri di ogni quadratica interpolante.

Facendo ciò sull'esempio di prima, preso $d_0 = -2$ e tracciando il grafico, si ha (assieme alla derivata prima e seconda):



da dove notiamo quindi di aver guadagnato la continuità al primo grado, ma ancora non al secondo. In verità le spline quadratiche non sono molto utili per via delle limitazioni che ci impongono, e vengono usate di rado in contesti reali. Passiamo quindi a polinomi di grado più alto, molto più di largo uso, cioè $s = 3$.

15.1.3 $s = 3$: spline cubica

Una soluzione ancora più "liscia" si può quindi avere sfruttando le cosiddette **spline cubiche**. Queste sono il tipo più comune di spline, e vengono usate in svariati contesti, fra cui ad esempio per tracciare linee "smussate" in computer grafica.

Definiamo quindi:

Definizione 15.2: Spline cubica

Dati $k+1$ punti (x_j, y_j) in un intervallo $[a,b]$ con $j = 0, ..., k$ si definisce spline cubica ($s = 3$) relativa ai $k+1$ punti la funzione definita a tratti $S_3(x)$ che rispetta le condizioni:

1. $S_3(x)$ è un polinomio di grado 3 se ristretto ad un intervallo $[x_{i-1}, x_i]$;
2. $S_3(x_i) = y_i$ per ogni punto $i = 0, ..., k$;
3. $S_3(x) \in C^2([a,b])$.

Vediamo quindi che le condizioni imposte equivalgono a dire:

$$\begin{cases} p_i(x_i) = y_i, \quad i = 1, ..., k \\ p_i(x_{i-1}) = y_{i-1}, \quad i = 1, ..., k \\ p_i'(x_i) + p_{i+1}'(x_i), \quad i = 1, ..., k-1 \\ p_i''(x_i) + p_{i+1}''(x_i), \quad i = 1, ..., k-1 \end{cases}$$

con una riga in più di condizioni rispetto al caso quadratico dato dal salto da continuità C^1 a C^2, cioè abbiamo complessivamente $4k-2$ condizioni. Di contro, guardando alla prima condizione si ha che ogni tratto dell'interpolazione avrà forma:

$$c_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \quad i = 1, ..., k$$

cioè $4k$ parametri complessivi.

Come nel caso quadratico, visto che $4k$ parametri per $4k-2$ condizioni sono troppi, dovremo introdurre 2 nuove condizioni, che modificheranno il tipo di problema che andremo a risolvere:

- **Spline naturale:** si considera questo tipo di spline quando non si ha alcun tipo di informazione oltre i punti da interpolare. In questo caso si assume quindi:

  $$p_1''(x_0) = p_k''(x_k) = 0$$

  cioè derivata seconda nulla agli estremi di interpolazione $x_0 = a$ e $x_k = b$ di $[a,b]$;

- **Spline periodica:** quando si ritiene che la funzione da approssimare sia periodica, si possono adottare le condizioni:

  $$p_1''(x_0) = p_k''(x_k), \quad p_1'(x_0) = p_k'(x_k)$$

- **Spline vincolata:** infine, se si conoscono le derivate prime negli estremi di interpolazione $x_0 = a$ e $x_k = b$ si possono imporre le condizioni:

  $$p_1'(x_0) = f'(x_0), \quad p_k'(x_k) = f'(x_k)$$

  Un altro tipo di spline vincolata potrebbe essere quello da usare nel caso in cui si conoscono derivata prima e seconda nel primo estremo di interpolazione $x_0 = a$. In questo caso si impongono le condizioni:

  $$p_1'(x_0) = f'(x_0) \quad p_1''(x_0) = f''(x_0)$$

  Questo equivale al caso che abbiamo considerato per la spline quadratica (chiamando $f'(x_0) = d_0$ e $f''(x_0) = s_0$), salvo aver dovuto introdurre un ulteriore grado di derivazione.

In ogni caso, per punti equispaziati i può dimostrare il seguente teorema:

Teorema 15.1: Esistenza di spline cubiche

Presi $x_0, ..., x_k$ nodi equispaziati su un intervallo $[a, b]$, cioè:

$$x_i = a + ih, \quad h = \frac{b-a}{k}, \quad i = 0, ..., k$$

e le valutazioni $y_0, ..., y_k$ di una certa funzione $f(x)$ nei nodi x_i, esiste unica la spline naturale $S_3(x)$ passante per i punti (x_i, y_i), $i = 0, ..., k$.

Vediamo quindi due procedimenti di calcolo per due dei casi appena definiti, cioè il primo (spline naturale) e l'ultimo (spline vincolata in x_0). Le formule per il calcolo degli altri due casi saranno in qualche modo ricavate in modo equivalente a quelle del primo caso.

- Un primo approccio può essere quello di risolvere ognuno dei k sottoproblemi di interpolazione fra x_{i-1} e x_i applicando l'interpolazione di Hermite, e portandosi dietro le derivate prime m_i (che ancora non conosciamo), interpretate come $f'(x_i)$. Si determinano quindi gli m_i imponendo le condizioni sulla derivata seconda agli estremi (abbiamo detto la *condizione naturale*).

  Presi quindi punti equispaziati come sopra, cioè:

  $$x_i = a + ih, \quad h = \frac{b-a}{k}, \quad i = 0, ..., k$$

  questo procedimento porta alla formazione di spline del tipo:

  $$p_i(x) = \left[f(x_{i-1}) + \left(m_{i-1} + \frac{2f(x_{i-1})}{h}\right)(x - x_{i-1})\right]\left(\frac{x - x_i}{h}\right)^2 + \left[f(x_i) + \left(m_i - \frac{2f(x_i)}{h}\right)(x - x_i)\right]\left(\frac{x - x_{i-1}}{h}\right)^2$$

  direttamente dall'interpolazione di Hermite, con gli m_i soluzioni del seguente sistema lineare:

  $$\begin{pmatrix} 2 & 1 & 0 & ... & 0 \\ 1 & 4 & 1 & ... & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & ... & 1 & 4 & 1 \\ 0 & ... & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_k \end{pmatrix} = \frac{3}{h} \begin{pmatrix} f(x_1) - f(x_0) \\ f(x_2) - f(x_0) \\ f(x_3) - f(x_1) \\ \vdots \\ f(x_{k-1}) - f(x_{k-2}) \\ f(x_k) - f(x_{k-1}) \end{pmatrix}$$

  da dove si nota la matrice A a *"striscia"*, data dall'interdipendenza fra i tratti della spline.

  Per la dimostrazione seguiremo i passaggi in quest'ordine:

  1. Prima troviamo una forma per il polinomio interpolante di Hermite $p_i(x)$ nei nodi x_{i-1} e x_i, noti i valori y_{i-1}, y_i e le derivate prime m_{i-1}, m_i;
  2. Poi calcoliamo la forma delle derivate seconde del polinomio interpolante agli estremi, in modo da preparare il prossimo passaggio;

3. Imponiamo le condizioni della spline cubica naturale, cioè derivate seconde agli estremi nulle e continuità C^2 nei punti intermedi.

Procediamo quindi con la dimostrazione.

1. Vorremo interpolare ogni tratto x_{i-1}, x_i con un polinomio $p_i(x)$ di terzo grado che rispetti:
$$\begin{cases} p_i(x_{i-1}) = f(x_{i-1}) \\ p_i(x_i) = f(x_i) \\ p_i'(x_{i-1}) = m_{i-1} \\ p_i'(x_i) = m_1 \end{cases}$$
dove i $k+1$ valori $m_0, m_1, ..., m_k$ sono le derivate prime che ancora non conosciamo. Avremo quindi da Hermite che vorremmo prendere le basi di Lagrange, che posto $x_i - x_{i-1} = h$ (come si ha da nodi equispaziati) sono:
$$e_{i-1}(x) = \frac{x - x_i}{x_{i-1} - x_i} = -\frac{x - x_i}{h}, \quad e_i(x) = \frac{x - x_{i-1}}{x_i - x_{i-1}} = \frac{x - x_{i-1}}{h}$$
di cui calcoliamo subito le derivate, in quanto ci serviranno in seguito:
$$e_{i-1}'(x) = -\frac{1}{h}, \quad e_i'(x) = \frac{1}{h}, \quad e_{i-1}''(x) = e_i''(x) = 0$$
Avremo quindi le basi di Hermite:
$$h_{0,i-1}(x) = (1 - 2e_{i-1}'(x_{i-1})(x - x_{i-1}))e_{i-1}^2(x) = \left(1 + 2\frac{x - x_{i-1}}{h}\right)\left(\frac{x - x_i}{h}\right)^2$$
$$h_{0,i}(x) = (1 - 2e_i'(x_i)(x - x_i))e_i^2(x) = \left(1 - 2\frac{x - x_i}{h}\right)\left(\frac{x - x_{i-1}}{h}\right)^2$$
$$h_{1,i-1}(x) = (x - x_{i-1})e_{i-1}^2(x) = (x - x_{i-1})\left(\frac{x - x_i}{h}\right)^2$$
$$h_{1,i}(x) = (x - x_i)e_i^2(x) = (x - x_i)\left(\frac{x - x_{i-1}}{h}\right)^2$$
da cui il polinomio interpolante finale, raccogliendo le basi quadrate comuni e qualche altro termine, è:
$$p_i(x) = h_{0,i-1}(x)\, y_{i-1} + h_{0,i}(x)\, y_i + h_{1,i-1}(x)\, m_{i-1} + h_{1,i}(x)\, m_i$$
$$= \left(1 + 2\frac{x - x_{i-1}}{h}\right)\left(\frac{x - x_i}{h}\right)^2 y_{i-1} + \left(1 - 2\frac{x - x_i}{h}\right)\left(\frac{x - x_{i-1}}{h}\right)^2 y_i$$
$$+ (x - x_{i-1})\left(\frac{x - x_i}{h}\right)^2 m_{i-1} + (x - x_i)\left(\frac{x - x_{i-1}}{h}\right)^2 m_i$$
$$= \left(y_{i-1} + \left(m_{i-1} + \frac{2y_{i-1}}{x_i - x_{i-1}}\right)(x - x_{i-1})\right)\left(\frac{x - x_i}{x_i - x_{i-1}}\right)^2$$
$$+ \left(y_i + \left(m_i - \frac{2y_i}{x_i - x_{i-1}}\right)(x - x_i)\right)\left(\frac{x - x_{i-1}}{x_i - x_{i-1}}\right)^2$$

che posto $y_i = f(x_i)$ è esattamente quanto dato prima:

$$p_i(x) = \left[f(x_{i-1}) + \left(m_{i-1} + \frac{2f(x_{i-1})}{h}\right)(x - x_{i-1})\right]\left(\frac{x - x_i}{h}\right)^2 + \left[f(x_i) + \left(m_i - \frac{2f(x_i)}{h}\right)(x - x_i)\right]\left(\frac{x - x_{i-1}}{h}\right)^2$$

2. A questo punto resta da ricavare il sistema lineare, che è ciò che ci darà i valori degli m_i. Per fare ciò calcoliamo la derivata seconda del polinomio interpolante, partendo dal calcolare le derivate secondo delle singole basi di Hermite:

$$h''_{0,i-1}(x) = \frac{2}{h^3}(h + 6x - 2x_{i-1} - 4x_i)$$

$$h''_{0,i}(x) = \frac{2}{h^3}(h - 6x + 2x_i + 4x_{i-1})$$

$$h''_{1,i-1}(x) = \frac{1}{h^2}(6x - 2x_{i-1} - 4x_i)$$

$$h''_{1,i}(x) = \frac{1}{h^2}(6x - 2x_i - 4x_{i-1})$$

da cui:

$$p''_i(x) = \frac{2}{h^3}(h + 6x - 2x_{i-1} - 4x_i)\, y_{i-1} + \frac{2}{h^3}(h - 6x + 2x_i + 4x_{i-1})\, y_i + \frac{1}{h^2}(6x - 2x_{i-1} - 4x_i)\, m_{i-1} + \frac{1}{h^2}(6x - 2x_i - 4x_{i-1})\, m_i$$

Ottenuta la derivata seconda del polinomio interpolante, calcoliamola nei punti x_{i-1} e x_i. Per x_{i-1}, ad esempio, si avrà:

$$\begin{aligned} p''_i(x_{i-1}) &= \frac{2}{h^3}(h + 6x_{i-1} - 2x_{i-1} - 4x_i)\, y_{i-1} + \frac{2}{h^3}(h - 6x_{i-1} + 2x_i + 4x_{i-1})\, y_i \\ &\quad + \frac{1}{h^2}(6x_{i-1} - 2x_{i-1} - 4x_i)\, m_{i-1} + \frac{1}{h^2}(6x_{i-1} - 2x_i - 4x_{i-1})\, m_i \\ &= \frac{2}{h^3}(h + 4x_{i-1} - 4x_i)\, y_{i-1} + \frac{2}{h^3}(h - 2x_{i-1} + 2x_i)\, y_i \\ &\quad + \frac{1}{h^2}(4x_{i-1} - 4x_i)\, m_{i-1} + \frac{1}{h^2}(2x_{i-1} - 2x_i)\, m_i \\ &= \frac{2}{h^3}(h - 4h)y_{i-1} + \frac{2}{h^3}(h + 2h)y_i + \frac{1}{h^2}(-4h)m_{i-1} + \frac{1}{h^2}(-2h)m_i \\ &= -\frac{6}{h^2}y_{i-1} + \frac{6}{h^2}y_i - \frac{4}{h}m_{i-1} - \frac{2}{h}m_i = \frac{6(y_i - y_{i-1})}{h^2} - \frac{4m_{i-1} + 2m_i}{h} \end{aligned}$$

e con passaggi simili si calcola in x_i, per cui:

$$p''_i(x_{i-1}) = \frac{6(y_i - y_{i-1})}{h^2} - \frac{4m_{i-1} + 2m_i}{h}$$

$$p''_i(x_i) = \frac{2m_{i-1} + 4m_i}{h} - \frac{6(y_i - y_{i-1})}{h^2}$$

3. Avremo a questo punto da applicare le condizioni della spline cubica naturale, cioè derivata seconda nulla agli estremi:

$$\begin{cases} p_1''(x_0) = 0 \implies 2m_0 + m_1 = \frac{3}{h}\left(f(x_1) - f(x_0)\right) \\ p_k''(x_k) = 0 \implies 2m_k + m_{k-1} = \frac{3}{h}\left(f(x_k) - f(x_{k-1})\right) \end{cases}$$

e derivate seconde continue nei punti intermedi:

$$p_i''(x_i) = p_{i+1}''(x_i) \implies m_{i-1} + 4m_i + m_{i+1} = \frac{3}{h} f(x_{i+1}) - f(x_{i-1})$$

da cui il sistema lineare che corredava la forma del polinomio di Hermite, e quindi la tesi. □

- Se si vuole imporre il vincolo "a sinistra" della derivata prima e seconda, si può procedere in maniera analoga a quanto fatto nel caso quadratico, cioe prendere i tratti polinomiali:

$$c_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \quad i = 1, ..., k$$

e trovare gli a_i, b_i, c_i e d_i in funzione di x_{i-1}, x_i e le due derivate consecutive in x_{i-1} (che chiamiamo d_{i-1} e s_{i-1}, rispettivamente primo e secondo grado).

Visto che trovare direttamente i quattro problemi può essere difficile, decidiamo invece di trovare le 4 basi ortogonali b_1, b_2, b_3 e b_4 (di cui le prime 3 equivalenti a quelle di Hermite, e l'ultima corrispondente alla condizione derivata seconda in x_{i-1}), che rispettano quindi le condizioni:

$$b_1 : \begin{cases} b_1(x_{i-1}) = 1 \\ b_1(x_i) = 0 \\ b_1'(x_{i-1}) = 0 \\ b_1''(x_{i-1}) = 0 \end{cases}, \quad b_2 : \begin{cases} b_2(x_{i-1}) = 0 \\ b_2(x_i) = 1 \\ b_2'(x_{i-1}) = 0 \\ b_2''(x_{i-1}) = 0 \end{cases}$$

$$b_3 : \begin{cases} b_3(x_{i-1}) = 0 \\ b_3(x_i) = 0 \\ b_3'(x_{i-1}) = 1 \\ b_3''(x_{i-1}) = 0 \end{cases}, \quad b_4 : \begin{cases} b_3(x_{i-1}) = 0 \\ b_3(x_i) = 0 \\ b_3'(x_{i-1}) = 0 \\ b_3''(x_{i-1}) = 1 \end{cases}$$

Queste si potranno ricavare, direttamente dall'imposizione delle condizioni, come:

1. $b_1\left(x, x_{i-1}, x_i\right) = a_{b1}\left(x_{i-1}, x_i\right) x^3 + b_{b1}\left(x_{i-1}, x_i\right) x^2 + c_{b1}\left(x_{i-1}, x_i\right) x + d_{b1}\left(x_{i-1}, x_i\right)$, con:
   - $a_{b1}\left(x_{i-1}, x_i\right) = \frac{1}{\left(x_{i-1} - x_i\right)^3}$;
   - $b_{b1}\left(x_{i-1}, x_i\right) = -3 a_{b1}\left(x_{i-1}, x_i\right) x_{i-1}$;
   - $c_{b1}\left(x_{i-1}, x_i\right) = 3 a_{b1}\left(x_{i-1}, x_i\right) x_{i-1}^2$;
   - $d_{b1}\left(x_{i-1}, x_i\right) = 1 - a_{b1}\left(x_{i-1}, x_i\right) x_{i-1}^3$.
2. Questa è banalmente $b_2\left(x, x_{i-1}, x_i\right) = 1 - b_1\left(x, x_{i-1}, x_i\right)$;
3. $b\left(x, x_{i-1}, x_i\right) = a_{b3}\left(x_{i-1}, x_i\right) x^3 + b_{b3}\left(x_{i-1}, x_i\right) x^2 + c_{b3}\left(x_{i-1}, x_i\right) x + d_{b3}\left(x_{i-1}, x_i\right)$, con:

- $a_{b3}(x_{i-1}, x_i) = \frac{x_{i-1}-x_i}{(x_i-x_{i-1})^3}$;
- $b_{b3}(x_{i-1}, x_i) = -3a_{b3}(x_{i-1}, x_i)\, x_{i-1}$;
- $c_{b3}(x_{i-1}, x_i) = 1 + 3a_{b3}(x_{i-1}, x_i)\, x_{i-1}^2$;
- $d_{b3}(x_{i-1}, x_i) = -a_{b3}(x_{i-1}, x_i)\, x_{i-1}^3 - x_{i-1}$.

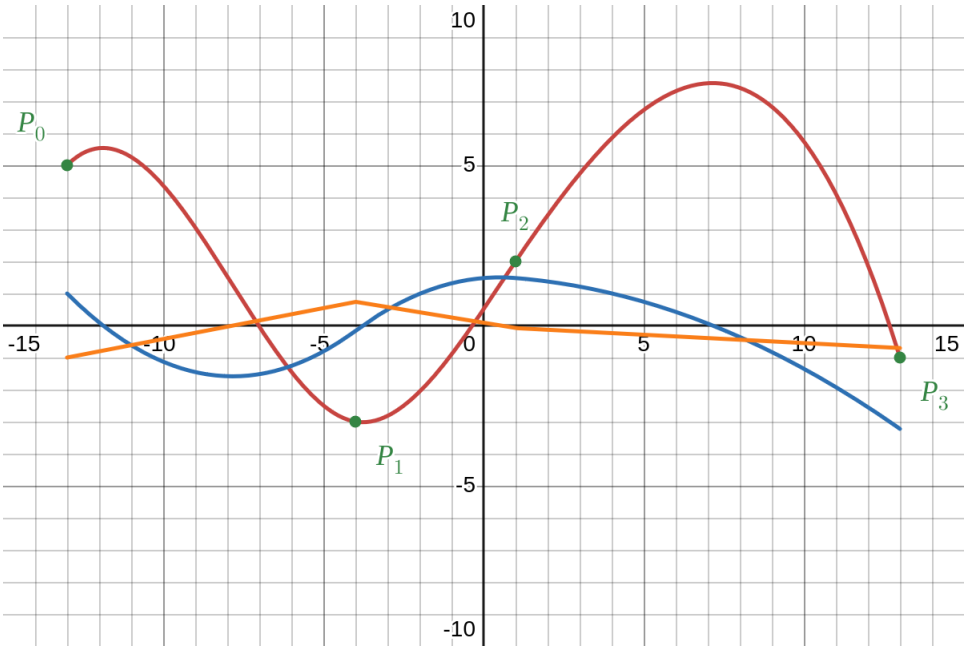
4. $b(x, x_{i-1}, x_i) = a_{b4}(x_{i-1}, x_i)\, x^3 + b_{b4}(x_{i-1}, x_i)\, x^2 + c_{b4}(x_{i-1}, x_i)\, x + d_{b4}(x_{i-1}, x_i)$, con:
   - $a_{b4}(x_{i-1}, x_i) = \frac{(x_i-x_{i-1})^2}{2(x_{i-1}-x_i)^3}$;
   - $b_{b4}(x_{i-1}, x_i) = \frac{1}{2} - 3a_{b4}(x_{i-1}, x_i)\, x_{i-1}$;
   - $c_{b4}(x_{i-1}, x_i) = 3a_{b4}(x_{i-1}, x_i)\, x_{i-1}^2 - x_{i-1}$;
   - $d_{b4}(x_{i-1}, x_i) = \frac{1}{2}x_{i-1}^2 - a_{b4}(x_{i-1}, x_i)\, x_{i-1}^3$.

A questo punto basterà definire ogni tratto polinomiale c_i come:

$$c_i(x) = b(x, x_{i-1}, x_i)\, y_{i-1} + b(x, x_{i-1}, x_i)\, y_i + b(x, x_{i-1}, x_i)\, d_{i-1} + b(x, x_{i-1}, x_i)\, s_{i-1}$$

dove gli d_{i-1} e s_{i-1} sono rispettivamente la derivata prima e seconda in ogni punto, data per il punto x_0 e ricavata dal tratto precedente per ogni tratto successivo.

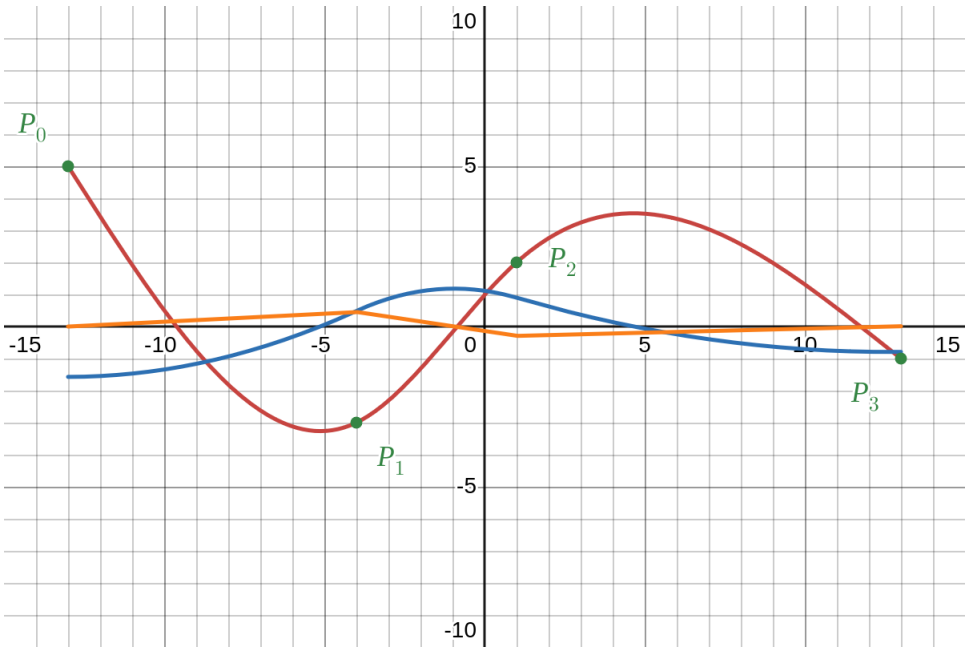
Un interpolazione di questo tipo per l'esempio considerato nelle scorse sezioni ha l'aspetto, prese $d_0 = 1$ e $s_0 = -1$ e tracciate derivata prima e seconda:



da dove notiamo che la derivata fino al secondo grado resta continua, da cui la condizione 3 è rispettata.

Vediamo che in verità si può risolvere in qualche modo anche lo scorso tipo di problema con questo metodo, semplicemente imponendo $s_0 = 0$ e cercando d_0 perché risulti $s_k = 0$.

Nell'esempio, questo si ottiene per $d_0 \approx -1.574$, con relativo grafico:



Esempi di questi tipi di interpolazione si possono trovare sempre su Desmos, al link `https://www.desmos.com/calculator/gm1idqauiy` (da qui sono state generate le figure).

15.2 Approssimazione ai minimi quadrati

Un altro approccio per l'approssimazione di insiemi di punti, specialmente nel caso questi siano imponenti in dimensioni, può essere dato dall'**approssimazione ai minimi quadrati**.

Un caso banale di questo tipo di approssimazione può essere quello della *regressione lineare*: dato un insieme di $k+1$ punti (x_j, y_j), si cerca la retta che minimizza la distanza quadratica da ogni punto. In qusto caso la retta prende il nome di **funzione modello**.

Generalizziamo quindi questo approssimazione in 2 direzioni:

- Ammettiamo che si abbiano più punti che funzioni modello (che possiamo intendere come le basi polinomiali usate finora);
- Amettiamo l'utilizzo di funzioni modello non polinomiali.

Formalmente, quindi, dati $k+1$ punti $(x_0, y_0), ..., (x_k, y_k)$ punti con $y_j = f(x_j)$ e $m+1$ funzioni modello $G_0(x), ..., G_m(x)$ con $m \leq x$ cercheremo l'approssimante $\Phi(x) \approx f(x)$ della forma:

$$\Phi(x) = \sum_{i=0}^{m} G_i(x) \cdot c_i, \quad c_i \in \mathbb{R}$$

con c_i da trovare, cioè la combinazione lineare ottima (definiremo cosa significa ottima fra poco) delle funzioni modello.

