Note — Calcolo Numerico

federico.poloni@unipi.it

Queste note sono pensate per fare da 'scaletta' della lezione per il corso per me mentre tengo le lezioni, e non necessariamente per essere usate come dispense o documento a sé stante.

Scelta degli argomenti, dimostrazioni ed esempi sono in parte presi dalle dispense di Luca Gemignani, Paolo Ghelardoni, Cecilia Magherini.

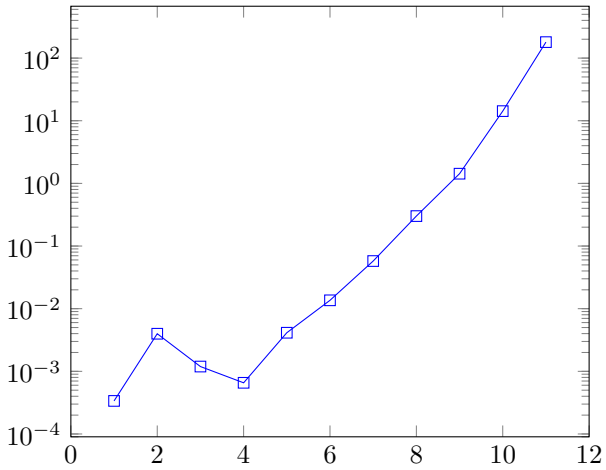
Capitolo 1

Motivazione

Vedremo algoritmi per risolvere al computer alcuni problemi che avete visto nei corsi di analisi e algebra lineare.

Problemi senza soluzione esatta Alcuni problemi non hanno una soluzione esatta data da una formula, per esempio $\int_0^1 e^{x^2} \mathrm{d}x$, ma si riescono a calcolare approssimazioni di queste soluzioni, per esempio approssimare un integrale tramite somme di Riemann. Vista la quantità di conti, meglio farlo fare a un computer.

Algoritmi lenti Non sempre gli algoritmi che si vedono in un corso 'teorico' funzionano bene su un computer. Esempio: calcolo del determinante tramite la formula di Laplace. Tempo di calcolo sul mio computer:



Più di 100 secondi per una matrice $11 \times 11 \implies$ poco utile in pratica.

Algoritmi inaccurati Un altro fattore è *come* il computer implementa certe operazioni. Numeri come $1/3$ o $\sqrt{2}$ hanno infinite cifre; non possiamo fare le operazioni "in colonna". Ci sono diversi modi di gestire i calcoli, ma per la maggior parte delle applicazioni l'esperienza ha insegnato che la strategia migliore è tenere (essenzialmente) solo un certo numero di cifre significative, per esempio 16. Questo significa che certe operazioni sono approssimate; per esempio

$\frac{1}{3} + \frac{1}{3} + \frac{1}{3} \approx 0.33333 + 0.33333 + 0.33333 = 0.99999 \neq 1$. Alcuni algoritmi vanno implementati con attenzione per evitare che queste approssimazioni che sembrano innocue abbiano un grosso impatto sul risultato. Per esempio: l'eliminazione di Gauss, o anche formule semplici come quella per risolvere un'equazione di secondo grado.

Importanza Risolvere problemi computazionali al computer è diventato sempre più importante negli ultimi anni. Pensate per esempio a chi deve progettare un aereo. Quanto spessa deve essere la parete di metallo che usate nelle ali? Se è troppo spessa, consumate più carburante del necessario. Se è troppo sottile, l'ala potrebbe rompersi. Una volta c'era un solo modo di scoprirlo: costruisci un prototipo di aereo, e vedi se cade o no. Oggi, la maggior parte di questi esperimenti possono essere rimpiazzati da simulazioni al computer. Va trovato il modo più efficiente di farle (un aereo è un oggetto molto complesso), ed è fondamentale capire se i risultati sono affidabili e se questi errori diventano così grandi da compromettere il risultato. Questo si applicherà, per voi, anche a simulare reazioni e processi chimici.

Programmazione L'altro scopo importante di questo corso è come *programmare* un computer, cioè scrivere delle istruzioni per far eseguire sequenze specifiche di operazioni: non vogliamo calcolare noi le somme di Riemann di $\int_0^1 e^{x^2} \mathrm{d}x$, ma fare in modo che il computer esegua questo calcolo per noi. E non vogliamo affidarci a un'app o un sito già fatto per ogni singolo problema, ma imparare come usare il computer per risolvere qualunque nuovo problema ci capiti davanti.

Capitolo 2

Equazioni non lineari (zeri di funzione)

Problema: data una funzione $f : [a, b] \to \mathbb{R}$, vogliamo calcolare uno *zero della funzione*, cioè un punto α tale che $f(\alpha) = 0$. Un esempio semplice è il calcolo di radici n-esime: calcolare $\sqrt[n]{a}$ è equivalente a trovare uno zero della funzione $f(x) = x^n - a$.

Ci potrebbero essere più soluzioni in un dato intervallo (a priori anche infinite!). Un modo conveniente di dimostrare che esiste *almeno* una soluzione in un dato intervallo $[a, b]$ è mostrare che è un *intervallo di separazione*, cioè $f(a)$ e $f(b)$ hanno segni opposti: $f(a) > 0, f(b) < 0$ oppure $f(a) > 0, f(b) < 0$. Un modo semplice di scrivere questa condizione è $f(a)f(b) < 0$. Difatti, ricordiamo questo teorema che avete visto ad analisi.

Teorema 2.1 (Teorema degli zeri)**.** *Sia $f \in \mathcal{C}^0([a, b])$. Se $f(a)$ e $f(b)$ hanno segno opposto (uno positivo e uno negativo), allora esiste almeno un reale $\alpha \in (a, b)$ tale che $f(\alpha) = 0$.*

Detto in altro modo, se f è una funzione continua e $[a, b]$ è un intervallo di separazione, allora esso contiene uno zero della funzione.

Questo non ci assicura che lo zero sia unico. Se riusciamo a mostrare che f è (strettamente) crescente o decrescente in $[a, b]$ (per esempio, per una $f \in \mathcal{C}^1$ mostrando che $f'(x) > 0$ per ogni $x \in (a, b)$ o $f'(x) < 0$ per ogni $x \in (a, b)$).

Esempio: per ogni $n > 2, a > 0$, consideriamo la funzione $f(x) = x^n - a$. Si ha $f(0) = -a < 0$, mentre per x sufficientemente grande si ha $f(x) > 0$ (difatti $x \to \infty$). Per esempio, $f(a) = a^n - a > 0$. Quindi $[0, a]$ è un intervallo di separazione, ed esiste una soluzione nell'intervallo. La soluzione è unica? La funzione f ha derivata $f'(x) = nx^{n-1}$; quindi $f'(x) > 0$ per ogni $x > 0$, e $f'(0) = 0$. Questo permette di concludere che la soluzione è una sola. Come dimostrarlo formalmente? Un modo è tramite il teorema di Rolle, che avete sicuramente visto ad analisi. Lo enunciamo qui, con delle ipotesi forse un pochino più restrittive ma che sono sufficienti per noi.

Teorema 2.2 (Teorema di Rolle)**.** *Sia $f : [a, b] \to \mathbb{R}$ una funzione di classe $\mathcal{C}^1$. Se $f(a) = f(b)$, allora esiste almeno un punto $c \in (a, b)$ in cui $f'(c) = 0$.*

Se per assurdo esistessero due punti $a, b \in [0, \infty)$ tali che $f(a) = f(b) = 0$ (supponiamo per semplicità di avere scelto i nomi in modo che $a < b$), allora esisterebbe un punto $c \in (a, b) \subset (0, \infty)$ tale che $f'(c) = 0$, ma questo è impossibile perché abbiamo detto che $f'(x) > 0$ per $x > 0$.

2.1 Metodo di bisezione

È un metodo che funziona con poche ipotesi sulla f: serve solo che $f \in \mathcal{C}^0([a, b])$, e che $[a, b]$ sia un intervallo di separazione.

Descrizione del metodo L'idea del metodo è la seguente: voglio costruire una sequenza di intervalli di separazione $[a_k, b_k]$ sempre più piccoli. Parto ponendo $a_1 = a, b_1 = b$. A questo punto, calcolo $c_1 = \frac{a_1+b_1}{2}$, il punto medio tra a_1 e b_1. Se $f(c_1) = 0$, allora ho trovato uno zero e posso terminare. Altrimenti, uno dei due intervalli $[a_1, c_1]$ e $[c_1, b_1]$ è un intervallo di separazione, come si può vedere facilmente considerando i casi possibili per i segni. Quindi pongo $[a_2, b_2] = [a_1, c_1]$ oppure $[c_1, b_1]$. A questo punto posso ripetere la procedura a partire da $[a_2, b_2]$ e continuare.

$a_1 = a, b_1 = b$;
for $k = 1, 2, \ldots$ **do**
  $c_k = \frac{a_k+b_k}{2}$;
  if $f(c_k) = 0$ **then**
    $\alpha = c_k$;
    break;
  else if $f(a_k)f(c_k) < 0$ **then**
    $a_{k+1} = a_k$, $b_{k+1} = c_k$;
  else
    $a_{k+1} = c_k$, $b_{k+1} = b_k$;
end

Esempio Prendiamo $f(x) = x^2 - 2$. L'intervallo $[a, b] = [0, 2]$ è un intervallo di separazione, quindi contiene (almeno) uno zero α di f. Sappiamo già in realtà che ne contiene esattamente uno, $\alpha = \sqrt{2}$.

k	a_k	c_k	b_k	$f(a_k)$	$f(c_k)$	$f(b_k)$
1	0	1	2	-2	-1	2
2	1	1.5	2	-1	0.25	2
3	1	1.25	1.5	-1	-0.4375	0.25
$\vdots$						

Terminazione Si può dimostrare formalmente (noi non lo faremo) che le successioni a_i, b_i, c_i convergono a un valore α che è uno zero di f. A noi però interessa poco cosa succede al limite, perché possiamo fare solo un numero finito di passi.

Supponiamo di voler determinare il valore α a meno di una tolleranza ε; in questo paragrafo vediamo come è possibile scegliere quanti passi fare in base a questo requisito.

È semplice vedere che ad ogni passo la lunghezza dell'intervallo si dimezza:

$$b_{k+1} - a_{k+1} = \frac{b_k - a_k}{2} = \frac{b_{k-1} - a_{k-1}}{4} = \cdots = \frac{b_1 - a_1}{2^k}$$

Quindi se ci fermiamo dopo n iterazioni del metodo abbiamo un intervallo di ampiezza $\frac{b_1-a_1}{2^{n-1}}$ che contiene *uno* zero α della funzione. Se vogliamo restituire *un* valore che sia la migliore approssimazione possibile (con le informazioni a disposizione) di α, la cosa migliore da fare è restituire il punto medio $c_n = \frac{a_n+b_n}{2}$. Difatti se α è un punto di cui sappiamo solo che sta nell'intervallo (a_n, b_n), di ampiezza $\ell = \frac{b_1-a_1}{2^{n-1}}$, la distanza tra α e il punto medio dell'intervallo è al massimo $\ell/2$. Se restituissi a_n o b_n invece nel caso peggiore l'errore è ℓ.

IMMAGINE

Quindi, dato il valore c_n restituito dopo n passi del metodo di bisezione, sappiamo che esiste uno zero α di f tale che

$$|c_n - \alpha| \leq \frac{b_1 - a_1}{2^n}.$$

Dato un valore $\varepsilon > 0$ possiamo calcolare quante iterazioni sono necessarie perché il termine di destra di questa uguaglianza sia minore di ε. Questo succede per il primo intero n tale che

$$\frac{b_1 - a_1}{2^n} \leq \varepsilon \iff 2^n \geq \frac{b_1 - a_1}{\varepsilon} \iff n \geq \log_2 \frac{b_1 - a_1}{\varepsilon}.$$

Un'altra idea naturale è fermarsi quando $f(c_k)$ è "sufficientemente piccolo". La condizione di arresto $f(c_k) = 0$ potrebbe non essere mai verificata esattamente, per colpa dei calcoli inaccurati in aritmetica di macchina, ma quando $f(c_k)$ è molto piccolo potremmo decidere di essere soddisfatti e fermare il calcolo. È però vero che se abbiamo trovato un valore c per cui $f(c)$ è piccolo, allora siamo vicini a uno zero α? È facile costruire funzioni per cui questo non succede: basta prendere una funzione positiva che diminuisce fino ad arrivare molto vicino a zero e poi risale. Però possiamo introdurre un *criterio euristico*, cioè un criterio non rigoroso (e di cui non è garantito il successo).

Supponiamo di avere calcolato la funzione in un punto c_k, ottenendo $f(c_k)$. Una delle idee fondamentali dell'analisi è che se "zoomiamo il grafico" e ne guardiamo una sezione sufficientemente piccola, una funzione f che sia *differenziabile* si avvicina molto alla sua retta tangente. Pertanto, se f è differenziabile in c_k,

$$f(x) \approx y = f(c_k) + f'(c_k)(x - c_k),$$

dove a destra dell'uguale abbiamo l'equazione della retta tangente in c_k. Un modo di trasformare questa relazione in un'uguaglianza esatta è tramite lo sviluppo di Taylor: se esiste la derivata seconda f'' in un intorno di c_k, allora per $x - c_k$ sufficientemente piccolo vale

$$f(x) = f(c_k) + f'(c_k)(x - c_k) + \frac{1}{2} f''(\xi)(x - c_k)^2,$$

dove ξ è un punto compreso nell'intervallo tra c_k e x (in qualunque ordine essi siano). Ci aspettiamo che l'ultimo termine sia piccolo, se stiamo guardando in

una sezione abbastanza piccola del grafico: se $x - c_k$ è piccolo, $(x - c_k)^2$ lo è ancora di più.

Supponiamo che ci sia uno zero $x = \alpha$ vicino a c_k; siamo in grado di valutare la distanza $|\alpha - c_k|$ usando le formule sopra? Possiamo ricavare dalla prima formula

$$|c_k - \alpha| \approx \frac{|f(c_k)|}{|f'(c_k)|}.$$

Rovesciando il ragionamento, se $\frac{|f(c_k)|}{|f'(c_k)|} \leq \varepsilon$ ci possiamo aspettare (ma non è garantito!) che c_k disti al più ε da uno zero α della funzione. Questo procedimento non è rigoroso però; può essere che la funzione f vari più di quanto ci aspettiamo; questo succede per esempio quando $f''(\xi)$ è grande, o la funzione non è differenziabile due volte.

Notiamo il fatto interessante che non importa che f sia piccolo in assoluto in c_k, ma che sia piccolo *rispetto alla sua derivata*. Valore piccolo e derivata piccola vuol dire semplicemente che la nostra funzione è molto "piatta", non per forza che siamo vicini a uno zero. (IMMAGINE) È utile cercare di avere un criterio di arresto che non dipenda dal valore della funzione, perché questi possono variare molto a seconda delle applicazioni: se la funzione di cui cerchiamo uno zero ha a che vedere con le distanze tra gli atomi di una molecola, per esempio, la scala sull'asse delle y sarà di circa $f(x) \approx 10^{-10}$ già di suo.

Ci manca ancora un ingrediente per completare questo criterio di arresto. Il valore di $f(c_k)$ viene calcolato lungo l'algoritmo, ma $f'(c_k)$ no, e calcolarlo può essere molto più complicato. Possiamo però rimpiazzarlo con un'altra approssimazione: se $c_k \in [a_k, b_k]$, allora

$$f'(c_k) \approx \frac{f(b_k) - f(a_k)}{b_k - a_k}.$$

Difatti questo è un rapporto incrementale fatto tra due punti sufficientemente vicini a c_k (IMMAGINE). Ricordiamo anche un altro teorema di analisi, il teorema di Lagrange:

Teorema 2.3. *Se $f : [a, b] \to \mathbb{R}$ è una funzione di classe $\mathcal{C}^1$, allora esiste un punto $\xi \in (a, b)$ tale che*

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}.$$

(Enunciamo questa versione per semplicità, in realtà forse nel corso di analisi avete visto il teorema con delle ipotesi leggermente diverse e meno restrittive). Questo dice che la quantità che stiamo calcolando è la derivata di f in un punto ξ, diverso da c_k (e dallo ξ del teorema precedente!), ma non troppo lontano da esso.

Quindi un possibile criterio di arresto è: se vogliamo calcolare α con un errore (circa) ε, possiamo fermare il metodo quando vale la disuguaglianza

$$|f(c_k)| \leq \left| \frac{f(b_k) - f(a_k)}{b_k - a_k} \right| \varepsilon$$

(*criterio di arresto sul residuo*).

Vantaggi e svantaggi del metodo di bisezione Vantaggi:

- Si applica a molte funzioni: richiede solo continuità di f.
- Funziona *sicuramente* se partiamo da un intervallo di separazione, restituendo un intervallo piccolo a piacere che contiene uno zero della funzione: come vedremo, non tutti i metodi offrono una garanzia di convergenza.
- Basso costo computazionale: dobbiamo calcolare la f solo su un *nuovo* punto ad ogni iterazione. Il costo (numero di operazioni) in questo caso sta nella valutazione di f (che potrebbe essere complicatissima), non nelle poche altre operazioni che facciamo come $\frac{a_n+b_n}{2}$. Se implementiamo tutto correttamente, ad ogni passo ci sarà una sola nuova valutazione di f.

Svantaggi:

- Serve avere a disposizione un intervallo di separazione da cui iniziare.
- La convergenza è lenta rispetto ad altri metodi (vedremo più avanti come misurarla).

2.2 Metodo del punto fisso

Un altro algoritmo che ci permette di cercare zeri di funzioni è il *metodo del punto fisso*, o di *iterazione funzionale*. Per questo metodo, è necessario fare delle manipolazioni algebriche per riscrivere l'equazione dalla forma $f(x) = 0$ alla forma $x = \Phi(x)$, per un'opportuna funzione Φ. Per esempio, se stiamo cercando uno zero di $x^3 - 2$, possiamo riscriverla in vari modi; per esempio:

- $x = x^3 - 2 + x$;
- $x = x - \frac{1}{6}(x^3 - 2)$;
- $x = \frac{2}{x^2}$.

Ognuno di queste scritture porta a una definizione diversa di $\Phi(x)$, la funzione che sta a destra dell'uguale. In tutti questi casi, uno zero α di $f(\alpha) = 0$ è anche un *punto fisso* di Φ, cioè soddisfa $\alpha = \Phi(\alpha)$. Tutte queste possibilità portano a scelte diverse di $\Phi(x)$, e quindi a metodi che hanno (potenzialmente) comportamenti molto diversi. Alcuni di questi metodi potrebbero generare successioni che convergono a α ed altri no. Studiamo in generale questi metodi, e poi vedremo dei modi furbi di scegliere la Φ.

Il metodo del punto fisso è fatto in questo modo: fissata una certa $\Phi : [a, b] \to \mathbb{R}$ e scelto $x_0 \in [a, b]$, costruiamo la successione

$$x_{n+1} = \Phi(x_n), \quad n = 0, 1, 2, \dots \tag{2.1}$$

Teorema del punto fisso Possiamo dimostrare questo risultato di convergenza:

Teorema 2.4 (Convergenza locale del metodo del punto fisso)**.** *Sia $\Phi \in \mathcal{C}^1([a, b])$, e $\alpha \in (a, b)$ un suo* punto fisso*, cioè un punto tale che $\Phi(\alpha) = \alpha$. Supponiamo che esista un intervallo chiuso $I = [\alpha - \rho, \alpha + \rho] \subseteq [a, b]$ tale che $|\Phi'(x)| < 1$ per ogni $x \in I$. Allora, il metodo* (2.1) *è tale che per ogni $x_0 \in I$ si ha*

- $x_n \in I$ *per ogni* $n \geq 0$ *(e quindi la successione è ben definita);*
- $\lim_{n\to\infty} x_n = \alpha$.

Dimostrazione. Sia $L = \max_{x\in I}|\Phi'(x)|$. Questo massimo esiste per il teorema di Weierstrass: $|\Phi'(x)|$ è una funzione continua (in quanto composizione di $|\cdot|$ e della funzione $\Phi'(x)$ che è continua in quanto $\Phi \in \mathcal{C}^1$) e l'intervallo I è chiuso e limitato. Inoltre $L < 1$.