- Ad esempio, la regressione lineare quella data dalle funzioni modello:

  $$G_0(x) = 1, \quad G_1(x) = x$$

  approssimante $k+1 = \#$ punti nel grafico;

- Potremmo avere altri tipi di funzioni modello. Ad esempio potremmo avere:

$$G_0(x) = e^{2x}, \quad G_1(x) = \sin\left(\frac{x}{2}\right), \quad G_2(x) = \frac{1}{3x^2 + 4}$$

  In questo caso l'approssimante avrebbe una forma del tipo:

$$\Phi(x) = c_0 e^{2x} + c_1 \sin\left(\frac{x}{2}\right) + \frac{c_2}{3x^2 + 4}$$

A questo punto il problema è capire cosa significa dire che $\Phi(x)$ passa *"vicino"* ai punti (x_j, y_j), cioè qual'è la funzione da ottimizzare per trovare i c_i ottimi in modo che risulti $\Phi(x) \approx f(x)$. Dobbiamo quindi definire una qualche funzione di errore $\psi(c_0, ..., c_m) : \mathbb{R}^{m+1} \to \mathbb{R}^+$.

Vediamo come definire tale funzione.

- Un primo approccio potrebbe essere il semplice *scarto*:

$$\psi(c_0, ..., c_m) = \sum_{i=0}^{k} (\Phi(x_i) - y_i)$$

  Questa ha il problema di poter dare cancellazione, cioè errori di punti diversi potrebbero annullarsi, per dare valori di ψ bassi quando in realtà la funzione è molto lontana da $f(x)$;

- Un approccio migliore è quindi quello di prendere lo *scarto assoluto*:

$$\psi(c_0, ..., c_m) = \sum_{i=0}^{k} \left|\Phi(x_i) - y_i\right|$$

  In questo caso non soffriremo di cancellazione, ma avremo il problema che ψ non è differenziabile (si avranno punti angolosi dati dal valore assoluto);

- L'approccio migliore risulta quindi:

$$\psi(c_0, ..., c_m) = \sum_{i=0}^{k} (\Phi(x_i) - y_i)^2$$

  cioè lo *scarto quadratico*, che risolve sia i problemi della cancellazione che della continuità C^1.

Il metodo dei minimi quadrati consisterà quindi nello scegliere:

$$c = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_m \end{pmatrix} \in \mathbb{R}^{m+1}$$

tale che:

$$c = \min_{c \in \mathbb{R}^{m+1}} \psi(c)$$

cioè c è punto di minimo di $\psi(c)$.

Dovremo quindi minimizzare:

$$\psi(c) = \sum_{i=0}^{k} \left(\Phi(x_i) - y_i\right)^2 = \sum_{i=0}^{k} \left(\sum_{j=0}^{m} c_j \cdot G_j(x_i) - y_i \right)^2$$

cioè in forma matriciale, posti:

$$A = \begin{pmatrix} G_0(x_0) & G_1(x_0) & ... & G_m(x_0) \\ \vdots & & & \vdots \\ G_0(x_0) & G_1(x_1) & ... & G_m(x_k) \end{pmatrix}, \quad c = \begin{pmatrix} c_0 \\ \vdots \\ c_m \end{pmatrix}, \quad b = \begin{pmatrix} y_0 \\ \vdots \\ y_k \end{pmatrix}$$

con $A \in \mathbb{R}^{(k+1)\times(m+1)}$, $c \in \mathbb{R}^{m+1}$, $b \in \mathbb{R}^{k+1}$, equivale a:

$$\psi(c) = \sum_{i=0}^{k} \left((Ac)_i - b_i\right)^2 = \left|Ac - b\right|_2^2$$

Stiamo quindi minimizzando la norma quadratica del residuo di $Ac = b$, che dalle dimensioni della matrice A e i vettori c e b è un sistema sovradeterminato (più equazioni che incognite):

$$c^* = \min_{c \in \mathbb{R}^{m+1}} |Ac - b|_2$$

Possiamo quindi risolvere interpretando il sistema come uno di quelli visti in 11.1, quindi sfruttando le **equazioni normali** o la **decomposizione QR**.

15.2.1 Implementazione MATLAB di un approssimatore ai quadrati minimi

Vediamo quindi come implementare questo approccio in MATLAB.

Abbiamo già a disposizione una funzione, `least_squares()`, definita in 12.2.9 per la risoluzione di sistemi sovradeterminati usando la decomposizione QR. Ci mancherà quindi da scrivere una funzione che calcola il valore della matrice A, valutando le funzioni in base G_i, e che campiona una certa funzione $f(x)$ in determinati punti x_i per generare un vettore di y_i, magari usando la `linspace()`, così da avere un insieme di nodi equispaziati del tipo:

$$x_i = l + ih, \quad i = 0, ..., k-1$$

$$h = \frac{u-l}{k-1}$$

Per il problema di come rappresentare le G_i usiamo dei *function handle*, dichiarati come segue:

```
% un function handle alla funzione f(x) = x^2
f = @(x) x.^2
% si usa come: f(2) -> 4
```

Una base potrà quindi essere costruita racchiudendo più handle di questo tipo in un *array di celle*.

Prendiamo per adesso la base:

$$G_0(x) = e^x, \quad G_1(x) = \frac{1}{x}, \quad G_2(x) = \sin(x)$$

Avremo allora una funzione del tipo:

```
function c = approx_trasc(f, l, u, n)
    % le funzioni in base
    funcs = {
      @(x) exp(x);
      @(x) 1/x;
      @(x) sin(x)
    };

    % costruisce la matrice G
    function G = build_G(X)
        m = height(funcs);

        G = zeros(n, m);

        for i = 1:n
            for j = 1:m
                G(i, j) = funcs{j}(X(:, i));
            end
        end
    end

    % campiona f
    X = linspace(l, u, n);
    Y = f(X);

    % costruisci G
    G = build_G(X);

    % usa il risolutore ai quadrati minimi
    c = least_squares(G, Y');
end
```

Attraverso questa funzione potremo calcolare i coefficienti c che minimizzano lo scarto quadratico sugli `n` punti fra `l` e `u`.

Prendiamo l'esempio della funzione:

$$f(x) = (2x - 3)(x - 3)(x - 4)(x - 5)$$

Definiremo questa, in MATLAB, come segue:

```
f = @(x) (2.*x - 3).*(x - 3).*(x - 4).*(x - 5);
```

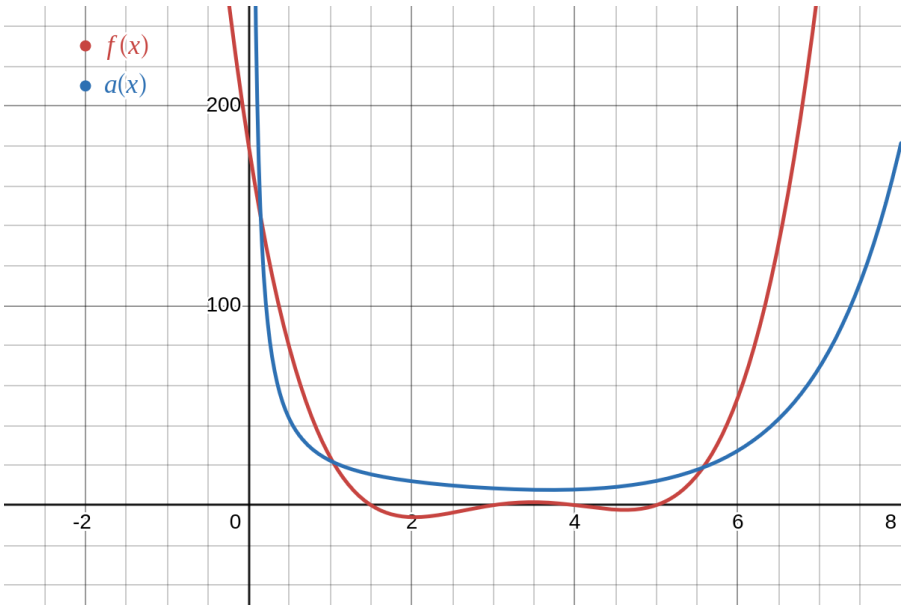
Potremo allora trovare i coefficienti c di un approssimazione in forma:

$$a(x) = c_0 \cdot e^x + c_1 \cdot \frac{1}{x} + c_2 \cdot \sin(x)$$

usando la funzione MATLAB appena implementata:

```
>> approx_trasc(f, 0.1, 6, 100)

ans =
    0.0596
   21.2591
    1.8092
```

cioè $c_0 = 0.0596$, $c_1 = 21.2591$, $c_2 = 1.8092$, che sul grafico dà:



Per un errore medio di:

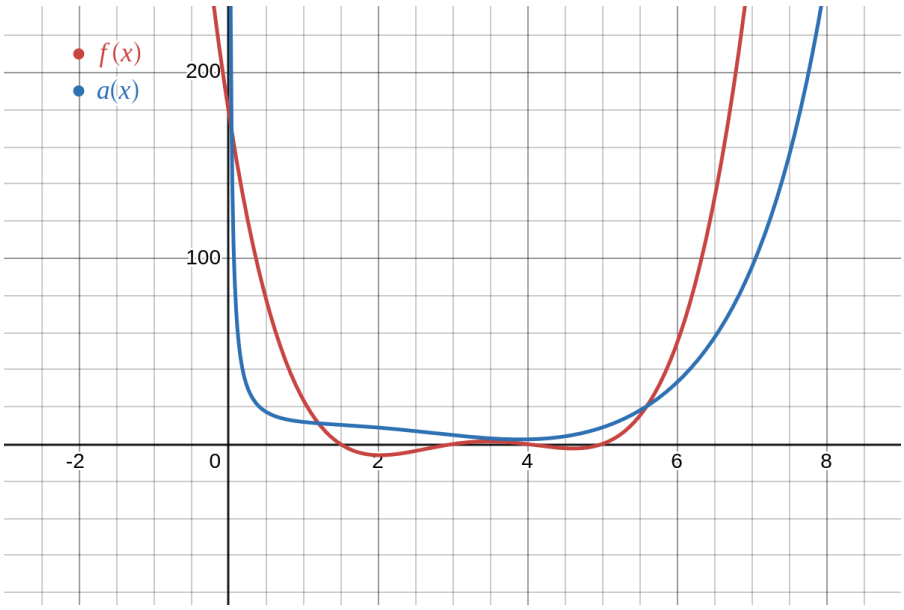
$$\varepsilon = \sqrt{\sum_{i=i}^{100} (f(x_i) - a(x_i))^2} \approx 163.141$$

Notiamo quindi che la funzione non passa necessariamente per nessuno dei punti campianati, ma riduce l'errore medio lungo tutta la regione campionata.

Potremmo renderci conto che la maggior parte dell'errore è data dal fatto che cerchiamo di approssimare la funzione nella regione $0.1 \sim 0.7$, che ci porta ad avere un errore più grande nella zona vicino all'asse x. Se fossimo interessati ad ottenere un'approssimazione più accurata in questa regione, potremmo richiamare la funzione aumentando il bound l:

```
>> approx_trasc(f, 0.7, 6, 100)

ans =
    0.0825
    7.4361
    5.0514
```

cioè $c_0 = 0.0825$, $c_1 = 7.4361$, $c_2 = 5.0514$, che sul grafico dà:



Per un errore medio di:

$$\varepsilon \approx 106.392$$

sul nuovo intervallo, che è già più accurato.

16 Lezione del 28-04-25

16.1 Approssimazione di integrali

Vorremo quindi approssimare funzioni del tipo:

$$I(\rho \cdot f) = \int_a^b f(x) \cdot \rho(x)\, dx$$

dove $f : [a,b] \to \mathbb{R}$ continua $\in C([a,b])$, mentre $\rho(x) : [a,b] \to \mathbb{R}$ sempre continua $\in C([a,b])$ è una funzione particolare, spesso la funzione unitaria o comunque una funzione detta **funzione peso** tale che:

- $\rho(x)$ è positiva:
$$\rho(x) \geq 0$$
- $\rho(x)$ rispetta le condizioni:
$$m_k = \int_a^b x^k \rho(x)\, dx < +\infty, \quad \forall k = 0, 1, ...$$

Veniamo alle motivazioni dell'approssimazione di integrali. Potremmo voler approssimare integrali del tipo:

$$\int_a^b f(x)\, dx \quad (\rho(x) = 1)$$

per una serie di motivi:

- Di molte funzioni non si può trovare un'espressione semplice della primitiva di f (funzioni ellittiche, funzioni su più variabili, ecc...);

- Anche se esiste la primitiva potrebbe essere particolarmente oneroso in termini di risorse computazioniali calcolarla o valutarla in un punto per ottenere l'integrale;
- Come nel caso dell'approssimazione e dell'interpolazione, ci sono casi in cui della f si conoscono solo alcuni punti, cioè non se ne ha un'espressione esplicita che possiamo integrare. Vedremo che sarà questo il caso più comune.

Prendiamo quindi il caso dove conosciamo una serie di punti $(x_i, f(x_i))$ di $f(x)$. L'idea per approssimare l'integrale sarà di considerare una formula del tipo:

$$\int_a^b f(x)\rho(x)\,dx \approx \sum_{i=0}^{n} f(x_i)\cdot a_i$$

con:

$$a \leq x_0 < x_1 < ... < x_n \leq b$$

In questo caso chiamiamo gli x_i **nodi** e gli a_i **pesi**, e la formula **formula di quadratura**. Si ha quindi che una formula di quadratura è univocamente determinata una volta decisi nodi e pesi.

Definiamo quindi formalmente:

Definizione 16.1: Formula di quadratura

Dati un intervallo $[a, b]$ e $\rho(x)$, definiamo $J_n(\circ)$ formula di quadratura su $(n+1)$ nodi $x_0, ..., x_n$ con pesi $a_0, ..., a_n$ la funzione:

- Di questa forma:
$$J_n : C([a,b]) \to \mathbb{R}$$
tale che:
$$f \to \sum_{i=0}^{n} f(x_i)\cdot a_i$$
- O equivalentemente:
$$J_n : \mathbb{R}^{n+1} \to \mathbb{R}$$
tale che:
$$\begin{pmatrix} f_0 \\ \vdots \\ f_n \end{pmatrix} \to \sum_{i=0}^{n} f_i \cdot a_i$$

Data la definizione di formula di quadratura, possiamo quindi definire l'errore:

Definizione 16.2: Errore della formula di quadratura

L'errore della formula di quadratura $J_n(\circ)$ si definisce come:

$$E_n(f) = \int_a^b f(x)\rho(x)\,dx - J_n(f) = \int_a^b f(x)\rho(x)\,dx - \sum_{i=0}^{n} f(x_i)\cdot a_i$$

anche questa tale che:

$$E_n : C([a,b]) \to \mathbb{R}$$

con:

$$f \to E_n(f)$$

Ossserviamo che sia J_n che E_n sono funzioni lineari, in quanto date $f_1, f_2 \in C([a,b])$ e $c_1, c_2 \in \mathbb{R}$:

$$J_n(c_1 f_1 + c_2 f_2) = \sum_{i=0}^{n} a_i (c_1 f_1 + c_2 f_2)(x_i) = \sum_{i=0}^{n} a_i (c_1 f_1(x_i) + c_2 f_2(x_i))$$

$$= \sum_{i=0}^{n} (a_i c_1 f_1(x_i) + a_i c_2 f_2(x_i)) = c_1 \sum_{i=0}^{n} a_i f_1(x_i) + c_2 \sum_{i=0}^{n} a_i f_2(x_i) = c_1 J_n(f_1) + c_2 J_n(f_2)$$

e analogamente con E_n. □

Potremmo quindi chiederci quando una formula di quadratura è accurata. Potremmo definire un primo indicatore detto **grado di precisione**.

Definizione 16.3: Grado di precisione

Data J_n formula di quadratura definiamo grado di precisione (a volte detto *grado di precisione algebrico*) il naturale $m \in \mathbb{N}$ tale che:

$$E_n(1) = E_n(x) = ... = E_n(x^m) = 0$$

presi i monomi $x_0, ..., x^m$, cioè per cui:

$$E_n(x^{m+1}) \neq 0$$

Osserviamo che per la proprietà di linearità di E_n e J_n appena dimostrata, si ha che J_n ha grado di precisione m se e solo se J_n integra esattamente tutti i polinomi di grado $\leq m$ (che altro non sono che combinazioni lineari dei monomi $x_0, ..., x^m$ appena considerati).

16.1.1 Formula dei trapezi

Prendiamo ad esempio $\rho = 1$, l'intervallo $[a,b] = [-1,1]$ con $n = 1$, per cui consideriamo solo gli estremi $x_0 = -1$, $x_1 = 1$. In questo caso avremo la formula di quadratura:

$$J_1(f) = a_0 f(-1) + a_1 f(1) \approx \int_{-1}^{1} f(x)\,dx$$

Potremmo chiederci qual'è la migliore scelta dei coefficienti di peso a_i. Questi saranno chiaramente quelli che massimizzano il grado di precisione m della formula di quadratura J_m. Vorremmo quindi imporre due condizioni:

$$\begin{cases} E_n(1) = 0 \\ E_n(x) = 0 \end{cases} \rightarrow \begin{cases} \int_{-1}^{1} 1\,dx - (a_0 + a_1) = 0 \\ \int_{-1}^{1} x\,dx - (-a_0 + a_1) = 0 \end{cases} \rightarrow \begin{cases} 2 - a_0 - a_1 = 0 \\ 0 + a_0 - a_1 = 0 \end{cases}$$

i da cui risolvendo il sistema si ha:

$$a_0, a_1 = 1$$

Il risultato sarà quindi che la formula di quadratura è:

$$J_1(f) = f(-1) + f(1)$$

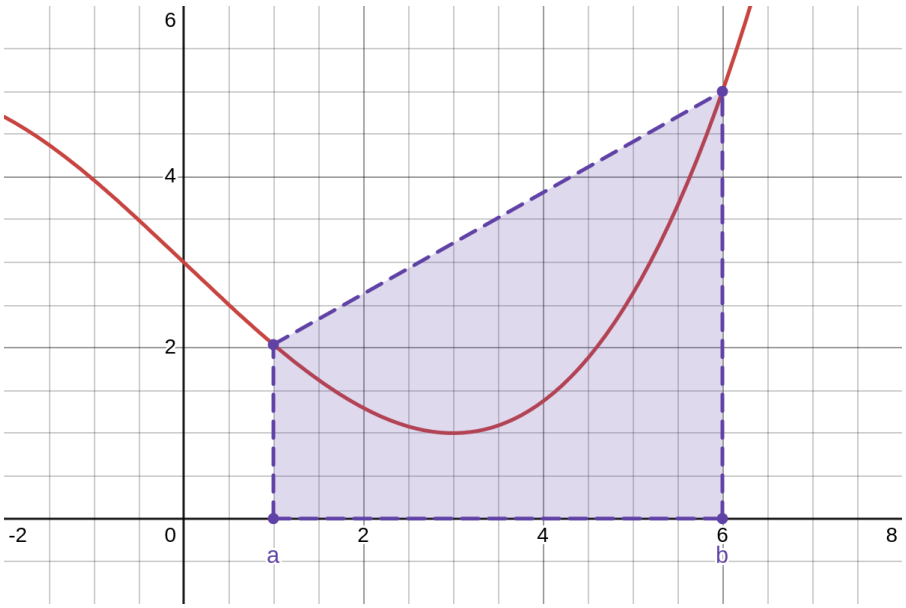
con grado di precisione almeno 1 ($m \geq 1$).

Verifichiamo infatti cosa accade per gradi $> m$, ad esempio grado 2:

$$E_n(x^2) = \int_{-1}^{1} x^2\,dx - (1+1) = \frac{2}{3} - 2 \neq 0$$

per cui m è esattamente 1.

Graficamente, questo non significherà altro che approssimare l'integrale fra -1 e 1 attraverso l'area sottesa alla retta passante per i punti $(-1, f(-1))$, $(1, f(1))$, cosa che chiaramente risulta inadeguata quando f è di grado maggiore a 1 (come ad esempio una parabola). Vediamo infatti su un grafico che dato a e b consideriamo l'area evidenziata in viola:



Quest'area è effettivamente un trapezio, motivo per cui la formula J_1 sul generico intervallo $[a, b]$ viene detta **formula dei trapezi**:

$$(f(a) + f(b))\,\frac{b-a}{2}$$

Possiamo dimostrare questa in 2 modi:

- Applicando direttamente l'area del trapezio, cioè:

$$(b+B)\frac{h}{2} = (f(a) + f(b))\,\frac{b-a}{2}$$

□

- Alternativamente possiamo ricorrere all'imposizione di un sistema simile a quello visto prima all'intervallo generico $[a, b]$:

$$\begin{cases} E_n(1) = 0 \\ E_n(x) = 0 \end{cases} \rightarrow \begin{cases} \int_a^b 1\,dx - a_0 - a_1 = 0 \\ \int_a^b x\,dx - a \cdot a_0 - b \cdot a_1 = 0 \end{cases}$$

$$\rightarrow \begin{cases} b - a = a_0 + a_1 \\ \frac{x^2}{2}\Big|_a^b = a \cdot a_0 + b \cdot a_1 \end{cases} \rightarrow \begin{cases} b - a = a_0 + a_1 \\ \frac{b^2}{2} - \frac{a^2}{2} = a \cdot a_0 + b \cdot a_1 \end{cases}$$

da cui si ottiene il sistema:

$$\begin{cases} a_0 + a_1 = b - a \\ a \cdot a_0 + b \cdot a_1 = \frac{b^2 - a^2}{2} \end{cases}$$

Prendiamo ad esempio a_0 dalla prima equazione:

$$a_0 = b - a - a_1$$

che sostituendo nella seconda equazione dà:

$$a(b - a - a_1) + ba_1 = \frac{b^2 - a^2}{2} \Rightarrow ab - a^2 - aa_1 + ba_1 = \frac{b^2 - a^2}{2}$$

$$\implies (b - a)a_1 = \frac{b^2 - a^2}{2} - ab + a^2 = \frac{a^2 - 2ab + b^2}{2} = \frac{(b - a)^2}{2}$$

dove guardando gli estremi si ha:

$$a_1 = \frac{b - a}{2}$$

Dalla prima equazione, poi, è immediato che anche:

$$a_0 = \frac{b - a}{2}$$

da cui la formula dei trapezi. □

16.1.2 Formula di Simpson

Vediamo di raffinare la formula dei trapezi aggiungendo un nodo in più, ad esempio il punto intermedio fra gli estremi dell'intervallo $[a, b]$. Nel caso $[-1, 1]$, questo non sarà altro che 0, per cui $x_0 = -1$, $x_1 = 0$, $x_2 = 1$. Avremo che la formula di quadratura è:

$$J_2(f) = a_0 f(-1) + a_1 f(0) + a_2 f(1)$$

e scegliamo gli $a_0, a_1, a_2 \in \mathbb{R}$ per massimizzare il grado di precisione, con procedimento analogo a prima:

$$\begin{cases} E_n(1) = 0 \\ E_n(x) = 0 \\ E_n(x^2) = 0 \end{cases} \rightarrow \begin{cases} 2 = a_0 + a_1 + a_2 \\ 0 = -a_0 + a_2 \\ \frac{2}{3} = a_0 + a_2 \end{cases}$$

da cui:

$$a_0 = a_2 = \frac{1}{3}, \quad a_1 = \frac{4}{3}$$

Il risultato sarà quindi che la formula di quadratura è:

$$J_2(f) = \frac{1}{3}\left(f(-1) + 4f(0) + f(1)\right)$$

detta anche **formula di Simpson** o di *Cavalieri-Simpson*.

Vediamo cosa succede per l'integrale del monomio di grado 3 attraverso la formula di Simpson:

$$E_2(x^3) = \int_{-1}^{1} x^3 \, dx - \frac{1}{3}\left(-1 + 4 \cdot 0 + 1\right) = 0$$

cioè abbiamo grado non solo ≥ 2, ma anche ≥ 3, per cui consideriamo il grado 4:

$$E_2(x^4) = \int_{-1}^{1} x^4\,dx - \frac{1}{3}(1 + 4 \cdot 0 + 1) = \frac{2}{5} - \frac{2}{3} \neq 0$$

per cui abbiamo che il grado di precisione della formula di Simpson è esattamente $m = 3$.

Infine generalizziamo la formula di Simspon all'intervallo $[a, b]$:

$$\frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Questa si dimostra per calcolo diretto, imponendo le condizioni sul generico intervallo $[a, b]$:

$$\begin{cases} E_n(1) = 0 \\ E_n(x) = 0 \\ E_n(x^2) = 0 \end{cases} \rightarrow \begin{cases} \int_a^b 1\,dx - a_0 - a_1 - a_2 = 0 \\ \int_a^b x\,dx - a \cdot a_0 - \frac{a+b}{2}a_1 - b \cdot a_2 = 0 \\ \int_a^b x^2\,dx - a^2 \cdot a_0 - \frac{(a+b)^2}{4}a_1 - b^2 \cdot a_2 = 0 \end{cases}$$

$$\rightarrow \begin{cases} b - a = a_0 + a_1 + a_2 \\ \frac{b^2}{2} - \frac{a^2}{2} = a \cdot a_0 + \frac{a+b}{2}a_1 + b \cdot a_2 \\ \frac{b^3}{3} - \frac{a^3}{3} = a^2 \cdot a_0 + \frac{(a+b)^2}{4}a_1 + b^2 \cdot a_2 \end{cases}$$

Abbiamo che dalla prima equazione si ricava:

$$a_0 = b - a - a_1 - a_2$$

mentre dalla seconda si ha, sostituendo:

$$\frac{b^2 - a^2}{2} = a(b - a - a_1 - a_2) + \frac{a+b}{2}a_1 + ba_2 =$$

$$= ab - a^2 - a \cdot a_1 - a \cdot a_2 + \frac{a+b}{2}a_1 + ba_2$$

$$\implies \frac{b^2 - a^2}{2} - ab + a^2 = -a \cdot a_1 - a \cdot a_2 + \frac{a+b}{2}a_1 + b \cdot a_2$$

$$\Rightarrow \frac{b^2 - a^2 - 2ab + 2a^2}{2} = a_1\left(-a + \frac{a+b}{2}\right) + a_2(b-a) \Rightarrow \frac{(b-a)^2}{2} = a_1\left(\frac{b-a}{2}\right) + a_2(b-a)$$

da cui:

$$a_1 = b - a - 2a_2$$

Sostituendo questo nella formula trovata prima per a_0 si trova quindi:

$$a_0 = b - a - b + a + 2a_2 - a_2 = a_2$$

cioè valgono le condizioni:

$$\begin{cases} a_0 = a_2 \\ a_1 = b - a - 2a_0 = b - a - 2a_2 \end{cases}$$

Usiamo queste per trovare a_0 (o equivalentemente a_2) dalla terza equazione. Avremo in questo caso:

$$\frac{b^3}{3} - \frac{a^3}{3} = a^2 \cdot a_0 + \frac{(a+b)^2}{4}a_1 + b^2 \cdot a_0 = a^2 \cdot a_0 + \frac{(a+b)^2}{4}(b - a - 2a_0) + b^2 \cdot a_0$$

$$= a^2 \cdot a_0 + b^2 \cdot a_0 + \frac{(a+b)^2}{4}(b-a) - \frac{(a+b)^2}{2}a_0$$

dove vorremo dividere i termini fra parte sinistra e destra come:

$$\frac{b^3-a^3}{3} - \frac{(a+b)^2}{4}(b-a) = a^2 \cdot a_0 + b^2 \cdot a_0 - \frac{(a+b)^2}{2}a_0$$

Valutiamo queste separatamente.

- **Parte sinistra**:

$$\frac{b^3-a^3}{3} - \frac{(a+b)^2}{4}(b-a) = \frac{b^3-a^3}{3} - \frac{a^2+2ab+b^2}{4}(b-a)$$

$$= \frac{b^3-a^3}{3} - \frac{a^2b - a^3 + 2ab^2 - 2a^2b + b^3 - ab^2}{4} = \frac{b^3-a^3}{3} + \frac{-b^3+a^3+a^2b-ab^2}{4}$$

$$= \frac{1}{12}b^3 - \frac{1}{12}a^3 + \frac{1}{4}a^2b - \frac{1}{4}ab^2 = \frac{1}{12}(b-a)^3$$

- **Parte destra**:

$$a^2 \cdot a_0 + b^2 \cdot a_0 - \frac{(a+b)^2}{2}a_0 = a^2 \cdot a_0 + b^2 \cdot a_0 - \left(\frac{a^2}{2} + ab + \frac{b^2}{2}\right)a_0$$

$$= \frac{a^2}{2}a_0 + \frac{b^2}{2}a_0 - ab \cdot a_0 = \frac{(b-a)^2}{2}a_0$$

da cui avremo quindi complessivamente:

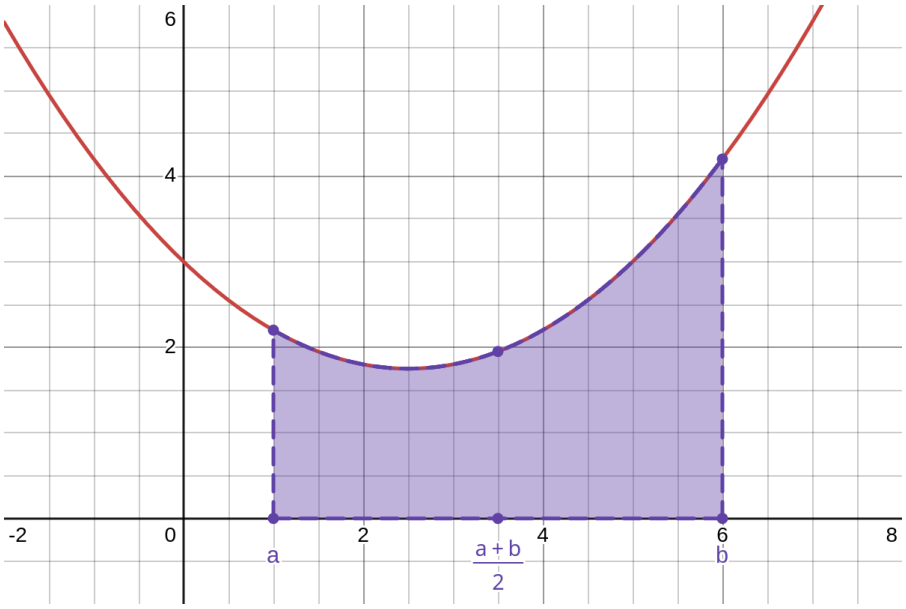
$$\frac{1}{12}(b-a)^3 = \frac{(b-a)^2}{2}a_0 \implies a_0 = \frac{b-a}{6}$$

da cui si ha immediatamente che:

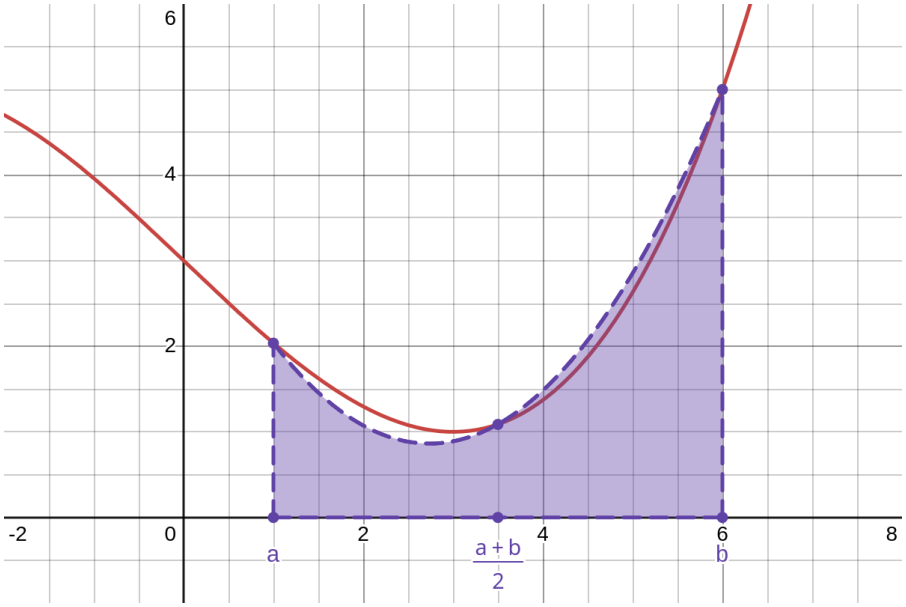
$$a_0 = a_2 = \frac{2}{3}(b-a)$$

che dà esattamente la formula di Simpson. □

Abbiamo quindi che anche Simspon si basa sul trovare implicitamente il polinomio interpolante e poi integrare quello, con la caratteristica aggiunta che anche al grado 3 l'integrale risulta esatto (anche se non lo è l'interpolante). In particolare, vediamo sul grafico che l'interpolante per una quadratica da il risultato esatto:



e che per una cubica, anche se l'interpolante non coincide con la funzione, si ha comunque risultato esatto. Questo è dato dal fatto che si creano due regioni discordi che si compensano fra di loro:



16.1.3 Formula Gaussiana

Rivediamo quindi il problema di massimizzare il grado di precisione dati i nodi $x_0, ..., x_n$. Questo significherà imporre:

$$\begin{cases} E_n(1) = 0 \\ E_n(x) = 0 \\ \vdots \\ E_n(x^{m+1}) = 0 \end{cases}$$

cioè $m+1$ equazioni per $a_0, ..., a_m$, cioè $m+1$ incognite. Questo porta a:

$$\implies \begin{cases} m_0 = a_0 + a_1 + ... + a_n \\ m_1 = a_0 x_0 + a_1 x_i + ... + a_n x_n \\ \vdots \\ m_n = a_0 x_0^n + a_1 x_1^n + ... + a_n x_n^n \end{cases}$$

dove gli m_i sono i risultati degli integrali dei monimoi x^i sull'intervallo $[a, b]$ da cui:

$$\implies \begin{pmatrix} 1 & 1 & ... & 1 \\ x_0 & x_1 & ... & x_n \\ x_0^2 & x_1^2 & ... & x_n^2 \\ \vdots & \vdots & & \vdots \\ x_0^n & x_1^n & ... & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} m_0 \\ \vdots \\ m_n \end{pmatrix} = V^T \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} m_0 \\ \vdots \\ m_n \end{pmatrix}$$

Questo non è altro che un sistema lineare $(n+1) \times (n+1)$ con matrice di Vandermonde, da cui $\det(V) \neq 0$ e quindi se $x_i \neq x_j$ per ogni $i \neq j$, la soluzione del sistema è unica ed esiste un'unica formula di quadratura sui nodi $x_0, ..., x_n$ che ha grado di precisione $m \geq n$.

Potremmo considerare il problema diverso (e più difficile) di cercare sia i *nodi* che i *pesi* in modo da massimizzare il grado di precisione m. In questo caso la formula di quadratura è sempre:

$$J_n(f) = \sum_{i=0}^{n} a_i f(x_i)$$

e le incognite sono $2n+2$, cioe gli $n+1$ nodi $x_0, ..., x_n$ e gli $n+1$ pesi $a_0, ..., a_n$.

Si impongono quindi $2n+2$ equazioni del tipo:

$$\begin{cases} E_n(1) = 0 \\ E_n(x) = 0 \\ \vdots \\ E_n(x^{2n+1}) = 0 \end{cases}$$

L'unica cosa che sarà data sarà l'intervallo $[a, b]$ (non gli x_i), per cui otterremo un sistema non lineare:

$$\begin{cases} a_0 + a_1 + ... + a_n = m_0 \\ a_0 x_0 + a_1 x_1 + ... + a_n x_n = m_1 \\ \vdots \\ a_0 x_0^{2n+1} + a_1 x_1^{2n+1} + ... + a_n x_n^{2n+1} = m_{2n+1} \end{cases}$$

Fortunatamente esiste un teorema, che diamo senza dimostrazione:

Teorema 16.1: Unicità della formula Gaussiana

Il sistema:

$$\begin{cases} a_0 + a_1 + ... + a_n = m_0 \\ a_0 x_0 + a_1 x_1 + ... + a_n x_n = m_1 \\ \vdots \\ a_0 x_0^{2n+1} + a_1 x_1^{2n+1} + ... + a_n x_n^{2n+1} = m_{2n+1} \end{cases}$$

ammette sempre, dato $x_i \neq x_j$ per ogni $i \neq j$, un'unica soluzione per ogni scelta di $[a, b]$, n e $\rho(x)$, con grado di precisione $\geq 2n+1$.

Definiamo tale formula di quadratura come:

Definizione 16.4: Formula Gaussiana

L'unica formula di quadratura su $[a, b]$ che verifica il sistema del teorema 16.1 (ovvero che massimizza il grado di precisione ottimizzando sia gli x_i che gli a_i) si dice **formula Gaussiana** su $[a, b]$ con $n+1$ nodi.

Prendiamo ad esempio il caso $\rho = 1$, $n = 1$, e $[a, b] = [-1, 1]$ (analogamente a prima ma con i nodi liberi).

$$\int_{-1}^{1} f(x)\, dx \approx a_0 f(x_0) + a_1 f(x_1)$$

con:

$$a \leq x_0 < x_1 \leq b$$

Imponiamo quindi:

$$\begin{cases} E_1(1) = 0 \\ E_1(x) = 0 \\ E_1(x^2) = 0 \\ E_1(x^3) = 0 \end{cases} \rightarrow \begin{cases} a_0 + a_1 = 2 \\ a_0 x_0 + a_1 x_1 = 0 \\ a_0 x_0^2 + a_1 x_1^2 = \frac{2}{3} \\ a_0 x_0^3 + a_1 x_1^3 = 0 \end{cases}$$

per risolvere il sistema ricaviamo prima a_0 e a_1 in funzione di x_0 e x_1 dalle prime 2 equazioni:

- a_0 si ha direttamente dalla prima equazione:

$$a_0 = 2 - a_1$$

- a_1 si ha dalla seconda equazione,, sostituendo la formula per a_0 appena trovata:

$$(2 - a_1)x_0 + a_1 x_1 = 2x_0 - a_1 x_0 + a_1 x_1 = 0 \implies a_1(x_1 - x_0) = \frac{-2x_0}{x_1 - x_0}$$

da cui abbiamo che vale:

$$a_0 = \frac{2x_1}{x_1 - x_0}, \quad a_1 = -\frac{2x_0}{x_1 - x_0}$$

Prendiamo quindi il sistema ridotto alle ultime 2 equazioni:

$$\begin{cases} \frac{2x_1 x_0^2}{x_1 - x_0} - \frac{2x_0 x_1^2}{x_1 - x_0} = \frac{2}{3} \\ \frac{2x_1 x_0^3}{x_1 - x_0} - \frac{2x_0 x_1^3}{x_1 - x_0} = 0 \end{cases} \rightarrow \begin{cases} 2x_1 x_0^2 - 2x_0 x_1^2 = \frac{2}{3}(x_1 - x_0) \\ 2x_1 x_0^3 - 2x_0 x_1^3 = 0 \end{cases}$$

dall'ultima di queste si ricava:

$$2x_1 x_0^3 - 2x_0 x_1^3 = 0 \Rightarrow 2x_1 x_0^3 = 2x_0 x_1^3 \Rightarrow x_0^2 = x_1^2 \Rightarrow x_0 = \pm x_1$$

Ci interesserà prendere $x_0 = -x_1$, da cui dalla prima equazione:

$$2x_1^3 + 2x_1^4 = \frac{4}{3}x_1 \implies x_1^2 = \frac{1}{3}$$

cioè:

$$x_0, x_1 = \pm\frac{\sqrt{3}}{3}$$

sostituendo nelle formule per a_0 e a_1 trovate prima, si ha poi immediatamente:

$$a_0, a_1 = 1,$$

cioè la formula Gaussiana è:

$$J_1(f) = f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right)$$

che vale fino al grado 3, in quanto al grado 4 si ha:

$$E_1(x^4) = \frac{2}{5} - \frac{2}{9} \neq 0$$

da cui il grado di precisione è esattamente 3. □

16.1.4 Errore nelle formule di quadratura

Riguardo all'errore già definito nella definizione 16.2, vale il seguente teorema:

Teorema 16.2: Teorema di Peano

Data una funzione $f(x) \in \mathbb{C}^{n+1}([a,b])$, cioè derivabile $n+1$ volte, con m grado di precisione della formula di quadratura J_n, allora l'errore E_n (vedi definizione 16.2) si può scrivere come:

$$E_n(f) = I(\rho, f) - J_n(f) = \frac{1}{m!}\int_a^b f^{(m+1)}(t) \cdot G(t)\, dt$$

dove la $G(t)$ è:

$$G(t) = E_n(s_m(x-t)) = I(\rho \cdot s_m(x-t)) - J_n(s_m(x-t))$$

e la $s_m(x)$ è:

$$s_m(x) = \begin{cases} x^m, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

cioè la $s_m(x-t)$ non è altro che la $s_m(x)$ spostata a destra di t.

Vale la definizione:

Definizione 16.5: Nucleo di Peano

La funzione $G(t)$ del teorema 16.2 è detta **nucleo di Peano**.

17 Lezione del 02-05-25

Avevamo dato la definizione del teorema di Peano (teorema 16.2), per l'errore della generica formula di quadratura J_n. Per ora quest'espressione ci risulta poco maneggevole in quanto richiede il calcolo dell'errore $G(t)$ sulla funzione $s_m(t)$, detto *nucleo di Peano* (definizione 16.5).

Riprendiamo quindi la definizione dell'errore dal teorema di Peano:

$$E_n(f) = \frac{1}{m!}\int_a^b f^{(m+1)}(t) \cdot G(t)\, dt$$

e accorgiamoci che se $G(t)$ non cambia segno in $[a,b]$ possiamo sfruttare il teorema della media integrale per dire che esiste un certo punto $\varepsilon \in [a,b]$ tale che:

$$E_n(f) = \frac{1}{m!} f^{(m+1)}(\varepsilon) \int_a^b G(t)\, dt$$

Cerchiamo adesso di trovare una forma più maneggevole per l'integrale del nucleo di Peano $G(t)$. Abbiamo che questa non dipende da f, e se valutiamo l'errore per $f(x) = x^{m+1}$ si ottiene:

$$E_n(x^{m+1}) = \frac{1}{m!}\int_a^b \frac{d^{m+1}}{dt^{m+1}} t^{m+1} G(t)\, dt = \frac{1}{m!}(m+1)! \int_a^b G(t)\, dt \implies \int_a^b G(t)\, dt = \frac{E_n(x^{m+1})}{m+1}$$

Prendiamo quindi questo come un **corollario** del teorema di Peano.

Osserviamo allora che se si vuole trovare una disuguaglianza del tipo:

$$|E_n(f)| \leq M$$

si può prendere:

$$M = \max_{x \in [a,b]} \left| f^{(m+1)}(x) \right| \cdot \frac{E_n(x^{m+1})}{m+1}$$

dove il primo termine può essere stimato con studi di funzione o maggiorazioni esplicite, mentre il secondo termine si può calcolare esplicitamente (lo facevamo per valutare manualmente il grado di una formula di integrazione).

17.0.1 Esempio: errore di Peano nella formula dei trapezi

Vediamo quindi cosa si ottiene quando $n = m = 1$, cioè nel caso della formula dei trapezi. In questo caso il nucleo è calcolato su $s_1(t)$:

$$s_1(x-t) = \begin{cases} 0, & x < t \\ x - t, & x \geq t \end{cases}$$

e quindi sarà:

$$G(t) = E_n(s_m(x-t)) = \int_a^b s_1(x-t)\,dx - J_1(s_1(x-t))$$

Prendiamo l'intervallo $[a,b] = [-1,1]$ e svolgiamo i conti. Si ha:

$$G(t) = \int_{-1}^{1} s_1(x-t)\,dx - s_1(-1-t) - s_1(1-t) = \int_{-1}^{1} (x-t)\,dx - 0 - 1 - t$$

dal fatto che $-1-t$ è sempre negativo e $1-t$ è sempre positivo. Proseguendo si ha:

$$= \frac{(x-t)^2}{2}\Big|_t^1 - 1 + t = \frac{(1-t)^2}{2} - 1 + t = \frac{1 + t^2 - 2t - 2 + 2t}{2} = \frac{t^2 - 1}{2} \leq 0, \quad \forall t \in [-1,1]$$

cioè il segno di $G(t)$ non cambia.

Questo si generalizza facilmente ad intervalli $[a,b]$ generici, in quanto con passaggi simili:

$$G(t) = \int_a^b s_1(x-t)\,dx - J_1(s_1(x-t)) = \int_t^b (x-t)\,dx - \frac{b-a}{2} s_m(a-t) - \frac{b-a}{2} s_m(b-t)$$

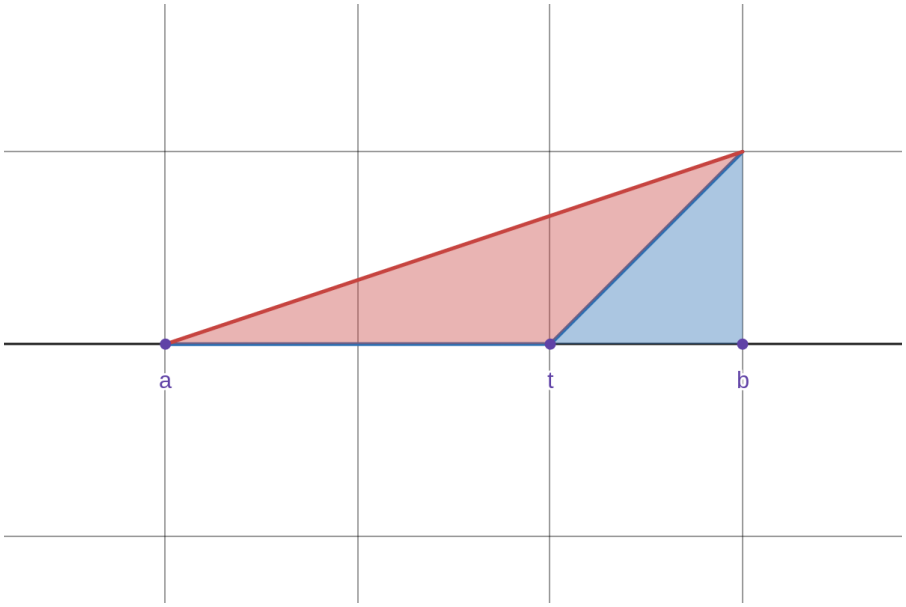
da cui si ha quindi, per la definizione di $s_1(x-t)$ appena data:

$$G(t) = \int_t^b (x-t)\,dx - \frac{b-a}{2} s_m(a-t) - \frac{b-a}{2} s_m(b-t) = \int_t^b (x-t)\,dx - \frac{b-a}{2}(b-t)$$

$$= \frac{(x-t)^2}{2}\Big|_t^b - \frac{b^2 - bt - ab + at}{2} = \frac{(b-t)^2}{2} - \frac{b^2 - bt - ab + at}{2}$$

$$= \frac{b^2 - 2bt + t^2 - b^2 + bt + ab - at}{2} = \frac{t^2 - bt - at}{2} = \frac{(t-a)(t-b)}{2} \leq 0, \quad \forall t \in [a,b]$$

Questo in realtà era chiaro osservando che il nucleo di Peano in questo caso valuta la differenza fra l'area in azzuro e la somma dell'area in azzurro e e dell'area in rosso nel seguente grafico:



che sarà chiaramente sempre ≤ 0.

Per la formula dei trapezi vale quindi, in generale:

$$E_1(f) = \frac{1}{m!} f^{(m+1)}(\varepsilon) \int_a^b G(t)\,dt = \frac{1}{m!} f^{(m+1)}(\varepsilon) \frac{E_n(x^{m+1})}{m+1} = \frac{f''(\varepsilon)}{2} E_1(x^2)$$

Preso $m = 1$. Vediamo quindi che per $f = x^2$:

$$E_1(x^2) = \int_a^b x^2\,dx - \frac{b-a}{2}(a^2+b^2) = \frac{x^3}{3}\Big|_a^b - \frac{ba^2+b^3-a^3-ab^2}{2}$$

$$= \frac{b^3-a^3}{3} - \frac{ba^2+b^3-a^3-ab^2}{2} = \frac{2b^3-2a^3-3ba^2-3b^3+3a^2+3ab^2}{6} = -\frac{(b-a)^3}{6}$$

da cui:

$$E_1(f) = -\frac{(b-a)^3}{12} \cdot f''(\varepsilon), \quad \varepsilon \in [a,b]$$

17.0.2 Esempio: errore di Peano nella formula di Simpson

Si dimostra che anche la formula di Simpson è tale per cui $G(t)$ non cambia segno, quindi vale:

$$E_2(f) = \frac{f^{(4)}(\varepsilon)}{4!} \cdot E_2(x^4)$$

Vediamo quindi che per l'intervallo $[-1,1]$ vale:

$$E_2(x^4) = \frac{2}{5} - \frac{2}{3} = -\frac{4}{15}$$

per cui:

$$E_2(f) = \frac{f^{(4)}(\varepsilon)}{24} \cdot \left(-\frac{4}{15}\right) = -\frac{1}{90} \cdot f^{(4)}(\varepsilon), \quad \epsilon \in [-1,1]$$

Lo stesso ragionamento si può chiaramente fare per $[a, b]$ generico, cioè:

$$E_2(x^4) = \int_a^b x^4 dx - \frac{b-a}{6}\left(a^4 + 4\frac{(a+b)^4}{16} + b^4\right) = \frac{b^5 - a^5}{5} - \frac{b-a}{24}\left(5a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + 5b^4\right)$$

$$= \frac{24b^5 - 24a^5 - 5\left(5a^4b + 4a^3b^2 + 6a^2b^3 + 4ab^4 + 5b^5 - 5a^5 - 4a^4b - 6a^3b^2 - 4a^2b^3 - 5ab^4\right)}{120}$$

$$= \frac{24b^5 - 24a^5 - 25a^4b - 20a^3b^2 - 30a^2b^3 - 20ab^4 - 25b^5 + 25a^5 + 20a^4b + 30a^3b^2 + 20a^2b^3 + 25ab^4}{120}$$

$$= -\frac{(b-a)^5}{120}$$

da cui:

$$E_2(f) = \frac{f^{(4)}(\varepsilon)}{24} \cdot \left(-\frac{(b-a)^5}{120}\right) = -\frac{(b-a)^5}{2880} \cdot f^{(4)}(\varepsilon), \quad \epsilon \in [a, b]$$

17.1 Formule di quadratura interpolatorie

Vediamo l'approccio all'approssimazione integrale che prevede scegliere $n+1$ nodi $x_0, ..., x_n$. In questo caso si utilizza come approssimazione di $\int_a^b f(x)\rho(x)\,dx$ la quantità:

$$\int_a^b f(x)\rho(x)\,dx \approx \int_a^b P_n(x)\rho(x)\,dx$$

con $P_n(x)$ polinomio di grado al più n tale che:

$$P_n(x_i) = f(x_i), \quad \forall i = 0, ..., n$$

cioè $P_n(x)$ polinomio interpolante.

Se f è sufficientemente regolare sappiamo che:

$$f(x) = P_n(x) + R_n(x), \quad R_n(x) = \frac{\Pi(x)}{(n+1)!} f^{(n+1)}(\varepsilon), \quad \varepsilon \in [a, b]$$

dal teorema 14.1.

Prendiamo quindi:

$$\int_a^b f(x)\rho(x)\,dx = \int_a^b P_n(x)\rho(x)\,dx + \int_a^b R_n(x)\rho(x)\,dx = \int_a^b \sum_{i=0}^n f(x_i)l_i(x) + \int_a^b R_n(x)\rho(x)\,dx$$

dove gli $l_i(x)$ sono i polinomi della base di Lagrange:

$$l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

da cui si può portare fuori la sommatoria:

$$\int_a^b f(x)\rho(x)\,dx = \sum_{i=0}^n f(x_i) \int_a^b l_i(x)\rho(x)\,dx + E_n(f)$$

che ha esattamente la forma di una formula di quadratura, dove gli $f(x_i)$ sono i nodi della formula $J_n(f)$ e gli:

$$\int_a^b l_i(x)\rho(x)\,dx$$

a moltiplicare sono i pesi, cioè quelli che avevamo finora chiamato a_i. L'errore $E_n(f)$ chiaramente si trascura.