Vogliamo dimostrare per induzione che

$$|x_n - \alpha| \leq L^n \rho \quad \text{per ogni } n \geq 0. \tag{2.2}$$

Per $n = 0$, la (2.2) diventa $|x_0 - \alpha| \leq \rho$, che è vera perché $x_0 \in I$. Supponiamo la (2.2) vera per un certo n, e dimostriamola per $n + 1$:

$$|x_{n+1} - \alpha| = |\Phi(x_n) - \Phi(\alpha)| = |\Phi'(\xi_n)||x_n - \alpha| \leq LL^n r = L^{n+1}\rho, \tag{2.3}$$

dove abbiamo usato il teorema di Lagrange, che dice che esiste un punto ξ_k nell'intervallo aperto di estremi x_n e α (in qualunque ordine essi siano) tale che $\Phi'(\xi_n) = \frac{\Phi(x_n)-\Phi(\alpha)}{x_n-\alpha}$. Visto che x_n e α stanno entrambi in I, anche $\xi_n \in I$ e quindi $|\Phi'(\xi_n)| \leq L$.

Questo completa la dimostrazione della (2.2). Da questa relazione si ha che $x_n \in I$, visto che $|x_n - \alpha| \leq L^n\rho \leq \rho$. Inoltre

$$0 \leq |x_n - \alpha| \leq L^n \rho$$

da cui per il teorema dei carabinieri segue che $\lim_{k\to\infty} x_n = \alpha$. □

Esempi Possiamo prendere $f(x) = x^3 - 2$, e le tre formulazioni alternative come problema di punto fisso viste sopra. Di queste, solo la seconda è localmente convergente; le altre hanno $|\Phi'(\alpha)| > 1$, e quindi non c'è speranza di applicare il teorema visto. (Anzi, con un ragionamento simile è possibile dimostrare che esiste un intervallo I tale che per tutte le successioni con $x_0 \in I \setminus \{\alpha\}$ esiste un n tale che $x_n \notin I$. Non vediamo questa dimostrazione.)

Osservazione 2.5. Se $\Phi(x) \in \mathcal{C}^1([a, b])$ e $|\Phi'(\alpha)| < 1$ per un certo punto fisso α, allora per continuità possiamo prendere un ρ sufficientemente piccolo da avere $|\Phi'(x)| < 1$ per ogni $x \in I = [\alpha - \rho, \alpha + \rho]$. Quindi le ipotesi del teorema di convergenza locale 2.4 sono verificate.

Esempio Esercizio: consideriamo l'equazione di punto fisso $x = \cos x$ (con x in radianti). Quante soluzioni positive ha? Sappiamo individuare esplicitamente un intervallo I in cui si applica il teorema del punto fisso?

Svolgimento: una soluzione corrisponde a uno zero di $f(x) = x - \cos x$. Questa funzione è tale che $f(0) = -1$, $f(\pi/2) = \pi/2$, quindi esiste almeno uno zero in $(0, \pi/2)$. La derivata di questa funzione è $f'(x) = 1 + \sin(x) \geq 0$, che si annulla solo in punti isolati, quindi la funzione è strettamente crescente. Pertanto c'è *solo* uno zero in $(0, \pi/2)$. Inoltre, non ci sono altri zeri in $[\pi/2, \infty)$ perché la funzione è strettamente positiva, $f(x) \geq x - 1 > 0$. La derivata $f'(x)$ è *strettamente* minore di 1 (in valore assoluto) al di fuori dei punti $-\pi/2, \pi/2$, e in generale tutti i punti della forma $\frac{1}{2}\pi + k\pi$, per $k \in \mathbb{Z}$. Quindi dobbiamo prendere un intervallo centrato in $\alpha \approx 0.7391$ che non contenga $-\pi/2 \approx -1.5708$ né $\pi/2 \approx 1.5708$.

Esercizio 2.6. Consideriamo il metodo del punto fisso applicato a una funzione lineare $\Phi(x) = mx + q$. Sotto quali condizioni su m e q il metodo converge?

Esercizio 2.7. Applichiamo il metodo del punto fisso all'equazione $x = \Phi(x) = \frac{1}{3} + \frac{2}{3}x^2$. Quali sono gli zeri della funzione? Per quali valori di x_0 possiamo assicurare che il metodo converge usando il teorema visto?

Vantaggi e svantaggi Svantaggi:

- Convergenza non garantita se non in un intorno della soluzione. È difficile calcolare quanto è grande questo intorno: i teoremi non ci danno una formula facile da calcolare.

Vantaggi del metodo:

- La convergenza è più "regolare" del metodo di bisezione, nel senso che la distanza da α scende sempre da un certo punto in poi; e a seconda della scelta della Φ, questo metodo può essere anche molto veloce (vedremo tra poco).

2.3 Convergenza lineare di un metodo iterativo

In questa sezione vediamo un modo di definire quanto velocemente converge un metodo numerico come quelli che abbiamo visto. Partiamo con un esempio che mostra cosa succede in un caso reale, in Figura 2.1.

Notare che abbiamo usato una *scala logaritmica* sull'asse delle y: con una scala lineare, non si riesce a vedere praticamente nulla già al di sotto di 10^{-2}, è tutto schiacciato sullo zero.

Convergenza lineare Abbiamo una successione di approssimazioni x_n di una certa soluzione esatta α, ognuna calcolata da quella precedente. La successione si dice *convergente* se l'errore $e_n := |x_n - \alpha|$ tende a 0. Ma quanto velocemente? L'esempio più semplice è quello in cui e_n si comporta come una successione geometrica:

$$e_0 = a,\ e_1 = ar,\ e_2 = ar^2,\ e_3 = ar^3, \dots$$

Ad ogni passo, l'errore precedente viene moltiplicato per r. Ovviamente serve $r < 1$ per avere convergenza. In un grafico come quello sopra, la successione e_k corrisponde a punti allineati lungo una retta di coefficiente angolare $\log r$ (che, ricordiamolo, è negativo): difatti la differenza tra due punti successivi è il valore costante $\log e_{n+1} - \log e_n = \log r < 0$.

Molti metodi hanno un comportamento simile "al limite", quindi introduciamo una definizione che formalizza questo comportamento. Diciamo che un metodo iterativo *converge linearmente* (o ha *ordine di convergenza* 1) quando

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n} = r \in (0, 1). \tag{2.4}$$

Questo vuol dire che "sufficientemente avanti" la successione degli errori si comporta come una successione geometrica. Questo corrisponde al comportamento che vediamo per alcune delle successioni nella figura 2.1: a parte i primissimi, i punti tendono ad essere allineati.

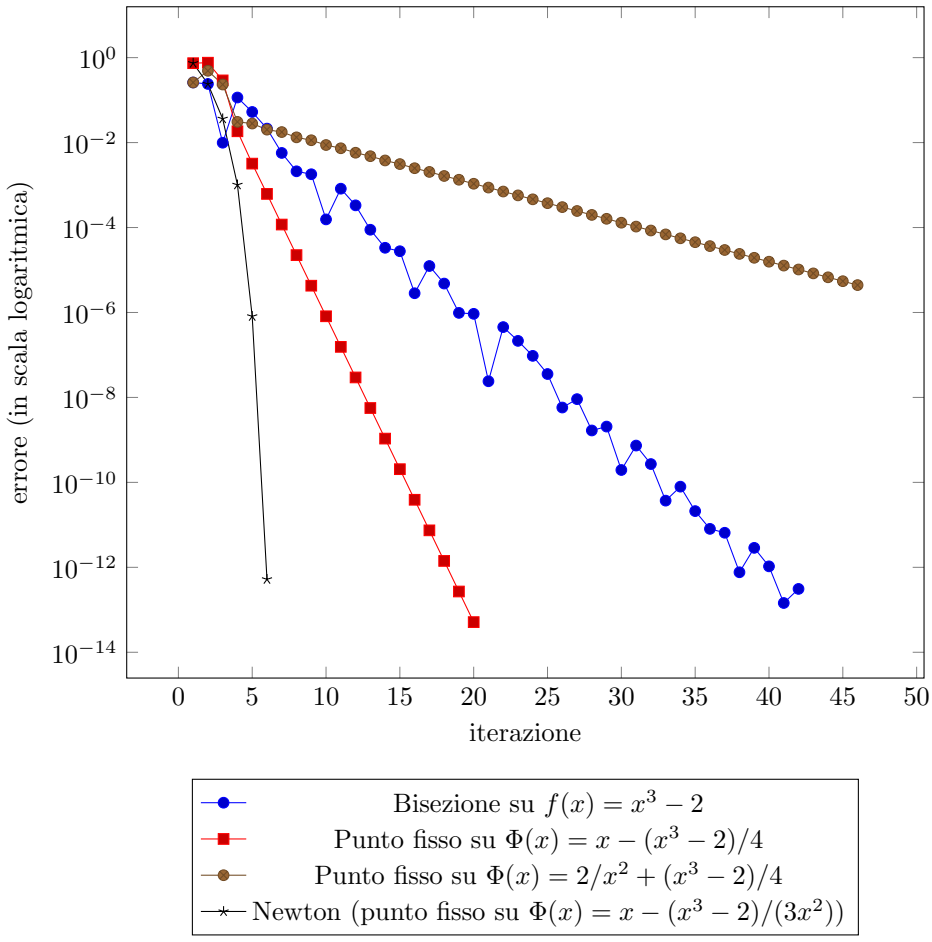


Figura 2.1: Convergenza di diversi metodi iterativi per trovare uno zero di $f(x) = x^3 - 2$.

La convergenza è più veloce quanto più piccolo è r, che si chiama *tasso di convergenza* (in inglese *convergence rate*), o più informalmente spesso si usa *velocità di convergenza*.

Tasso di convergenza del metodo di punto fisso Riprendendo il conto fatto nell'equazione (2.3), abbiamo

$$\lim_{n\to\infty} \frac{|x_{n+1}-\alpha|}{|x_n-\alpha|} = \lim_{n\to\infty} |\Phi'(\xi_n)| = |\Phi'(\alpha)| \in [0,1).$$

L'ultima uguaglianza vale perché abbiamo $0 \le |\xi_n - \alpha| \le |x_n - \alpha|$, e quindi anche $\xi_n \to \alpha$. Inoltre $x \mapsto |\Phi'(x)|$ è una funzione continua (perché composizione della funzione valore assoluto, che è continua, e della funzione Φ', che è continua perché $\Phi \in \mathcal{C}^1$). Quindi possiamo passare al limite.

Quindi la convergenza è *almeno lineare*. Perché "almeno"? Perché può succedere che $|\Phi'(\alpha)| = 0$, allora punti successivi tendono ad essere allineati in verticale, e la successione va a zero più velocemente di qualunque retta sul grafico in scala logaritmica. Questo succede per la quarta successione in Figura 2.1, che converge più velocemente degli altri metodi. Studieremo questo metodo nel dettaglio più avanti.

Esercizio 2.8. Consideriamo il metodo del punto fisso applicato a una funzione lineare $\Phi(x) = mx + q$. Quanto vale il tasso di convergenza r?

Esercizio 2.9. Supponiamo di avere una funzione con uno zero $\alpha = 1$, e di avere un metodo numerico scritto in modo sbagliato che produce una successione x_k che converge a $\lim x_k = 2$. Quanto vale il limite r in (2.4)?

Dall'esercizio precedente, ricaviamo che $r \ge 1$ può essere un'indicazione che il metodo non sta convergendo, o sta convergendo alla soluzione sbagliata!

Tasso di convergenza del metodo di bisezione Nel caso del metodo di bisezione, non riusciamo a dimostrare una convergenza secondo questa definizione. Difatti può anche succedere che l'errore e_k cresca da un passo all'altro: è facile costruire una situazione in cui lo zero α è più vicino a c_k che a c_{k+1} (IMMAGINE). Il comportamento non monotono si vede anche nella Figura 2.1.

Riusciamo però a dimostrare che $e_n \le \varepsilon_n$, dove ε_n è una successione che converge linearmente: difatti abbiamo visto che

$$e_n \le \underbrace{\frac{b_1 - a_1}{2^n}}_{=\varepsilon_n},$$

e naturalmente

$$\lim \frac{\varepsilon_{n+1}}{\varepsilon_n} = \frac{1}{2}.$$

Diciamo allora che il metodo di bisezione ha *convergenza pressoché lineare*. Su un grafico in scala logaritmica, gli errori e_n stanno al di sotto della retta data dai punti ε_n, e solitamente non se ne discostano troppo. Questo è confermato anche dalla nostra figura.

2.4 Metodo di Newton

Il metodo di Newton è un metodo per la ricerca di zeri che offre convergenza più veloce, ma richiede di essere in grado di calcolare non solo $f \in \mathcal{C}^1([a,b])$, ma anche la sua derivata f'.

L'idea è la seguente. Ad ogni passo, data un'approssimazione x_n di una soluzione, calcoliamo x_{n+1} calcolando uno zero della *retta tangente* al grafico di $f(x)$ in x_n. L'equazione di questa retta è

$$y(x) = f(x_n) + f'(x_n)(x - x_n),$$

quindi x_{n+1} è il punto tale che

$$0 = y(x_{n+1}) = f(x_n) + f'(x_n)(x_{n+1} - x_n),$$

ossia, risolvendo,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Il metodo di Newton è un metodo di punto fisso con funzione $\Phi(x) = x - \frac{f(x)}{f'(x)}$. Notiamo che questa funzione può non essere ben definita se $f'(x) = 0$: quando incontriamo un'iterata x_n per cui si ha $f'(x_n) = 0$, il metodo di Newton fallisce e non riusciamo a calcolare l'iterata successiva.

Sotto l'ipotesi aggiuntiva $f'(\alpha) \neq 0$, riusciamo a dimostrare la convergenza locale del metodo di Newton.

Teorema 2.10. *Sia $f \in \mathcal{C}^2([a,b])$, e sia $\alpha \in (a,b)$ tale che $f(\alpha) = 0$ e $f'(\alpha) \neq 0$. Allora, il metodo di Newton converge localmente: cioè, esiste un intervallo $I = [\alpha - \rho, \alpha + \rho]$ tale che per ogni successione generata dal metodo con $x_0 \in I$ si ha*

- *$x_n \in I$ per ogni $n \geq 0$;*
- *$\lim_{n\to\infty} x_n = \alpha$.*

Dimostrazione. Poiché $f'(\alpha) \neq 0$ e la funzione derivata $f'(x)$ è continua, in un intorno sufficientemente piccolo di α si ha $f'(x) \neq 0$. Inoltre, possiamo calcolare

$$\Phi'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}; \tag{2.5}$$

pertanto se $f'(\alpha) \neq 0$ abbiamo $\Phi'(\alpha) = 0$. Come nell'Osservazione 2.5, possiamo affermare che in un intorno sufficientemente piccolo di α avremo $\Phi'(x) < 1$. Possiamo quindi trovare un intervallo I centrato in α in cui la $\Phi(x)$ esiste (il denominatore non si annulla) e $\Phi'(x) < 1$ per ogni $x \in I$. In questo intervallo possiamo applicare il teorema del punto fisso e quindi ottenere la tesi. □

Vantaggi e svantaggi del metodo di Newton Svantaggi:

- Richiede di saper calcolare non solo f ma anche f': con Matlab, tipicamente dovremo passare due funzioni anziché una.
- Costo computazionale maggiore: ad ogni passo, dobbiamo valutare f e f' una volta.

- Convergenza non garantita.

Vantaggi:

- Nella maggior parte dei casi, converge molto più velocemente degli altri metodi.

Il comportamento che si osserva tipicamente dal metodo di Newton è un certo periodo di "oscillazioni iniziali" e poi una convergenza molto veloce; appena x_n si avvicina a α e l'errore e_n va sotto una certa soglia, esso comincia ad andare a zero molto velocemente.

Ordine di convergenza di un metodo iterativo Diciamo invece che si ha *convergenza superlineare* quando il limite $\lim_{n\to\infty} \frac{e_{n+1}}{e_n}$ è uguale a zero: questo vuol dire che la successione degli errori e_n va a zero più velocemente di una serie geometrica, qualunque sia la sua ragione r. In un grafico in scala logaritmica, la differenza di ordinata tra due punti successivi $\log e_{n+1} - \log e_n$ tende a $-\infty$, quindi il grafico diventa più ripido di qualsiasi retta.

Per studiare la velocità di convergenza di metodi superlineari in modo più preciso, possiamo introdurre un'altra definizione: dato un $p > 1$, diciamo che un metodo ha *ordine di convergenza* p se vale

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^p} = r \in (0, \infty).$$

Notare che non abbiamo più il requisito $r < 1$ che c'era quando $p = 1$: in tutti questi casi la convergenza è superlineare, anche se r è grande. Difatti, se un metodo ha ordine di convergenza $p > 1$ allora vale

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n} = \lim_{n\to\infty} \frac{e_{n+1}}{e_n^p} \cdot \lim_{n\to\infty} e_n^{p-1} = r \cdot 0 = 0. \tag{2.6}$$

Esempio La successione $x_n = 3 \cdot 10^{-2^n}$ converge a 0 con ordine di convergenza 2: difatti si ha

$$\frac{e_{n+1}}{e_n^2} = \frac{3 \cdot 10^{-2^{n+1}} - 0}{(3 \cdot 10^{-2^n} - 0)^2} = \frac{3 \cdot 10^{-2^{n+1}}}{9 \cdot 10^{-2^{n+1}}} = \frac{1}{3}.$$

I primi elementi della successione sono $x_0 = 3 \cdot 10^{-1}$, $x_1 = 3 \cdot 10^{-2}$, $x_2 = 3 \cdot 10^{-4}$, $x_3 = 3 \cdot 10^{-8}$, $x_4 = 3 \cdot 10^{-16}$, cioè $x_n = 0,00 \ldots 03$, dove prima del 3 ci sono 2^n zeri in totale (incluso quello prima della virgola). Il numero di zeri raddoppia ad ogni passo, e questo fa sì che la successione converga molto più velocemente che non una successione che converge linearmente. In un grafico in scala logaritmica, gli errori tendono a scendere come un'esponenziale, e la distanza (in verticale) tra due punti successivi $\log e_{n+1} - \log e_n$ raddoppia ad ogni passo.

Diciamo che un metodo numerico converge con ordine *almeno* p se vale

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^p} = r \in [0, \infty).$$

Questa volta abbiamo incluso lo zero tra i possibili valori ammessi per r. Notare che questo concetto di "convergenza di ordine almeno k" si comporta come ci aspettiamo dalle parole che abbiamo usato: se un metodo converge con ordine k, allora converge anche di ordine almeno h per ogni $h \leq k$. (La dimostrazione di questo fatto è analoga alla (2.6).)

Convergenza quadratica

Teorema 2.11 (Convergenza almeno quadratica del metodo di Newton)**.** *Supponiamo $f \in \mathcal{C}^2([a,b])$. Se il metodo di Newton converge a $\alpha \in (a,b)$ e se $f'(\alpha) \neq 0$ (*radice semplice*), allora l'errore $e_n = |x_n - \alpha|$ soddisfa*

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^2} = r \in [0,\infty).$$

(cioè il limite esiste ed è finito.)

Dimostrazione. Usiamo uno sviluppo di Taylor (con resto di Lagrange) in x_n per scrivere l'uguaglianza

$$0 = f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + \frac{f''(\xi_n)}{2}(\alpha - x_n)^2$$

che vale per uno ξ_n compreso tra α e x_n.

Dividiamo per $f'(x_n)$ per ottenere

$$\frac{f''(\xi_n)}{2f'(x_n)}(\alpha - x_n)^2 = -\frac{f(x_n)}{f'(x_n)} - \alpha + x_n = x_{n+1} - \alpha.$$

Allora

$$\frac{x_{n+1} - \alpha}{(x_n - \alpha)^2} = \frac{f''(\xi_n)}{2f'(x_n)}.$$

Vogliamo ora passare al limite per $n \to \infty$. Sappiamo che $0 \leq |\xi_n - \alpha| \leq |x_n - \alpha|$, da cui per il teorema dei carabinieri concludiamo che $|\xi_n - \alpha| \to 0$ e quindi $\xi_n \to \alpha$. Allora, di nuovo per continuità di queste funzioni, $f''(\xi_n) \to f''(\alpha)$ e $f'(x_n) \to f'(\alpha)$ (che è diverso da zero); quindi

$$\lim_{n\to\infty} \frac{x_{n+1} - \alpha}{(x_n - \alpha)^2} = \lim \frac{f''(\xi_n)}{2f'(x_n)} = \frac{f''(\alpha)}{2f'(\alpha)} = c \in [0,\infty).$$

□

Criterio di arresto Come per il metodo di bisezione, un criterio di arresto euristico è: fermiamoci quando $\left|\frac{f(x_n)}{f'(x_n)}\right| \leq \varepsilon$; ci aspettiamo (senza garanzie!) che questo avvenga quando $|x_n - \alpha| \leq \varepsilon$. Per come è definito il metodo, questo corrisponde a $|x_n - x_{n+1}| \leq \varepsilon$, cioè, ci fermiamo quando due iterate successive sono abbastanza vicine.

Zeri di molteplicità superiore a 1 Cosa succede al metodo di Newton se la funzione a cui lo applichiamo ha $f'(\alpha) = 0$? Introduciamo innanzitutto una definizione: diciamo che *f ha uno zero di molteplicità m in α* se si ha

$$f(\alpha) = f'(\alpha) = f''(\alpha) = \cdots = f^{(m-1)}(\alpha) = 0, \quad \text{ma } f^{(m)}(\alpha) \neq 0.$$

Questa definizione coincide con quella che avete già visto per i polinomi. Difatti, è possibile dimostrare questo risultato:

Lemma 2.12. *Se $f \in \mathcal{C}^m([a,b])$ ha uno zero di molteplicità m in α, allora si ha $f(x) = (x-\alpha)^m g(x)$, dove $g(x)$ è una funzione continua in $[a,b]$ con $g(\alpha) \neq 0$.*

Dimostrazione. Definiamo $g(x) = \frac{f(x)}{(x-\alpha)^m}$. Questa funzione è definita dappertutto tranne che in $x = \alpha$, ma possiamo estenderla per continuità. Per farlo, dobbiamo calcolare il limite

$$\lim_{x\to\alpha} \frac{f(x)}{(x-\alpha)^m}.$$

Per calcolare questo limite, utilizziamo m volte il teorema di De l'Hopital:

$$\begin{aligned}\lim_{x\to\alpha} \frac{f(x)}{(x-\alpha)^m} &= \lim_{x\to\alpha} \frac{f'(x)}{m(x-\alpha)^{m-1}} = \lim_{x\to\alpha} \frac{f''(x)}{m(m-1)(x-\alpha)^{m-2}} = \dots \\ &= \lim_{x\to\alpha} \frac{f^{(m)}(x)}{m!\cdot 1} = \frac{f^{(m)}(\alpha)}{m!} \neq 0.\end{aligned}$$

(Tutti i limiti tranne l'ultimo sono della forma $\frac{0}{0}$.) □

Se inseriamo la forma $f = (x-\alpha)^m g(x)$ nella formula (2.5) per la derivata di $\Phi(x)$, e supponiamo $g \in \mathcal{C}^2$, vediamo che anche Φ e Φ' si possono estendere per continuità, e si ha

$$\Phi'(\alpha) = \frac{m-1}{m} < 1$$

(vi invito a provare a fare i conti a casa, non sono complicati).

Quindi, quando α è uno zero di molteplicità $m > 1$, il metodo di Newton converge localmente ad α, ma questa volta solo linearmente.

Metodo di Newton modificato Data una funzione f che ha uno zero di molteplicità m in α, se conosciamo il valore di m possiamo fare una modifica del metodo che converge di nuovo quadraticamente (*metodo di Newton modificato*):

$$x_{n+1} = \underbrace{x_n - m\frac{f(x_n)}{f'(x_n)}}_{:=\Phi_m(x_n)} \quad n = 0, 1, 2, \dots \tag{2.7}$$

(notare la m di fronte al secondo termine). In questo caso, ripetendo i calcoli otteniamo che $\Phi'_m(\alpha) = 0$. Si può inoltre dimostrare che il metodo (2.7) converge (localmente) con ordine almeno 2.

Esercizi Esempio: metodo di Newton su $f(x) = mx + q$ (equazione lineare) $\to$ convergenza in un passo.

Esempio: metodo di Newton su $f(x) = x^2 - a \to$ buono come metodo anche per calcolare radici quadrate a mano. Esempio: calcolando $\sqrt{17}$ con $x_0 = 4$, già x_2 ha 5 cifre significative esatte.

Esempio: metodo di Newton su $f(x) = x^2 \to$ la convergenza diventa lineare con ragione $\frac{1}{2}$.

Esempio: metodo di Newton modificato su $f(x) = x^2$ per recuperare convergenza quadratica (converge di nuovo in un passo).

Varianti del metodo di Newton Non sempre si ha a disposizione (in modo facile da calcolare) la derivata di una funzione f di cui vogliamo trovare gli zeri. Per questo esistono alcune varianti che cercano di "imitare" il metodo di Newton ma senza dover calcolare derivate.

Metodo delle corde È l'iterazione $x_{n+1} = x_n - \frac{f(x_n)}{c}$, dove c è un valore costante fissato. Per esempio, si può prendere $c = f'(x_0)$, se lo si conosce; questo richiede di calcolare la f' una volta sola anziché una volta per passo come nel metodo di Newton. Geometricamente, questo metodo corrisponde a rimpiazzare le rette tangenti che si usano nel metodo di Newton con rette che hanno coefficiente angolare costante c. Il metodo non converge sempre, e quando lo fa ha convergenza almeno lineare (idea per dimostrarlo: anche questo è un metodo di punto fisso $x_{n+1} = \Phi(x_n)$; quanto fa $\Phi'(\alpha)$?).

Metodo delle secanti È l'iterazione che si ottiene rimpiazzando, nel metodo di Newton, $f'(x_n)$ con il rapporto incrementale calcolato sugli ultimi due punti,

$$\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

In questo modo si rimpiazza la derivata con un oggetto più economico da calcolare; difatti abbiamo già calcolato $f(x_n)$ e $f(x_{n-1})$ nei passi precedenti del metodo, quindi dobbiamo fare solo due sottrazioni e una divisione (che in realtà diventa una moltiplicazione, quando si va a scrivere il metodo per esteso). Anche questo metodo converge localmente, e il suo di convergenza è $p = \frac{1+\sqrt{5}}{2} \approx 1.618$ (non lo dimostriamo, è più complesso: difatti non è un metodo del tipo $x_{n+1} = \Phi(x_n)$, perché ogni valore dipende dalle *due* iterate precedenti). Per applicarlo, serve partire da *due* punti iniziali x_0 e x_1.

[Note per lezione Matlab:

- Descrizione sintassi vettori e matrici: `A = [1 2;3 4]`
- Accesso elementi con `A(1,2)`. Accesso elementi oltre i limiti.
- `size(), length()`
- Plot: esempio con plot della funzione quadrato.
- Remark che esiste `1:n` già fatto.
- Funzioni già pronte per bisezione, punto fisso, Newton. Provarle su $f(x) = x^2 - 2$.
- Calcolo dell'errore come `abs(xs - sqrt(2))`.
- Grafici in scala logaritmica.

]

Capitolo 3

Aritmetica di macchina

Se provate a implementare i metodi precedenti su un computer con Matlab, noterete un comportamento abbastanza particolare: i metodi spesso smettono di convergere attorno a 10^{-15} o 10^{-16}. Per comprendere cosa sta succedendo, dobbiamo studiare un po' meglio come i numeri reali vengono rappresentati in un computer.

Rappresentazione in base Scegliamo un intero $\beta > 1$ che sarà la *base* della nostra rappresentazione. Quella che segue è sostanzialmente la notazione scientifica che già conoscete.

Teorema 3.1 (rappresentazione scientifica in base β)**.** *Fissato un intero $\beta > 1$ (la* base*), ogni numero reale $x \neq 0$ si può scrivere come*

$$x = \pm\beta^p \sum_{i=1}^{\infty} c_i \beta^{-i}, \tag{3.1}$$

*dove i c_i (*cifre*) sono interi $0 \leq c_i < \beta$.*

Questa scrittura è unica se aggiungiamo le condizioni che $c_1 \neq 0$, e che c_i non sono tutti uguali a $\beta - 1$ da un certo punto in poi.

Per esempio, $x = -764.88888\ldots$ (periodico) si scrive in base $\beta = 10$ come

$$x = -10^3(7 \cdot 10^{-1} + 6 \cdot 10^{-2} + 4 \cdot 10^{-3} + 8 \cdot 10^{-4} + 8 \cdot 10^{-5} + 8 \cdot 10^{-6} + \ldots)$$

Terminologia: p si chiama *esponente* di x, la quantità nella sommatoria si chiama *mantissa*.

Non ci interessa qui dimostrare questo teorema: "sappiamo che è vero" fin dalla scuola primaria, e più si va verso fatti base e più bisogna essere puntigliosi e formali nelle dimostrazioni. Però ha senso soffermarsi sulle questioni di unicità.

La prima condizione $c_1 \neq 0$ serve a escludere scritture alternative con zeri iniziali, per esempio

$$-764.88888 = -10^5(0 \cdot 10^{-1} + 0 \cdot 10^{-2} + 7 \cdot 10^{-3} + 6 \cdot 10^{-4} + 4 \cdot 10^{-5} + \ldots)$$

Questa rappresentazione senza zeri iniziali si chiama *normalizzata*.

La seconda è per escludere scritture come $0.999999\ldots$ (periodico): se vi ricordate come si sommano le serie, questa scrittura è uguale a 1, ed è un modo diverso di scriverlo che vogliamo escludere.

Numeri di macchina Su un computer, possiamo rappresentare solo una quantità finita di numeri.

Definizione 3.2. *L'insieme dei* numeri di macchina (o floating-point) normalizzati, $\mathbb{F}(\beta, t, m, M)$ *è l'insieme dei numeri della forma*

$$\pm\beta^p \sum_{i=1}^{t} c_i \beta^{-i}, \quad p \in \{m, m+1, \ldots, M\}$$

Ci sono due grossi cambiamenti rispetto alla (3.1): t cifre (anziché infinite) nella mantissa, e un range finito per gli esponenti, da m a M.

Esempio Prendiamo i numeri di macchina $\beta = 10$, $t = 2$, $m = -3$, $M = 3$.

Il numero di macchina positivo più piccolo è $10^{-3}(1 \cdot 10^{-1} + 0 \cdot 10^{-2}) = 0.00010$. I numeri di macchina successivi si ottengono incrementando le cifre: $0.00011, 0.00012, \ldots, 0.00099$. Difatti numeri intermedi, ad esempio 0.000101, non si possono scrivere con sole due cifre più un esponente. A questo punto abbiamo elencato tutti i numeri con esponente -3; il numero di macchina ancora successivo ha esponente -2: $10^{-2}(1 \cdot 10^{-1} + 0 \cdot 10^{-2}) = 0.0010$. Si prosegue con $0.0011, 0.0012, \ldots, 0.0099$. Poi seguono tutti i numeri di macchina con esponente -1, 0, 1, 2, 3. I numeri di macchina più grandi sono $970, 980, 990$: esponente 10^3, mantisse più grandi possibili $0.97, 0.98, 0.99$. Numeri come 989 richiedono tre cifre e quindi non stanno nell'insieme dei numeri di macchina normalizzati.

I numeri con esponente -3 hanno "distanza" 10^{-5} l'uno dal successivo; i numeri con esponente -2 hanno "distanza" 10^{-4} l'uno dal successivo, fino ai numeri con esponente 3 che hanno distanza 10.

Se segniamo sulla retta reale i numeri di macchina quindi vediamo che i numeri più vicini allo zero (p piccolo) hanno uno spazio minore tra l'uno e l'altro; non sono "tacche equispaziate". Vediamo però che queste spaziature variabili sono proprio quello che consente una buona approssimazione relativa.

Se x è un numero di macchina, anche $-x$ lo è; quindi per enunciare le proprietà successive ci restringiamo ai numeri positivi; saranno valide anche per numeri negativi, ed è facile adattare le dimostrazioni.

Numero di macchina successivo, più piccolo e più grande Dato un numero di macchina normalizzato positivo $x = \beta^p \sum_{i=1}^{t} c_i \beta^{-i}$, il numero di macchina immediatamente successivo è quello che si ottiene aggiungendo 1 all'ultima cifra, cioè $x + \beta^{p-t}$. Questo è chiaro se l'ultima cifra è diversa da $\beta - 1$; se l'ultima cifra è $\beta - 1$, aggiungendo 1 all'ultima cifra ci sono dei riporti, e la mantissa del risultato potrebbe avere $t + 1$ cifre; però l'ultima cifra è 0 e quindi si può omettere.

Il numero (positivo) più piccolo rappresentabile, che chiamiamo ω, si ottiene scegliendo $p = m$ e mantissa $1000\ldots0$. Il numero più grande rappresentabile, Ω, si ottiene scegliendo $p = M$ e mantissa con tutte cifre $\beta - 1$.

Altri valori rappresentabili Un computer oltre a questi numeri rappresenta alcuni valori speciali:

- Lo zero (difficile fare senza, e non è un numero normalizzato!)

- $+\infty, -\infty, -0$: vengono aggiunti, con regole aritmetiche ispirate dall'analisi come $1/-\infty = -0$, per far sì che gli algoritmi possano funzionare anche in alcuni casi limite.
- `NaN`, che viene restituito come "codice d'errore" da alcune operazioni non valide come $0/0$.
- Alcuni numeri in più compresi tra 0 e ω, chiamati *numeri denormalizzati*; non ci interessano qui.

Approssimazione con numeri di macchina

Teorema 3.3. *Dato un numero reale $x \in [-\Omega, -\omega] \cup [\omega, \Omega]$, esiste un numero di macchina $\tilde{x}$ tale che*

$$\frac{|\tilde{x} - x|}{|x|} < \beta^{1-t}. \tag{3.2}$$

Dimostrazione. Possiamo assumere che x sia positivo (il caso negativo è analogo), e che non sia esso stesso un numero di macchina (altrimenti è ovvio). Allora x è compreso tra due numeri di macchina successivi, chiamiamoli $\underline{x}$ e $\overline{x}$. In particolare, $\underline{x}$ si ottiene troncando la rappresentazione (3.1), cioè arrestando la sommatoria a t anziché a ∞, e ha lo stesso esponente p del numero x. Possiamo scegliere se prendere $\tilde{x} = \underline{x}$ (troncamento o arrotondamento verso 0), $\tilde{x} = \overline{x}$ (arrotondamento verso infinito), o quello dei due che ha un errore minore (arrotondamento al più vicino); tutti questi forniscono arrotondamenti che soddisfano la (3.2). In ogni caso, visto che $x \in (\underline{x}, \overline{x})$, si ha

$$|\tilde{x} - x| < \overline{x} - \underline{x} = \beta^{p-t}. \tag{3.3}$$

Inoltre, visto che la prima cifra di x è almeno 1 e le altre sono positive o nulle,

$$x \geq \beta^{p-1}. \tag{3.4}$$

Dividendo membro a membro queste due disuguaglianze (notare che i versi sono quelli giusti per farlo!) si ha la tesi. □

Errore relativo Data un'approssimazione $\tilde{x}$ di un numero reale $x \neq 0$, il suo *errore relativo* è

$$\varepsilon = \frac{\tilde{x} - x}{x}. \tag{3.5}$$

È un oggetto molto naturale da considerare: l'errore assoluto $|\tilde{x} - x|$ da solo non dice nulla: possiamo fare degli esempi dalla "vita reale" di errori su lunghezze e prezzi. Per esempio, aver misurato una lunghezza con un errore di 0.5 cm da solo non vuol dire nulla: è molto diverso se questa lunghezza è la distanza dalla terra alla luna, o una tolleranza di fabbricazione sulla cover del vostro cellulare.

Possiamo riscrivere la (3.5) come $\tilde{x} = x(1 + \varepsilon)$. Quindi la (3.2) dice che per ogni x in quegli intervalli esiste un numero di macchina $\tilde{x} = x(1 + \varepsilon)$ che lo approssima con un errore relativo che soddisfa $|\varepsilon| \leq \beta^{1-t}$. La quantità $\mathsf{u} = \beta^{1-t}$ (*precisione di macchina*) non dipende da x, ma solo dall'insieme di numeri di macchina scelto.

Numeri a doppia precisione Esiste uno standard (IEEE 754) per l'aritmetica di macchina che specifica quali parametri scegliere e il risultato di ogni operazione. Il formato più comune è quello noto come `double`, `float64` o `binary64`. Corrisponde a $\beta = 2, t = 53, m = -1022, M = 1023$. Ogni numero viene rappresentato in 64 bit (=valori 0/1). Con questi valori ogni numero compreso tra $\omega \approx 2.2 \cdot 10^{-308}$ e $\Omega \approx 1.8 \cdot 10^{308}$ viene rappresentato con errore relativo $\mathtt{u} \approx 2.2 \cdot 10^{-16}$. Matlab (che useremo per programmare in questo corso) usa questo formato.

C'è anche un altro formato comune, chiamato `single` o `float32`, che ha un errore relativo di $\approx 10^{-8}$. È più impreciso, ma permette di memorizzare più numeri nello stesso spazio (32 bit l'uno).

Quando scriviamo (per esempio) `x = 0.3` in Matlab, questo numero non viene memorizzato esattamente: non appartiene a $\mathbb{F}(2, 53, -1022, 1023)$, ma anzi in base 2 è il numero periodico $0.0\overline{1001}$. Questo numero periodico viene troncato a $t = 53$ cifre. Quindi il numero con cui il computer lavora non è *esattamente* 0.3, bensì

$$\tilde{x} = 0.2\underbrace{999999999999999}_{\text{15 volte 9}}8889776975374843459576368331909179687 5,$$

che ha un errore relativo minore di $2.2 \cdot 10^{-16}$, come promesso dal teorema.

Per fare qualche esperimento: `https://www.exploringbinary.com/floating-point-converter/`.

Operazioni di macchina Una volta memorizzati due numeri, possiamo chiedere al computer di calcolarne la somma, per esempio se scrivete in Matlab `x = 0.3; y = 0.4; x + y` viene visualizzato un risultato.

Il computer memorizza approssimazioni $\tilde{x}$ e $\tilde{y}$ di 0.3 e 0.4, che soddisfano $\tilde{x} = 0.3(1 + \varepsilon_1), \tilde{y} = 0.4(1 + \varepsilon_2)$, con $|\varepsilon_i| \leq \mathtt{u}$. Ma c'è una terza fonte di errore: anche se $\tilde{x}$ e $\tilde{y}$ sono numeri di macchina, la loro somma $\tilde{x} + \tilde{y}$ non lo è per forza; quindi va approssimata anche lei con un numero di macchina. L'operazione "calcola la somma di $\tilde{x}$ e $\tilde{y}$ e rimpiazzala con il numero di macchina più vicino" viene indicata con $\tilde{x} \oplus \tilde{y}$. Analogamente definiamo $\ominus, \odot, \oslash$. Quindi $\tilde{x} \oplus \tilde{y} = (\tilde{x} + \tilde{y})(1 + \varepsilon)$, per un opportuno errore ε che soddisfa $|\varepsilon| \leq \mathtt{u}$, e analogamente per le altre operazioni.

Underflow/overflow Eseguendo queste operazioni, si possono incontrare numeri più grandi di Ω in valore assoluto. Questi numeri vengono rimpiazzati con $+\infty$ o $-\infty$. Questo fenomeno si chiama *overflow*. Similmente, quando le operazioni producono numeri più piccoli di ω in valore assoluto, questi vengono rimpiazzati con 0 (*underflow*).

Esempi Consideriamo di nuovo il sistema di numerazione con $\beta = 10, t = 2, m = -3, M = 3$. Qual è il risultato delle seguenti operazioni (arrotondando al più vicino)? $0.1 \ominus 0.001$; $1.01 \ominus 1.01$; $(0.77 \oplus 0.44) \oplus 1$; $0.077(\oplus 0.044 \oplus 1)$; $(1 \oslash 3) \otimes 3$; $500 \oplus 500$; $0.001 \odot 0.001$; $50 \oplus 0.01$.

Qual è il più piccolo intero che non è un numero di macchina in questo sistema?

Commenti Perché è stato scelto questo sistema di numerazione? Esistono alternative, come fare i conti con razionali esatti (ma i numeratori diventano presto *molto* grandi anche quando si fanno operazioni semplici) o tenere traccia degli errori calcolando esplicitamente degli intervalli di inclusione per ogni quantità calcolata (ma gli intervalli diventano presto *molto* grandi anche quando si fanno operazioni semplici). Alla fine questo è quello che si è affermato negli anni come il più comodo con cui fare i conti nelle applicazioni. Ciononostante ci sono delle approssimazioni, per cui è importante comprendere teoricamente il loro impatto.

Capitolo 4

Analisi dell'errore

Supponiamo di voler usare il calcolatore per calcolare una quantità $y = f(x)$ che dipende da un numero reale x. Supponiamo per ora che f sia una funzione razionale (cioè che si scrive combinando solo le quattro operazioni). Per esempio, $f(x) = x^2 - 3/x$, e vogliamo calcolare il suo valore nel punto $x = 0.2$. Per esempio con Matlab

```
x = 0.2;
y = x^2 - 3/x
```

Ci sono due diverse fonti di errore che fanno sì che la quantità calcolata non sia il risultato esatto.