Osserviamo che per costruzione le formule interpolatorie hanno grado di precisione almeno n (chiaramente dal fatto che il polinomio interpolante di grado n di un polinomio di grado n è esattamente sé stesso).

Viceversa, dati $n+1$ nodi c'è una sola formula di quadratura che ha grado di precisione $\geq n$ su qui nodi, e deve essere la formula di quadratura interpolatoria.

17.1.1 Esempio: formule di Newton-Cotes

L'esempio più classico delle formule di quadratura interpolatorie sono le formule di **Newton-Cotes**, che si basano sull'ipotesi di prendere nodi equispaziati su a, b. Un prima scelta è riguardo all'inclusione o meno degli estremi:

- Quando gli estremi (a, b) sono inclusi si parla di formule di Newton-Cotes *chiuse*;
- Quando gli estremi (a, b) sono esclusi si parla di formule di Newton-Cotes *aperte*.

Consideriamo per adesso le formule **chiuse**, per cui:

$$x_0 = a, \quad x_1 = h, \quad ..., \quad x_i = a + ih, \quad ..., \quad x_n = b$$

con:

$$h = \frac{b-a}{n}$$

detto **passo** della formula.

I pesi a_i saranno quindi:

$$a_i = \int_a^b l_i(x)\, dx$$

Le formule di Newton-Cotes equivalgono, per il grado 1 alle formule dei trapezi e per il grado 2 alle formule di Simosib.

Una proprietà importante delle formule di Newton-Cotes è che per ogni scelta di n vale che $G(t)$ nucleo di Peano non cambia segno su $[a, b]$, ergo si può usare la formula semplificata per l'errore di Peano:

$$E_n(f) = \frac{f^{(m+1)}(\varepsilon)}{(m+1)!} \cdot E_n(x^{m+1})$$

Sul grado n vale un fenomeno analogo al fenomeno di Runge (sezione 14.3.2). Per $n > 7$ (cioè se si hanno più di 8 nodi) le formule di Newton-Cotes iniziano ad avere pesi di segno alternato (mentre per $n \leq 7$ sono sempre positivi), cioè si inizia ad avere cancellazione numerica e valutare le formule diventa instabile numericamente.

Per esprimere l'errore delle formule di Newton-Cotes abbiamo quindi bisogno di trovarne l'errore $E_n(x^{m+1})$, cioè l'errore per il monomio di grado $m+1$ con m grado di precisione. In questo caso si assume un'espressione del tipo:

$$E_n(f) = \begin{cases} c_n \cdot h^{n+2} \cdot f^{(n+1)}(\varepsilon), & n \text{ dispari} \\ c_n \cdot h^{n+3} \cdot f^{(n+2)}(\varepsilon), & n \text{ pari} \end{cases}$$

Vediamo ad esempio:

- **Formula dei trapezi:** in questo caso si ha $n = 1$, $h = b - a$, da cui l'errore è esprimibile come:

$$E_1(f) = -\frac{(b-a)^3}{12} \cdot f''(\varepsilon) = -\frac{h^3}{12} \cdot f''(\varepsilon)$$

- **Formula di Simpson** in questo caso si ha $n = 2$, $h = \frac{b-a}{2}$, da cui l'errore è esprimibile come:

$$E_2(f) = -\frac{(b-a)^5}{2880} \cdot f^{(4)}(\varepsilon) = -\frac{h^5}{90} \cdot f^{(4)}(\varepsilon)$$

Un'altra osservazione da fare è che il grado di precisione e l'ordine della potenza di h è lo stesso per una formula con grado n pari e quella con grado dispari $n+1$. In genere, a parte il caso dei trapezi, si considerano più spesso le formule di Newton-Cotes con n pari.

Vediamo nel dettaglio l'andamento del grado degli errori per qualche grado dopo lo 0. Innanzitutto, bisogna fare una precisazione per cosa significa prendere la formula di Newton-Cotes di grado 0. Avevamo che per formula *chiusa* si intende quella che prende $n + 1$ punti estremi inclusi, mentre la formula *aperta* è quella che prende $n + 1$ punti esclusi gli estremi. Tali punti si ottengono ad esempio come:

- **Formula chiusa:**

$$x_i = a + ih, \quad h = \frac{b-a}{n}, \quad i = 0, 1, ..., n$$

- **Formula aperta:**

$$x_i = a + (i+1)h, \quad h = \frac{b-a}{n+2}, \quad i = 0, 1, ..., n$$

Da questo risulta chiaro che sarà impossibile prendere una formula chiusa che include entrambi gli estremi quando si hanno a disposizione $n + 1 = 1$ punti per il grado $n = 0$. Divideremo quindi fra:

- **Somma di Riemann sinistra:**

$$J_0^l(f) = (b-a) \cdot f(a)$$

- **Somma di Riemann destra:**

$$J_0^r(f) = (b-a) \cdot f(b)$$

Per queste il grado è $m = 0$ e l'errore per $f = x$ risulta proporzionale al grado 2, e non al 3 come ci si aspetterebbe (per il punto in meno di cui non possiamo sfruttare la simmetria). Ad esempio rispetto alla somma di Riemann sinistra si ha:

$$E_0^l(x) = \int_a^b x\,dx - (b-a) \cdot a = \frac{b^2 - a^2}{2} - ab + a^2 = \frac{b^2 - a^2 - 2ab + 2a^2}{2} = \frac{(b-a)^2}{2}$$

Potremo quindi considerare la formula aperta, che equivarrà alla cosiddetta **formula del punto medio**:

$$J_0^m(f) = (b-a) \cdot f\left(\frac{a+b}{2}\right)$$

Vediamo la particolarità che questa ha effettivamente grado $m = 1$, in quanto:

$$E_0^m(x) = \int_a^b x\,dx - (b-a)\cdot\frac{a+b}{2} = \frac{b^2-a^2}{2} - \frac{b^2-a^2}{2} = 0$$

Valutiamo quindi l'errore in x^2 come:

$$E_0^m(x^2) = \frac{b^3-a^3}{3} - (b-a)\left(\frac{a+b}{2}\right)^2 = \frac{b^3-a^3}{3} - (b-a)\frac{a^2+2ab+b^2}{4}$$

$$= \frac{b^3-a^3}{3} - \frac{a^2b+2ab^2b^3-a^3-2a^2b-ab^2}{4} = \frac{b^3-a^3}{3} + \frac{a^2b-ab^2-b^3+a^3}{4}$$

$$= \frac{b^3-4a^3+3a^2b-4ab^2-3b^3+3a^3}{12} = \frac{b^3-3ab^2+3a^2b-a^3}{12} = \frac{(b-a)^3}{12}$$

cioè otteniamo un errore proporzionale a h^3 (lo stesso grado del metodo dei trapezi), e almeno per x^2 opposto a quello dato dal metodo dei trapezi.

Possiamo quindi compilare una tabella che associa i gradi di approssimazione ai gradi dell'errore per le formule di Newton-Cotes chiuse e aperte:

Grado	**Formula**		**Errore**	
	Aperta	**Chiusa**	**Aperta**	**Chiusa**
0	$2hf_0$	$2hf_0$	3	2
1	$\frac{3}{2}h(f_0+f_1)$	$\frac{1}{2}h(f_0+f_1)$	3	3
2	$\frac{4}{3}h(2f_0-f_1+2f_2)$	$\frac{1}{3}h(f_0+4f_1+f_2)$	5	5
3	$\frac{1}{12}h(11f_0+f_1+f_2+11f_3)$	$\frac{3}{8}h(f_0+3f_1+3f_2+f_3)$	5	5

17.1.2 Esempio: formule Gaussiane

Riprendiamo quindi anche le formule Gaussiane espresse come formule di quadratura interpolatorie.

Innanzitutto si può dire che anche per queste il nucleo di Peano $G(t)$ non cambia segno su $[a,b]$, cioè si può usare la formula di errore data dal corollario.

Un'altra proprietà è che i pesi delle formule Gaussiane sono sempre positivi, ergo non ci sono problemi di instabilità numerica per n grande.

Veniamo quindi a come ricavarle. Fissato l'intervallo $[a,b]$ e la funzione peso $\rho(x)$ si ha che le formule Gaussiane sono completamente determinate. I nodi delle formule gaussiane si possono scrivere come zeri di particolari polinomi, detti *polinomi ortogonali*. Più precisamente sono gli zeri della successione di polinomi $\{q_n\}$, con $\deg(q_n) = n$, determinati da:

$$< q_n, p >= 0, \quad \forall p : \deg(p) < n$$

sfruttando la definizione di prodotto scalare $< \cdot,\cdot >$ sullo spazio polinomiale $\mathbb{R}[x]$ data nei corsi di algebra lineare:

$$< a, b >= \int_a^b a(x)b(x)\,dx$$

da cui:

$$< q_n, p >= \int_a^b q_n(x)p(x)\rho(x)\,dx$$

- Nel caso specifico che stavamo calcolando, con $[a, b]$ e $\rho(x) = 1$ si trova l'insieme dei **polinomi di Legendre**.
- Usando $[a, b]$ e la funzione peso:

$$\rho(x) = \frac{1}{\sqrt{1 - x^2}}$$

si ottiene invece il cosiddetto **polinomio di Chebyshev di prima specie**, le cui radici sono i nodi di Chebyshev (che avevamo già incontrato nella sezione 14.3.3).

Esistono metodi ad-hoc per trovare gli zeri dei polinomi ortogonali su $[-1, 1]$. Una volta trovati questi si possono ottenere quelli su $[a, b]$ con il cambio di variabili:

$$[-1, 1] \to [a, b], \quad x \to \frac{b - a}{2} x + \frac{b + a}{2}$$

cioè una semplice operazione di scalatura lineare.

18 Lezione del 05-05-25

Continuiamo la trattazione dell'approssimazione di integrali attraverso le formule di quadratura interpolatorie. Un problema che avevamo fino adesso è che se h è grande, allora $E_n(f)$ è grande per qualche termine proporzionale a h^d, dove d è il grado dell'errore.

Un'idea potrebbe essere di aumentare n, quindi aumentare i punti campionati per ridurre il passo h, ma come abbiamo visto questo è instabile (risente di fenomeni simili al fenomeno di Runge).

18.0.1 Formule di Newton-Cotes generalizzate

Decidiamo quindi di spezzare l'integrale su sottointervalli, e in ciascuno di essi applicare una formula di quadratura. Questo metodo ci porterà alle formule di Newton-Cotes **generalizzate**, anche dette *composite*.

Quindi, posto ad esempio un certo intervallo $[a, b]$ contenente i punti ordinati c, d, e, dividiamo l'integrale:

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^d f(x)\,dx + \int_d^e f(x)\,dx + \int_e^b f(x)\,dx$$

e si applica la formula di quadratura ad ogni sottointervallo, sommando.

Più formalmente, avremo che si divide $[a, b]$ in L sottointervalli equispaziati, con:

$$x_0 = a, \quad x_1 = x_0 + \frac{b - a}{L}, \quad ..., \quad x_L = b$$

e si divide l'integrale come:

$$\int_a^b f(x)\,dx = \sum_{i=1}^{L} \int_{x_{i-1}}^{x_i} f(x)\,dx$$

In ogni intervallo $[x_{i-1}, x_i]$ si applica quindi una formula di Newton Cotes con $n + 1$ nodi.

Avremo quindi bisogno, in ogni sottointervallo, di:

$$n + 1 - 2 = n - 1$$

nodi aggiuntivi oltre agli estremi dell'intervallo per poter applicare la formula di quadratura $J_n(f)$. Il numero di nodi totali dovrà quindi essere:

$$L + 1 + (n - 1) \cdot L = nL + 1$$

Osserviamo che il numero di nodi corrisponde al numero di valutazioni di f, e quindi in genere è un'indicazione del costo computazionale di una formula.

18.0.2 Esempio: formula dei trapezi generalizzata

Prendiamo la forma che otteniamo per $n = 1$:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{L} \frac{b-a}{2L}\left(f(x_{i-1}) + f(x_i)\right) = \frac{b-a}{2L}\sum_{i=1}^{L}\left(f(x_{i-1}) + f(x_i)\right)$$

$$= \frac{b-a}{2L}\left(f(x_0) + 2\sum_{i=1}^{L-1} f(x_i) + f(x_L)\right) = J_1^{(G)}(f)$$

18.0.3 Esempio: formula di Cavalieri-Simspon

Vediamo quindi la generalizzazione della formula di Cavalieri, cioè ciò che otteniamo per $n = 2$:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{L} \frac{b-a}{6L}\left(f(x_{i-1}) + 4f\left(\frac{x_{i-1}+x_i}{2}\right) + f(x_i)\right)$$

$$= \frac{b-a}{6L}\left(f(x_0) + 2\sum_{i=1}^{L-1} f(x_i) + 4\sum_{i=1}^{L} f\left(\frac{x_{i-1}+x_i}{2}\right) + f(x_L)\right) = J_2^{(G)}(f)$$

18.0.4 Errore nelle formule generalizzate

In ogni intervallo della forma $[x_{i-1}, x_i]$ conosciamo l'errore, che è quello della formula di Newton-Cotes che stiamo utilizzando:

$$e \cdot h^d \cdot f^{(d-1)}(\varepsilon), \quad f \in C^{(d-1)}([a,b])$$

L'errore sulla formula totale $J_n^{(G)}(f)$ sarà quindi:

$$E_n^{(G)}(f) = I(f) - J_n^{(G)}(f) = \sum_{i=1}^{L} c \cdot h^d \cdot f^{(d-1)}(\varepsilon_i), \quad \varepsilon_i \in [x_{i-1}, x_i]$$

Vediamo quindi che l'unico termine dipendente dall'intervallo è $f^{(d-1)}(\varepsilon)$, per cui possiamo dire:

$$= \sum_{i=1}^{L} c \cdot h^d \cdot L\frac{f^{(d-1)}(\varepsilon_i)}{L} = ch^dL \cdot f^{(d-1)}(\varepsilon), \quad \varepsilon \in [a,b]$$

sfruttando il teorema della media integrale.

Abbiamo quindi gli errori:

- **Formula dei trapezi:**

$$E_1^{(G)}(f) = -\frac{(b-a)^3}{12L^2} \cdot f''(\varepsilon)$$

- **Formula di Cavalieri-Simpson:**

$$E_2^{(G)}(f) = -\frac{(b-a)^5}{2880L^4} \cdot f^{(4)}(\varepsilon), \quad \varepsilon \in [a,b]$$

e il caso generale:

$$E_n^{(G)}(f) = \begin{cases} c_n \cdot \frac{(b-a)^{n+2}}{L^{n+1}} \cdot f^{(n+1)}(\varepsilon), & n \text{ dispari} \\ c_n \cdot \frac{(b-a)^{n+3}}{L^{n+2}} \cdot f^{(n+2)}(\varepsilon), & n \text{ pari} \end{cases}$$

18.1 Formule di quadratura sulle derivate

Potremmo considerare formule di quadratura che coinvolgono le derivate di f. Si possono quindi generalizzare i concetti di grado di precisione, scelta dei nodi e pesi ottimali anche per formule di quadratura del tipo:

$$\int_a^b f(x)\rho(x)\,dx = \sum_{i=0}^{n_1} f(x_i) + \sum_{i=0}^{n_2} f'(y_i) + \sum_{i=0}^{n_3} f''(z_i)$$

chiaramente restringendo l'applicazione delle formule a funzioni sufficientemente derivabili.

Vediamo ad esempio come trovare i pesi ottimi per l'approssimazione con:

$$\int_0^1 f(x)\,dx = I(f) \approx a_1 f(0) + a_2 f'\left(\frac{1}{3}\right) + a_3 f'\left(\frac{2}{3}\right) + f(1)$$

sull'intervallo $[a,b] = [0,1]$.

Imponendo errore nullo ai primi 4 gradi si otterrà:

$$\begin{cases} E_1(1) = 0 \\ E_1(x) = 0 \\ E_1(x^2) = 0 \\ E_1(x^3) = 0 \end{cases} \implies \begin{cases} \int_0^1 1\,dx = 1 = a_1 + a_4 \\ \int_0^1 x\,dx = \frac{1}{2} = a_2 + a_3 + a_4 \\ \int_0^1 x^2\,dx = \frac{1}{3} = \frac{2}{3}a_2 + \frac{4}{3}a_3 + a_4 \\ \int_0^1 x^3\,dx = \frac{1}{4} = \frac{1}{3}a_2 + \frac{4}{3}a_3 + a_4 \end{cases}$$

Ricaviamo allora dalle prime 3:

$$\begin{cases} a_4 = 1 - a_1 \\ a_2 = \frac{1}{2} - a_3 - a_4 \\ a_3 = \frac{3}{4}\left(\frac{1}{3} - \frac{2}{3}a_2 - a_4\right) = \frac{1}{4} - \frac{1}{2}a_2 - \frac{3}{4}a_4 \end{cases}$$

da cui sostituendo ulteriormente:

$$\begin{cases} a_4 = 1 - a_1 \\ a_2 = \frac{1}{2} + \frac{1}{2}a_4 - a_4 = \frac{1}{2} - \frac{1}{2}a_4 \\ a_3 = \frac{1}{4} - \frac{1}{2}\left(\frac{1}{2} - a_3 - a_4\right) - \frac{3}{4}a_4 = \frac{1}{2}a_3 - \frac{1}{4}a_4 \end{cases}$$

sostituendo tutto nella quarta si ha:

$$\frac{1}{4} = \frac{1}{3}\left(\frac{1}{2} - \frac{1}{2}a_4\right) + \frac{4}{3}\cdot -\frac{1}{2}a_4 + a_4 = \frac{1}{6} + \frac{1}{6}a_4 \implies a_4\frac{1}{2}$$

da cui immediatamente:

$$a_1 = \frac{1}{2}, \quad a_2 = \frac{1}{4}, \quad a_3 = -\frac{1}{4}, \quad a_4 = \frac{1}{2},$$

da cui:

$$J_3(f) = \frac{f(0)}{2} + \frac{f'\left(\frac{1}{3}\right)}{4} - \frac{f''\left(\frac{2}{3}\right)}{4} + \frac{f(1)}{2}$$

Per determinare il grado di precisione vediamo l'errore per x^4:

$$E_3(x^4) = \int_0^1 x^4\,dx - \left(\frac{1}{2}\cdot 0 + \frac{1}{4}\cdot 4\cdot\left(\frac{1}{3}\right)^3 - \frac{1}{4}\cdot 4\cdot\left(\frac{2}{3}\right)^3 + \frac{1}{2}\right) = \frac{1}{5} - \left(\frac{1}{27} - \frac{8}{27} + \frac{1}{2}\right) = \frac{1}{5} - \frac{13}{54} \neq 0$$

da cui il grado di precisione è esattamente 3.

18.2 Implementazione MATLAB dei metodi di approssimazione di integrali

Possiamo quindi implementare quanto abbiamo studiato finora in funzioni MATLAB che ci permettano di approssimare gli integrali di certi function handle su dati intervalli.

Definiamo in particolare una funzione di approssimazione composita di grado 1 (metodo dei trapezi) e 2 (metodo di Cavalieri-Simpson).

18.2.1 Implementazione MATLAB del metodo dei trapezi

Potremo implementare direttamente la funzione descritta in 18.0.2, con un pò di logica aggiuntiva per tracciare un grafico delle rette approssimazione, come:

```
function j = trapez_int(f, a, b, L, print)
    if nargin < 5
        print = false
    end

    xi = linspace(a, b, L + 1)';
    yi = f(xi);

    if print
        % stampa
        clf;
        hold on;

        % funzione
        fplot(f, [a, b], 'r');

        % punti
        plot(xi, yi, 'ob');

        % rette
        for i = 1:L
            x0 = xi(i); x1 = xi(i + 1);
            y0 = yi(i); y1 = yi(i + 1);

            m = (y1 - y0) / (x1 - x0);
```

```
            r = @(x) m * (x - x0) + y0;

            fplot(r, [x0, x1], 'b')
        end

        hold off;
    end

    % integra
    j = yi(1) + yi(L + 1) + 2 * sum(yi(2:L));
    j = j * (b - a) / (2 * L);
end
```

18.2.2 Implementazione MATLAB del metodo di Cavalieri-Simpson

Anche qui otremo implementare direttamente la funzione descritta in 18.0.3, con un pò di logica aggiuntiva per tracciare un grafico delle parabole di approssimazione, come:

```
function j = simpson_int(f, a, b, L, print)
    if nargin < 5
        print = false;
    end

    xi = linspace(a, b, L + 1)';
    pi = xi(1:L) + (b - a) / (2 * L);
    yi = f(xi);
    ypi = f(pi);

    if print
        % stampa
        clf;
        hold on;

        % funzione
        fplot(f, [a, b], 'r');

        % punti
        plot(xi, yi, 'ob');

        % parabole
        for i = 1:L
            x0 = xi(i); x1 = xi(i + 1);
            p = pi(i); yp = ypi(i);
            y0 = yi(i); y1 = yi(i + 1);

            p = polyfit([x0, p, x1], [y0, yp, y1], 2);
            q = @(x) polyval(p, x);

            fplot(q, [x0, x1], 'b')
        end

        hold off;
    end

    % integra
    j = yi(1) + 2 * sum(yi(2:L)) + 4 * sum(ypi) + yi(L + 1);
    j = j * (b - a) / (6 * L);
end
```

dove notiamo che manteniamo esplicitamente i punti intermedi separati dai punti estremi degli intervalli di integrazione, ergo L determina il numero di intervalli di integrazione.

Potremo quindi provare gli algoritmi, ad esempio sulla funzione:

$$f(x) = \sin(x) + \tan\left(\frac{x}{15}\right) + e^{-x}$$

sull'intervallo $[a, b]] = [0, 15]$.

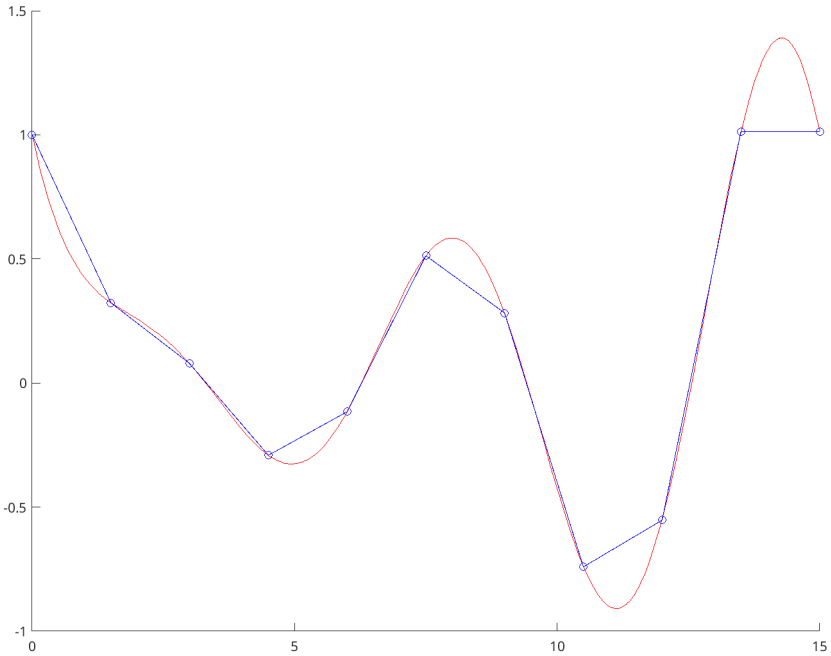
Iniziamo col definire le variabili e un valore di *ground truth*, usando la `integral()` fornita da MATLAB:

```
>> f = @(x) sin(x) .* tan(x / 15) + exp(-x); % definiamo f
>> a = 0; b = 15;
>> i = integral(f, a, b)

i =

    2.2912
```

Potremo quindi provare il metodo dei trapezi come:

```
>> jt = trapez_int(f, a, b, 10)

jt =

    2.2734
```

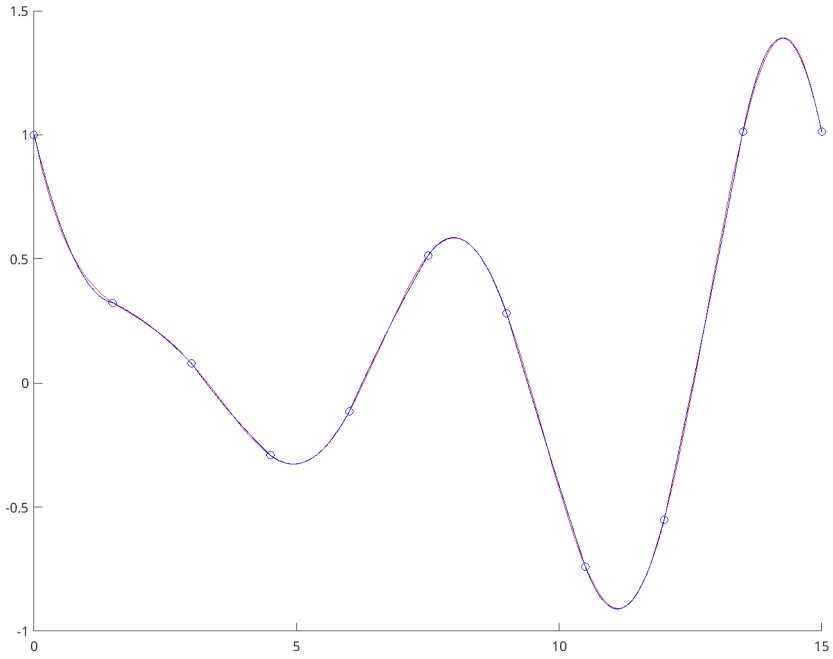
Da cui otteniamo un valore di $i - j_t \approx 0.0178$, e il grafico:



Vediamo quindi se l'approssimazione col metodo di Cavalieri-Simpson è più accurata:

```
>> js = simpson_int(f, a, b, 10)

js =

    2.2940
```

In questo caso otteniamo il valore $i - j_s \approx -0.0028$, e il grafico:



cioè notiamo che sia il grafico che l'appossimazione integrale sono molto pià vicini alla funzione originale.