Errore inerente Il computer lavora non con il dato di partenza esatto x, ma con una sua approssimazione $\tilde{x}$ affetta da un errore relativo ε. Questo succede sicuramente tutte le volte che x non è un numero di macchina, perché per utilizzarlo il computer deve approssimarlo con un numero di macchina $\tilde{x} = x(1 + \varepsilon)$, commettendo un errore relativo $|\varepsilon| \leq \mathtt{u}$. Però ci possono essere altre fonti di errori: per esempio, x potrebbe essere a sua volta il risultato di operazioni precedenti, o potrebbe venire da una misurazione nel mondo reale (quindi affetto da un errore che tipicamente è molto più grande di $\mathtt{u}$). Quindi il computer in realtà può calcolare solo $f(\tilde{x})$ commettendo un *errore inerente* pari a

$$e_{in} = \frac{f(\tilde{x}) - f(x)}{f(x)}. \tag{4.1}$$

IMMAGINE illustrativa: grafico di una funzione in floating point "per punti".

Errore algoritmico Per quanto visto sopra, il computer può effettuare le operazioni aritmetiche solo approssimando il loro risultato con numeri di macchina; quindi può calcolare non $f(\tilde{x})$, bensì un'altra funzione $h(\tilde{x})$. Per esempio, dando i comandi più sopra, invece di $f(\tilde{x}) = x^2 - 3/x$, il computer calcolerà $h(\tilde{x}) = \tilde{x} \otimes \tilde{x} \ominus 3 \oslash \tilde{x}$. La differenza tra queste due funzioni determina un *errore algoritmico*

$$e_{alg} = \frac{h(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}.$$

Errore totale Possiamo dare un'espressione approssimata per l'*errore totale*, cioè

$$e_{tot} = \frac{h(\tilde{x}) - f(x)}{f(x)}.$$

"Approssimata" perché (1) assumiamo che entrambi questi errori siano piccoli, e (2) ignoriamo termini che contengono il prodotto di due errori. Un po' come approssimare $\exp(x) \approx 1 + x$ per valori di x piccoli, ignorando i termini di grado superiore. Nel dettaglio, usiamo il simbolo $a \doteq b$ per indicare che a e b sono uguali a patto di ignorare "termini di ordine superiore" (cosa siano esattamente dipende dal contesto: nel nostro caso, termini che sono il prodotto di due o più errori).

Teorema 4.1.

$$e_{tot} \doteq e_{in} + e_{alg}.$$

Dimostrazione. Notiamo innanzitutto che riarrangiando la (4.1) si ottiene

$$f(\tilde{x}) = f(x)(1 + e_{in}).$$

Ora possiamo scrivere

$$\begin{aligned} e_{tot} &= \frac{h(\tilde{x}) - f(x)}{f(x)} = \frac{h(\tilde{x}) - f(\tilde{x})}{f(x)} + \frac{f(\tilde{x}) - f(x)}{f(x)} \\ &= e_{alg}\frac{f(\tilde{x})}{f(x)} + e_{in} = e_{alg}(1 + e_{in}) + e_{in} \\ &= e_{alg} + e_{in} + e_{alg}e_{in} \doteq e_{alg} + e_{in}. \end{aligned}$$

□

Approssimazione al prim'ordine dell'errore inerente Supponiamo che f sia derivabile due volte, e ricordiamo che abbiamo $\tilde{x} = x(1 + \varepsilon)$. Allora,

$$f(\tilde{x}) = f(x) + f'(x)(\tilde{x} - x) + \frac{f''(\xi)}{2}(\tilde{x} - x)^2 = f(x) + f'(x)\varepsilon x + \frac{f''(\xi)}{2}\varepsilon^2 x^2$$

da cui

$$\frac{f(\tilde{x}) - f(x)}{f(x)} = \frac{f'(x)x}{f(x)}\varepsilon + \frac{f''(\xi)}{2f(x)}\varepsilon^2 x^2 \doteq \frac{f'(x)x}{f(x)}\varepsilon.$$

La quantità

$$\kappa_{f,x} = \left|\frac{f'(x)x}{f(x)}\right|$$

è detta *numero di condizionamento* della funzione f nel punto x. Mostra di quanto un piccolo errore su x viene "amplificato" dal calcolo di f.

Esempio La funzione $f(x) = \frac{x}{1-x}$ ha numero di condizionamento

$$\kappa_{f,x} = \left|\frac{f'(x)x}{f(x)}\right| = \left|\frac{\frac{1}{(1-x)^2}x}{\frac{x}{1-x}}\right| = \frac{1}{|1-x|}.$$

Questa quantità è grande quando $x \approx 1$. Un errore relativo piccolo, per esempio $x = 0.999$, $\tilde{x} = 0.9991$, causa un errore relativo grande $\frac{f(\tilde{x})-f(x)}{f(x)}$.

Condizionamento delle quattro operazioni Possiamo studiare il condizionamento delle quattro operazioni rispetto a perturbazioni dei dati in ingresso. Notiamo che le operazioni dipendono da due valori in ingresso, quindi ha senso calcolare *due* numeri di condizionamento (che in realtà spesso saranno uguali, per simmetria). Per esempio, $f(x,y) = x \cdot y$; cosa succede perturbando x in $x(1+\varepsilon_x)$?

$$\kappa_{f,x} = \frac{\frac{\partial f(x,y)}{\partial x} x}{f(x)} = \frac{y \cdot x}{xy} = 1.$$

Quindi la moltiplicazione è un'operazione ben condizionata.

Possiamo fare un calcolo simile per $f(x,y) = x + y$. (Questo include anche la differenza, perché $x - y = x + (-y)$.)

$$\kappa_{f,x} = \frac{\frac{\partial f(x,y)}{\partial x} x}{f(x)} = \frac{1 \cdot x}{x+y} = \frac{x}{x+y}.$$

Questo errore diventa molto grande quando la somma $x+y$ è molto più piccola (in valore assoluto) degli addendi x e y. In questo caso, piccole perturbazioni (relative) agli addendi x, y possono portare a una perturbazione molto grande (relativa) sulla somma $x+y$.

Esempio: $x = 1.001$, $y = -1$. $\tilde{x} = 1.0011$. La perturbazione relativa sul dato in ingresso è $\frac{\tilde{x}-x}{x} \approx 10^{-4}$, ma la perturbazione relativa sulla differenza è mille volte più grande,

$$\frac{f(\tilde{x},y) - f(x,y)}{f(x,y)} \approx 10^{-1}.$$

E per $f(x,y) = x/y \to$ sempre ben condizionata.

Approssimazione al prim'ordine dell'errore algoritmico Data un'espressione, per esempio $z = f(x,y) = x^2 - y^2$ (anche con più di un "valore di ingresso"), possiamo stimarne l'errore algoritmico. Dati numeri di macchina $\tilde{x}, \tilde{y}$, possiamo scrivere

$$\begin{aligned} h(\tilde{x},\tilde{y}) &= \tilde{x} \otimes \tilde{x} \ominus \tilde{y} \otimes \tilde{y} = \tilde{x}^2(1+\varepsilon_1) \ominus \tilde{y}^2(1+\varepsilon_2) \\ &= \left(\tilde{x}^2(1+\varepsilon_1) - \tilde{y}^2(1+\varepsilon_2)\right)(1+\varepsilon_3) \\ &\doteq \underbrace{\tilde{x}^2 - \tilde{y}^2}_{\text{valore esatto } f(\tilde{x},\tilde{y})} + \underbrace{\tilde{x}^2\varepsilon_1 - \tilde{y}^2\varepsilon_2 + (\tilde{x}^2 - \tilde{y}^2)\varepsilon_3}_{\text{errore algoritmico}}. \end{aligned}$$

Nell'ultima parentesi abbiamo omesso tutte le quantità "del secondo ordine", vale a dire che contengono il prodotto di due (o più) ε_i: visto che sappiamo che $|\varepsilon_i| \leq \mathsf{u}$ per $i = 1,2,3$, queste quantità sono dell'ordine di u^2 e sono presumibilmente molto più piccole di tutti gli altri numeri coinvolti.

Visto che tutto quello che sappiamo è che $|\varepsilon_i| \leq \mathsf{u}$ per $i = 3,4,5$, l'unica stima che possiamo fare è

$$\frac{|h(\tilde{x},\tilde{y}) - f(\tilde{x},\tilde{y})|}{|f(\tilde{x},\tilde{y})|} = \frac{|\tilde{x}^2\varepsilon_1 - \tilde{y}^2\varepsilon_2 + (\tilde{x}^2 - \tilde{y}^2)\varepsilon_3|}{|\tilde{x}^2 - \tilde{y}^2|} \leq \frac{x^2}{|\tilde{x}^2 - \tilde{y}^2|}\mathsf{u} + \frac{y^2}{|\tilde{x}^2 - \tilde{y}^2|}\mathsf{u} + \mathsf{u}.$$

Un calcolo analogo si può impostare per sequenze più lunghe di operazioni: l'idea è sempre espandere le definizioni, eliminare tutti i termini con più di un

ε_i, e stimare l'errore tramite valori assoluti. Per algoritmi più complessi questo calcolo diventa presto proibitivo, purtroppo.

Notare che l'errore algoritmico dipende non dalla funzione in sé, ma dalla sequenza di operazioni che usiamo per calcolarlo: per esempio $f(x,y) = (x+y)(x-y)$ è la stessa funzione matematica, ma $h_2(\tilde{x},\tilde{y}) = (\tilde{x} \oplus \tilde{y}) \otimes (\tilde{x} \ominus \tilde{y})$ porta a un calcolo dell'errore algoritmico che a priori può essere completamente diverso.

Un'altra osservazione è che gli algoritmi che causano problemi solitamente sono quelli che contengono differenze tra due valori molto vicini: in questo caso per esempio i termini $\frac{x^2}{|x^2-y^2|}\mathsf{u}$, $\frac{y^2}{|x^2-y^2|}\mathsf{u}$ possono essere molto grandi, se il denominatore è molto minore del numeratore.

Errore analitico Anche supponendo di avere valori di x, y esatti e ignorando gli errori dovuti all'aritmetica di macchina, spesso i nostri algoritmi non restituiscono la soluzione esatta, ma solo una sua approssimazione. Per esempio, se vogliamo calcolare $\sqrt{2}$ tramite il metodo di punto fisso con una certa funzione $\Phi(x)$ (che supponiamo razionale, cioè ottenuta solo componendo le quattro operazioni), e ci arrestiamo dopo 3 passi, la funzione che calcoliamo in aritmetica esatta non è $f(x) = \sqrt{2}$, bensì $g(x) = \Phi(\Phi(\Phi(x_1)))$.

Il numero $\sqrt{2}$ in generale non è razionale, quindi non abbiamo speranza di calcolarlo esattamente con solo le quattro operazioni.

In generale, supponiamo di voler calcolare a partire da un valore in ingresso $\tilde{x}$ (già perturbato da errore in ingresso) una funzione $f(\tilde{x})$ (questa volta non per forza una funzione razionale), e di avere un algoritmo che calcola una sua approssimazione $g(\tilde{x})$; per esempio $f(x) = \sqrt{x}$ e $g(x) = \Phi(\Phi(\Phi(1)))$, dove $\Phi(t)$ è la funzione $\Phi(t) = \frac{1}{2}(t + \frac{x}{t})$ (quella risultante dal metodo di Newton). Possiamo definire allo stesso modo un *errore analitico*

$$e_{an} = \frac{g(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}.$$

Il computer questa volta calcolerà una versione modificata $h(x)$ di $g(x)$, anziché di $f(x)$, ottenuta con operazioni di macchina. Con una dimostrazione analoga a quella sopra ma con tre termini anziché due, possiamo dimostrare che

$$e_{tot} = \frac{h(\tilde{x}) - f(x)}{f(x)} \doteq e_{in} + e_{an} + e_{alg}.$$

Gli errori e_{in} ed e_{alg} sono dovuti all'aritmetica di macchina (almeno assumendo che non ci siano altre perturbazioni sui dati in ingresso), e quindi possiamo aspettarci (in molti casi!) che siano un qualche multiplo di u: il fattore che limita l'accuratezza che possiamo ottenere è l'accuratezza dell'aritmetica di macchina. Invece e_{an} va studiato algoritmo per algoritmo, e potrebbe essere anche molto più grande, e quindi dominare la somma. Abbiamo già studiato l'errore analitico dei metodi di ricerca di zeri, dicendo che converge a zero con il numero di passi e dandone delle stime. Allo stesso modo analizzeremo l'errore analitico di vari algoritmi che studieremo.

In un problema più facile (che si risolve solo con le quattro operazioni) come risolvere un sistema di equazioni lineari, abbiamo formule esatte e quindi $e_{an} = 0$. In problemi più difficili come risolvere equazioni differenziali o calcolare integrali, e_{an} è non-zero, e spesso è più grande degli altri errori, quindi è la componente più importante.

Capitolo 5

Equazioni lineari e autovalori

5.1 Richiami di algebra lineare

Prodotto matrice-vettore $A\mathbf{x}$: definito come $(Ax)_i = \sum_{j=1}^{n} A_{ij}x_j$ (riga per colonna); più geometricamente, crea una combinazione lineare $\mathbf{v}_1 x_1 + \mathbf{v}_2 x_2 + \cdots + \mathbf{v}_n x_n$) delle colonne $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ di A.

Un *sistema lineare* è il problema inverso del prodotto: data $A \in \mathbb{R}^{n\times n}$ e $\mathbf{b} \in \mathbb{R}^n$, trovare il vettore di coefficienti $\mathbf{x}$ che servono per scrivere $\mathbf{b}$ come combinazione lineare delle colonne di A.

Partiamo studiando sistemi quadrati, cioè $A \in \mathbb{R}^{n\times n}$ (o $\mathbb{C}^{n\times n}$). Un sistema ha *una e una sola* soluzione quando le righe/colonne di A formano una base di $\mathbb{R}^n$ (in particolare, quando sono linearmente indipendenti). Ci concentriamo su sistemi che soddisfano questa ipotesi. Dall'algebra lineare, sappiamo che se A è invertibile esiste una matrice $A^{-1} \in \mathbb{R}^{n\times n}$ tale che la soluzione si scrive come prodotto $\mathbf{x} = A^{-1}\mathbf{b}$. Questa matrice è unica e soddisfa $A^{-1}A = AA^{-1} = I$, la matrice con uni sulla diagonale e zeri altrove.

Non possiamo scrivere $\mathbf{x} = \mathbf{b}A^{-1}$ (dimensioni non compatibili per il prodotto; l'ordine dei fattori conta!), né $\mathbf{x} = \frac{\mathbf{b}}{A}$ (non vuol dire nulla, e non mi specifica l'ordine!).

In Matlab, esiste una funzione `inv(A)` che calcola l'inversa, quindi potremmo scrivere `inv(A) * b`; però questo metodo è più costoso (e spesso anche più inaccurato) di altri algoritmi. C'è una notazione diversa per risolvere un sistema, `x = A \ b`. Occhio: la barra è la "backslash". Per non confondere le barre, pensate a questa operazione come a una sorta di "divisione da un lato": c'è una barra di frazione, e la `A` sta al di sotto.

Intuitivamente, calcolare un'inversa richiede risolvere gli n sistemi lineari $A^{-1}e_1, A^{-1}e_2, \ldots A^{-1}e_n$; per questo è più lento.

Determinanti Altri algoritmi da evitare sono quelli basati su determinanti, come il *metodo di Cramer*. Il *determinante* è una particolare funzione di una matrice quadrata; le proprietà che ci interessa ricordare qui sono queste:

- Una matrice è invertibile se e solo se $\det(A) \neq 0$.

- Se una matrice è *triangolare*, il suo determinante è il prodotto degli elementi sulla diagonale.
- $\det(AB) = \det(A)\det(B)$ (teorema di Binet).

Tipicamente i determinanti sono più lenti da calcolare, e per matrici grossine vanno spesso in overflow/underflow; per esempio, $\det(10I_{400\times 400})$. In questa sezione vedremo invece gli algoritmi "migliori" che Matlab utilizza per risolvere sistemi lineari.

5.2 Condizionamento della soluzione di sistemi lineari

Partiamo studiando il condizionamento della soluzione di sistemi lineari. Per farlo, introduciamo alcuni strumenti teorici.

Norme vettoriali Notate che l'analisi dell'errore che avevamo fatto assumeva di avere funzioni $f : \mathbb{R} \to \mathbb{R}$, funzioni $y = f(x)$ di un reale x. La soluzione di un sistema lineare invece è una funzione $x = f(A, b)$ che prende una matrice e un vettore e restituisce un vettore. Potremmo studiare separatamente il condizionamento di ogni componente rispetto a ogni componente dell'input, ma si preferisce un approccio diverso che passa attraverso il misurare "distanze" tra vettori e matrici.

Lo strumento teorico che ci serve è una *norma vettoriale*, cioè una funzione che "assomiglia al valore assoluto" per vettori.

Si definisce *norma vettoriale* una funzione $f : \mathbb{C}^n \to \mathbb{R}$ che ha queste proprietà.

1. $f(v) \geq 0$ per ogni vettore $v \in \mathbb{C}^n$, e l'uguaglianza vale solo per il vettore zero.
2. $f(\alpha v) = |\alpha| f(v)$ per ogni vettore $v \in \mathbb{C}^n$ e scalare $\alpha \in \mathbb{C}$.
3. $f(v + w) \leq f(v) + f(w)$ per ogni $v, w \in \mathbb{C}^n$.

Notate che queste rispecchiano le proprietà del valore assoluto; per esempio l'ultima è la disuguaglianza triangolare. Difatti non è complicato verificare che il valore assoluto è una norma per $n = 1$.

Una norma di solito non si indica con $f(v)$, ma con $\|v\|$ (due stanghette).

Le norme più usate sono le seguenti.

- Norma-1: $\|v\|_1 = \sum_{i=1}^{n} |v_i|$.
- Norma-2 (o Euclidea): $\|v\|_2 = \sqrt{\sum_{i=1}^{n} |v_i|^2} = \sqrt{v^* v}$.
- Norma infinito: $\|v\|_\infty = \max_{i \in \{1,2,\ldots,n\}} |v_i|$.

Si dimostra che tutte e tre soddisfano le proprietà qui sopra. L'unica un po' più difficile è la disuguaglianza triangolare per la norma-2; le altre potete provare a farle come esercizio.

Esempio: calcola le tre norme di $\begin{bmatrix} 2 \\ -3 \\ -1 \end{bmatrix}$.

Data una norma, $\|v - w\|$ fornisce un modo di misurare la distanza tra v e w. Ogni norma fornisce valori diversi, che danno modi leggermente diversi di definire questa distanza. In generale, si dimostra che date due norme qualunque queste differiscono per una costante; vale a dire, esistono due reali $c_1, c_2 > 0$ tali che *per ogni* vettore $v \in \mathbb{C}^n$ si ha

$$c_1\|v\|_\alpha \le \|v\|_\beta \le c_2\|v\|_\alpha.$$

Queste costanti spesso sono una funzione della dimensione; per esempio, per ogni $v \in \mathbb{C}^n$ si ha

$$\|v\|_0 \le \|v\|_2 \le \sqrt{n}\|v\|_0.$$

(esercizio: dimostrarlo.)

Esempio: disegnare le "sfere" $\|v\| = 1$ in $\mathbb{R}^2$ nelle tre norme.

Norme matriciali Similmente ai vettori, possiamo definire norme su matrici. Si dice *norma matriciale* una funzione $f : \mathbb{C}^{n\times n} \to \mathbb{R}$ che soddisfa queste proprietà:

1. $f(A) \ge 0$ per ogni matrice $A \in \mathbb{C}^{n\times n}$, e l'uguaglianza vale solo per la matrice zero.
2. $f(\alpha A) = |\alpha| f(A)$ per ogni $A \in \mathbb{C}^{n\times n}$ e scalare $\alpha \in \mathbb{C}$.
3. $f(A + B) \le f(A) + f(B)$ per ogni $A, B \in \mathbb{C}^{n\times n}$.
4. $f(AB) \le f(A)f(B)$ per ogni $A, B \in \mathbb{C}^{n\times n}$.

Rispetto al caso dei vettori, abbiamo aggiunto una proprietà che lega la norma al prodotto di matrici. Notate stavolta una differenza rispetto al valore assoluto; non abbiamo $\|AB\| = \|A\|\|B\|$. (Sarebbe impossibile ottenere norme con questa proprietà più forte.)

Norme matriciali indotte È possibile costruire una norma matriciale a partire da ogni norma vettoriale in questo modo. Fissata una norma vettoriale $\|\cdot\|$ (per esempio le norme $1, 2, \infty$) definiamo

$$\|A\|_p = \max_{\|v\|_p=1} \|Av\|_p. \tag{5.1}$$

In generale la matrice A manderà gli infiniti vettori con norma uguale a 1 in vettori di lunghezza diversa; prendiamo il più lungo, e definiamolo come la norma. Si può dimostrare (non lo faremo) che questa definizione soddisfa tutte le proprietà di una norma matriciale.

La definizione fatta in questo modo serve per assicurare un'ulteriore proprietà.

Teorema 5.1. *La norma matriciale $\|A\|_p$ definita qui sopra soddisfa $\|Av\|_p \le \|A\|_p\|v\|_p$ per ogni matrice $A \in \mathbb{C}^{n\times n}$ e vettore $v \in \mathbb{C}^n$.*

Notare che mi serve usare *la stessa* norma: per esempio se misuro i vettori in una norma p con $p \in \{1, 2, \infty\}$, dovrò usare la norma matriciale costruita usando la norma p nella (5.1).

Dimostrazione. Prima un caso particolare: se $v = 0$, allora anche $Av = 0$ e sia il membro di sinistra che quello di destra si annullano. Possiamo quindi proseguire considerando $v \neq 0$, e quindi $\|v\| \neq 0$.

Mi basta considerare il vettore $u = \frac{1}{\|v\|}v$. Questo vettore ha norma uguale a 1, per le proprietà delle norme (posso "portare fuori" lo scalare $\frac{1}{\|v\|}$); quindi

$$\frac{1}{\|v\|}\|Av\| = \|Au\| \leq \|A\|,$$

ed eliminando il denominatore otteniamo la tesi. □

Norma di Frobenius Non tutte le norme matriciali si ottengono da questa costruzione. Un altro esempio è la *norma di Frobenius*,

$$\|A\|_F = \sqrt{\sum_{i,j=1}^{n} |A_{ij}|^2}.$$

Questa funzione soddisfa tutte le proprietà di una norma matriciale, ma non è una norma matriciale *indotta*. Un modo veloce di vederlo è considerando la norma della matrice identità: $\|I\|_F = \sqrt{n}$, ma per una norma matriciale indotta segue dalla definizione che $\|I\| = 1$.

Norme, autovalori e raggio spettrale Data una matrice quadrata A, ricordiamo che quando $Av = v\lambda$ per un qualche vettore $v \in \mathbb{C}^n$ (diverso dal vettore nullo!) e scalare $\lambda \in \mathbb{C}$ si dice che λ è un *autovalore* e v è un *autovettore* di A. Avete visto ad algebra lineare diverse proprietà degli autovalori. Prendendo norme, abbiamo che

$$\|v\||\lambda| = \|v\lambda\| = \|Av\| \leq \|A\|\|v\|.$$

Possiamo semplificare $\|v\| \neq 0$, quindi $|\lambda| \leq \|A\|$ per ogni autovalore.

Data una matrice $A \in \mathbb{C}^{n\times n}$ (quindi gli autovalori esistono sempre), si chiama *spettro* l'insieme dei suoi autovalori, e *raggio spettrale* (e si indica $\rho(A)$) il valore assoluto più grande degli autovalori, $\rho(A) = \max_{\lambda \text{ autoval.}} |\lambda|$. Notate che questo non è per forza un autovalore; per esempio potremmo avere A con autovalori $\{-2, i, -i\}$, e quindi $\rho(A) = 2$ non è un autovalore.

In ogni caso, dalla formula qui sopra segue

$$\rho(A) \leq \|A\|$$

per ogni norma matriciale indotta.

Formule per le norme matriciali $1, 2, \infty$ È abbastanza scomodo calcolare le norme matriciali indotte usando la loro definizione (c'è un massimo su un insieme infinito di vettori...). Per le norme $1, 2, \infty$ ci sono delle formule più

semplici. Le enunciamo senza dimostrazione.

$$\|A\|_\infty = \max_{i=1}^{n} \sum_{j=1}^{n} |A_{ij}|,$$
$$\|A\|_1 = \max_{j=1}^{n} \sum_{i=1}^{n} |A_{ij}|,$$
$$\|A\|_2 = \rho(A^T A)^{1/2}.$$

(Qui $\rho(\cdot)$ è di nuovo il raggio spettrale.)

Esempio: calcolare le tre norme (quattro con Frobenius) sulla matrice

$$A = \begin{bmatrix} -2 & -1 \\ -2 & 1 \end{bmatrix},$$

e verificare (aiutandosi eventualmente con Matlab) che $\rho(A) \leq \|A\|_p$ per $p = 1, 2, \infty$. (La disuguaglianza è vera anche per $\|A\|_F$, ma la dimostrazione che abbiamo fatto funziona solo per norme matriciali indotte.)

Condizionamento della soluzione di sistemi lineari La soluzione di sistemi lineari è un problema che ha come "input" A, b e come "output" x. Ha senso chiederci come cambia x se perturbiamo A oppure b (o anche tutti e due insieme). Analogamente al caso scalare, possiamo definire errori relativi in termini di norme: $\frac{\|\tilde{x}-x\|}{\|x\|}$, $\frac{\|\tilde{A}-A\|}{\|A\|}$, $\frac{\|\tilde{b}-b\|}{\|b\|}$ (occhio che $\|\frac{\tilde{b}-b}{b}\|$ non vuol dire niente, non possiamo dividere per vettori!)

Qui vediamo il caso più semplice, cosa succede quando perturbiamo il vettore dei termini noti b. Supponiamo di avere un sistema lineare con A invertibile e $b \neq 0$, e di avere un vettore $\tilde{b} = b + f$ che è una perturbazione di b (con $b, f \in \mathbb{R}^n$). Siano x e $\tilde{x}$ rispettivamente le soluzioni di $Ax = b$ e $A\tilde{x} = \tilde{b} = b + f$. Possiamo calcolare

$$\|\tilde{x} - x\| = \|A^{-1}(b + f) - A^{-1}b\| = \|A^{-1}f\| \leq \|A^{-1}\|\|f\| = \|A^{-1}\|\|\tilde{b} - b\|,$$

se usiamo una norma vettoriale e la corrispondente norma matriciale indotta. Similmente abbiamo

$$\|b\| = \|Ax\| \leq \|A\|\|x\|.$$

Combinando le due disuguaglianze (occhio ai versi!) abbiamo

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|\tilde{b} - b\|}{\|b\|}. \tag{5.2}$$

Questa disuguaglianza è valida non solo "al prim'ordine", ma per tutti gli A, x, b. La quantità $\kappa(A) = \|A\|\|A^{-1}\|$ si definisce "numero di condizionamento" della matrice A (con un piccolo abuso di notazione: condizionamento di una matrice $\neq$ condizionamento di un problema).

La quantità $\kappa(A)$ è sempre maggiore di 1, perché per ogni norma matriciale indotta $1 = \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\|$. Una matrice si dice "ben condizionata" se questa quantità è vicina a 1, e "mal condizionata" se è molto maggiore di 1 (qualche migliaio almeno, di solito; non c'è una soglia precisa).

Così per conoscenza, si può dimostrare che $\kappa(A)$ limita anche l'errore inerente dovuto a perturbazioni della matrice A:

$$\frac{\|\tilde{x} - x\|}{\|x\|} \dot{\leq} \|A\|\|A^{-1}\| \frac{\|\tilde{A} - A\|}{\|A\|}.$$

Occhio al punto sopra il $\leq$: a differenza del precedente, in questo caso si tratta di un risultato solo a meno di termini dell'ordine del quadrato di $\frac{\|\tilde{A}-A\|}{\|A\|}$: se $\frac{\|\tilde{A}-A\|}{\|A\|}$ è grande, questa disuguaglianza può essere ben lontana dall'essere verificata.

5.3 Condizionamento del calcolo di autovalori (*)

Un altro problema classico dell'algebra lineare è il calcolo di autovalori e autovettori. Non vediamo nel dettaglio algoritmi per farlo, perché sono molto più tecnici del resto del corso; ci fideremo di `eig(A)` di Matlab. Però, almeno per studiare il problema teoricamente, è importante vedere quale è il condizionamento di questa operazione, in modo da sapere quando anche Matlab rischia di calcolare un risultato errato.

Ci limitiamo a un caso più facile, quello di *autovalori semplici*: un autovalore si dice *semplice* se la sua molteplicità geometrica e algebrica è uguale a 1, cioè, se è uno zero semplice del polinomio caratteristico $\det(\lambda I - A)$. In questo caso, esistono e sono unici (a meno di multipli) un autovalore destro $\mathbf{x}$ e uno sinistro $\mathbf{y}^*$ associati a λ; cioè, $A\mathbf{x} = \mathbf{x}\lambda$ e $\mathbf{y}^*A = \lambda\mathbf{y}^*$. Si può dimostrare che per un autovalore semplice $\mathbf{x}$ e $\mathbf{y}^*$ non sono mai ortogonali. Il risultato di perturbazione che enunciamo dipende dal coseno dell'angolo che essi formano, cioè

$$\cos\theta = \frac{|\mathbf{y}^*\mathbf{x}|}{\|\mathbf{y}\|_2\|\mathbf{x}\|_2}.$$

Teorema 5.2 (Perturbazione di autovalori (*))**.** *Sia λ un autovalore* semplice *della matrice $A \in \mathbb{C}^{n\times n}$. Sia $\tilde{A}$ una perturbazione di A; allora esiste un autovalore $\tilde{\lambda}$ di $\tilde{A}$ tale che*

$$|\tilde{\lambda} - \lambda| \dot{\leq} \frac{1}{\cos\theta}\|\tilde{A} - A\|_2,$$

dove θ è l'angolo tra l'autovettore destro $\mathbf{x}$ e quello sinistro $\mathbf{y}^*$ associati a λ.

Occhio al punto sopra il $\leq$: anche in questo caso il risultato è valido solo al prim'ordine, cioè ignorando termini dell'ordine di $\|\tilde{A} - A\|_2^2$.

In particolare, quando A è una matrice simmetrica gli autovalori destri e sinistri coincidono, cioè $\mathbf{x} = \mathbf{y}$; quindi il coseno è uguale a 1 e il calcolo degli autovalori di una matrice simmetrica è sempre un'operazione ben condizionata.

Dimostrazione. Scriviamo $\tilde{A} - A = \varepsilon E$, dove $\|E\|_2 = 1$ e $\varepsilon = \|\tilde{A} - A\|_2$. Consideriamo la funzione $A(t) = A + tE$. È possibile dimostrare usando il teorema della funzione implicita che esistono funzioni differenziabili $\lambda(t), \mathbf{x}(t)$ tali che $\lambda(t)$ è un autovalore di $A(t)$ e $\mathbf{x}(t)$ è il suo autovettore, e tali che $\lambda(0) = \lambda$, $\mathbf{x}(0) = \mathbf{x}$. Quindi, in particolare, vale la relazione

$$A(t)\mathbf{x}(t) = \mathbf{x}(t)\lambda(t).$$

Possiamo derivare entrambi i termini rispetto a t, per ottenere

$$\underbrace{\dot{A}(t)}_{=E}\mathbf{x}(t) + A(t)\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(t)\lambda(t) + \mathbf{x}(t)\dot{\lambda}(t).$$

Valutiamo in $t = 0$, e otteniamo

$$E\mathbf{x} + A\dot{\mathbf{x}}(0) = \dot{\mathbf{x}}(0)\lambda + \mathbf{x}\dot{\lambda}(0)$$

Moltiplichiamo a sinistra per $\mathbf{y}^*$, e usiamo il fatto che $\mathbf{y}^*A = \lambda\mathbf{y}^*$ per semplificare due termini della relazione

$$\mathbf{y}^*E\mathbf{x} + \mathbf{y}^*A\dot{\mathbf{x}}(0) = \lambda\mathbf{y}^*\dot{\mathbf{x}}(0) + \mathbf{y}^*\mathbf{x}\dot{\lambda}(0).$$

Quindi otteniamo

$$\dot{\lambda}(0) = \frac{1}{\mathbf{y}^*\mathbf{x}}\mathbf{y}^*E\mathbf{x}.$$

Possiamo scrivere, a meno di termini di ordine superiore in ε, lo sviluppo di Taylor

$$\tilde{\lambda} = \lambda(\varepsilon) \doteq \lambda + \varepsilon\dot{\lambda}(0) = \lambda + \varepsilon\frac{1}{\mathbf{y}^*\mathbf{x}}\mathbf{y}^*E\mathbf{x} \tag{5.3}$$

da cui

$$|\tilde{\lambda} - \lambda| = \varepsilon\frac{1}{\mathbf{y}^*\mathbf{x}}\mathbf{y}^*E\mathbf{x}$$

Per concludere dimostriamo che $|\mathbf{y}^*E\mathbf{x}| \leq \|\mathbf{y}\|_2\|\mathbf{x}\|_2$: usiamo prima la disuguaglianza di Cauchy-Schwarz e poi la definizione di norma-2 per avere

$$|\mathbf{y}^*E\mathbf{x}| \leq \|\mathbf{y}\|_2\|E\mathbf{x}\|_2 \leq \|\mathbf{y}\|_2\underbrace{\|E\|_2}_{=1}\|\mathbf{x}\|_2.$$

□

Nel caso di autovalori non semplici, le perturbazioni agli autovalori possono essere molto più grandi. Per esempio, è possibile dimostrare che la matrice $n \times n$ che ha uni sulla sopradiagonale, ε in posizione $(n, 1)$, e zero in tutte le altre posizioni

$$A(\varepsilon) = \begin{bmatrix} 0 & 1 & & \\ & 0 & \ddots & \\ & & \ddots & 1 \\ \varepsilon & & & 0 \end{bmatrix}$$

ha come autovalori le radici n-esime complesse di ε, che hanno tutte modulo $(\varepsilon)^{1/n}$. Quindi

$$|\tilde{\lambda} - \lambda| = (\varepsilon)^{1/n},$$

che è molto più grande di ε. Per esempio, se $\varepsilon = 10^{-16}$, $n = 8$, abbiamo $\varepsilon^{1/n} = 10^{-2}$: una perturbazione alla matrice $A(0)$ di norma 10^{-16} fa spostare gli autovalori di una distanza 10^{-2}.

5.4 Teorema dei cerchi di Gershgorin (*)

È possibile ottenere anche un risultato di inclusione per gli autovalori, cioè un risultato teorico che ci dice in quali regioni del piano complesso si possono trovare gli autovalori di una matrice. Per introdurlo, definiamo i *cerchi di Gershgorin* di una matrice $A \in \mathbb{C}^{n \times n}$ come gli insiemi

$$K_i = \left\{ z \in \mathbb{C} \colon |z - A_{ii}| \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |A_{ij}| \right\}, \quad i = 1, 2, \ldots, n.$$

Le equazioni definiscono l'interno di n cerchi, che hanno centro negli elementi diagonali di A e raggio uguale alla somma dei moduli degli elementi al di fuori della riga.

Teorema 5.3 (Teorema dei cerchi di Gershgorin)**.** *Sia $A \in \mathbb{C}^{n \times n}$, e λ un suo autovalore. Allora λ appartiene all'unione dei cerchi K_i definiti sopra (al variare di $i = 1, 2, \ldots, n$).*

Dimostrazione. Prendiamo un autovettore $\mathbf{x}$ associato a λ. Scrivendo componente per componente la relazione $A\mathbf{x} = \mathbf{x}\lambda$ otteniamo

$$\sum_{j=1}^{n} A_{ij} x_j = x_i \lambda, \quad i = 1, 2, \ldots, n.$$

Riarrangiando termini e prendendo i moduli otteniamo

$$|\lambda - A_{ii}||x_i| = |x_i \lambda - A_{ii} x_i| = \left| \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij} x_j \right| \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |A_{ij}||x_j|.$$

Questa uguaglianza è valida per ogni $i = 1, 2, \ldots, n$. Scegliamo ora come i l'indice p (o uno degli indici) tale che il modulo $|x_p|$ sia massimo. Poiché $\mathbf{x} \neq 0$, avremo $|x_p| \neq 0$. Possiamo quindi riscrivere l'uguaglianza precedente per $i = p$ e dividere per $|x_p|$ entrambi i lati, ottenendo

$$|\lambda - A_{pp}| \leq \sum_{\substack{j=1 \\ j \neq p}}^{n} |A_{pj}| \frac{|x_j|}{|x_p|} \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |A_{pj}|.$$

L'ultima disuguaglianza segue dal fatto che $|x_p|$ è massimo. Questa catena di disuguaglianze dimostra che λ sta all'interno del cerchio K_p. Poiché λ sta all'interno di almeno un cerchio, sta anche all'interno della loro unione. □

5.5 Soluzione di equazioni lineari: casi speciali

Sistemi diagonali Il caso più semplice è quello di A *diagonale*, vale a dire

$$A = \begin{bmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{nn} \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

In questo caso i prodotti sono semplici da fare, e $Ax = b$ implica

$$x_i = \frac{b_i}{A_{ii}}, \quad i = 1, 2, \ldots, n.$$

Il costo chiaramente è di n operazioni aritmetiche.

Sistemi triangolari Un caso particolare è quello in cui la matrice è *triangolare inferiore*, cioè contiene zeri al di sopra della diagonale ($A_{ij} = 0$ se $i < j$), oppure è *triangolare superiore* ($A_{ij} = 0$ se $i > j$).

(Una matrice può essere al tempo stesso *triangolare* e *quadrata*!)

Ricordiamo che una matrice triangolare è invertibile se gli elementi sulla sua diagonale sono tutti diversi da zero. Per dimostrarlo, per esempio notiamo che una matrice è invertibile se tutti i suoi autovalori sono diversi da zero, e che gli autovalori di una matrice triangolare sono uguali agli elementi sulla diagonale.

Se la matrice è triangolare inferiore, possiamo risolvere il sistema per *sostituzione in avanti* partendo dalla prima equazione, che contiene solo la prima incognita:

$$\begin{aligned}
x_1 &= \frac{b_1}{A_{11}}, \\
x_2 &= \frac{b_2 - A_{21}x_1}{A_{22}}, \\
&\vdots \qquad \vdots \\
x_n &= \frac{b_n - A_{n1}x_1 - A_{n2}x_2 - \cdots - A_{n,n-1}x_{n-1}}{A_{nn}}.
\end{aligned}$$

Notare che ad ogni passo compaiono a destra dell'uguale solo valori x_i che abbiamo già calcolato nei passi precedenti. Possiamo contare il numero di operazioni in ogni riga, ottenendo

$$1 + 3 + 5 + \cdots + (2n - 1) = n^2$$

(questa ultima uguaglianza si può dimostrare per induzione).

Diciamo che la complessità del metodo è di n^2 operazioni aritmetiche. Notare che in questo caso non ci sono funzioni sconosciute da valutare, quindi questa è davvero tutta la complessità. Inoltre queste operazioni calcolano esattamente la soluzione, quindi quello che abbiamo chiamato "errore analitico" è zero.

Spesso quando si parla di complessità ci interessa solo l'ordine di grandezza, e quindi si scrive $O(n^2)$. Come in analisi, questa è una notazione che include complessità come $2n^2$, $\frac{1}{3}n^2 + 2n - 5$, ecc. Si intende che lavoriamo nel limite $n \to \infty$, quindi ignoriamo potenze *inferiori* della n che crescono più lentamente di n^2.

Per una matrice triangolare superiore, la tecnica è analoga, ma dobbiamo partire dall'ultima equazione e risolvere "andando al contrario" (*sostituzione*

all'indietro):

$$\begin{aligned} x_n &= \frac{b_n}{A_{nn}}, \\ x_{n-1} &= \frac{b_{n-1} - A_{n-1,n}x_n}{A_{n-1,n-1}}, \\ &\vdots \qquad\quad \vdots \\ x_1 &= \frac{b_1 - A_{1,n}x_n - A_{1,n-1}x_{n-1} - \cdots - A_{1,2}x_2}{A_{11}}. \end{aligned}$$

La complessità è di nuovo n^2 operazioni.

Se una matrice triangolare ha molti elementi uguali a zero (si dice che è *sparsa*),

5.6 Eliminazione di Gauss e fattorizzazione LU

Si chiama *fattorizzazione* LU di A un modo di scrivere $A = LU$, dove L è triangolare inferiore con 1 sulla diagonale, e U è triangolare superiore. Vediamo ora un metodo di calcolare una fattorizzazione LU di una matrice $A \in \mathbb{C}^{n\times n}$; questo algoritmo è sostanzialmente una variante dell'eliminazione di Gauss che avete già visto ad algebra lineare.