19 Lezione del 09-05-25

19.1 Approssimazione di equazioni non lineari

Vogliamo risolvere equazioni del tipo:

$$f(x) = 0, \quad f : \mathbb{R} \to \mathbb{R}$$

con f non lineare ($f(x) \neq ax + b$).

Osserviamo che in generale non c'è un'espressione analitica, quindi una formula chiusa, per tutti i punti $\alpha \in \mathbb{R}$ tali che $f(\alpha) = 0$, quindi che risolvono l'equazione.

Diamo quindi la definizione elementare:

Definizione 19.1: Radice

Per un sistema $f(x) = 0$, $\alpha \in \mathbb{R}$ tale che $f(\alpha) = 0$ è detta radice o zero di f.

Vorremo quindi cercare delle approssimazioni di α, e più precisamente vogliamo considerare metodi numerici che generano successioni $\{x_n\}_{n\in\mathbb{N}}$ che sperabilmente hanno la proprietà:

$$\lim_{n\to+\infty} x_n = \alpha$$

In particolare vedremo metodi del tipo:

$$x_{n+1} = \phi_n(x_n, x_{n-1}, ..., x_{n-k+1})$$

La funzione ϕ_n accetta k argomenti (punti) e viene detta **funzione di iterazione**. Se $\phi_n = \phi$, $\forall n \in \mathbb{N}$ parliamo di **metodi stazionari**.

Diamo quindi la definizione di **convergenza**:

Definizione 19.2: Convergenza

Un metodo iterativo per risolvere $f(x)$ si dice convergente per k punti iniziali $x_0, x_{-1}, ..., x_{1-k}$ se la successione generata:

$$x_{n+1} = \phi_n(x_n, x_{n-1}, ..., x_{n-k+1})$$

verifica:

$$\lim_{n \to +\infty} x_n = \alpha$$

con α radice di f.

Possiamo generalizzare l'idea di convergenza a convergenza *di un certo ordine*:

Definizione 19.3: Ordine di convergenza

Si dice che un metodo iterativo ha convergenza di ordine $p \geq 1$ se $\exists c < +\infty$ (costante finita), $c \neq 0$ tale per cui:

$$\lim_{n \to +\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^p} = c$$

dove nel caso $p = 1$ si richiede anche $0 < c < 1$.

Osserviamo quindi che la definizione dice che, asintoticamente, si ha $|e_{n+1}| \approx |e_n|^p$. Da questo è chiaro che se e_n è un numero piccolo ($|e_n| \in (0, 1)$), per $p \geq 2$ (o $p = 1$ con $0 < c < 1$, come richiesto dalla definizione) e_{n+1} sarà ancora più piccolo.
In particolare si ha quindi che:

- Per $p = 1$ si parla di convergenza **lineare**, cioè l'errore si riduce di un fattore c ad ogni passo;
- Per $p = 2$ si parla di convergenza **quadratica**, e via dicendo.

Possiamo parlare di **convergenza locale**:

Definizione 19.4: Convergenza locale

Si cha un metodo converge localmente (con ordine p) se $\exists S \subset \mathbb{R}$, con $\alpha \in S$, tale che per ogni scelta di $x_0, ..., x_{1-k} \in S$, x_n converge ad α (con ordine p).

19.1.1 Metodi grafici

Riassumendo, si ha quindi che è importante saper localizzare/stimare dove sono le radici di una certa f di cui ci interessa $f(x) = 0$. Per questo scopo si possono usare strumenti di analisi del grafico di una funzione (almeno nel caso scalare):

1. Se f continua e $f(a) \cdot f(b) < 0 \implies \exists$ almeno una radice in $[a, b]$ (che potrebbe essere più di una).
2. Si può studiare la derivata, ammesso $f \in C'([a, b])$, e quindi del segno di f.

Esempio

Poniamo ad esempio di avere:

$$f(x) = x\log(x) + \frac{1}{3}$$

e di porci la domanda di trovare le radici approssimate di $f(x)$. Innanzitutto restringiamo il dominio a $x > 0$, e prendiamo i limiti agli estremi:

$$\lim_{x\to 0^+} x\log(x) + \frac{1}{3} = \frac{1}{3}, \quad \lim_{x\to+\infty} x\log(x) + \frac{1}{3} = +\infty$$

Prendiamo quindi la derivata:

$$f'(x) = \log(x) + 1 \implies \begin{cases} f'(x) \geq 0, & x \geq \frac{1}{e} \\ f'(x) \leq 0, & 0 < x \leq \frac{1}{e} \end{cases}$$

e quindi $f'\left(\frac{1}{e}\right) = 0$, e la funzione descresce in $\left(0, \frac{1}{e}\right)$ per poi crescere in $\left(\frac{1}{e}, +\infty\right)$.

Calcolando f in $\frac{1}{e}$ si ha:

$$f\left(\frac{1}{e}\right) = \frac{1}{e}\log\left(\frac{1}{e}\right) + \frac{1}{3} = \frac{1}{3} - \frac{1}{e} < 0$$

per cui si avranno necessariamente due radici, comprese, nelle regioni:

$$\alpha_1 \in \left(0, \frac{1}{e}\right), \quad \alpha_2 \in \left(\frac{1}{e}, +\infty\right)$$

3. Si può procedere per separazione grafica. Data:

$$f(x) = g(x) - h(x)$$

dove $g(x)$ e $h(x)$ hano grafico noto, può essere conveniente sfruttare:

$$f(x) = 0 \Leftrightarrow g(x) = h(x)$$

cioè cercare i punti di intersezione fra i grafici di g e h.

Esempio

Poniamo di avere:

$$f(x) = 5x^2 - 2e^x$$

cioè:

$$\begin{cases} h(x) = 2e^x \\ g(x) = 5x^2 \end{cases}$$

A sinistra del grafico, cioè per $\mathbb{R}^-$, ci sarà chiaramente una qualche soluzione α_1.

A destra del grafico si ha invece che e^x va ad infinito più velocemente di e^x, cioè i grafici possono:

(a) O non toccarsi mai;

(b) O toccarsi due volte (con e^x che interseca x^2 in una fase iniziale dove va più lentamente, e quindi lo interseca di nuovo quando lo vince);

(c) O toccarsi una volta sola, come caso critico del caso precedente.

Se quindi troviamo un punto $\tilde{x} \in (0, +\infty)$ dove $2e^{\tilde{x}} < 5\tilde{x}^2$, siamo nel caso 2. Prendiamo allora $\tilde{x} = 2$, per cui:

$$\begin{cases} g(2) = 20 \\ h(2) = 2e^2 = 2e^2 < 2 \cdot 3^2 = 18 \end{cases}$$

per cui chiaramente avremo tre radici:

$$\alpha_1 \in (-\infty, 0), \quad \alpha_2 \in (0, 2), \quad \alpha_3 \in (2, +\infty)$$

Iniziamo quindi a vedere i metodi iterativi veri e propri.

19.1.2 Metodo di bisezione

Il metodo di **bisezione** o *ricerca binaria* (anche *ricerca dicotomica*) consiste nel prendere un intervallo $[a, b]$ tale che $f(a) \cdot f(b) < 0$ (come nel primo esempio grafico della scorsa sezione).

In questo caso si prende come prima approssimazione:

$$x_1 = \frac{x_0 + x_{-1}}{b}, \quad x_0 = a, \quad x_{-1} = b$$

Si valuta quindi il segno di $f(x_1)$. Se $f(x_1) \cdot f(x_0) < 0$, allora si scarta b e si riparte con l'intervallo $[x_0, x_1]$, altrimenti si scarta x_0 e si riparte con l'intervallo $[x_1, x_{-1}]$.

Questo procedimento chiaramente è iterabile, e converge eventualmente ad *una* radice α, che non è detta essere l'unica.

La formula che si ottiene è la seguente:

$$x_{n+1} = \frac{x_{n-1} + x_n}{2}$$

Per quanto riguarda l'errore (al caso *idealmente* pessimo), abbiamo che questo si dimezza ad ogni passaggio:

$$|x_{n+1} - \alpha| < \frac{b-a}{2^n} \implies \lim_{n \to +\infty} |x_n - \alpha| = 0$$

Questa stima è la migliore che possiamo dare, in quanto la funzione di errore $x_{n+1} - \alpha$ non è monotona. In ogni caso, può esserci utile a limitare l'errore con un numero minimo di passaggi al caso peggiore.

Infatti, se vogliamo:

$$|x_n - \alpha| < t_{ol}$$

dobbiamo usare n iterazioni, con n che verifica:

$$\frac{b-a}{2^n} < t_{ol} \implies n > \log_2\left(\frac{b-a}{t_{ol}}\right) \implies n = \left\lceil \log_2\left(\frac{b-a}{t_{ol}}\right) \right\rceil$$

approssimato al primo naturale superiore.

Ad esempio, se $b - a = 1$, servono $n = 10$ passi per ottenere $t_{ol} = 10^{-3}$, $n = 20$ per $t_{ol} = 10^{-6}$, e quindi in genere serviranno un numero sempre maggiore di misurazioni per ottenere precisioni migliori.

19.1.3 Implementazione MATLAB del metodo di bisezione

Vediamo l'implementazione MATLAB del metodo di bisezione:

```
function m = bisect(f, a, b, k, tol)
    if nargin < 5
        tol = 0.01;
    end

    if nargin < 4
        k = 100;
    end

    for i = 1:k
        % print info
        fprintf("\nIteration %d\n", i);
        fprintf("\ta: %.4f\n", a);
        fprintf("\tb: %.4f\n", b);

        % update
        m = (a + b) / 2;

        if f(a) * f(m) > 0
            a = m;
        else
            b = m;
        end

        % tolerance
        if abs(b - a) < tol
            break;
        end
    end
end
```

Dove notiamo di aver adottato il criterio di stop:

$$|x_{k+1} - x_k| < t_{ol}$$

per una tolleranza t_{ol}, di default 0.1.

Inoltre, notiamo che per la valutazione del segno di $f(m)$, teniamo conto del segno di $f(a)$, in quanto vogliamo mantenere la radice sempre nell'intervallo $[a, b]$.

Vediamo ad esempio come applicare questo metodo per trovare una radice della funzione:

$$f(x) = (x-3)^4 - 2$$

sull'intervallo $[a, b] = [3, 5]$, con una differenza di step fra iterazioni al minimo di 0.01.

In MATLAB definiremo f come una function handle:

```
>> f = @(x) (x - 3) .^ 4 - 2

f =

  function_handle with value:

    @(x)(x-3).^4-2
```

e quindi applicheremo l'algoritmo appena definito:

```
>> bisect(f, 3, 5)

```

```
Iteration 1
  a: 3.0000
  b: 5.0000

Iteration 2
  a: 4.0000
  b: 5.0000

Iteration 3
  a: 4.0000
  b: 4.5000

Iteration 4
  a: 4.0000
  b: 4.2500

Iteration 5
  a: 4.1250
  b: 4.2500

Iteration 6
  a: 4.1875
  b: 4.2500

Iteration 7
  a: 4.1875
  b: 4.2188

Iteration 8
  a: 4.1875
  b: 4.2031

ans =

    4.1953
```

Si ottiene quindi 4.1953, che per $f(4.1953) \approx 0.04$ è abbastanza vicino considerata la differenza di step minima imposta. Concedendo all'algoritmo più passaggi, si otterrano chiaramente soluzioni via via più accurate.

19.1.4 Metodo delle secanti

Il metodo delle secanti si basa su un idea geometrica, cioè quella di tracciare la *retta* secante di f fra a e b, ovvero quella passante per i punti $(a, f(a))$, $(b, f(b))$. Di questa si trova quindi l'intersezione con l'asse delle ascisse, che chiamiamo ad esempio x_1:

$$x_1 = x_b - f(x_b) \cdot \frac{x_b - x_a}{f(b) - f(x_a)}$$

A questo punto basterà iterare finche il punto di intersezione non è abbastanza vicino ad α.

La formula che si ottiene è la seguente:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Vale riguardo all'errore il seguente teorema:

Teorema 19.1: Errore del metodo delle secanti

Se $f \in C^2([a,b])$ ammette radice, allora il metodo converge localmente con ordine:

$$p = \frac{1+\sqrt{5}}{2} > 1$$

Questo solitamente risulta in un errore migliore del lineare ma peggiore del quadratico.

19.1.5 Implementazione MATLAB del metodo delle secanti

Vediamo quindi un implementazione MATLAB del metodo delle secanti scalare, per function handle:

```
function m = secant_method(f, a, b, k, tol)
    if nargin < 5
        tol = 0.01;
    end

    if nargin < 4
        k = 100;
    end

    for i = 1:k
        % print info
        fprintf("\nIteration %d\n", i);
        fprintf("\ta: %.4f\n", a);
        fprintf("\tb: %.4f\n", b);

        % update
        m = a - f(a) * ((b - a) / (f(b) - f(a)));
        a = b;
        b = m;

        % tolerance
        if abs(b - a) < tol
            break;
        end
    end
end
```

Dove si usa lo stesso criterio di stop di prima.

Vediamo l'algoritmo in azione per la stessa funzione dell'esmpio 19.1.3:

```
>> secant_method(f, 3, 5, 100, 0.001) % step piu' piccolo

Iteration 1
  a: 3.0000
  b: 5.0000

Iteration 2
  a: 5.0000
  b: 3.2500

Iteration 3
  a: 3.2500
  b: 3.4684

```

```
Iteration 4
  a: 3.4684
  b: 13.1076

Iteration 5
  a: 13.1076
  b: 3.4702

[...]

Iteration 23
  a: 4.4099
  b: 4.1135

Iteration 24
  a: 4.1135
  b: 4.1703

Iteration 25
  a: 4.1703
  b: 4.1911

Iteration 26
  a: 4.1911
  b: 4.1892

ans =

    4.1892
```

dove si è ridotto lo step per far fronte alle fluttuazioni che il metodo incontra attorno alla radice, per via della derivata particolarmente "piatta" che $f(x)$ ha vicino a 3. In particolare, si noti il salto che viene effettuato fra le iterazioni 4 e 5, che porta il valore da ~ 3 a ~ 13, cosa che chiaramente richiede alcuni passaggi per venire corretta.

19.2 Metodi stazionari a un punto

I **metodi stazionari a un punto** sono metodi della forma:

$$\begin{cases} x_0, \text{ dato o generato casualmente} \\ x_{n+1} = \phi(x_n) \end{cases}$$

In questo il metodo di Jacobi e di Gauss-Seidel sono metodi stazionari a un punto.

Il modo in cui si sceglie phi è (in analogia ai metodi per sistemi lineari) una funzione che verifica:

$$\phi(\alpha) = \alpha, \quad \forall \alpha : f(\alpha) = 0$$

Un modo canonico per costruire ϕ con questa proprietà è considerare:

$$\phi(x) = x - g(x)f(x)$$

con $g(x)$ tale che $g(x) \neq 0$ "vicino" ad α.

Si vede che:

$$\phi(\alpha) = \alpha - g(\alpha)f(\alpha) = \alpha$$

e chiaramente il fatto che $g(\alpha) \neq 0$ indica che:

$$\alpha \text{ punto fisso di } \phi \Leftrightarrow \alpha \text{ radice di } f$$

Potremmo quindi chiederci quando $\{\phi(x_n)\}_{n\in\mathbb{N}}$ è convergente. Sfruttiamo per questo il seguente teorema:

Teorema 19.2: Teorema di convergenza locale

Se si ha un intervallo $I \subseteq \mathbb{R}$, con $\alpha \in I$, $\phi(\alpha) = \alpha$, $\phi \in C^1(I)$ e esistono $\rho \in \mathbb{R}^+$ e $k \in (0,1)$ tali che:

$$|\phi'(x)| \leq k, \quad \forall x \in [\alpha - \rho, \alpha + \rho]$$

Allora valgono:

1. $\forall x_0 \in [\alpha - \rho, \alpha + \rho] \implies x_n \in [\alpha - \rho, \alpha + \rho]$;
2. $\forall x_0 \in [\alpha - \rho, \alpha + \rho]$ si ha:
$$\lim_{n\to+\infty} x_n = \alpha$$
3. α è l'unico punto fisso di $\phi(x)$ in $[\alpha - \rho, \alpha + \rho]$.

Vediamo di dimostrare.

1. Il punto 1) si dimostra per induzione. Poste le ipotesi, il caso $n = 0$ è banale (si prende lo stesso punto). Vogliamo quindi porre l'errore, attraverso il teorema di Lagrange:
$$|x_{n+1} - \alpha| = |\phi(x_n) - \phi(\alpha)| = |x_n - \alpha| \cdot |\phi'(\varepsilon)|, \quad \varepsilon \in [x_n, \alpha]$$
di questo abbiamo che il primo termine ($|x_n - \alpha|$) è $< \rho$, e il secondo ($|\phi'(\varepsilon)|$) è $<\leq k$, per cui l'errore successivo è:
$$k\rho < \rho$$
cioè:
$$x_{n+1} \in [\alpha - \rho, \alpha + \rho]$$
che è la tesi. □
2. Il punto 2) si dimostra a partire dallo stesso passaggio di prima:
$$|x_{n+1} - \alpha| = |x_n - \alpha| \cdot |\phi'(\varepsilon)| \leq k|x_n - \alpha|$$
maggiorando $|\phi'(\varepsilon)|$ con k. Per il calcolo esplicito si ha quindi che all'n-esimo passaggio si ha:
$$|x_n - \alpha| \leq k^{n+1} \cdot \rho$$
e quindi basterà dire:
$$\lim_{n\to+\infty} |x_{n+1} - \alpha| \leq \lim_{n\to+\infty} k^{n+1} \cdot \rho = 0$$
che è la tesi. □
3. Infine, il punto 3) si dimostra supponendo per assurdo che $\exists \tilde{\alpha} \in [\alpha - \rho, \alpha + \rho]$, $\tilde{\alpha} \neq \alpha$ tale che $\phi(\tilde{\alpha}) = \tilde{\alpha}$. In questo caso varrà:
$$|\alpha - \tilde{\alpha}| = |\phi(\alpha) - \phi(\tilde{\alpha})| = |\alpha - \tilde{\alpha}| \cdot |\phi'(\varepsilon)| \leq |\alpha - \tilde{\alpha}| \cdot k < |\alpha - \tilde{\alpha}|$$
che guardando agli estremi è un assurdo. □

Osserviamo quindi che se $|\phi'(\alpha)| < 1$, il metodo converge localmente perché per continuità della derivata $\exists \rho, k$ tali che $\rho > 0$, $k \in (0,1)$ e $|\phi'(x)| \leq k \; \forall x \in [\alpha - \rho, \alpha + \rho]$.

Ulteriori osservazioni si possono fare sul tipo di convergenza: questa può essere **monotona** o **alternata**. Infatti per l'errore vale:

$$(x_{n+1} - \alpha) = (x_n - \alpha) \cdot \phi'(\varepsilon), \quad \varepsilon \in [x_n, \alpha]$$

- Se $\phi'(x) > 0$ su $[\alpha - \rho, \alpha + \rho]$, allora gli errori $x_{n+1} - \alpha$ e $x_n - \alpha$ hanno lo stesso segno, cioè la convergenza è monotona.
- Altrimenti, se $\phi'(x) < 0$ su $[\alpha - \rho, \alpha + \rho]$, allora gli errori $x_{n+1} - \alpha$ e $x_n - \alpha$ hanno segno discorde, cioè la convergenza è alternata.

Possiamo poi dare il seguente teorema:

Teorema 19.3: Teorema sull'ordine di convergenza

Sia $\phi(x) \in C^p(I)$ e α punto fissso di ϕ (cioè $\alpha = \phi(\alpha)$) con $\alpha \in I$. Allora $\exists \rho > 0$ tale per cui $\forall x_0 \in [\alpha - \rho, \alpha + \rho]$ la successione $\{x_n\}$ converge con ordine $p \geq 1$ ad α se e solo se vale:

$$\phi'(\alpha) = \phi''(\alpha) = ... = \phi^{(p-1)}(\alpha) = 0, \quad \phi^{(p)}(\alpha) \neq 0$$

Osserviamo che p deve essere chiaramente un numero intero (un indice di derivata). Dimostriamo quindi le due coimplicazioni.

$\Leftarrow$: Abbiamo che $x_{n+1} = \phi(x_n)$. Sviluppando con Taylor nel punto α si ha:

$$x_{n+1} = \phi(\alpha) + \phi'(\alpha)(x_n - \alpha) + ... + \phi^{(p-1)}(\alpha)\frac{(x_n - \alpha)^{p-1}}{(p-1)!} + \phi^{(p)}(\varepsilon)\frac{(x_n - a)^p}{p!}$$

$$= \phi(\alpha) + \phi^{(p)}(\varepsilon)\frac{(x_n - \alpha)^p}{p!}, \quad \varepsilon \in [\alpha, x_n]$$

Quindi:

$$x_{n+1} - \alpha = \phi^{(p)}(\varepsilon)\frac{(x_n - \alpha)^p}{p!} \implies \frac{x_{n+1} - \alpha}{(x_n - \alpha)^p} = \frac{\phi^{(p)}(\varepsilon)}{p!}$$

$$\implies \lim_{n \to +\infty} \frac{x_{n+1} - \alpha}{(x_n - \alpha)^p} = \lim_{n \to +\infty} \frac{\phi^{(p)}(\varepsilon)}{p!}$$

Dato che $x_n \to \alpha$, si ha che:

$$\lim_{n \to +\infty} \phi^{(p)}(\varepsilon) = \phi^{(p)}(\alpha)$$

e:

$$\lim_{n \to +\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^p} = \frac{\phi^{(p)}(\alpha)}{p!} \neq 0$$

$\Rightarrow$: Prendiamo $1 \leq r \leq p - 1$ e facciamo vedere che:

$$\phi^{(r)}(\alpha) = 0$$

per induzione su r.

Se $r = 1$, si ha che:

$$\lim_{n\to+\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|} = \lim_{n\to+\infty} \frac{|\phi(x_n) - \phi(\alpha)|}{|x_n - \alpha|} = \lim_{n\to+\infty} \frac{|x_n - \alpha|\phi'(\varepsilon)}{|x_n - \alpha|} = \phi'(\alpha)$$

Prendiamo quindi per vero che $0 = \phi'(\alpha) = \phi''(\alpha) = ... = \phi^{(r-1)}(\varepsilon)$, e dimostriamo che vale anche per r. Per fare questo prenderemo lo sviluppo di Taylor in α troncato all'ordine r:

$$\phi(x_n) - \phi(\alpha) = (x_n - \alpha)\phi'(\alpha) + \frac{(x_n - \alpha)^2}{2}\phi''(\alpha) + ... + \frac{(x_n - \alpha)^{r-1}}{(r-1)!}\phi^{(r-1)}(\alpha) + \frac{(x_n - \alpha)^r}{r!}\phi^{(r)}(\varepsilon)$$

$$= \frac{(x_n - \alpha)^r}{2!}\phi^{(r)}(\varepsilon), \quad \varepsilon \in [x_n, \alpha]$$

da cui:

$$\frac{x_{n+1} - \alpha}{(x_n - \alpha)^r} = \frac{\phi^{(r)}(\varepsilon)}{r!} \implies \lim_{n\to+\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^r} = \lim_{n\to+\infty} \frac{\phi^{(r)}(\varepsilon)}{r!} = \frac{\phi^{(r)}(\alpha)}{r!} \implies \phi^{(r)}(\alpha) = 0$$

□

20 Lezione del 12-05-25

Vediamo alcuni esempi sui medi stazionari ad un punto.

20.0.1 Esempio: convergenza locale 1

Prendiamo la funzione:

$$f(x) = x\log(x) - 1$$

Innanzitutto dimostriamo che questa ha un'unica radice nell'intervallo $[0, +\infty)$, e poi studiamo la convergenza locale dei seguenti metodi:

1. L'espressione esplicita di x:
$$\phi_1(x) = \frac{1}{\log(x)}$$
2. L'espressione analoga, ma meno immediata (prendiamo la x da dentro il logaritmo):
$$\log(x) = \frac{1}{x} \implies \phi_2(x) = e^{\frac{1}{x}}$$
3. Ciò che troveremmo applicando la formula usuale:
$$\phi(x) = x - g(x)f(x) \implies \phi_3(x) = \frac{x+1}{\log(x)+1}$$
dove $g(x)$ è:
$$g(x) = \frac{1}{\log(x)+1}$$
in quanto:
$$\phi_3(x) = x - g(x)f(x) = x - \frac{x\log(x)+1}{\log(x)+1} = \frac{x\log(x)+1-x\log(x)-1}{\log(x)+1} = \frac{x+1}{\log(x)+1}$$

Vediamo quindi di dimostrare l'unicità della soluzione. Abbiamo che la derivata prima $f'(x)$ è:

$$f'(x) = \log(x) + 1 \implies \begin{cases} f'(x) < 0, & x < \frac{1}{e} \\ f'(x) > 0, & x > \frac{1}{e} \end{cases}$$

e si ha un minimo in $\frac{1}{e}$. Vediamo quindi il limite in 0^+:

$$\lim_{x \to 0} x\ log(x) - 1 = -1$$

Cioè la funzione parte da -1 in $x = 0$, scende fino al minimo $\frac{1}{e}$ e poi sale, incontrando l'asse delle ascisse una volta sola, nel punto α a cui siamo interessati.

Vorremo avere un punto $> \alpha$ per circoscrivere la regione in cui questo si trova. Per semplificare i calcoli, proviamo $x = e$, da cui:

$$f(e) = e \log(e) - 1 = e - 1 > 0$$

da cui individuiamo la regione:

$$\alpha \in \left[\frac{1}{e}, e\right]$$

Verifichiamo quindi le convergenze dei vari metodi proposti.

1. Prendiamo la derivata:

$$\phi_1'(x) = \left(\frac{1}{\log(x)}\right)' = \frac{1}{x}\left(-\frac{1}{\log^2(x)}\right) = -\frac{1}{x \log^2(x)}$$

   e valutiamone il modulo in α:

$$|\phi_1'(\alpha)| = \left|\frac{1}{\alpha \log^2(\alpha)}\right| = \left|\frac{1}{\log(\alpha)}\right| = |\alpha| \in \left[\frac{1}{e}, e\right]$$

   sfruttando $\alpha \log(\alpha) = 1$.