Ricordiamo come funziona l'eliminazione di Gauss (senza considerare scambi di righe, per ora). Partiamo definendo $U_1 = A$, e vogliamo applicare ripetutamente delle trasformazioni in modo da generare una sequenza di matrici $U_2, U_3, \ldots, U_n$, dove l'ultima matrice U_n è triangolare superiore. Supponiamo di aver già fatto $k-1$ passi ed essere al $k$esimo. Abbiamo generato a partire da A una matrice $A_k \in \mathbb{C}^{n\times n}$ che è parzialmente in forma triangolare superiore: in ogni colonna $j < k$, gli elementi sotto la diagonale sono nulli.

$$U_k = \begin{bmatrix} * & \ldots & * & * & * & \ldots & * \\ 0 & \ddots & * & * & * & \ldots & * \\ \vdots & \ddots & * & * & * & \ldots & * \\ 0 & 0 & 0 & (U_k)_{kk} & * & \ldots & * \\ 0 & 0 & 0 & (U_k)_{k+1,k} & * & \ldots & * \\ 0 & 0 & 0 & \vdots & * & \ldots & * \\ 0 & 0 & 0 & (U_k)_{n,k} & * & \ldots & * \end{bmatrix}$$

(abbiamo chiamato $(A_k)_{ij}$ gli elementi della matrice A_k). Nel k-esimo passo,

introdurremo zeri anche nella k-esima colonna, ottenendo una nuova matrice

$$U_{k+1} = \begin{bmatrix} * & \dots & * & * & * & \dots & * \\ 0 & \ddots & * & * & * & \dots & * \\ \vdots & \ddots & * & * & * & \dots & * \\ 0 & 0 & 0 & (U_k)_{kk} & * & \dots & * \\ 0 & 0 & 0 & 0 & * & \dots & * \\ 0 & 0 & 0 & \vdots & * & \dots & * \\ 0 & 0 & 0 & 0 & * & \dots & * \end{bmatrix}.$$

Lo facciamo sottraendo ogni riga $i > k$ un opportuno multiplo della k-esima riga, in modo da eliminare l'elemento in posizione (i, k). Tale multiplo quindi dev'essere $\frac{(U_k)_{ik}}{(U_k)_{kk}}$. Le prime k righe rimangono invariate.

Matrici elementari di Gauss La parte che probabilmente non avete visto ad algebra lineare è che questa trasformazione si può vedere come una moltiplicazione a sinistra per una matrice: $U_{k+1} = E_k U_k$. La matrice E_k si può scrivere come

$$E_k = I - v_k e_k^T,$$

dove v_k è il vettore tale che

$$(v_k)_i = \begin{cases} 0 & i \leq k, \\ \frac{(U_k)_{ik}}{(U_k)_{kk}} & i > k, \end{cases}$$

e e_k è il k-esimo vettore della base canonica. La matrice E_k differisce dalla matrice identità solo per una sequenza di elementi diversi da zero nella k-esima colonna sotto la diagonale:

$$E_k = \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & -v_{k+1,k} & 1 & & \\ & & & -v_{k+2,k} & & 1 & \\ & & & \vdots & & & \ddots \\ & & & -v_{n,k} & & & 1 \end{bmatrix}.$$

Una matrice di questo tipo si chiama *matrice elementare di Gauss*: nel dettaglio, una matrice si chiama *matrice elementare di Gauss* se si scrive (per un qualche k) nella forma $I - v_k e_k^T$, dove $(v_k)_i = 0$ per ogni $i \leq k$.

Difatti, è semplice verificare che per ogni vettore $x \in \mathbb{R}^n$

$$E_k \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ x_{k+1} - v_{k+1,k} x_k \\ \vdots \\ x_n - v_{n,k} x_k \end{bmatrix}.$$

[TODO: fare un esempio 4×4 su Matlab a questo punto, per chiarire meglio il funzionamento prima di entrare in dettagli sulle matrici elementari.]

Dopo k passi dell'eliminazione di Gauss, possiamo scrivere la matrice ottenuta A_{k+1} come

$$U_{k+1} = E_k A_k = E_k E_{k-1} U_{k-1} = \cdots = E_k E_{k-1} \dots E_1 A_1 = E_k E_{k-1} \dots E_1 U_1.$$

Notiamo che le matrici E_k sono tutte invertibili: difatti sono triangolari inferiori con elementi uguali a 1 sulla diagonale. Moltiplicando a sinistra entrambi i lati per la matrice $E_1^{-1} E_2^{-1} \dots E_k^{-1}$ otteniamo la fattorizzazione

$$E_1^{-1} E_2^{-1} \dots E_k^{-1} U_{k+1} = U_1 = A.$$

Dimostriamo alcuni risultati che ci portano a un'espressione esplicita del prodotto di queste inverse.

Lemma 5.4 (Proprietà delle matrici elementari di Gauss)**.** *Valgono le seguenti due proprietà:*

1. $E_k^{-1} = I + v_k e_k^T$.
2. $E_1^{-1} E_2^{-1} \dots E_k^{-1} = I + v_1 e_1^T + v_2 e_2^T + \cdots + v_k e_k^T$.

Dimostrazione. 1. Basta verificare che

$$E_k(I + v_k e_k^T) = (I - v_k e_k^T)(I + v_k e_k^T) = I - v_k e_k^T + v_k e_k^T - v_k \underbrace{e_k^T v_k}_{=0} e_k^T = I.$$

Difatti il prodotto scalare $e_k^T v_k$ fa zero perché v_k ha uno zero in posizione k.

2. Possiamo verificarlo per induzione: il caso $k = 1$ l'abbiamo dimostrato al passo precedente. Supponendo che sia vero per un certo $k - 1$, si ha

$$E_1^{-1} E_2^{-1} \dots E_k^{-1} = (I + v_1 e_1^T + v_2 e_2^T + \cdots + v_{k-1} e_{k-1}^T)(I + v_k e_k^T).$$

Ora espandiamo le parentesi e utilizziamo il fatto che $e_i^T v_k = 0$ per $i < k$, vero perché v_k ha le prime componenti uguali a zero.

□

Questo mostra che per calcolare il prodotto $E_1^{-1} E_2^{-1} \dots E_k^{-1}$ non dobbiamo effettuare nessuna ulteriore operazione aritmetica: basta "scrivere" gli elementi v_{ik} nel punto giusto della matrice.

Dopo k passi dell'eliminazione di Gauss quindi vale l'identità

$$A = E_1^{-1} E_2^{-1} \dots E_k^{-1} U_{k+1}.$$

La matrice U_{k+1} ha zeri nella parte triangolare inferiore delle prime k colonne, e la matrice $E_1^{-1} E_2^{-1} \dots E_k^{-1}$ è triangolare inferiore e contiene elementi diversi

da 0 e 1 solo nelle prime k colonne:

$$A = \underbrace{\begin{bmatrix} 1 & & & & & & \\ v_{21} & \ddots & & & & & \\ \vdots & \vdots & 1 & & & & \\ \vdots & \vdots & v_{k+1,k} & 1 & & & \\ \vdots & \vdots & v_{k+2,k} & 0 & 1 & & \\ \vdots & \vdots & \vdots & 0 & 0 & \ddots & \\ v_{n1} & \dots & v_{n,k} & 0 & \dots & \dots & 1 \end{bmatrix}}_{E_1^{-1}E_2^{-1}\dots E_k^{-1}} \underbrace{\begin{bmatrix} U_{11} & U_{12} & \dots & & & & U_{1,n} \\ 0 & \ddots & \dots & \dots & \dots & \dots & \vdots \\ 0 & & U_{kk} & \dots & \dots & \dots & \vdots \\ 0 & \dots & 0 & U_{k+1,k+1} & \dots & \dots & U_{k+1,n} \\ 0 & \dots & 0 & U_{k+2,k+1} & & & \vdots \\ 0 & \dots & 0 & \vdots & & & \vdots \\ 0 & \dots & 0 & U_{n,k+1} & \dots & \dots & U_{n,n} \end{bmatrix}}_{U_{k+1}}.$$

Qui abbiamo indicato con v_{ik} l'entrata i-esima del vettore v_k.

Supponiamo per ora di poter effettuare l'eliminazione di Gauss senza mai incontrare divisioni per zero sulla diagonale (e, per ora, senza mai fare scambi di righe/colonne — poi vedremo come trattarli). Dopo $n-1$ passi (l'ultimo passo $k = n$ non serve!), abbiamo ottenuto l'identità

$$A = LU,$$

$U = A_n$ è una matrice triangolare superiore, e

$$L = E_1^{-1}E_2^{-1}\dots E_{n-1}^{-1} = \begin{bmatrix} 1 & & & & & \\ v_{21} & 1 & & & & \\ v_{31} & v_{32} & 1 & & & \\ v_{41} & v_{42} & v_{43} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \ddots & \\ v_{n1} & v_{n2} & v_3 & \dots & v_{n,n-1} & 1 \end{bmatrix},$$

è una matrice triangolare inferiore.

Il ragionamento che abbiamo appena fatto mostra che (*se* non incontriamo fallimenti nell'eliminazione di Gauss) allora esiste una fattorizzazione LU della matrice A.

Esempio: scrittura in Matlab, facendo direttamente (all'infuori di un ciclo for) i primi passaggi.

Esempio: scrittura in Matlab dell'eliminazione di Gauss tramite cicli for. Usiamo un'unica variabile A che sovrascriviamo ad ogni passo.

```
function [L, U] = fattorizzazioneLU(A)

[m, n] = size(A);
if m ~= n
    error('A dev''essere quadrata');
end

L = eye(n);  % qui scriveremo al passo k  inv(L1)...inv(Lk)
U = A;  % qui scriveremo al passo k la matrice A_{k+1}
for k = 1:n-1
    % invariante: ad ogni passo, vale L*U == A
```

```
    if U(k,k) = 0
        error('Pivot nullo');
    end
   L(k+1:n, k) = U(k+1:n, k) / U(k, k);
   U(k+1:n, k) = 0;
   U(k+1:n, k+1:n) = U(k+1:n, k+1:n) - L(k+1:n,k)*U(k,k+1:n);
end
```

Costo computazionale dell'eliminazione di Gauss Il grosso del costo è dato dall'aggiornamento del blocco $A_{k:n,k:n}$ ad ogni passo del ciclo for; questo richiede $2(n-1)^2$ operazioni al primo passo, $2(n-2)^2$ al secondo, $\dots 2 \cdot 1^2$ al $n-2$-esimo passo, $2 \cdot 0^2$ al $n-1$-esimo passo. Un'identità algebrica (che possiamo dimostrare per induzione) ci dice che

$$0^2 + 1^2 + 2^2 + \cdots + (n-1)^2 = \frac{1}{6}(n-1)n(2n-1) = \frac{1}{3}n^3 + O(n^2).$$

I termini che abbiamo omesso sono *di ordine inferiore*, cioè hanno potenze più basse della n. Similmente, se teniamo traccia delle altre operazioni (ad esempio, il calcolo degli L_{ij}) otteniamo termini di ordine inferiore. Quindi il costo della fattorizzazione LU (o dell'eliminazione di Gauss) è $\frac{2}{3}n^3 + O(n^2)$ operazioni aritmetiche.

Notare che nel nostro codice non scriviamo mai esplicitamente moltiplicazioni `Ek*Uk`, ma scriviamo del codice che ne calcola il risultato usando la struttura particolare di queste matrici E_k. Se invece lo scrivessimo come prodotto, Matlab userebbe l'algoritmo generico per calcolare un prodotto matrice-matrice, che costa $2n^3 + O(n^2)$. Una sola moltiplicazione di questo tipo costa più che non tutto il nostro algoritmo!

Soluzione di sistemi lineari tramite la fattorizzazione LU Una volta calcolata $A = LU$, abbiamo decomposto A in un prodotto di matrici più semplici, e possiamo usare questa decomposizione per risolvere sistemi lineari.

Notiamo inoltre che la matrice L è sempre invertibile (perché ha 1 sulla diagonale), e in più abbiamo $\det(A) = \det(LU) = \det(L)\det(U)$, quindi U è invertibile se e solo se A lo è. Quindi possiamo risolvere sistemi lineari con matrici L e U. Abbiamo $\mathbf{b} = A\mathbf{x} = LU\mathbf{x}$.

L'algoritmo è composto di tre passi:

1. Calcolo la fattorizzazione $A = LU$.
2. Calcolo il vettore $\mathbf{y} = U\mathbf{x}$ risolvendo il sistema lineare $\mathbf{b} = L\mathbf{y}$ (sostituzione in avanti).
3. Calcolo il vettore $\mathbf{x}$ risolvendo il sistema lineare $\mathbf{y} = U\mathbf{x}$ (sostituzione all'indietro).

Il costo del primo passo è di $\frac{2}{3}n^3 + O(n^2)$ operazioni aritmetiche. Il secondo e il terzo costano n^2 operazioni aritmetiche, quindi sono molto più economici e "spariscono" dentro l'$O(n^2)$. In totale quindi abbiamo $\frac{2}{3}n^3 + O(n^2)$.

Un'altra osservazione interessante è che se dobbiamo risolvere molti sistemi lineari con vettori $\mathbf{b}$ diversi ma la stessa matrice A, possiamo riutilizzare la stessa fattorizzazione LU più volte, e quindi effettuare il passo più costoso una volta sola.

Esistenza della fattorizzazione LU Cosa può andare storto? Una sola cosa: quando dobbiamo dividere per $(U_k)_{kk}$ (elemento *pivot*), questo potrebbe essere zero. Possiamo dimostrare un criterio che mostra quando questo succede.

Definiamo le *sottomatrici principali di testa* di una matrice A come $A_{1:k,1:k}$. Questa notazione (che possiamo usare anche in Matlab) significa prendere le righe dalla 1 alla k e le colonne dalla 1 alla k della matrice A.

Esempio: sottomatrici principali di una "matrice a freccia"; sono tutte uguali a matrici identità.

Teorema 5.5 (Esistenza della fattorizzazione LU)**.** *Data una matrice $A \in \mathbb{C}^{n\times n}$, supponiamo che le sottomatrici principali di testa $A_{1:k,1:k}$ di A, per $k = 1, \dots, n-1$, siano tutte invertibili. Allora, è possibile portare a termine l'algoritmo di fattorizzazione LU senza mai incontrare pivot nulli, e quindi esiste una fattorizzazione LU di A.*

Dimostrazione. Supponiamo di aver effettuato i primi $k-1$ passi dell'eliminazione di Gauss, ed essere pronti ad effettuare il k-esimo.

$$A = \underbrace{\begin{bmatrix} 1 & & & & & & \\ v_{21} & \ddots & & & & & \\ \vdots & \vdots & 1 & & & & \\ \vdots & \vdots & v_{k,k-1} & 1 & & & \\ \vdots & \vdots & v_{k,k-1} & 0 & 1 & & \\ \vdots & \vdots & \vdots & 0 & 0 & \ddots & \\ v_{n1} & \dots & v_{n,k-1} & 0 & \dots & \dots & 1 \end{bmatrix}}_{E_1^{-1}E_2^{-1}\dots E_{k-1}^{-1}} \underbrace{\begin{bmatrix} * & \dots & * & * & * & \dots & * \\ 0 & \ddots & * & * & * & \dots & * \\ \vdots & \ddots & * & * & * & \dots & * \\ 0 & 0 & 0 & (U_k)_{kk} & * & \dots & * \\ 0 & 0 & 0 & (U_k)_{k+1,k} & * & \dots & * \\ 0 & 0 & 0 & \vdots & * & \dots & * \\ 0 & 0 & 0 & (U_k)_{n,k} & * & \dots & * \end{bmatrix}}_{U_k}$$

Notiamo che le prime k *righe* (non solo $k-1$!) di L e di U sono già quelle definitive: non verranno più modificate nei passi successivi dell'algoritmo. Inoltre, se facciamo il prodotto tra le prime k righe di L e le prime k colonne di U otteniamo $A_{1:k,1:k}$; cioè

$$A_{1:k,1:k} = L_{1:k,1:k}U_{1:k,1:k} :$$

se prendiamo le sottomatrici principali di testa di L e U, queste forniscono una fattorizzazione LU di $A_{1:k,1:k}$. In particolare, abbiamo

$$\det A_{1:k,1:k} = (\det L_{1:k,1:k})(\det U_{1:k,1:k}) = (1 \cdot 1 \cdot \dots \cdot 1)(U_{11}U_{22}\cdots U_{kk}).$$

Quindi se $\det A_{1:k,1:k} \neq 0$, allora l'elemento U_{kk} è diverso da zero e possiamo effettuare il passo k-esimo dell'eliminazione di Gauss. Visto che ho $n-1$ passi, mi basta guardare fino a $A_{1:n-1,1:n-1}$ (la matrice A stessa potrebbe essere singolare, e quindi potrei avere $U_{nn} = 0$, ma tanto non devo più effettuare divisioni). □

Notiamo che quello che abbiamo dimostrato qui sopra in realtà è un "se e solo se": se $A_{1:k,1:k}$ è il primo minore principale non invertibile, allora possiamo effettuare i primi $k-1$ passi, e quindi, $U_{11}, \dots, U_{k-1,k-1}$ sono invertibili, ma arrivati al passo k si ha che $\det A_{1:k,1:k} = 0$ implica $U_{kk} = 0$ e quindi l'eliminazione di Gauss fallisce.

Categorie di matrici per cui esiste sempre la fattorizzazione LU Ci sono alcune categorie di matrici particolari per cui esiste sempre la fattorizzazione LU. Una sono le matrici *dominanti diagonali*. Una matrice $A \in \mathbb{C}^{n\times n}$ si dice *dominante diagonale per righe* se

$$|A_{ii}| > \sum_{j\neq i} |A_{ij}|, \quad i = 1, 2, \ldots, n$$

vale a dire, in ogni riga il valore assoluto dell'elemento sulla diagonale è maggiore della somma di tutti gli altri.

Teorema 5.6. *Una matrice dominante diagonale è invertibile.*

Dimostrazione. Supponiamo invece (per assurdo) che $A\mathbf{x} = 0$ per un qualche vettore $\mathbf{x} \neq 0$. Allora, $\mathbf{x}$ è un autovettore con autovalore $\lambda = 0$. Per il teorema dei cerchi di Gershgorin, dovrà esistere un indice $p \in \{1, 2, \ldots, n\}$ tale che $0 \in K_p$. Ma questa relazione vuol dire che

$$|A_{pp}| = |0 - A_{pp}| \leq \sum_{j\neq i} |A_{ij}|,$$

e quindi contraddice la dominanza diagonale. □

Osserviamo che se A è dominante diagonale, allora lo sono anche tutte le sue sottomatrici principali di testa (le disuguaglianze diventano più forti perché stiamo omettendo dei termini!). Quindi se A è dominante diagonale allora ammette fattorizzazione LU.

Analogamente, possiamo definire matrici dominanti diagonali per colonne, quando

$$|A_{jj}| > \sum_{i\neq j} |A_{ij}|, \quad j = 1, 2, \ldots, n.$$

Una matrice A è dominante diagonale per colonne se A^T è dominante diagonale per righe, quindi valgono gli stessi risultati.

Esempio: la matrice

$$\begin{bmatrix} -5 & 2 & 2 \\ 1 & 3 & -1 \\ 1 & 1 & 3 \end{bmatrix}$$

è dominante diagonale per righe, ma non per colonne (valgono uguaglianze). Le sue sottomatrici principali di testa sono di nuovo pred. diag. per righe (e questa volta anche per colonne).

Stabilità e necessità del pivoting Su un computer, difficilmente gli "zeri" vengono calcolati esattamente come zero! L'eliminazione di Gauss tecnicamente fallisce solo se c'è un pivot *esattamente* uguale a zero, ma un pivot molto vicino a zero è comunque problematico. Non facciamo un'analisi dettagliata dell'errore algoritmico, ma vediamo direttamente un esempio in cui le cose vanno storte. Supponiamo di avere (per un $\varepsilon > 0$ molto piccolo) la matrice

$$\begin{bmatrix} \varepsilon & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}.$$

Questa matrice è ben condizionata (il condizionamento nelle tre norme è minore di 10), ma dopo il primo passo di eliminazione di Gauss otteniamo

$$\begin{bmatrix} \varepsilon & 1 & 1 \\ 0 & 1-\frac{1}{\varepsilon} & 1-\frac{1}{\varepsilon} \\ 0 & 1-\frac{1}{\varepsilon} & -1-\frac{1}{\varepsilon} \end{bmatrix}.$$

Se ε è molto piccolo, $\frac{1}{\varepsilon}$ è molto grande. In particolare, se per esempio $\varepsilon = 2^{-60}$, il numero di macchina più vicino sia a $1-\frac{1}{\varepsilon}$ che a $-1-\frac{1}{\varepsilon}$ è $\frac{1}{\varepsilon}$, e quindi le ultime due righe diventano uguali quando andiamo a scrivere quei numeri su un calcolatore: pertanto l'algoritmo di fattorizzazione LU si blocca affermando (erroneamente) che la matrice è singolare. Ma in realtà la matrice è molto lontana dall'esserlo. Anche senza arrivare a valori così estremi, dobbiamo sommare un numero molto grande a tutti gli elementi della sottomatrice $A_{2:3,2:3}$, e poi andare a fare una sottrazione tra due valori molto grandi e molto vicini tra loro, al passo successivo dell'eliminazione. Questo causa perdita di precisione e problemi numerici.

La soluzione, in questo caso e in molti problemi simili, è scambiare tra loro le righe. In particolare, scambiarle in modo da avere in posizione (k,k) (cioè come pivot) il numero *più grande* in valore assoluto tra quelli della sottomatrice $(U_k)_{k:n,k}$.

Notare che questo è qualcosa di molto diverso rispetto a quello che facevate ad algebra lineare facendo i conti a mano: se per esempio i numeri della prima colonna erano $2, -4, 3, 1$, tipicamente volevate 1 come pivot in modo da non dover lavorare con i denominatori. Questa volta invece l'obiettivo è diverso; vogliamo pivot più grandi possibili, quindi scegliamo -4 come pivot.

Matrici di permutazione La fattorizzazione che vedremo corrisponde essenzialmente a calcolare la fattorizzazione LU di una matrice ottenuta dalla A permutando le righe in un ordine opportuno: quindi per esempio

$$A = \begin{bmatrix} A_{1,:} \\ A_{2,:} \\ A_{3,:} \\ A_{4,:} \end{bmatrix}, \quad PA = \begin{bmatrix} A_{4,:} \\ A_{2,:} \\ A_{1,:} \\ A_{3,:} \end{bmatrix}.$$

Abbiamo indicato la matrice di destra con PA perché effettivamente è ottenuta moltiplicando la matrice A a sinistra per una matrice $P \in \mathbb{R}^{n\times n}$: questa è la matrice ottenuta dalla matrice identità applicando alcuni scambi di righe (gli stessi applicati in A); ad esempio,

$$P = \begin{bmatrix} \mathbf{e}_4^T \\ \mathbf{e}_2^T \\ \mathbf{e}_1^T \\ \mathbf{e}_3^T \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Una matrice di questo tipo è detta *matrice di permutazione*, perché il suo effetto su vettori e matrici è quello di permutarne le entrate:

$$P\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_4 \\ x_2 \\ x_1 \\ x_3 \end{bmatrix}.$$

La stessa cosa succede quando effettuiamo il prodotto PA; la stessa permutazione viene applicata calcolando ogni colonna separatamente, quindi l'effetto netto è quello di permutare le righe di A.

Eliminazione di Gauss con pivoting (parziale) Partiamo da una matrice $P_1 = I$. Al primo passo dell'eliminazione di Gauss, una volta individuata la riga p dove sta l'elemento con $|A_{p1}| = \max|A_{i1}|$, scambiamo la riga p e la riga 1 in A. Questo corrisponde a rimpiazzare la matrice A con una matrice P_2A, dove P_2 è la matrice che differisce dall'identità solo per lo scambio delle righe 1 e p.

Similmente, prima di ogni passo k, individuiamo la riga dove sta il pivot $|(U_k)_{pk}| = \max|(U_k)_{k:n,k}|$, e scambiamo la riga k con la riga p nella matrice P_kA. Quella che otteniamo quindi è un'altra matrice ottenuta permutando le righe della matrice A, che possiamo quindi scrivere come $P_{k+1}A$, dove P_k è un'opportuna matrice di permutazione.

Notiamo che fino a questo punto l'algoritmo di fattorizzazione LU ha lavorato indipendentemente sulle righe da $k : n$: se queste righe fossero state in un altro ordine nella matrice A, avremmo ottenuto le stesse matrici L_k e U_k, solo con gli elementi calcolati nelle ultime righe in ordine diverso. Quindi per ottenere il risultato dei primi k passi di fattorizzazione LU su questa matrice $P_{k+1}A$, dobbiamo semplicemente prendere le matrici L_k e U_k che abbiamo già calcolato, e scambiare le righe p e k negli elementi che abbiamo calcolato finora. (Sulla matrice L_k, questo scambio si limita alle prime $k-1$ colonne, quelle contenenti gli elementi che abbiamo calcolato: gli elementi uguali a 1 sulla diagonale restano al loro posto.)

A questo punto, effettuiamo normalmente un passo di eliminazione di Gauss sulle matrici ottenute, ottenendo due nuove matrici L_{k+1} e U_{k+1}. Per come abbiamo lavorato, vale ad ogni passo la relazione $P_kA = L_kU_k$.

Vediamo insieme il codice Matlab di questo algoritmo.

```
function [L, U, P] = fattorizzazioneLU_pivoting(A)
% fattorizzazione LU con pivoting
% restituisce matrici tali che L*U = P*A
[m, n] = size(A);
if m ~= n
    error('A deve essere quadrata');
end

L = eye(n);
P = eye(n);
U = A;
for k = 1:n-1
   % ad ogni iterazione, abbiamo L*U == P*A

   % calcola riga p del pivot
   [valore, posizione] = max(abs(U(k:n,k)));
   % passa da un indice in k:n ad uno in 1:n
   p = posizione + k-1;
   if valore == 0
       error('matrice esattamente singolare');
   end
```

```
    % esegue gli scambi
    U([p k], 1:n) = U([k p], 1:n);
    L([p k], 1:k-1) = L([k p], 1:k-1);
    P([p k], 1:n) = P([k p], 1:n);

    % prosegue con l'eliminazione di Gauss
    L(k+1:n, k) = U(k+1:n, k) / U(k, k);
    U(k+1:n, k) = 0;
    U(k+1:n, k+1:n) = U(k+1:n, k+1:n) - L(k+1:n,k)*U(k,k+1:n);
end
```

Notiamo che `posizione` restituisce un indice all'interno del vettore di $n-k+1$ elementi $U_{k:n,n}$: se per esempio l'elemento di modulo massimo fosse il primo, `posizione = 1`, però questa posizione 1 corrisponde alla riga k delle matrici con cui stiamo lavorando. Per convertire questa posizione in un indice di riga, dobbiamo sommare $k-1$.

Qual è il costo computazionale di questo algoritmo? Le uniche operazioni che abbiamo aggiunto sono la ricerca del massimo e gli scambi di righe. Queste sono operazioni che richiedono un certo tempo per essere effettuate sul computer, però non sono operazioni aritmetiche, strettamente parlando. Quindi il costo in termini di operazioni aritmetiche è di nuovo di $\frac{2}{3}n^3$ operazioni aritmetiche.

In Matlab, già esiste `[L, U, P] = lu(A)`.

Stabilità dell'eliminazione di Gauss con pivoting Non vediamo un'analisi completa della stabilità dell'eliminazione di Gauss (che è abbastanza complessa), ma il messaggio principale è che l'errore algoritmico è circa dello stesso ordine dell'errore inerente (e quindi l'algoritmo è "il migliore possibile") a patto che gli elementi della L e della U non crescano troppo durante l'algoritmo (rispetto agli elementi di A, nel caso della U).

Il pivoting assicura che gli elementi della L siano tutti minori o uguali a 1 in valore assoluto. Però anche con il pivoting esistono esempi particolarmente sfortunati in cui $\|U\|/\|A\|$ cresce come 2^n, dove n è la dimensione. Nella maggior parte dei casi, però, l'eliminazione di Gauss con pivoting è perfettamente stabile. È l'algoritmo più usato per risolvere sistemi lineari; in Matlab anche `A \ b` utilizza questo algoritmo.

Soluzione di sistemi lineari tramite fattorizzazione LU con pivoting La fattorizzazione LU con pivoting restituisce quindi L, U, P tali che $LU = PA$. P è una matrice di permutazione. Si può dimostrare che le matrici di permutazione sono *ortogonali*, cioè $P^{-1} = P^T$: per invertire la matrice basta trasporla.

Pertanto possiamo convertire la scrittura in una fattorizzazione $P^T LU = A$, e usarla per risolvere sistemi lineari: da $P^T LU\mathbf{x} = \mathbf{b}$ possiamo chiamare $\mathbf{y} = U\mathbf{x}$, $\mathbf{z} = L\mathbf{y}$ e risolvere nell'ordine

$$P^T\mathbf{z} = \mathbf{b}, \quad L\mathbf{y} = \mathbf{z}, \quad U\mathbf{x} = \mathbf{y}.$$

Il primo dei tre sistemi lineari si risolve semplicemente moltiplicando per l'inversa, $\mathbf{z} = (P^T)^{-1}\mathbf{b} = P\mathbf{b}$; gli altri due per sostituzione in avanti e all'indietro come in precedenza.

Calcolo del determinante mediante fattorizzazione LU con pivoting Abbiamo visto nell'introduzione del corso che usare le formule classiche dell'algebra lineare (espansione di Laplace, formule ricorsive sui minori) non è un modo efficace di calcolare il determinante; ha un costo computazionale che cresce troppo velocemente. Vediamo qui invece un metodo basato sulla fattorizzazione LU per calcolarlo.

Teorema 5.7. *Sia $PA = LU$ una fattorizzazione LU con pivoting della matrice $A \in \mathbb{C}^{n\times n}$. Allora,*

$$\det(A) = (-1)^s U_{11}U_{22}\dots U_{nn},$$

dove s è il numero di scambi di riga effettuati durante il calcolo della fattorizzazione.

Dimostrazione. Usando il teorema di Binet, abbiamo

$$\det P \det A = \det PA = \det LU = \det L \det U. \tag{5.4}$$

Poiché L e U sono triangolari, i loro determinanti sono i prodotti degli elementi lungo la diagonale:

$$\det L = 1 \cdot 1 \cdot 1 \cdot \dots \cdot 1 = 1; \det U = U_{11}U_{22}\dots U_{nn}.$$

La matrice di permutazione P è ottenuta applicando alla matrice identità gli scambi di riga effettuati lungo la fattorizzazione; sappiamo dall'algebra lineare che ogni scambio di riga fa cambiare il segno del determinante, quindi

$$\det P = (-1)^s \det I = (-1)^s.$$

Possiamo quindi ricavare $\det A$ dalla (5.4), ottenendo la tesi. □

5.7 Fattorizzazioni per matrici simmetriche

Fattorizzazione LDL^T Supponiamo di effettuare l'eliminazione di Gauss su una matrice simmetrica, cioè $A = A^T$. Non è difficile vedere che dopo il primo passo la matrice U_2 ottenuta non è più simmetrica; la "struttura" della matrice viene distrutta. Esiste però una variante dell'eliminazione di Gauss che evita questo problema e lavora unicamente con matrici simmetriche. L'idea è la seguente: ad ogni passo, anziché definire $U_2 = E_1U_1$, definiamo $U_2 = E_1U_1E_1^T$ (moltiplicando a destra e a sinistra). Questa matrice è simmetrica, difatti $(E_1U_1E_1^T)^T = (E_1^T)^T(U_1)^TE_1^T = E_1U_1E_1^T$. In più, considerando la posizione degli zeri in E_1^T, è facile verificare che la prima colonna continua a contenere zeri. Ma allora anche la prima riga contiene zeri, e abbiamo ottenuto qualcosa della forma

$$U_2 = \begin{bmatrix} A_{11} & 0 & 0 & 0 \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \end{bmatrix}.$$

Il blocco indicato con $\star$ è simmetrico, e possiamo continuare nello stesso modo, definendo $U_3 = E_2U_2E_2^T$, e così via. Alla fine della procedura, otteniamo una matrice diagonale, che possiamo chiamare D. Più nel dettaglio, abbiamo

$$D = E_{n-1}E_{n-2}\dots E_2E_1AE_1^TE_2^T\dots E_{n-2}^TE_{n-1}^T.$$

Moltiplicando per le inverse delle E_k (che esistono) dai lati opportuni, abbiamo

$$A = \underbrace{(E_1^{-1}E_2^{-1}\dots E_{n-1}^{-1})}_{=L} D \underbrace{(E_{n-1}^T)^{-1}(E_{n-2}^T)^{-1}\dots(E_2^T)^{-1}(E_1^T)^{-1}}_{=L^T} = LDL^T.$$

Quella che abbiamo ottenuto è una particolare fattorizzazione LU in cui la matrice U è uguale a DL^T, il prodotto di L per una matrice diagonale. Si chiama *fattorizzazione* LDL^T.

Costo computazionale della fattorizzazione LDL^T Possiamo impostare un'analisi del costo computazionale analoga a quella che abbiamo fatto per la fattorizzazione LU. Abbiamo di nuovo degli aggiornamenti della forma

$$A_{k+1:n,k+1:n} \leftarrow A_{k+1:n,k+1:n} - L_{k+1:n,k}A_{k,k+1:n} = A_{k+1:n,k+1:n} - A_{k+1:n,k}\frac{1}{A_{kk}}A_{k,k+1:n};$$

la differenza fondamentale però è che questa volta le matrici coinvolte sono simmetriche. Quindi non è necessario calcolare tutti gli elementi: basta calcolare quelli della parte triangolare inferiore (inclusa la diagonale), e ignorare la parte triangolare superiore che può essere ricavata per simmetria. Questo riduce il costo computazionale alla metà: da $\frac{2}{3}n^3 + O(n^2)$ a $\frac{1}{3}n^3 + O(n^2)$ operazioni aritmetiche.

In Matlab per applicare quest'ultimo trucco di riempire la parte superiore per simmetria è necessario scrivere cicli `for` direttamente; non possiamo lavorare con matrici a blocchi. Vi invito però, come esercizio, a provare a scrivere una funzione che implementa la fattorizzazione LDL^T, nella versione senza pivoting, usando cicli `for` per questo aggiornamento.

Pivoting e fattorizzazione LDL^T In una fattorizzazione LDL^T, il pivoting è un punto abbastanza delicato; difatti scambiando righe si perde la simmetria. Anche scambiare righe e colonne (moltiplicando PAP^T) da solo non basta. In Matlab, la funzione `[L,D,P] = ldl(A)` produce una fattorizzazione $LDL^T = P^TAP$, però (per avere una versione più stabile) la matrice D può avere dei blocchi 2×2 lungo la diagonale. Anche questa volta, però, abbiamo una categoria particolare di matrici per cui possiamo dimostrare che la fattorizzazione funziona sempre senza bisogno di ricorrere al pivoting.

Matrici SPD (simmetriche e positive definite) Una matrice $A \in \mathbb{R}^{n\times n}$ (serve che sia reale, questa volta!) si definisce *simmetrica e positiva definita* (SPD) se:

1. è simmetrica, cioè $A = A^T$;
2. è positiva definita, cioè $\mathbf{x}^TA\mathbf{x} > 0$ per ogni vettore $\mathbf{x} \in \mathbb{R}^n$ tale che $\mathbf{x} \neq 0$.

Spesso si dice solo "positiva definita" sottintendendo "simmetrica".

Si può dimostrare il seguente risultato (non lo vediamo qui; talvolta si fa nei corsi di algebra lineare):

Teorema 5.8. *Una matrice simmetrica A è definita positiva se e solo se tutti i suoi autovalori sono reali (strettamente) positivi.*

Valgono le seguenti proprietà.

1. Una matrice definita positiva ha tutti gli elementi sulla diagonale (strettamente) positivi; difatti $A_{kk} = e_k^T A e_k > 0$. (Notare che questo *non* è un "se e solo se": esistono matrici simmetriche che hanno tutti gli elementi sulla diagonale positivi, ma non sono positive definite! Per esempio, $A = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right]$).

2. Se A è semidefinita positiva e M è invertibile, allora lo è anche MAM^T: difatti $x^T MAM^T x = (M^T x)^T A (M^T x) > 0$.

Utilizzando queste due proprietà, è facile dimostrare che ad ogni passo della fattorizzazione LDL^T il pivot A_{kk} è invertibile. Quindi una matrice SPD ammette sempre fattorizzazione LDL^T, e gli elementi diagonali d_{kk} sono tutti (strettamente) positivi. In più, è possibile dimostrare che questa fattorizzazione è stabile senza bisogno di fare pivoting; gli elementi della L e della D "non crescono troppo" rispetto agli elementi di A.

Fattorizzazione di Cholesky Si può riscrivere la fattorizzazione LDL in un formato leggermente diverso: per la matrice diagonale D abbiamo che $D = D^{1/2} D^{1/2}$, dove $D^{1/2}$ è la matrice diagonale che ha elementi $\sqrt{d_{kk}}$. Quindi

$$A = LDL^T = (LD^{1/2})(D^{1/2} L^T) = R^T R,$$

dove R è una matrice triangolare superiore. Questa fattorizzazione (che si può scrivere per ogni matrice SPD) si chiama *fattorizzazione di Cholesky*. Ci permette di scrivere A come il prodotto di una matrice triangolare inferiore e di una triangolare superiore, che sono una la trasposta dell'altra.

Per calcolarla basta aggiungere n radici quadrate alle $\frac{1}{3}n^3 + O(n^2)$ operazioni aritmetiche della LDL^T. In Matlab, c'è il comando `R = chol(A)`.

Esercizio Dato un parametro $\alpha \in \mathbb{C}$, consideriamo la matrice

$$A_\alpha = \begin{bmatrix} 1 & & & & -\alpha \\ -\alpha & 1 & & & \\ & -\alpha & 1 & & \\ & & \ddots & \ddots & \\ & & & -\alpha & 1 \end{bmatrix} \in \mathbb{C}^{n \times n}.$$

1. Per quali valori di α il teorema dei cerchi di Gershgorin ci permette di concludere che la matrice è invertibile?

2. Per quali valori di α è soddisfatta la condizione di esistenza della fattorizzazione LU?

3. Sapreste trovare la fattorizzazione LU di A?

4. Per quali valori di α la matrice è invertibile?

Soluzione:

1. I cerchi di Gershgorin hanno tutti centro 1 e raggio $|\alpha|$. Se $|\alpha| < 1$, allora 0 non appartiene all'unione dei cerchi di Gershgorin, quindi non è un autovalore della matrice A; pertanto A è invertibile. Se invece $|\alpha| \geq 1$, allora 1 sta all'interno dell'unione dei cerchi, quindi il teorema non ci permette di concludere nulla: 0 potrebbe essere autovalore oppure no.

2. Le sottomatrici di testa fino all'ordine $n-1$ sono tutte triangolare inferiori con 1 sulla diagonale; quindi sono invertibili per ogni valore di $\alpha \in \mathbb{C}$.

3. Vediamo due modi di calcolare la fattorizzazione: il primo è quello di eseguire l'eliminazione di Gauss, e ricavare L e U rispettivamente dai moltiplicatori e dalla matrice ottenuta. [TODO: vedi note tablet]

   Il secondo è quello di notare che (dalla nostra dimostrazione dell'esistenza della fattorizzazione LU) abbiamo che $A_{1:n-1,1:n-1} = L_{1:n-1,1:n-1}U_{1:n-1,1:n-1}$, e questa è una fattorizzazione LU di questa sottomatrice. Ma una fattorizzazione LU di $A_{1:n-1,1:n-1}$ è facile da determinare: poiché è triangolare inferiore con 1 sulla diagonale, possiamo prendere $L_{1:n-1,1:n-1} = A_{1:n-1,1:n-1}$ e $U_{1:n-1,1:n-1} = I$ (matrice identità di ordine $n-1$). Possiamo estendere questa fattorizzazione a una fattorizzazione LU della matrice A? [TODO: vedi note tablet]

4. Dalla fattorizzazione LU calcolata, abbiamo $\det A = \det U = 1 \cdot 1 \cdot \cdots \cdot 1 \cdot (1 - \alpha^n) = 1 - \alpha^n$. Quindi il determinante si annulla se e solo se α è una radice n-esima complessa dell'unità. Notiamo che questo risultato è coerente con quanto visto sopra con il teorema dei cerchi di Gershgorin.

5.8 Metodi iterativi per sistemi lineari

Matrici sparse Spesso nelle applicazioni compaiono matrici che hanno molti elementi uguali a zero. Per esempio, alcuni metodi di soluzione di equazioni differenziali conducono a matrici *tridiagonali*, oppure a matrici che hanno elementi diversi da zero solo lungo alcune diagonali (non per forza vicine a quella principale). In una matrice $n \times n$ di questo tipo, il numero di elementi diversi da zero ("non-zeri") è proporzionale a n, (possiamo scriverlo come $O(n)$), quindi molti meno degli n^2 elementi.

Alcune operazioni su matrici sparse si possono effettuare più velocemente. Per esempio, per calcolare il prodotto con un vettore, Av, mi basta considerare nella somma gli elementi di A diversi da zero. Il costo computazionale di questo prodotto quindi è di circa $2nnz$ elementi, dove con nnz indichiamo il "numero di non-zeri" della matrice A.

In Matlab, abbiamo alcuni comandi per gestire matrici sparse. Per esempio, `sparse(A)` restituisce una copia della matrice A, memorizzata in un formato diverso che elenca i soli "non-zeri". Similmente, `sparse(m, n)` e `speye(m)` creano, in questo formato, rispettivamente una matrice nulla e l'identità. Lavorare con questo formato comincia a diventare conveniente solo quando la A ha una densità di non-zeri molto bassa, 10% o meno: se una matrice triangolare ha semplicemente zeri nella parte triangolare inferiore, o superiore, ma a parte questo è densa, allora i non-zeri sono circa il 50%, e i costi collegati alla gestione delle matrici sparse sono maggiori dei risparmi ottenuti usando questo formato.