   Abbiamo quindi che α è potenzialmente grande fino a e, quindi non convergente. Per convincercene, prendiamo il punto 1:

$$f(1) = \log(1) - 1 = -1$$

   da cui sicuramente, in quanto $f(\alpha)$ sarà $0 > -1$ e quindi $\alpha > 1$:

$$|\phi_1'(\alpha)| > 1$$

   e il metodo non converge.

2. Prendiamo la derivata:

$$\phi_2'(x) = \left(e^{\frac{1}{x}}\right)' = -\frac{1}{x^2} e^{\frac{1}{x}}$$

   e valutiamone il modulo in α:

$$|\phi_2'(\alpha)| = \left|-\frac{\alpha}{\alpha^2}\right| = -\frac{1}{\alpha}$$

   sfruttando $\alpha = e^{\frac{1}{\alpha}}$.

   Da questo abbiamo che:

$$|\phi_2'(\alpha)| < 1$$

   in quanto $\alpha \in [1, e]$, e quindi il metodo converge localmente, in modo lineare ($\phi_2'(\alpha) \neq 0$).

3. Prendiamo la derivata:
$$\phi_3'(x) = \left(\frac{x+1}{\log(x)+1}\right)' = \frac{(\log(x)+1) - (x+1)\frac{1}{x}}{(\log(x)+1)^2} = \frac{\log(x) - \frac{1}{x}}{(\log(x)+1)^2}$$
$$= \frac{x\log(x) - 1}{x(\log(x)+1)^2} = \frac{f(x)}{x(\log(x)+1)^2}$$
da cui si ha $f(x)$ al denominatore, e quindi sicuramente:
$$\phi_3'(\alpha) = 0$$
e si converge con ordine maggiore o uguale al lineare.
Vediamo la derivata seconda per confermare l'ordine di convergenza:
$$\phi_3''(x) = \frac{(\log(x)+1)^3 x - (x\log(x) - 1)\left[(\log(x)+1)^2 + \frac{2(\log(x)+1)}{x}\right]}{x^2(\log(x)+1)^4}$$
che sostituendo α diventa:
$$\phi_3''(\alpha) = \frac{\alpha(\log(\alpha)+1)^3}{\alpha^2(\log(\alpha+1)^4)} = \frac{1}{\alpha(\log(\alpha)+1)} \neq 0$$
cioè l'ordine di convergenza è 2.

20.0.2 Esempio: convergenza locale 2

Prendiamo adesso un esempio dove ci è data la funzione di punto fisso:
$$\phi(x) = 1 + a\log(x) + b\log^2(x), \quad a, b \in \mathbb{R}$$
con punto fisso $\alpha = 1$. Siamo interessati a valutarne l'ordine di convergenza al variare dei parametri a e b, e determinare quando la convergenza è lineare e monotona. Prendiamo quindi la derivata:
$$\phi'(x) = \frac{a}{x} + 2b\frac{\log(x)}{x} \implies \phi'(1) = a$$
e il metodo converge localmente per $a \in (-1, 1)$. In particolare, il metodo converge in modo monotono (localmente) quando $a \in (0, 1)$, e per $a = 0$ converge in modo superlineare.

Valutiamo quindi l'ordine di convergenza per $a = 0$:
$$\phi(x) = 1 + b\log^2(x)$$
da cui le derivate:
$$\phi'(x) = 2b\frac{\log(x)}{x} \implies \phi'(\alpha) = 0$$
$$\phi''(x) = \frac{2b}{x^2} - 2b\frac{\log(x)}{x^2} = 2b\frac{(1 - \log(x))}{x^2} \implies \phi''(1) = 2b$$
cioè per $b \neq 0$ la convergenza è quadratica, e per $b = 0$ la convergenza è in un passo ($x_1 = \alpha = 1$ per ogni x_0).

Nel caso $a = 0$ per studiare la convergenza monotona dobbiamo studiare il segno di $\phi'(x)$ vicino ad $\alpha = 1$.

$b > 0$: vale $\phi'(x) > 0$ su $(1, 1+\rho)$ con ρ sufficientemente piccolo, quindi si ha convergenza monotona con $x_0 \in (1, 1+\rho)$.

$b < 0$: vale $\phi'(x) > 0$ su $(1-\rho, 1)$ con ρ sufficientemente piccolo, quindi si ha convergenza monotona con $x_0 \in (1-\rho, 1)$.

20.1 Metodo di Newton

Vediamo un particolare metodo stazionario ad un punto, cioè il **metodo di Newton**:

$$\phi(x) = x - \frac{f(x)}{f'(x)}$$

e quindi il passo iterativo è:

$$\begin{cases} x_0, \quad \text{dato} \\ x_{n+1} = \phi(x_n) = x_n - \frac{f(x_n)}{f'(x_n)} \end{cases}$$

L'interpretazione geometrica del metodo di Newton è quella classica, cioè si prende la retta tangente che si stacca da x_i e si chiama x_{i+1} l'intersezione di questa con l'asse delle ascisse. Per questo viene detto anche *metodo delle tangenti*.

Il metodo di Newton è solitamente molto veloce, ma può presentare comporamenti non ottimali nel caso di funzioni particolarmente *"schiacciate"*, cioè con derivata $|f'(x)| << 1$ vicino ad α, fra cui ad esempio le polinomiali x^n con $n >> 1$.

Per valutare questi punti sfavorevoli diamo la definizione di **radice di ordine r**:

Definizione 20.1: Radice di ordine r

Sia $f : [a, b] \to \mathbb{R}$ continua e $\alpha : f(\alpha) = 0$. In questo caso α è detta radice (o zero) di ordine $r > 0$ se vale:

$$\lim_{x \to \alpha} \frac{f(x)}{(x - \alpha)^r} = c < +\infty \ (c \neq 0)$$

Quindi, se $f(x) \in C^r([a, b])$, allora α è radice di ordine r se:

$$f(\alpha) = f'(\alpha) = ... = f^{(r-1)}(\alpha) = 0, \quad f^{(r)}(\alpha) \neq 0$$

In particolare, se $r = 1$, la radice si dice **semplice**.

Riguardo al teorema di Newton, vale quindi il teorema:

Teorema 20.1: Convergenza del metodo di Newton

Se α è radice semplice, e $f \in C^2([a, b])$, allora il metodo converge localmente con ordine $p \geq 2$.

In particolare varrà:

$$\lim_{n \to +\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^2} = \frac{1}{2} \frac{f''(\alpha)}{f'(\alpha)}$$

per cui:

$$\begin{cases} f''(\alpha) = 0 \implies p > 2 \\ f''(\alpha) \neq 0 \implies p = 2 \end{cases}$$

Vediamo di dimostrare il teorema. Inizieremo prendendo la funzione di punto fisso:

$$\phi(x) = x - \frac{f(x)}{f'(x)} \implies \phi'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}$$

che sostituendo α diventa:

$$\phi'(\alpha) = \frac{f(\alpha)f''(\alpha)}{(f'(\alpha))^2} = 0$$

per cui la convergenza è almeno quadratica.

Continuando si ha:

$$\phi''(x) = \frac{(f'(x)f''(x) + f(x)f'''(x))(f'(x))^2 - f(x)f''(x)(2f'(x)f''(x))}{(f'(x))^4}$$

che sostituendo α diventa:

$$\phi''(\alpha) = \frac{f'(\alpha)f''(\alpha)\,(f'(\alpha))^2}{(f'(\alpha))^4} = \frac{f''(\alpha)}{f'(\alpha)}$$

da cui:

$$\begin{cases} f''(\alpha) \neq 0 \implies p = 2 \\ f''(\alpha) = 0 \implies p > 2 \end{cases}$$

In particolare, si può prendere:

$$\frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^2} = \frac{|\phi(x_n) - \phi(\alpha)|}{|x_n - \alpha|^2} = \frac{|\phi(\alpha) + \phi'(\alpha)(x_n - \alpha) + \frac{1}{2}\phi''(\varepsilon)(x_n - \alpha)^2 - \phi(\alpha)|}{|x_n - \alpha|^2}$$

sfruttando lo sviluppo di Taylor di $\phi(x_n)$ attorno a α con errore di Laplace:

$$\phi(x_n) = \phi(\alpha) + \phi'(\alpha)(x_n - \alpha) + \frac{1}{2}\phi''(\alpha)(x_n - \alpha)^2$$

da cui:

$$= \frac{|\phi'(\alpha)(x_n - \alpha) + \frac{1}{2}\phi''(\varepsilon)(x_n - \alpha)^2|}{|x_n - \alpha|^2} = \frac{1}{2}\phi''(\varepsilon) \xrightarrow{n \to +\infty} \frac{1}{2}\phi''(\alpha) = \frac{1}{2}\frac{f''(\alpha)}{f'(\alpha)}$$

con $n \to +\infty$, noto $\varepsilon \in [x_n, \alpha]$, per cui si ha l'ultima tesi del teorema. □

Osserviamo che nel metodo di Newton si può guardare direttamente a $f(x)$ e alle sue derivate in α per studiare la convergenza del metodo. Con i generici metodi di punto fisso dovevamo invece guardare alla funzione di punto fisso $\phi(x)$.

Vediamo quindi il caso delle radici multiple, cioé $f(x)$ in forma:

$$f(x) = g(x)(x - \alpha)^r, \quad g(\alpha) \neq 0$$

Si ha che $r \in \mathbb{N}$, $r > 1$ e:

$$f'(\alpha) = 0$$

e quindi il metodo di newton si ridefinisce come:

$$\phi(x) = \begin{cases} x - \frac{f(x)}{f'(x)}, & x \neq \alpha \\ \alpha, & x = \alpha \end{cases}$$

Ricordiamo che:

$$\phi'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}$$

dove $f'(x)$ compare al denominatore, quindi non si può valutare in α.

Prendiamo allora le derivate successive di $f(x)$:

1. $f(x) = g(x)(x-\alpha)^r$
2. $f'(x) = g'(x)(x-\alpha)^r + rg(x)(x-\alpha)^{r-1} = (x-\alpha)^{r-1}(g'(x)(x-\alpha) + rg(x))$
3. $f''(x) = g''(x)(x-\alpha)^r + 2rg'(x)(x-\alpha)^{r-1} + r(r-1)g(x)(x-\alpha)^{r-1}$

$$= (x-\alpha)^{r-2}(g''(x)(x-\alpha)^2 + 2rg'(x)(x-\alpha) + r(r-1)g(x))$$

e sostituiamo in $\phi'(x)$:

$$\phi'(x) = \frac{g(x)(x-\alpha)^r \cdot (x-\alpha)^{r-2}(g''(x)(x-\alpha^2) + rg'(x)(x-\alpha) + r(r-1)g(x))}{((x-\alpha)^{r-1}(g'(x)(x-\alpha) + rg(x)))^2}$$

$$= \frac{g(x)(g''(x)(x-\alpha^2) + 2rg'(x)(x-\alpha) + r^2g^2(x) - rg^2(x)}{(g'(x)(x-\alpha) + rg(x))^2}$$

da cui:

$$\phi'(\alpha) = \frac{r^2g^2(\alpha) - rg^2(\alpha)}{r^2g^2(\alpha)} = 1 - \frac{1}{r} \neq 0$$

quando $r > 1$, perciò:

$$|\phi'(\alpha)| = 1 - \frac{1}{r} < 1$$

e la convergenza è lineare, dove più è alto r più è lenta la convergenza.

Una proprietà è che se si conosce r (l'ordine della radice) si può modificare Newton per ritrovare la convergenza quadratica. In particolare si può dimostrare che:

$$x_{n+1} = x_n - r \cdot \frac{f(x_n)}{f'(x_n)} = \phi(x_n)$$

è tale per cui $\phi'(\alpha) = 0$, e quindi la convergenza è almeno quadratica.

20.1.1 Implementazione MATLAB del metodo di Newton

Vediamo infine come si implementa il metodo di Newton su MATLAB. Il codice avrà l'aspetto:

```
function x0 = newton_method(f, x0, k, tol)
    if nargin < 5
        tol = 0.01;
    end

    if nargin < 4
        k = 100;
    end

    % differentiat

    % e
    syms x;
    diff_f = str2func(['@(x)', char(diff(f(x)))]);

    for i = 1:k
        % print info
        fprintf("\nIteration %d\n", i);
        fprintf("\tx: %.4f\n", x0);

        % update
```

```
        x_old = x0;
        x0 = x0 - f(x0) / diff_f(x0);

        % tolerance
        if abs(x0 - x_old) < tol
            break;
        end
    end
end
```

Usiamo questo algoritmo per risolvere lo stesso problema degli esempi 19.1.3 e 19.1.5, che ricordiamo era trovare la radice in $[a, b] = [3, 5]$ di:

$$f(x) = (x-3)^4 - 2$$

Avremo quindi:

```
>> f = @(x) (x - 3) .^ 4 - 2;
>> newton_method(f, 3.1)

Iteration 1
  x: 3.1000

Iteration 2
  x: 503.0750

Iteration 3
  x: 378.0563

Iteration 4
  x: 284.2922

Iteration 5
  x: 213.9691

[...]

Iteration 21
  x: 5.1467

Iteration 22
  x: 4.6606

Iteration 23
  x: 4.3546

Iteration 24
  x: 4.2171

Iteration 25
  x: 4.1902

ans =

    4.1892
```

Dove notiamo sono richiesti un numero significativo di passaggi, sopratutto per l'esplosione del valore fino a ~ 500 alla seconda iterazione. Vedremo nello specifico come risolvere problemi di questo tipo nella sezione 21.1.3.

Inoltre, vediamo di essere dovuti partire da $x_0 = 3.1$, in quanto in $x_0 = 3$ si avrebbe avuto derivata prima $f'(3) = 0$.

21 Lezione del 16-05-25

21.1 Intervallo di convergenza

Vorremo estendere la definizione di convergenza data in 19.2 in un certo intervello $[a, b]$.

Iniziamo con un esempio:

21.1.1 Esempio: convergenza del metodo di Newton

Prendiamo la funzione:

$$f(x) = 3x^2 e^{-x} - 1$$

e poniamo:

$$f(x) = 0$$

cioè equivalentemente uguagliamo le due funzioni:

$$3x^2 = e^x$$

che sono rispettivamente una parabola simmetrica rispetto all'asse y e un esponenziale.

Queste intersezioni saranno 3, che chiamiamo $\alpha_{1,2,3}$. Avremo che, scelti i punti $x_{0,1,2}$:

$$x_0 = -\frac{1}{2}, \quad x_1 = 1, \quad x_2 = 5$$

si avrà la convergenza nei rispettivi punti $\alpha_{1,2,3}$.

21.1.2 Convergenza in un intervallo

Una domanda che potremo farci, in relazione all'ultimo esempio, è quando possiamo assicurare che un metodo converge $\forall x_0 \in [a, b]$.

Esiste il seguente teorema:

Teorema 21.1: Teorema di convergenza in un intervallo

Se $\phi(x) \in C^1([a, b])$ e:

1. $|\phi'(x)| < 1, \quad \forall x \in [a, b]$;
2. $\phi(x) \in [a, b], \quad \forall x \in [a, b]$;

allora $\phi(x)$ converge su $[a, b]$.

Dove l'ipotesi in più rispetto al teorema di convergenza locale (teorema 19.2) è la 2.

21.1.3 Condizioni di convergenza globale per Newton

Nello specifico per Newton, la convergenza in un intervallo si assicura attraverso condizioni su concavità e convessità.

Esiste infatti il seguente teorema:

Teorema 21.2: Convergenza globale per Newton

Se $\alpha \in [a,b]$ è radice semplice di $f \in C^2([a,b])$, e f, f'' sono di segno costante su $[a,b]$, il metodo di Newton converge globalmente in maniera superlineare (≥ 2) $\forall x_0$ che verifica:

$$f(x_0) \cdot f''(x_0) > 0$$

Dove si mette enfasi su *globalmente*, in quanto il teorema 20.1 ha valenza solo locale. Sostanzialmente, questo teorema ci è utile per fare stime iniziali, probabilmente arbitrariamente distanti dall'origine, che convergano con velocità superlinare.

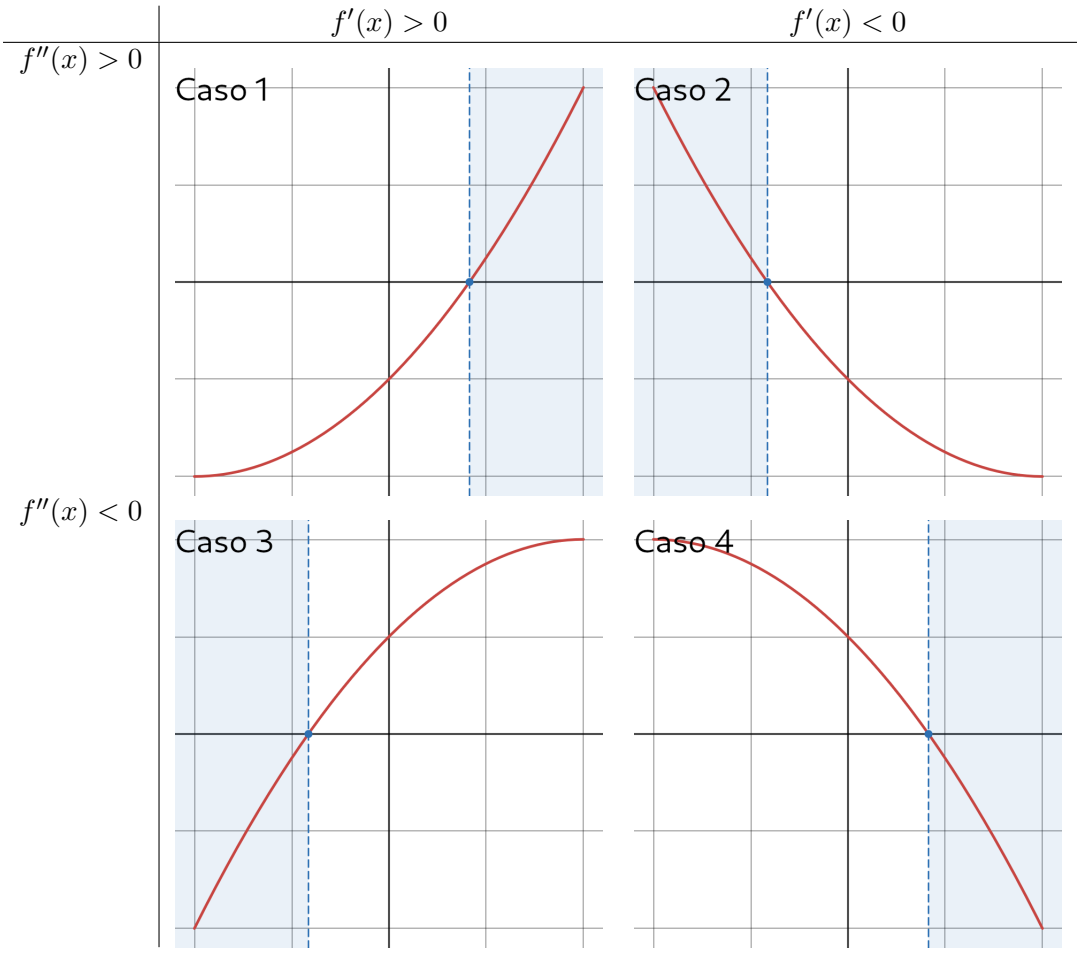
Osserviamo che:

- Se la funzione è **concava**, chiediamo $f(x_0) < 0$;
- Se la funzione è **convessa**, chiediamo $f(x_0) > 0$.

In particolare distinguiamo quindi i 4 casi:

1. $f' > 0$, $f'' > 0$ convessa crescente, vorremo partire a *destra* di α;
2. $f' < 0$, $f'' > 0$ convessa decrescente, vorremo partire a *sinistra* di α;
3. $f' > 0$, $f'' < 0$ concava crescente, vorremo partire a *sinistra* di α;
4. $f' < 0$, $f'' < 0$ concava decrescente, vorremo partire a *destra* di α;

Cioè riassumendo:



21.1.4 Esempio: ricerca di un intervallo di convergenza

Prendiamo la funzione:

$$f(x) = x + \log(x)$$

e cerchiamo un intervallo dove questa ha una radice, in particolare fornendo un punto di partenza x_0 per cui Newton risulta convergente.

Chiaramente quello che cerchiamo sono le intersezioni dei grafici:

$$\log(x) = -x$$

che saranno una sola, compresa fra 0 e 1, perciò:

$$\alpha \in (0, 1)$$

Vediamo quindi se si applicano le condizioni di convergenza per Newton. Calcoliamo innanzitutto le derivate:

$$f'(x) = 1 + \frac{1}{x}, \quad > 0 \text{ su } (0, 1)$$

$$f''(x) = -\frac{1}{x^2}, \quad < 0 \text{ su } (0, 1)$$

Per 1 la condizione non è soddisfatta, in quanto si troverà a *destra* di α (e trovandoci nel caso 3, vorremo prendere punti a *sinistra*).

Dovremo quindi cercare un $x_0 \in (0, \alpha)$, ovvero $x_0 : f(x_0) < 0$. Proviamo $\frac{1}{e}$:

$$f\left(\frac{1}{e}\right) = \frac{1}{e} - 1 < 0$$

per cui possiamo scegliere $x_0 = \frac{1}{e}$ ed essere sicuri, per il teorema 21.2, di convergere ad α.

21.1.5 Esempio: miglioramento della convergenza con Newton

Avevamo visto nell'esempio 20.1.1 come Newton poteva dare convergenza non ottimale per alcuni valori.

In particolare, avevamo preso la funzione:

$$f(x) = (x - 3)^4 - 2$$

e stavamo cercando la radice fra 3 e 5. Notiamo che in questo intervallo $f''(x) > 0$, e qundi vogliamo partire da un punto x_0 tale per cui $f(x_0) > 0$. Prendiamo ad esempio 5, con $f(5) = 14$:

```
>> newton_method(f, 5)

Iteration 1
  x: 5.0000

Iteration 2
  x: 4.5625

Iteration 3
  x: 4.3029

Iteration 4
```

```
  x: 4.2033

Iteration 5
  x: 4.1895

ans =

    4.1892
```

dove notiamo che troviamo una soluzione in sole 5 iterazioni, contro le 25 di cui avevamo avuto bisogno partendo da $x_0 = 3.1$ nell'esempio precedente.

21.2 Equazioni non lineari in $\mathbb{R}^m$

Vediamo quindi di considerare sempre equazioni non lineari, ma in più variabili (in particolare m).

Cercheremo quindi $f(x) = 0$ per $f : \mathbb{R}^m \to \mathbb{R}^m$, cioè:

$$f = \begin{pmatrix} f_1(x_1, ..., x_m) \\ \vdots \\ f_m(x_1, ..., x_m) \end{pmatrix}$$

e vorremo trovare $\alpha \in \mathbb{R}^m$, cioè:

$$\alpha = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{pmatrix}$$

tale che:

$$f(\alpha) = \begin{pmatrix} f_1(\alpha_1, ..., \alpha_m) \\ \vdots \\ f_m(\alpha_1, ..., \alpha_m) \end{pmatrix} = 0$$

Questo problema corrisponde effettivamente a cercare soluzioni a sistemi di m equazioni in m incognite, non lineari.

In particolare, ci può essere utile per trovare i punti stazionari di funzioni in più variabili, ovvero le soluzioni di $\nabla F(x) = 0$, con $F(x) : \mathbb{R}^m \to \mathbb{R}$ cioè:

$$\nabla F(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} F(x_1, ..., x_m) \\ \vdots \\ \frac{\partial}{\partial x_m} F(x_1, ..., x_m) \end{pmatrix}$$

Nel caso dei sistemi lineari, avevamo la condizione semplice:

$$f(x) = b - Ax, \quad x = A^{-1}b$$

mentre in questo caso non avremo a disposizione tali semplificazioni.

Ci potremo quindi chiedere come risolvere $f(x)$ nel caso non lineare. La soluzione sarà quella di generalizzare i metodi di punto fisso a $\mathbb{R}^m$, definendoli come:

$$x = \phi(x), \quad \phi : \mathbb{R}^m \to \mathbb{R}^m$$

e quindi usare la solita formula di aggiornamento:

$$x^{(n+1)} = \phi(x^{(n)})$$

Una possibile formula di aggiornamento può essere:

$$\phi(x) = x - G(x) \cdot f(x)$$

dove $G(x)$ è:

$$G(x) : \mathbb{R}^m \to \mathbb{R}^{m \times m}$$

cioè:

$$G(x) = \begin{pmatrix} G_{11}(x) & ... & G(1m)(x) \\ \vdots & \vdots & \vdots \\ G_{m1}(x) & ... & G(mm)(x) \end{pmatrix}$$

e si cerca $G(x)$ tale che $G(\alpha)$ è non singolare, ovvero

$$\det(G(\alpha)) \neq 0$$

Quindi si considera l'iterazione:

$$\begin{cases} x^{(0)} \in \mathbb{R}^m, \quad \text{dato} \\ x^{(n+1)} = x^{(n)} - G(x^{(n)}) f(x^{(n)}) = \phi(x^{(n)}) \end{cases}$$

Ridiamo la definizione di convergenza in $\mathbb{R}^n$:

Definizione 21.1: Convergenza multivariabile

La successione $\{x^{(n)}\}_{n \in \mathbb{N}}$ converge ad $\alpha \in \mathbb{R}^m$ se:

$$\lim_{n \to +\infty} |x^{(n)} - \alpha|_2 = 0$$

In particolare, la convergenza ha **ordine** $p > 0$ se vale:

$$\lim_{n \to +\infty} \frac{|x^{(n+1)} - \alpha|_2}{|x^{(n)} - \alpha|_2^p} = c$$

con $c \neq 0$, $c < +\infty$, e in particolare nel caso $p = 1$ è $C \in (0, 1)$, cioè si ha la stessa situazione della definizione 19.3.

Nel caso esistano e siano continue le derivate delle componenti di $\phi(x)$ si introduce il **Jacobiano** di ϕ, come la funzione $J_\phi : \mathbb{R}^m \to \mathbb{R}^{m \times m}$ definita da:

$$J_\phi(x) = \begin{pmatrix} \frac{\partial \phi_1}{\partial x_1}(x) & ... & \frac{\partial \phi_1}{\partial x_m}(x) \\ \vdots & \vdots & \vdots \\ \frac{\partial \phi_m}{\partial x_1}(x) & ... & \frac{\partial \phi_m}{\partial x_m}(x) \end{pmatrix}$$

oppure per componenti:

$$[J_\phi(x)]_{ij} = \frac{\partial \phi_i}{\partial x_j}(x), \quad i, j = 1, ..., m$$

Potremo quindi enunciare il seguente teorema:

Teorema 21.3: Convergenza locale multivariabile

Sia $\phi(x) \in C^1(\Omega)$, $\Omega \subseteq \mathbb{R}^m$ con $\alpha \in \Omega$ tale che $\phi(\alpha) = \alpha$. Se:

$$\rho\left(J_\phi(\alpha)\right) < 1$$

cioè il raggio spettrale del Jacobiano di ϕ in α è < 1, allora $\exists \delta > 0$ tale che:

$$\forall x^{(0)} : |x^{(0)} - \alpha|_2 < \delta$$

in cui la successione:

$$x^{(n+1)} = \phi(x^{(n)})$$

converge ad α, ed α è l'unico punto fisso di ϕ in:

$$\{z \in \mathbb{R}^m : |z - \alpha|_2 < \delta\}$$

Osserviamo quindi che se per una qualsiasi norma si trova che:

$$|J_\phi(\alpha)| < 1$$

allora vale la tesi del teorema, ovvero la convergenza locale. Viceversa, se si trova che:

$$|J_\phi(\alpha)| \geq 1$$

non si può concludere nulla.

21.2.1 Esempio: convergenza multivariabile

Prendiamo la funzione, con $m = 2$:

$$f(x_1, x_2) = \begin{pmatrix} x_1 - \frac{1}{4}(x_1^2 + x_2^2) \\ x_2 + x_1 - 2 \end{pmatrix}$$

e cerchiamone i punti $f(\alpha) = 0$, cioè che soddisfano:

$$\begin{cases} x_1 - \frac{1}{4}(x_1^2 + x_2^2) = 0 \\ x_2 + x_1 - 2 = 0 \end{cases}$$

Dalla f, posto $f(x) = 0$ si derivano le identità:

$$\begin{cases} x_1 = \frac{1}{4}(x_1^2 + x_2^2) \\ x_2 = 2 - x_1 \end{cases}$$

per cui si può considerare $\phi(x)$:

$$\phi(x) = \begin{pmatrix} \frac{1}{4}(x_1^2 + x_2^2) \\ 2 - x_1 \end{pmatrix}$$

e prendere l'aggiornamento dal metodo stazionario ad un punto:

$$x^{(n+1)} = \phi(x^{(n)})$$

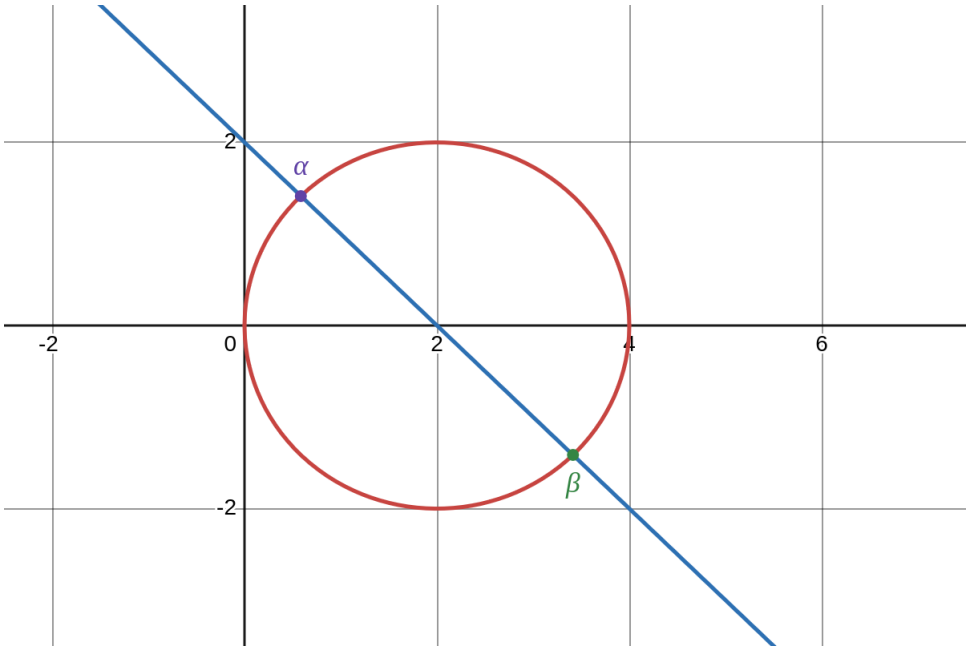
Prima di valutare la convergenza, cerchiamo di trovare i punti α in maniera analitica.
Per la prima equazione, avremo che:

$$x - \frac{1}{4}(x^2 + y^2) = 0 \Rightarrow -\frac{(x-2)^2}{4} - \frac{y^2}{4} + 1 = 0$$

cioè si trova la circonferenza di centro $(2, 0)$ e raggio 2.
Per la seconda, si ha invece semplicemente la retta:

$$y = 2 - x$$

Sul grafico, questo ha l'aspetto:



Le soluzioni di $f(x) = 0$ saranno quindi i punti di intersezione fra la circonferenza e la retta, che vediamo essere 2, uno sopra l'asse x e l'altro sotto. Consideriamone solo il primo, che chiamiamo (α_1, α_2).

Calcoliamo quindi il Jacobiano:

$$J_\phi(x) = \begin{pmatrix} \frac{x_1}{2} & \frac{x_2}{2} \\ -1 & 0 \end{pmatrix}$$

Vediamo che questa ha norma ≥ 1, per cui bisogna calcolare gli autovalori.
Guardiamo quindi al polinomio caratteristico:

$$\left(\frac{x_1}{2} - \lambda\right)(-\lambda) + \frac{x_2}{2} = \lambda^2 - \frac{x_1}{2}\lambda + \frac{x_2}{2} \rightarrow \lambda_{1,2} = \frac{\frac{x_1}{2} \pm \sqrt{\frac{x_1^2}{4} - 2x_2}}{2}$$

cioè:

$$\lambda_{1,2} = \frac{\frac{\alpha_1}{2} \pm \sqrt{\frac{\alpha_1^2}{4} - 2\alpha_2}}{2}$$

Valutando la posizione generica di α, si ha che:

$$\alpha_1 < 1, \quad \alpha_2 \in (1, 2)$$

che formalmente si ottiene notando che $\alpha_1 \geq 1$ porta ad un assurdo, e $\alpha_2 = 2 - \alpha_1$.

Da questo deriverà che il discriminante:

$$\Delta = \frac{\alpha_1^2}{4} - 2\alpha_2 < 0$$

e cioè $\lambda_{1,2}^{\alpha}$ sono necessariamente numeri complessi in forma:

$$\lambda_{1,2}^{\alpha} = \frac{\alpha_1}{4} \pm i\sqrt{\frac{\alpha_2}{2} - \frac{\alpha_1^2}{16}}$$

Il modulo di uno di questi (prendiamo il primo) sarà:

$$|\lambda_1^{\alpha}|^2 = \frac{\alpha_1^2}{16} + \frac{\alpha_1}{2} - \frac{\alpha_1^2}{16} = \frac{\alpha_1}{2}$$

per cui il raggio spettrale sarà $\frac{\alpha_1}{2} < 1$ (dal fatto che $\alpha_1 < 1$), e quindi il metodo convergente.

21.2.2 Esempio: convergenza multivariabile con MATLAB

Facciamo una trattazione meno analitica e più numerica della convergenza nello scorso esempio.

Avevamo individuato due radici, chiamiamole α (già ampiamente discussa) e β. Queste si calcono esplicitamente prendendo dalla seconda equazione:

$$x_2 + x_1 - 2 = 0 \implies x_2 = 2 - x_1$$

e sostituendo nella prima:

$$x_1 - \frac{1}{4}(x_1^2 + (2 - x_1)^2) = -\frac{x_1^2}{2} + 2x_1 - 1$$

da cui:

$$\alpha = (2 - \sqrt{2}, \sqrt{2}), \quad \beta = (2 + \sqrt{2}, -\sqrt{2})$$

Potremo chiaramente rifare le stesse considerazioni di prima con i valori effettivi di α, trovando gli stessi risultati. Ci basti sapere che il punto α fa (come avevamo visto) da attrattore, mentre il metodo non converge in β, scelto l'aggiornamento:

$$\phi(x) = \begin{pmatrix} \frac{1}{4}(x_1^2 + x_2^2) \\ 2 - x_1 \end{pmatrix}$$

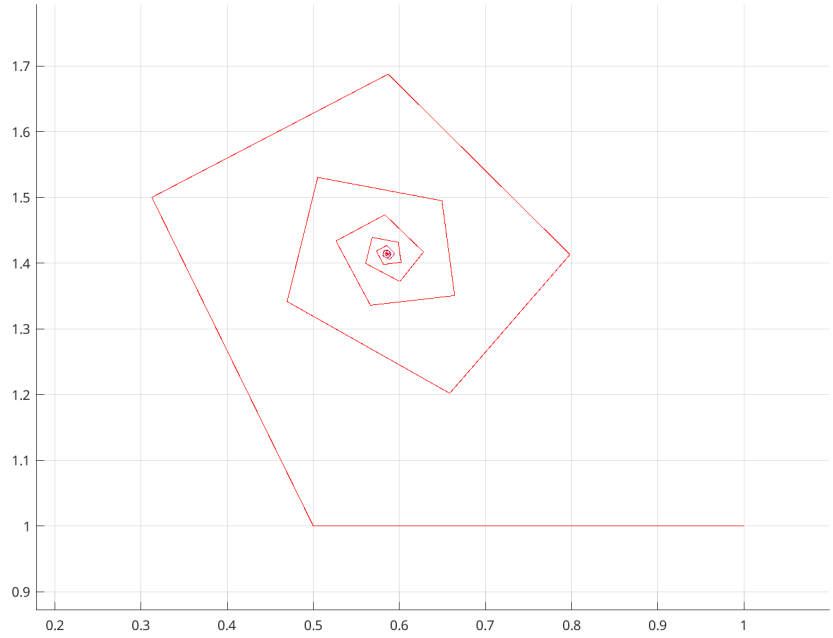
Vediamo di implementare in MATLAB un risolutore che applichi nella pratica tale funzione. Questo si tradurrà nel dire:

```
function x0 = nonlin_r2_ex(x0)
    % fixed point
    phi1 = @(x1, x2) 0.25 * (x1.^2 + x2.^2);
    phi2 = @(x1, x2) 2 - x1;

    % iterations
    k = 100;

    for i = 1:k
        x1 = zeros(2, 1);
```

```
        x1(1) = phi1(x0(1), x0(2));
        x1(2) = phi2(x0(1), x0(2));

        fprintf("\nx0: (%.4f, %.4f)\n", x0(1), x0(2));
        fprintf("x1: (%.4f, %.4f)\n", x1(1), x1(2));

        plot([x0(1), x1(1)], [x0(2), x1(2)], 'r-');
        x0 = x1;
    end
end
```

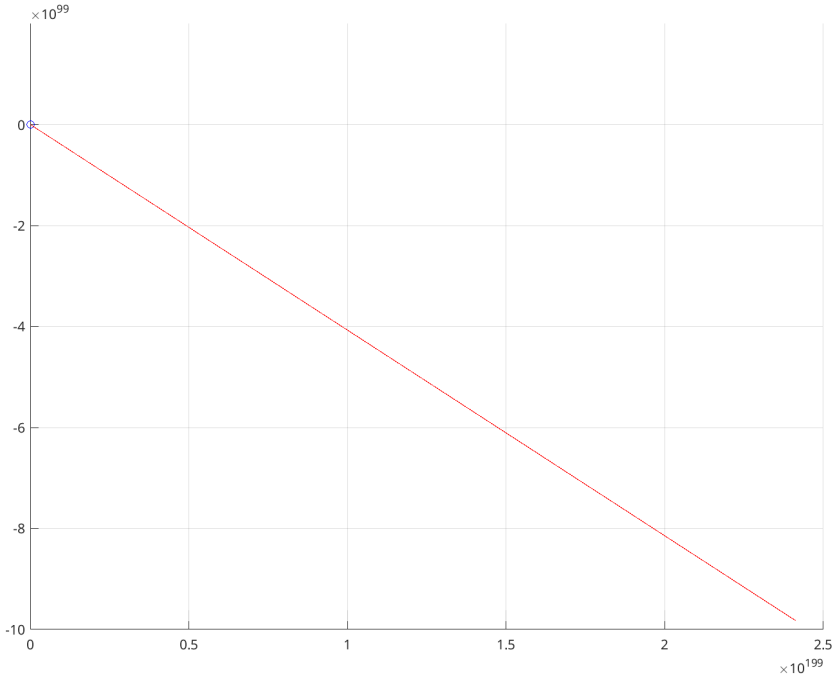
Dove notiamo che si stampano alcune informazioni intermedie sui punti attraversati, e si traccia il percorso seguito su un grafico.

- Se si sceglie come punto di inizio $(1, 1)$, con l'intenzione di convergere ad α, si ha il seguente andamento:



da dove si nota chiaramente che in un numero limitato di passaggi ci si avvicina abbastanza velocmente ad α;

- Cercare di fare la stessa cosa ad esempio dal punto $(4, -1)$ non dà risultati simili, in quanto notiamo dal grafico:



che il risultato esplode velocemente all'infinito, e quindi chiaramente il metodo non converge.

21.3 Metodo di Newton-Raphson

Vediamo quindi la versione multivariabile del metodo di Newton-Raphson.

Quindi, se abbiamo $f : \mathbb{R}^m \to \mathbb{R}^m$ il metodo di Newton si generalizza nel seguente modo:

$$x^{(n+1)} = x^{(n)} - J_f(x^{(n)})^{-1} f(x^{(n)})$$

Osserviamo che ad ogni passo si valuta $J_f(x^{(n)})$ e $f(x^{(n)})$, si risolve il sistema lineare:

$$J_f(x^{(n)})y = f(x^{(n)})$$

ed infine si chiama $x^{(n+1)} = x^{(n)} - y$.

L'aggiornamento complessivo sarà quindi:

$$\begin{cases} x^{(0)}, \quad \text{dato} \\ x^{(n+1)} = x^{(n)} - J_f(x^{(n)})^{-1} f(x^{(n)}) \end{cases}$$

Per quanto riguarda la convergenza si ha il seguente risultato:

Teorema 21.4: Convergenza del metodo di Newton-Rapson

Sia $f \in C^2(\Omega)$, $\Omega \subseteq \mathbb{R}^m$, $\alpha \in \Omega$ tale che $f(\alpha) = 0$. Se $J_f(\alpha)$ è non singolare, allora $\exists s \subseteq \Omega$ intorno di α tale che $\forall x^{(0)} \in S$ la successione generata dal metodo di Newton-Raphson a partire da $x^{(0)}$ converge e vale:

$$|x^{(n+1)} - \alpha|_2 \leq \beta |x^{(n)} - \alpha|_2^2$$

ovvero la convergenza è almeno quadratica (quindi superlineare).

21.3.1 Esempio: convergenza del metodo di Newton-Raphson

Vediamo quindi come valutare la convergenza del metodo di Newton-Raphson per il sistema:

$$f(x_1, x_2) = \begin{pmatrix} \frac{1}{3}(x_1 - x_2) + x_1^2 \\ \frac{1}{3}(-x_1 + x_2) + x_1 x_2 \end{pmatrix}$$

Vogliamo quindi determinare le radici di f, che si otterrano come:

$$\begin{cases} \frac{x_1 - x_2}{3} + x_1^2 = 0 \\ \frac{x_2 - x_1}{3} + x_1 x_2 = 0 \end{cases} \Rightarrow \begin{cases} x_2 = 3x_1^2 + x_1 \\ \left(\frac{-x_1 + x_1 + 3x_1^2}{3}\right) + x_1(3x_1^2 + x_1) = 0 \end{cases} \Rightarrow \begin{cases} x_2 = 3x_1^2 + x_1 \\ 2x_1^2 + 3x_1^3 = 0 \end{cases}$$

da cui si ottengono i due punti:

$$\alpha = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \beta = \begin{pmatrix} -\frac{2}{3} \\ \frac{2}{3} \end{pmatrix}$$

Vorremo quindi valutare il Jacobiano:

$$J_f(x_1, x_2) = \begin{pmatrix} \frac{1}{3} + 2x_1 & -\frac{1}{3} \\ -\frac{1}{3} + x_2 & \frac{1}{3} + x_1 \end{pmatrix}$$

Che nei due punti α e β risulta:

α :

$$J_f(\alpha) = \begin{pmatrix} \frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

da cui $\det = 0$, e la convergenza non è superlineare;

β :

$$J_f(\beta) = \begin{pmatrix} -1 & -\frac{1}{3} \\ \frac{1}{3} & -\frac{1}{3} \end{pmatrix}$$

da cui $\det \neq 0$, e la convergenza è superlineare;

21.3.2 Esempio: convergenza del metodo di Newton-Raphson con MATLAB

Vediamo di confermare quanto abbiamo detto sopra sfruttando uno strumento di calcolo. Potrebbe venirci in mente di "colorare" su un grafico le regioni che convergono nei punti α e β, e di valutarne la velocità di convergenza.

Iniziamo con un implementazione del metodo di Newton-Raphson che sfrutta il risolutore di MATLAB per essere il più veloce possibile:

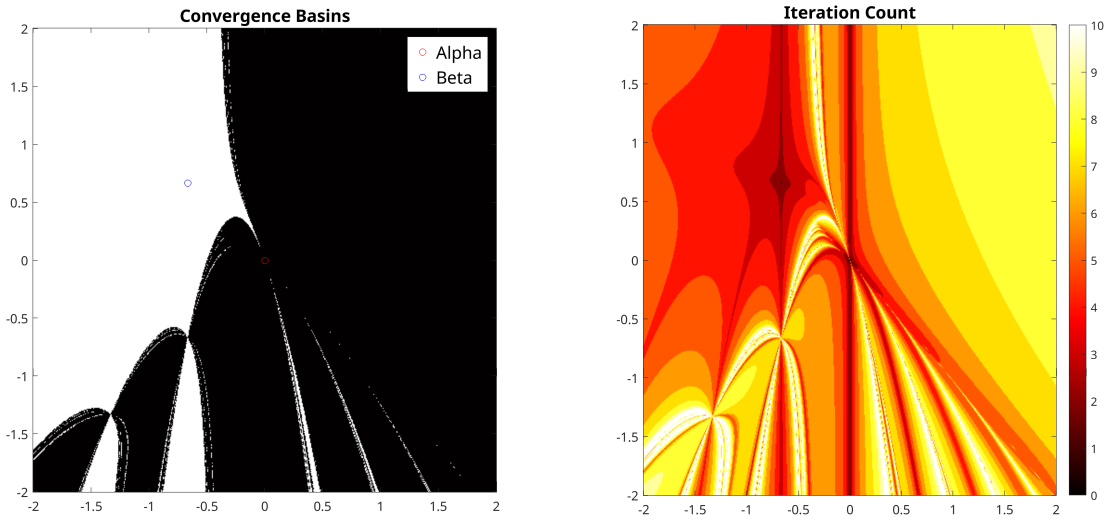
```
function [x0, i] = q_nraph(x0)
    for i = 1:100
        f0 = f_handle(x0);
        j0 = j_handle(x0);

        % solve j0 y = f0
        y0 = j0 \ f0;
        x1 = x0 - y0;

        if norm(x1 - x0) < 0.01
            break;
        end

        x0 = x1;
    end
end
```

e usiamo quindi le funzioni per la colorazione di grafici che MATLAB offre:

```
function raphson_ex()
    x1 = sym('x1');
    x2 = sym('x2');

    function [x0, i] = q_nraph(x0)
        for i = 1:100
            f0 = f_handle(x0);
            j0 = j_handle(x0);

            % solve j0 y = f0
            y0 = j0 \ f0;
            x1 = x0 - y0;

            if norm(x1 - x0) < 0.01
                break;
            end

            x0 = x1;
        end
    end

    function c = classify(p)
        c = 1;

        if norm(alpha - p) < 0.05
            c = 0;
        end
    end

    % problem vars
    f = [(1 / 3) * (x1 - x2) + x1^2; (1 / 3) * (x2 - x1) + x1 * x2];
    j = jacobian(f, [x1, x2]);

    % use handles
    f_handle = matlabFunction(f, 'Vars', {[x1; x2]});
    j_handle = matlabFunction(j, 'Vars', {[x1; x2]});


    alpha = [0; 0];
    beta = [-2/3; 2/3];
```

```
41
42      % grid vars
43      n = 500;
44
45      conv_p = zeros(n);
46      conv_k = zeros(n);
47
48      % sample grid
49      [xg, yg] = meshgrid(linspace(-2, 2, n), linspace(-2, 2, n));
50
51      for x = 1:n
52          for y = 1:n
53              x0 = [xg(x, y); yg(x, y)];
54
55              % print info
56              fprintf("\nCurrent: %.4f, %.4f\n", x0(1), x0(2));
57
58              [p, k] = q_nraph(x0);
59
60              conv_p(x, y) = classify(p);
61              conv_k(x, y) = k;
62
63              fprintf("\n=> converges to %d\n", conv_p(x, y));
64          end
65      end
66
67      figure;
68
69      % plot convergence basins
70      subplot(1,2,1);
71      imagesc(linspace(-2,2,n), linspace(-2,2,n), conv_p);
72      colormap(hot);
73      caxis([0, 1]);
74      title('Convergence Basins');
75      axis xy; axis equal tight;
76      % colorbar;
77
78      % plot solutions
79      hold on;
80      plot(alpha(1), alpha(2), 'ro', 'DisplayName', 'Alpha');
81      plot(beta(1), beta(2), 'bo', 'DisplayName', 'Beta');
82      hold off;
83
84      legend;
85
86      % plot convergence iterations
87      subplot(1,2,2);
88      imagesc(linspace(-2,2,n), linspace(-2,2,n), conv_k);
89      colormap(hot);
90      caxis([0, 10]);
91      title('Iteration Count');
92      axis xy; axis equal tight;
93      colorbar;
94  end
```

L'esecuzione di questo script ci fornisce in pochi secondi il seguente grafico:



Da dove si nota che la regione di convergenza in β ha numeri di iterazione in media molto più piccoli rispetto alla regione di convergenza di α.

Questo corrisponde con quanto avevamo trovato in maniera analitica, cioè che:

α : $\det = 0$, e la convergenza non è superlineare;

β : $\det \neq 0$, e la convergenza è superlineare;

21.3.3 Varianti del metodo di Newton-Raphson

Abbiamo che ad ogni passo del metodo di Newton-Raphson dobbiamo:

1. Valutare $J_f(x^{(n)})$; ovvero m^2 funzioni non lineari;
2. Risolvere $J_f(x^{(n)})y = f(x^{(n)})$, sistema lineare $m \times m$ (complessità $O(m^3)$).

Quando m è particolarmente grande siamo disposti a fare qualcosa di più economico al prezzo di perdere l'ordine di convergenza molto favorevole di Newton-Rhapson.

1. La prima strategia è quella di non aggiornare mai il Jacobiano (**Newton semplificato**). Il Jacobiano viene quindi calcolato solo in $x^{(0)}$ punto iniziale, e poi si usa sempre lo stesso nelle iterazioni successive, cioè l'aggiornamento diventa:

$$x^{(n+1)} = x^{(n)} - J_f(x^{(0)})^{-1} f(x^{(n)})$$

   Per la risoluzione del sistema, si può quindi sfruttare la fattorizzazione LU del Jacobiano appena nominato, e quindi svolgere complessivamente due operazioni $O(m^2)$.

2. La seconda strategia viene detta **metodo di Jacobi nonlineare**. Ad ogni passo si sostituisce $J_f(x^{(n)})$ con la matrice diagonale:

$$\text{diag}\left(\frac{\partial f}{\partial x_1}(x^{(n)}), ..., \frac{\partial f}{\partial x_m}(x^{(n)})\right)$$

   così che:

(a) Bisogna fare m valutazioni di funzioni non lineari;

(b) Risolvere il sistema lineare costa $O(m)$.

L'aggiornamento diventa quindi qualcosa del tipo:

$$x_i^{(n+1)} = x_i^{(n)} - \frac{f_i(x^{(n)})}{\frac{\partial f}{\partial x_i}(x_1^{(n)}, ..., x_m^{(n)})}$$

3. La terza strategia viene detta **metodo di Gauss-Seidel nonlineare**.

   In questo caso l'aggiornamento è sostanzialmente analogo al metodo di Jacobi, con la differenza:

$$x_i^{(n+1)} = x_i^{(n)} - \frac{f_i(x_1^{(n+1)}, ..., x_{i-1}^{(n+1)}, x_i^{(n)}, ...x_m^{(n)})}{\frac{\partial f}{\partial x_i}(x_1^{(n+1)}, ..., x_{i-1}^{(n+1)}, x_i^{(n)}, ...x_m^{(n)})}$$

   dove le entrate $x^{(n+1)}$ vengono calcolate una per volta attraverso le stesse considerazioni fatte per il metodo di Gauss-Seidel lineare.

Altre varianti sono date dai cosiddetti metodi **quasi-Newton**, dove si sostituisce $J_f(x)$ con qualcosa con cui è più semplice (meno costoso) risolvere sistemi lineari e che richiedono meno valutazioni di funzioni non lineari.

22 Lezione del 19-05-25

22.1 Calcolo di autovalori e autovettori

Vediamo quindi il problema di, data una matrice $A \in \mathbb{C}^{n \times n}$, trovare autocoppie (λ, v), con $\lambda \in \mathbb{C}$, $v \in \mathbb{C}^n \setminus \{0\}$ che verificano:

$$Av = \lambda v$$

Vedremo 2 metodi iterativi che approcciano 2 versioni di questo problema:

1. Calcolo di una singola autocoppia (λ, v), ad esempio una coppia in cui λ è l'autovalore di modulo massimo;

2. Calcolo di tutte le autocoppie (λ, v), ovvero una autocoppia per ogni autovalore distinto. Non approfondiremo questo metodo per questioni di complessità.

Osserviamo che gli autovalori sono necessariamente n contate le le loro molteplicità μ, in quanto rappresentano le soluzioni di un polinomio di grado n, e altro non possono fare per via del teorema fondamentale dell'algebra.

Osserviamo poi che gli autovettori sono definiti a meno di moltiplicazione per scalare, cioè sono sempre infiniti gli autovettori associati ad un singolo autovalore λ (tutti quelli nell'autopazio).

22.1.1 Metodo delle potenze

Vediamo il metodo 1) per ottenere una singola autocoppia, e in particolare quella di modulo (autovalore) massimo. Questo può essere utile per:

- Trovare il raggio spettrale di una matrice;
- Trovare la norma 2 di una matrice come:

$$|A|_2 = \sqrt{\rho(A^H A)}$$

Supponiamo che A sia diagonalizzabile. Questa è l'ipotesi che ci assicura che ogni autovalore ha un autovettore associato linearmente indipendente. Prendiamo quindi gli autovalori $\lambda_1, ..., \lambda_n$ posti che soddisfano:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq ... \geq |\lambda_n|$$

dove il primo è in maggiorazione stretta (altrimenti non si distinguerebbe un unico autovalore di modulo massimo).

Dato che A è diagonalizzabile, abbiamo dai fondamenti di algebra lineare che esiste una base di $\mathbb{C}^n$ fatta da autovettori di A, ovvero $\exists v^{(1)}, ..., v^{(n)} \in \mathbb{C}^n$ tali che:

$$Av^{(j)} = \lambda_j v^{(j)}, \quad j = 1, ..., n$$

e per ogni $z \in \mathbb{C}^n$ vale che $\exists! c_1, .., c_n \in \mathbb{C}$ tali che:

$$z = \sum_{j=1}^{n} c_j v^{(j)} = c_1 v^{(1)} + ...c_n v^{(n)}$$

L'idea fondamentale del metodo è che, preso $z \in \mathbb{C}^n$ a caso, abbiamo:

$$Az = A(c_1 v^{(1)} + ... + c_n v^{(n)}) = c_1 A v^{(1)} + ... + c_n A v^{(n)} = c_1 \lambda_1 v^{(1)} + ... + c_n \lambda_n v^{(n)}$$

Continuando ad iterare si ha:

$$A^2 z = A(Az) = A(c_1 \lambda_1 v^{(1)} + ... + c_n \lambda_n v^{(n)})$$

$$= c_1 \lambda_1 A v^{(1)} + ... + c_n \lambda_b A v^{(n)} = c_1 \lambda_1^2 v^{(1)} + ... + c_n \lambda_n^2 v^{(n)}$$

e quindi continuando ad iterare si ha che:

$$A^k z = c_1 \lambda_1^k v^{(1)} + ... + c_n \lambda_n^k v^{(n)}$$

Il vantaggio è che l'autovalore di modulo massimo compare chiaramente con contribuzioni sempre più grandi, in quanto:

$$= \lambda_1^k \left(c_1 v^{(1)} + c_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k v^{(2)} + ... + c_n \left(\frac{\lambda_n}{\lambda_1} \right)^k v^{(n)} \right) \sim \lambda_1^k, \quad k \to +\infty$$

Più formalmente, abbiamo che vale il teorema:

Teorema 22.1: Convergenza del metodo delle potenze

Sia considerata la successione:

$$\begin{cases} z^{(0)} = z \\ z^{(k+1)} = Az^{(k)} = A^{k+1}z^{(0)} = A^{k+1}z \end{cases}$$

con $A \in \mathbb{C}^{n\times n}$ hermitiana ($A = A^H$), con autovalori:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq ... \geq |\lambda_n| > 0$$

e sia $h \in \{1, ..., n\}$ tale per cui $v_h^{(1)} \neq 0$. Se $z^{(0)} \in \mathbb{C}^n$ è tale che:

$$\left[V^{(1)}\right]^H z^{(0)} \neq 0$$

allora la successione è tale che:

$$\lim_{k\to+\infty} \frac{z^{(k)}}{z_h^{(k)}} = \tilde{v}^{(1)}$$

multiplo scalare di $v^{(1)}$ (in particolare $\tilde{v}^{(1)} = \frac{v^{(1)}}{v_h^{(1)}}$). Inoltre, per l'autovalore vale:

$$\lim_{k\to+\infty} \frac{\left[z^{(k)}\right]^H Az^{(k)}}{\left[z^{(k)}\right]^H z^{(k)}} = \lambda_1$$

Osserviamo che, dato $x \in \mathbb{C}^n$ e considerato il *quoziente di Rayleigh*:

$$R(x) = \frac{x^H Ax}{x^H x}$$

Se x verifica $Ax = \lambda x$ (cioè è autovettore) allora:

$$R(x) = \frac{x^H Ax}{x^H x} = \frac{x^H \lambda x}{x^H x} = \lambda \frac{x^H x}{x^H x} = \lambda$$

Notiamo la motivazione dell'assunto di matrice hermitiana. Di base, per il teorema spettrale si ha che:

$$A = A^H \implies A \text{ diagonalizzabile}$$

e in particolare si può scegliere una base ortonormale di autovettori, cioè trovare $v^{(1)}, ..., v^{(n)} \in \mathbb{C}^n$ tali che:

$$Av^{(j)} = \lambda_j v^{(j)}, \quad \left[v^{(i)}\right]^H v^{(j)} = < v^{(i)}, v^{(j)} > = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

cioè equivalentemente:

$$A = VDV^H$$

con D diagonalizzabile e V normale data dagli autovettori in colonna.

Dimostriamo quindi il teorema. Dato che $A = A^H$ hermitiana si ha che:

$$z^{(0)} = \sum_{j=1}^{n} c_j v^{(j)}$$

ed inoltre abbiamo:

$$0 \neq \left[v^{(1)}\right]^H z^{(0)} = \sum_{j=1}^{n} c_j \left[v^{(1)}\right]^H v^{(j)} = c_1 \left[v^{(1)}\right]^H v^{(1)} = c_1$$

e quindi $c_1 \neq 0$.

Dimostriamo allora le due tesi, su autovettore (1) a autovalore massimo (2):

1. Come fatto in precedenza prendiamo:

$$z^{(k)} = A^k z^{(0)} = \lambda_1^k \left(c_1 v^{(1)} + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k v^{(2)} + ... + c_n \left(\frac{\lambda_n}{\lambda_1}\right)^k v^{(n)} \right)$$

$$= \lambda_1^k \left(c_1 v^{(1)} + \sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_i}\right)^k v^{(j)} c_j \right)$$

   dove vorremo dimostrare che il termine a destra scompare per $k \to +\infty$.

   Prendiamo quindi il limite:

$$\lim_{k\to+\infty} \frac{\lambda_1^k \left(c_1 v^{(1)} + \sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_i}\right)^k v^{(j)} c_j \right)}{\lambda_1^k \left(c_1 v_h^{(1)} + \sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_i}\right)^k v_h^{(j)} c_j \right)}$$

$$\lim_{k\to+\infty} \frac{c_1 v^{(1)} + \sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_i}\right)^k v^{(j)} c_j)}{c_1 v_h^{(1)} + \sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_i}\right)^k v_h^{(j)} c_j} = \lim_{n\to+\infty} \frac{c_1 v^{(1)}}{c_1 v_h^{(1)}} = \frac{v^{(1)}}{v_h^1}$$

   che è la prima tesi.

2. Prendiamo quindi:

$$\lim_{k\to+\infty} \frac{\left[z^{(k)}\right]^H A z^{(k)}}{\left[z^{(k)}\right]^H z^{(k)}} = \lim_{k\to+\infty} \frac{\left[\frac{z^{(k)}}{z_h^{(k)}}\right]^H A \frac{z^{(k)}}{z_h^{(k)}}}{\left[\frac{z^{(k)}}{z_h^{(k)}}\right]^H A \frac{z^{(k)}}{z_h^{(k)}}} = \frac{\left[\frac{v^{(1)}}{v_h^{(1)}}\right]^H A \frac{v^{(1)}}{v_h^{(1)}}}{\left[\frac{v^{(1)}}{v_h^{(1)}}\right]^H A \frac{v^{(1)}}{v_h^{(1)}}}$$

   che moltiplicando sopra e sotto per $\left(v_h^{(1)}\right)^2$ dà:

$$= \frac{[v^{(1)}]^H A v^{(1)}}{[v^{(1)}]^H v^{(1)}} = \lambda_1$$

   da cui la seconda tesi. □

Osserviamo che h è stato usato perché dividiamo per $z_h^{(k)}$ che per k abbastanza grandi è $\neq 0$, in quanto vicino a $v_h^{(1)}$.

Osserviamo poi che nell'implementazione pratica il metodo delle potenze $z^{(k)}$ viene diviso per la sua norma, così da ottenere solo vettori di norma 1 ed evitare problemi di overflow o underflow.

22.1.2 Criterio di stop

Vediamo la condizione di stop del metodo delle potenze. In genere si usa:

$$|R(z^{(k)} - R(z^{(k-1)}))| < \epsilon$$

con ad esempio $\epsilon = 10^{-8}$.

Testiamo gli autovalori con $R()$ e non gli autovettori con:

$$|z^{(k)} - z^{(k+1)}|_2 < \epsilon$$

in quanto non sappiamo verso quale multiplo scalare degli autovettori ci stiamo dirigendo, per cui la differenza ad ogni passaggio potrebbe essere molto grande.

22.1.3 Implementazione MATLAB del metodo delle potenze

Vediamo quindi, da quello che abbiamo detto, come si può implementare una funzione per il calcolo dell'autocoppia massima in MATLAB:

```
function [z0, e] = max_eigen(A, z0, k, tol)
    if nargin < 4
        tol = 0.01;
    end

    if nargin < 3
        k = 100;
    end

    % eigenvector thru power metod
    for i = 1:k
        z1 = A * z0;
        z1 = z1 / norm(z1);

        fprintf("\nStep %d:\n", i);
        disp(z1);

        if norm(z1 - z0) < tol
            break;
        end

        z0 = z1;
    end

    % eigenvalue
    e = z0' * A * z0;
end
```

Notiamo che per ricavare l'autovalore a termine iterazione abbiamo usato il famigerato **quoziente di Rayleigh**:

$$\lambda = z_0^H A z_0$$

che è sostanzialmente una versione attenta al segno del prendere la norma di Az_0.

Il costo è quello di una norma ($O(n)$) e di un prodotto matrice vettore ($O(n^2)$)ad ogni iterazione, cioè $O(k \cdot n^2)$.

22.1.4 Commenti sull'ipotesi di convergenza

Possiamo fare un commento sulle ipotesi del teorema 22.1.

- Non è strettamente necessario che $A = A^H$ hermitiana, in quanto si può dimostrare che il metodo funziona anche con A solo diagonalizzabile;
- L'assunzione $\left[v^{(1)}\right]^H z^{(0)} \neq 0$ è virtualmente vera (o meglio vera con probabilità 1) per qualsiasi $z^{(0)} \in \mathbb{C}^n$ scelto a caso. Inoltre, nel caso di calcolo a macchina si ha che gli errori di arrotondamento stessi aiutano ad uscire dall'ortogonalità, anche se si parte da tale condizione.
- L'assunzione $|\lambda_1| > |\lambda_2|$ non è invece rinunciabile, in quanto si possono costruire esempi di successioni non convergenti nel caso di più autovalori dominanti.

  Prendiamo l'esempio:

  $$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

  con 2 autovalori di modulo 1. Prendiamo:

  $$z^{(0)} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

  Si trova che l'aggiornamento scambia semplicemente le componenti:

  $$Az^{(0)} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

  e cosi via all'infinito.

  Gli autovettori, calcolati a mano, sono semplicemente:

  $$v^{(1)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad v^{(2)} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Osserviamo quindi che per eseguire il metodo delle potenze, serve soltanto essere in grado di valutare il prodotto matrice vettore Az. Questo può essere utile nel caso sopra citato, quando si ha una matrice definita come risultato di operazioni aritmetiche, magari costose.

Ad esempio, se si vuole stimare:

$$|A|_2 = \sqrt{\rho(A^H A)}$$

si può pensare di applicare il metodo delle potenze su $A^H A$, senza nemmeno dover calcolare $A^H A$ ma applicando direttamente i prodotti matrici vettore $A^H Az$.

22.2 Metodi per più autovalori

22.2.1 Autovalori minimi

Supponiamo di essere interessati all'autocoppia $(\lambda_n, v^{(n)})$, associata all'autovalore λ_n di modulo minimo, e supponiamo:

$$|\lambda_1| \geq ... \geq |\lambda_{n-2}| \geq |\lambda_{n-1}| > |\lambda_n|$$

L'idea è di applicare il metodo delle potenze ad A^{-1}, infatti A^{-1} ha autovalori:

$$\left|\frac{1}{\lambda_n}\right| > \left|\frac{1}{\lambda_{n-1}}\right| \geq \left|\frac{1}{\lambda_{n-2}}\right| \geq ... \geq \left|\frac{1}{\lambda_1}\right|$$

ed inoltre ha come autovettori $v^{(1)}, ..., v^{(n)}$. Quidi le successioni generate dal metodo delle potenze su A^{-1} convergono all'autocoppia:

$$\left(\frac{1}{\lambda_n}, v^{(n)}\right)$$

Chiamiamo questo metodo anche **metodo delle potenze inverse**.

Dove chiaramente non vale la pena di calcolare l'inversa ma piuttosto conviene risolvere il sistema:

$$Ay^{(k)} = z^{(k-1)}$$

ad esempio sfruttando l'eliminazione di Gauss con pivoting. Per rendere tutto più efficiente, conviene anzi precalcolare una fattorizzazione LU di A prima di iniziare il metodo.

In questo caso avremo:

$$A = \Pi LU$$

cioè si fa una fattorizzazione LU con pivoting dato dalla matrice di permutazione Π, così che ad ogni passaggio il prodotto $A^{-1}z$ diventa equivalente a risolvere un sistema lineare triangolare. In questo caso il costo diventa $O(n^3)$ per la fattorizzazione più $O(k \cdot n^2)$ per la risoluzione dei sistemi triangolari.

22.2.2 Implementazione MATLAB del metodo delle potenze inverse

Vediamo quindi come si può implementare una funzione per il calcolo dell'autocoppia minima in MATLAB:

```
function [z0, e] = min_eigen(A, z0, k, tol)
    if nargin < 4
        tol = 0.01;
    end

    if nargin < 3
        k = 100;
    end

    % eigenvector thru power metod
    for i = 1:k
        z1 = A \ z0;
        z1 = z1 / norm(z1);

        if norm(z1 - z0) < tol
            break;
        end

        z0 = z1;
    end

    % eigenvalue
    e = z0' * A * z0;
end
```

22.2.3 Autovalori intermedi

Vediamo quindi come modificare il metodo delle potenze per trovare un autovalore intermedio, cioè preso $\sigma \in \mathbb{C}$ scalare arbitrario calcolare l'autocoppia (λ, v) con λ più vicino a σ.

In questo caso si applica il metodo delle potenze alla matrice:

$$A \sim (A - \sigma \cdot I)^{-1}$$

che ha gli stessi autovettori di A e come autovalori ha quantità della forma:

$$\lambda'_j = \frac{1}{\lambda_j - \sigma}$$

Questo si dimostra prendendo assumendo (λ, v) autocoppia, quindi $Av = \lambda v$, e calcolando $(A - \sigma I)v$:

$$(A - \sigma I)x = Ax - \sigma Ix = Ax - \sigma x = \lambda x - \sigma x = (\lambda - \sigma)x$$

guardando agli estremi, e invertendo, si ottiene:

$$(A - \sigma I)x = (\lambda - \sigma)x \implies (A - \sigma I)^{-1}x = \frac{1}{\lambda - \sigma}x$$

che rende $\left(\frac{1}{\lambda - \sigma}, v\right)$ autocoppia, da cui la tesi. □

In questo caso l'autovalore di modulo più grande diventa quello più vicino a σ, per cui basta applicare il metodo delle potenze.

22.2.4 Implementazione MATLAB del metodo delle potenze per autovalori intermedi

Vediamo quindi l'implementazione MATLAB dell'ultimo algoritmo descritto:

```
function [v, e] = close_eigen(A, s, z0, k, tol)
    if nargin < 5
        tol = 0.01;
    end

    if nargin < 4
        k = 100;
    end

    n = height(A);
    B = inv(A - eye(n) * s);

    [v, m] = max_eigen(B, z0, k, tol);
    e = 1 / m + s;
end
```

23 Lezione del 23-05-25

Continuiamo a vedere i metodi per più autovalori.

23.0.1 Metodo QR per gli autovalori

Introduciamo quindi il **metodo QR per gli autovalori**, che non va confuso col metodo QR *per i minimi quadrati* (visto in 12.2), anche se chiaramente si basa sulla fattorizzazione QR.

Questo è quindi un metodo per trovare tutti gli autovalori di $A \in \mathbb{C}^{n \times n}$ e un insieme di autovettori associati a ciascun autovalore. Abbiamo quindi come input A, de come uscita V e D, con D diagonale di autovalori:

$$D = \begin{pmatrix} \lambda_1 & ... & 0 \\ & \ddots & \\ 0 & ... & \lambda_n \end{pmatrix}$$

e V tale che:

$$A = VDV^{-1}$$

L'algoritmo che risolve questo problema viene ideato negli anni '70, ed è considerato fra i più importanti degli ultimi anni. Vedremo una versione *"vanilla"*, quindi semplificata, dell'algoritmo, in quanto le implementazioni moderne (MATLAB o librerie come *LAPACK*)non sarebbero effettivamente utili se non per le ottimizzazioni che sono state fatte nel tempo.

L'idea di fondo è creare una successione di matrici:

$$\{A_k\}_{k \in \mathbb{N}}, \quad A_0 = A$$

tutte simili ad A, cioè tali che:

$$\forall k, \ \exists Q \ : \ Q^{-1} A_k Q = A$$

con Q invertibile ed unitaria, cioè:

$$Q^{-1} = Q^H$$

In questo modo ogni A_k ha gli stessi autovalori di A, mentre gli autovettori di $A_K = Q^H A Q$ saranno $Q^h v$, dove v è autovettore di A. Inoltre, chiamando:

$$A_\infty = \lim_{k \to +\infty} A_k$$

si ha che A_∞ è unna matrice di cui si possono calcolare gli autovettori e autovalori in modo efficiente. Nello specifico, A_∞ è una matrice triangolare.

23.0.2 Calcolo di autovalori e autovettori per matrici triangolari

Motiviamo quindi quest'ultima affermazione studiando il caso particolare delle matrici triangolari. Poniamo T triangolare superiore:

$$T = \begin{pmatrix} t_{11} & ... & t_{1n} \\ & \ddots & \vdots \\ 0 & & t_{nn} \end{pmatrix}$$

gli autovalori sono facili, in quanto risulteranno:

$$\{t_{11}, t_{22}, ..., t_{nn}\}$$

gli autovettori sono più difficili, ma vediamo che il primo è banale:

$$Tv_1 = \begin{pmatrix} t_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = t_{11} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \implies v_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Cerchiamo quini il generico autovettore v_j assumedo $t_{ii} \neq t_{jj}$, $\forall i \neq j$ (cioè l'ipotesi degli autovalori distinti) e cerchiamo una soluzione diversa da 0 del sistema:

$$(T - t_{jj}I)v_j = 0$$

che avrà l'aspetto:

$$\begin{pmatrix} t_{11} - t_{jj} & t_{12} & & \dots & & & & t_{1n} \\ & \ddots & & \dots & & & & \vdots \\ & & t_{j-1,j-1} - t_{jj} & \dots & & & & \vdots \\ & & & 0 & \dots & & & \vdots \\ & & & & t_{j+1,j+1} - t_{jj} & \dots & & \vdots \\ & & & & & \ddots & & \vdots \\ & & & & & & t_{nn} - t_{jj} & \end{pmatrix} v_j = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{pmatrix}$$

Riscrivendo a blocchi abbiamo:

$$\begin{pmatrix} (T_1) & z & (*) \\ & 0 & (*) \\ & & (T_2) \end{pmatrix} \underbrace{\begin{pmatrix} (v_j^{(1)}) \\ \gamma \\ v_j^{(2)} \end{pmatrix}}_{v_j} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \Leftrightarrow \begin{cases} T_1 v_j^{(1)} + \gamma z + (*) v_j^{(2)} = 0 \\ \dots \\ T_2 v_j^{(2)} = 0 \end{cases}$$

Notiamo che:

$$T_2 v_j^{(2)} = 0$$

è un sistema $(n - j - 1) \times (n - j - 1)$ ha matrice invertibile, per cui:

$$v_j^{(2)} = 0$$

e la parte indicata con asterischi del sistema non ci interessa, per cui possiamo dire:

$$T_1 v_j^{(1)} + \gamma z = 0$$

che è un sistema $(j - 1) \times j$, cioè sottodeterminato. Scegliamo ad esempio $\gamma = 1$ per fissare una soluzione, e otteniamo:

$$T_1 v_j^{(1)} = -z \implies v_j^{(1)} = -T_1^{-1} z$$

Per cui abbiamo ricostruito l'intero vettore v_j come:

$$v_j = \begin{pmatrix} -T_1^{-1} z \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Iteriamo quindi per gli $j = 1, ..., n$ indici e troviamo tutti gli autovettori.

Dal punto di vista del costo, per trovare v_j si deve risolvere il sistema $T_1 x = -z$ di dimensione $(j-1) \times (j-1)$ e triangolare, che costa $O(j^2)$. Col fatto che iteriamo per j volte abbiamo complessivamente:

$$O\left(\sum_{j=1}^{n} j^2\right) = O(n^3)$$

23.0.3 Metodo QR per la successione

Vediamo quindi come formare la successione vera e propria. L'idea è che prendiamo $A_0 = A$, e per $k = 1, 2, ...$ prendiamo:

$$A_{k-1} = Q_{k-1} R_{k-1}$$

$$A_k = R_{k-1} Q_{k-1}$$

In effetti le matrici A_k sono simili:

$$A_{k+1} = R_k Q_k = Q_k^H Q_k R_k Q_k = Q_k^H A_k Q_k$$

espandendo R_k, e quindi A_{k+1} è simile ad A_k. Iterando il ragionamento si ottiene che A_{k+1} è simile ad A_0 e quindi ad A.

In particolare:

$$A_{k+1} = Q_k^H A_k Q_k = Q_k^H Q_{k-1}^H A_{k-1} Q_{k-1} Q_k = ... = \underbrace{Q_k^H Q_{k-1}^H ... Q_1^H}_{\tilde{Q}_k^H} A \underbrace{Q_1 ... Q_{k-1} Q_k}_{\tilde{Q}_k}$$

e quindi:

$$A_{k+1} = \tilde{Q}_k^H A \tilde{Q}_k$$

da cui immediatamente la similarità con $\tilde{Q}_k$ matrice di traduzinoe.

A giustificare questo abbiamo il teorema:

Teorema 23.1: Validità del metodo QR agli autovalori

Presa $A \in \mathbb{R}^{n \times n}$ e supposto:

$$|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$$

allora:

$$\lim_{n \to +\infty} A_k = T = \begin{pmatrix} t_{11} & ... & t_{1n} \\ & \ddots & \vdots \\ 0 & & t_{nn} \end{pmatrix}$$

dove i blocchi T_{jj} sono 1×1 per autovalori reali e 2×2 per autovalori complessi coniugati.

23.0.4 Implementazione MATLAB del metodo QR per gli autovalori

Vediamo quindi l'implementazione MATLAB della versione semplificata del metodo che abbiamo visto. Innanzitutto ci dotiamo di una funzione per il calcolo delle autocoppie di matrici triangolari superiori T come abbiamo visto in 23.0.2:

```
function [V, D] = eigu(T)
    n = height(T);

    D = diag(diag(T));

    V = zeros(n);

    for j = 1:n
        Tu = T(1:j - 1, 1:j - 1) - T(j, j) * eye(j - 1);
        z = T(1:j - 1, j);
        Viu = - (Tu \ z);

        V(1:j - 1, j) = Viu;
        V(j, j) = 1;
    end
end
```

che includeremo come sottofunzione. A questo punto basterà implementare il passo iterativo:

```
function [V, D] = qr_eigen(A, k)
    function [V, D] = eigu(T)
        n = height(T);

        D = diag(diag(T));

        V = zeros(n);

        for j = 1:n
            Tu = T(1:j - 1, 1:j - 1) - T(j, j) * eye(j - 1);
            z = T(1:j - 1, j);
            Viu = - (Tu \ z);

            V(1:j - 1, j) = Viu;
            V(j, j) = 1;
        end
    end

    n = height(A);
    Qt = eye(n);

    for i = 1:k
        [Q, R] = qr_decomp(A);
        A = R * Q;
        Qt = Qt * Q;
    end

    % tronca al triangolo superiore
    T = triu(A);

    [V, D] = eigu(T);
    V = Qt * V;
end
```

23.0.5 Precisazioni sul metodo QR per gli autovalori

Vediamo che i termini sotto a cui abbiamo definito l'algoritmo sono estremamente restrittivi (autovalori distinti, ecc...). Chiaramente nelle implementazioni reali ci sono, oltre che alle ottimizzazioni, vari accorgimenti che rendono l'algoritmo robusto alle varie casistiche (autovalori ripetuti, ecc...), mantenendo il costo cubico $O(n^3)$.

Alcuni accorgimenti sono:

1. Si deve tenere il numero di iterazioni N sotto controllo, e in particolare viene garantito che:
$$N = O(n)$$

2. Calcolare la fattorizzazione QR costa $O(n^3)$, se la calcoliamo ad ogni passo otteniamo un costo $O(n^4)$ che non è ottimale. Nella versione finale si fa un preprocessing sulla matrice A che la riduce alla cosiddetta *forma di Hessenberg* che rende il costo del calcolo della fattorizzazione quadratico $O(n^2)$ anziché cubico, così che l'algoritmo complessivo costi $O(n^3)$;

3. Nella soluzione che abbiamo visto si fanno assunzioni sugli autovalori (autovettori distinti, moduli distinti, ecc...). Queste possono essere aggirate, e il metodo QR completo non ha restrizioni sulla matrice A.