Ve lo dico solo per conoscenza, ma non vediamo dettagli su come gestire matrici sparse in questo corso.

I metodi che abbiamo visto finora per risolvere sistemi lineari non sono l'ideale per matrici sparse, perché spesso l'eliminazione di Gauss elimina anche la sparsità, introducendo molti elementi diversi da zero. Per esempio, data una matrice della forma

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & & & & \\ * & & * & & & \\ * & & & * & & \\ * & & & & * & \\ * & & & & & * \end{bmatrix}$$

già dopo il primo passaggio otteniamo una matrice con molti più non-zeri di quella di partenza:

$$\begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{bmatrix}.$$

Esistono metodi per risolvere sistemi lineari che riescono a sfruttare il fatto che A sia una matrice sparsa, ottenendo un minore costo computazionale rispetto a $O(n^3)$. Questi metodi sono molto simili a quelli che abbiamo visto nella prima sezione: producono, passo dopo passo, una successione di vettori, $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \dots, \mathbf{x}^{(3)}, \dots$ (ogni $\mathbf{x}^{(i)}$ è un vettore; mettiamo l'indice in alto per non confonderci con gli elementi). Sotto condizioni opportune (che vedremo), questa successione converge alla soluzione $\mathbf{x}$ di $A\mathbf{x} = \mathbf{b}$.

Metodi iterativi generici Descriviamo ora un modo per ottenere un metodo iterativo a partire da uno *splitting* della matrice A, cioè una scrittura del tipo $A = M - N$, dove $M \in \mathbb{C}^{n\times n}$ è una matrice invertibile. La soluzione $\mathbf{x} \in \mathbb{C}^{n\times n}$ del sistema lineare soddisfa

$$A\mathbf{x} = \mathbf{b} \iff (M-N)\mathbf{x} = \mathbf{b} \iff M\mathbf{x} = N\mathbf{x} + \mathbf{b} \iff \mathbf{x} = M^{-1}(N\mathbf{x} + \mathbf{b}) \tag{5.5}$$

Questo suggerisce di impostare un'iterazione di punto fisso

$$\begin{cases} \mathbf{x}^{(1)} \in \mathbb{C}^n & \text{fissato;} \\ \mathbf{x}^{(k+1)} = M^{-1}(N\mathbf{x}^{(k)} + \mathbf{b}) & k = 1, 2, 3, \dots \end{cases} \tag{5.6}$$

Notiamo che questa iterazione si può implementare in modo efficiente se abbiamo un modo efficiente di risolvere il sistema lineare $M\mathbf{x}^{(k+1)} = N\mathbf{x}^{(k)} + \mathbf{b}$: per esempio, se M è triangolare possiamo usare un processo di sostituzione. Tipicamente *non* vogliamo calcolare M^{-1} (né tantomeno $M^{-1}N$), perché questo andrebbe a costare più di quello che possiamo permetterci.

La (5.6) è una versione "multidimensionale" del metodo di punto fisso $\mathbf{x}_{k+1} = \Phi(\mathbf{x}_k)$ che abbiamo già visto. Parlando di quel metodo, avevamo visto che spesso la convergenza avviene solo quando $\mathbf{x}^{(1)}$ è sufficientemente vicino alla soluzione esatta $\mathbf{x}$. Per questi metodi, invece, la convergenza solitamente *non* dipende dal punto iniziale, come vedremo.

Partiamo introducendo una definizione. Diciamo che il metodo iterativo è *convergente* se *per ogni scelta* di $\mathbf{x}^{(1)} \in \mathbb{C}^n$ la successione generata dal metodo converge alla soluzione $\mathbf{x}$ del sistema lineare.

Per studiare la convergenza, definiamo $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}$ l'errore (assoluto) al passo k. Usando la (5.5) e la (5.6) abbiamo per ogni $k = 1, 2, \dots$

$$\mathbf{e}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x} = M^{-1}(N\mathbf{x}^{(k)} + \mathbf{b}) - M^{-1}(N\mathbf{x} + \mathbf{b}) = M^{-1}(N\mathbf{x}^{(k)} - N\mathbf{x}) = M^{-1}N\mathbf{e}^{(k)}. \tag{5.7}$$

Definiamo la *matrice di iterazione* del metodo come $H = M^{-1}N$.

Teorema 5.9. *Il metodo iterativo* (5.6) *è convergente se e solo se* $\rho(H) < 1$.

Dimostrazione. In questo corso dimostriamo il teorema solo nel caso particolare in cui H è diagonalizzabile (ma in realtà è vero sempre).

Usando ripetuatamente la (5.7) is ha

$$\mathbf{e}^{(k+1)} = H\mathbf{e}^{(k)} = HH\mathbf{e}^{(k-1)} = \cdots = H^k\mathbf{e}^{(1)}.$$

Supponiamo che la matrice H sia diagonalizzabile, cioè che esistano una matrice V invertibile e D diagonale tali che $H = VDV^{-1}$; sappiamo dall'algebra lineare che gli elementi diagonali di D sono proprio gli autovalori di H. Possiamo scrivere

$$H^k = \underbrace{(VDV^{-1})(VDV^{-1})\dots(VDV^{-1})}_{k \text{ volte}} = VD^kV^{-1}.$$

Se $\rho(H) < 1$, allora $|D_{ii}| < 1$ per ogni $i = 1, 2, \dots, n$, e quindi $\lim_{k\to\infty} D^k = 0$. Sostituendo più sopra abbiamo allora anche $\lim_{k\to\infty} \mathbf{e}^{(k+1)} = 0$.

Questo conclude (con l'ipotesi aggiuntiva che k sia diagonalizzabile) la dimostrazione che se $\rho(H) < 1$ allora il metodo è convergente. Visto che questo è un "se e solo se", vogliamo però dimostrare anche l'implicazione opposta. Per fare questo, ci basta dimostrare che se H ha un autovalore $H\mathbf{v} = \mathbf{v}\lambda$, $|\lambda| \geq 1$, allora esiste una scelta di $\mathbf{x}^{(1)}$ per cui l'errore $\mathbf{e}^{(k)}$ non converge a zero. Per farlo, prendiamo $\mathbf{x}^{(1)} = \mathbf{x} + \mathbf{v}$. Allora abbiamo $\mathbf{e}^{(1)} = \mathbf{x}^{(1)} - \mathbf{x} = \mathbf{v}$, e

$$\mathbf{e}^{(k+1)} = H^k\mathbf{v} = \mathbf{v}\lambda^k \tag{5.8}$$

(notare infatti che ogni volta che facciamo un prodotto $H\mathbf{v}$ otteniamo il vettore $\mathbf{v}\lambda$, quindi...)

Se $|\lambda| \geq 1$ (e $\mathbf{v} \neq 0$, visto che $\mathbf{v}$ è un autovettore), allora la quantità $\mathbf{v}\lambda^k$ non converge a zero, che era quello che volevamo dimostrare. □

Visto che $\rho(H) \leq \|H\|$ per ogni norma matriciale indotta, abbiamo anche il seguente risultato.

Corollario 5.10. *Se* $\|H\|_p \leq 1$ *per una qualunque norma matriciale indotta, allora il metodo iterativo* (5.6) *è convergente.*

Questo *non* è un "se e solo se"! In particolare, possiamo anche avere situazioni in cui alcune norme matriciali indotte sono minori di 1 e altre maggiori di 1.

È possibile dimostrare anche che per ogni norma vettoriale vale

$$\lim_{k\to\infty} \frac{\|\mathbf{e}^{(k+1)}\|}{\|\mathbf{e}^{(k)}\|} \leq \rho(H),$$

con uguaglianza per quasi tutti i vettori di partenza $\mathbf{x}^{(1)}$. Pertanto il metodo converge linearmente. La convergenza è tanto più veloce (pendenza della retta in scala logaritmica più vicina a $-\infty$) quanto più $\rho(H)$ è piccolo.

Criteri di arresto per metodi iterativi per sistemi lineari Abbiamo due scelte naturali come criteri di arresto per il metodo:

1. Ci fermiamo quando $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)}\| \leq \varepsilon$, per una certa norma e un valore $\varepsilon > 0$ fissato.
2. Ci fermiamo quando $\|A\mathbf{x}^{(k+1)} - \mathbf{b}\| \leq \varepsilon$.

Cosa possiamo affermare sull'errore $\mathbf{e}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x}$ quando è verificato uno di questi criteri? Analizziamoli a partire dal primo.

Teorema 5.11 (Errore di un metodo iterativo)**.** *Consideriamo il metodo iterativo* (5.6)*; supponiamo che sia convergente (quindi $\rho(H) < 1$) e che per un certo valore di k sia verificato (in una certa norma vettoriale) il criterio di arresto $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)}\| \leq \varepsilon$. Allora,*

$$\|\mathbf{e}^{(k+1)}\| \leq \|H(I-H)^{-1}\|\varepsilon,$$

dove la norma è la norma matriciale indotta dalla norma vettoriale usata.

Dimostrazione. Notiamo innanzitutto che la matrice $I - H$ è invertibile, se $\rho(H) < 1$: difatti, se per un vettore $\mathbf{v} \neq \mathbf{0}$ valesse $(I-H)\mathbf{v} = \mathbf{0}$, allora $H\mathbf{v} = \mathbf{v}$; quindi $\mathbf{v}$ sarebbe un autovettore di H con autovalore 1, ma questo è impossibile per la definizione di raggio spettrale.

Abbiamo

$$\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)} = (\mathbf{x}^{(k)} - \mathbf{x}) - (\mathbf{x}^{(k+1)} - \mathbf{x})) = \mathbf{e}^{(k)} - \mathbf{e}^{(k+1)} = \mathbf{e}^{(k)} - H\mathbf{e}^{(k+1)} = (I-H)\mathbf{e}^{(k)}.$$

Quindi quando ci arrestiamo

$$\begin{aligned}\|\mathbf{e}^{(k+1)}\| &= \|H\mathbf{e}^{(k)}\| = \|H(I-H)^{-1}(\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)})\| \\ &\leq \|H(I-H)^{-1}\|\|\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)}\| \leq \|H(I-H)^{-1}\|\varepsilon. \qquad \square\end{aligned}$$

Si può vedere che la quantità $\|H(I-H)^{-1}\|$ può essere molto grande; in particolare questo succede quando $\rho(H)$ è vicino a 1. Quindi questo criterio di arresto non è particolarmente buono: diventa meno accurato quanto più lentamente converge il metodo.

Residuo e stabilità all'indietro (*) Se $\mathbf{x}$ è la soluzione esatta del sistema lineare $A\mathbf{x} = \mathbf{b}$, allora chiaramente $\|A\mathbf{x} - \mathbf{b}\| = 0$. Ma se per un vettore $\tilde{\mathbf{x}} \in \mathbb{C}^n$ il *residuo* $\mathbf{r} = A\tilde{\mathbf{x}} - \mathbf{b}$ è piccolo, questo implica che $\tilde{\mathbf{x}}$ e $\mathbf{x}$ sono vicini? Vediamo un risultato quantitativo che dice cosa succede.

Teorema 5.12 (Stima sul residuo)**.** *Siano $A \in \mathbb{C}^{n\times n}$, $\mathbf{b} \in \mathbb{C}^n$. Sia $\mathbf{x}$ la soluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$, sia $\tilde{\mathbf{x}} \in \mathbb{R}^n$ un altro vettore, e sia $\mathbf{r} = A\tilde{\mathbf{x}} - \mathbf{b}$. Allora,*

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}.$$

Dimostrazione. Abbiamo $A\tilde{\mathbf{x}} = \mathbf{b} + \mathbf{r}$. Allora, $\tilde{\mathbf{x}}$ è la soluzione del sistema lineare $A\tilde{\mathbf{x}} = \widehat{\mathbf{b}}$, in cui abbiamo perturbato il termine noto in $\widehat{\mathbf{b}} = \mathbf{b} + \mathbf{r}$. Pertanto possiamo applicare la disuguaglianza (5.2), che ci dà la tesi. □

Questo risultato non si applica solo all'iterata $\mathbf{x}^{(k+1)}$ prodotta da un metodo iterativo, ma a una qualunque soluzione approssimata di un sistema lineare, non importa come l'abbiamo ottenuta. È quindi particolarmente interessante perché ci permette di ottenere una stima esplicita sull'errore.

Metodi di Jacobi e Gauss–Seidel Due metodi iterativi sono particolarmente comuni per la loro semplicità.

Scriviamo $A = D - E - F$, dove D è la parte diagonale di A, E è la parte strettamente triangolare inferiore (con un segno $-$), e F è la parte triangolare superiore.

Il *metodo di Jacobi* si ottiene scegliendo $M = D$. È applicabile quando D è invertibile (cioè quando $A_{ii} \neq 0$ per ogni i). La soluzione di sistemi lineari con M è particolarmente veloce perché M è diagonale.

Il *metodo di Gauss–Seidel* si ottiene scegliendo $M = D - E$. È applicabile, di nuovo, quando $A_{ii} \neq 0$ per ogni i. La soluzione di sistemi lineari con M è particolarmente veloce perché M è triangolare.

Implementazione del metodo di Jacobi Partiamo scrivendo la relazione $M\mathbf{x}^{(k+1)} = \mathbf{b} + N\mathbf{x}^{(k)}$:

$$\begin{bmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{nn} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} - \begin{bmatrix} 0 & A_{12} & \dots & A_{1n} \\ A_{21} & 0 & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \dots & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix}$$

La i-esima riga corrisponde a

$$A_{ii}x_i^{(k+1)} = b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} A_{ij}x_j^{(k)}$$

da cui possiamo ricavare $x_i^{(k+1)}$ ottenendo

$$x_i^{(k+1)} = \frac{b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} A_{ij}x_j^{(k)}}{A_{ii}} = \frac{b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} A_{ij}x_j^{(k)}}{A_{ii}}, \quad i = 1, 2, \dots, n.$$

Queste equazioni sono indipendenti tra loro per ogni i, possiamo risolverle una dopo l'altra in qualunque ordine per ottenere $\mathbf{x}^{(k+1)}$ a partire da $\mathbf{x}^{(k)}$. Questo costituisce un'iterazione del metodo.

Il costo computazionale è di $2n$ operazioni aritmetiche per ogni i, quindi in totale $2n^2$ per ogni iterazione (il numero di iterazioni che servono per ottenere convergenza non è noto a priori, anzi, il metodo potrebbe non convergere del tutto). Inoltre, se so a priori che alcuni elementi A_{ij} sono zero, posso restringere la somma agli elementi non-nulli; questo modifica il costo totale in $2nnz(A)$.

Implementazione del metodo di Gauss–Seidel Di nuovo, partiamo scrivendo la relazione $M\mathbf{x}^{(k+1)} = \mathbf{b} + N\mathbf{x}^{(k)}$:

$$\begin{bmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ A_{31} & \ddots & \ddots & \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} - \begin{bmatrix} 0 & A_{12} & \dots & A_{1n} \\ & 0 & \ddots & \vdots \\ & & \ddots & A_{n-1,n} \\ & & & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix}.$$

L'equazione corrispondente alla riga i-esima è

$$\sum_{j=1}^{i} A_{ij} x_j^{(k+1)} = b_i - \sum_{j=i+1}^{n} A_{ij} x_j^{(k)}.$$

Risolvendola per $x_i^{(k+1)}$ otteniamo

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{n} A_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} A_{ij} x_j^{(k)}}{A_{ii}}, \quad i = 1, 2, \dots, n. \tag{5.9}$$

Possiamo risolvere queste equazioni una dopo l'altra per $i = 1, 2, \dots, n$ (in quest'ordine!) per calcolare tutti gli elementi di $\mathbf{x}^{(k+1)}$. Questo corrisponde a risolvere il sistema lineare $M\mathbf{x}^{(k+1)} = \mathbf{b} + N\mathbf{x}^{(k)}$ per sostituzione in avanti; il sistema è triangolare inferiore, quindi non ci stupisce che questo sia possibile!

La (5.9) differisce dalla corrispondente formula per il metodo di Jacobi solo per il fatto che abbiamo $x_j^{(k+1)}$ anziché $x_j^{(k)}$ nella prima sommatoria. Questo corrisponde ad usare immediatamente gli elementi più nuovi $x_j^{(k+1)}$ appena li abbiamo calcolati invece di aspettare l'iterazione successiva del metodo. Intuitivamente, questo metodo fornisce un'approssimazione migliore della soluzione, perché usiamo approssimazioni più accurate ad ogni passo. Questa intuizione non sempre corrisponde a realtà; esistono matrici per cui il metodo di Jacobi converge più velocemente di quello di Gauss–Seidel. (Lo vedremo in laboratorio.)

Un'osservazione interessante che il metodo di Gauss–Seidel può essere implementato con *una sola* variabile (vettore) $\mathbf{x}$ che contiene ad ogni passo le approssimazioni più recenti degli elementi di $\mathbf{x}$; ad ogni passo $i = 1, 2, \dots, n$ rimpiazziamo un elemento $x_i^{(k)}$ con $x_i^{(k+1)}$:

$$\mathbf{x}^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_{n-1}^{(k)} \\ x_n^{(k)} \end{bmatrix} \overset{i=1}{\to} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k)} \\ \vdots \\ x_{n-1}^{(k)} \\ x_n^{(k)} \end{bmatrix} \overset{i=2}{\to} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_{n-1}^{(k)} \\ x_n^{(k)} \end{bmatrix} \overset{i=3}{\to} \dots \overset{i=n-1}{\to} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k)} \\ x_3^{(k)} \\ \vdots \\ x_{n-1}^{(k+1)} \\ x_n^{(k)} \end{bmatrix} \overset{i=n}{\to} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k)} \\ x_3^{(k)} \\ \vdots \\ x_{n-1}^{(k+1)} \\ x_n^{(k+1)} \end{bmatrix} = \mathbf{x}^{(k+1)}.$$

Lavorando in questo modo, in ogni momento abbiamo a disposizione tutti gli elementi che ci servono per calcolare l'elemento successivo.

Convergenza per matrici A dominanti diagonali Non è facile prevedere a priori per quali matrici A questi due metodi convergono. Ci sono matrici in

cui uno converge e l'altro no, per esempio. Un risultato che possiamo dimostrare è il seguente.

Teorema 5.13. *Se A è dominante diagonale, i metodi di Jacobi e Gauss–Seidel sono applicabili e convergenti.*

Dimostrazione. Innanzitutto notiamo che i metodi sono applicabili; difatti $A_{ii} \neq 0$:

$$|A_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |A_{ij}| \geq 0, \quad i = 1, 2, \ldots, n.$$

Vogliamo dimostrare che $\rho(H) < 1$, in modo da avere la convergenza. Scriviamo il polinomio caratteristico

$$\det(H - \lambda I) = \det(M^{-1}(N - \lambda M)) = \det M^{-1} \det(N - \lambda M).$$

Poiché $\det M^{-1} \neq 0$, il polinomio caratteristico si annulla solo quando $\det(N - \lambda M) = 0$, cioè $N - \lambda M$ è singolare. Quindi ci basta mostrare che $N - \lambda M$ è invertibile quando $|\lambda| \geq 1$.

La matrice $N - \lambda M$ ha elementi

$$N - \lambda M = - \begin{bmatrix} \lambda A_{11} & A_{12} & \ldots & A_{1n} \\ A_{21} & \lambda A_{22} & \ldots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \ldots & \lambda A_{nn} \end{bmatrix}$$

nel caso del metodo di Jacobi (rispetto a $-A$, la diagonale è moltiplicata per λ), e

$$N - \lambda M = - \begin{bmatrix} \lambda A_{11} & A_{12} & \ldots & A_{1n} \\ \lambda A_{21} & \lambda A_{22} & \ldots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda A_{n1} & \lambda A_{n2} & \ldots & \lambda A_{nn} \end{bmatrix}$$

nel caso del metodo di Gauss–Seidel (la parte triangolare inferiore è moltiplicata per λ). In entrambi i casi, questa matrice è predominante diagonale: difatti, la predominanza diagonale di A assicura che

$$|\lambda A_{ii}| = |\lambda||A_{ii}| > \sum_{j\neq i} |\lambda||A_{ij}|$$

e il membro di destra (quando $|\lambda| \geq 1$) è più grande del termine corrispondente agli elementi fuori dalla diagonale di $N - \lambda M$. □

5.9 Laboratorio su metodi iterativi per sistemi lineari

- Fatto insieme: scrivere una funzione che esegue k iterazioni del metodo di Jacobi. Lasciato a voi: cambiare criterio di arresto; scrivere il metodo di Gauss–Seidel. Fornire matrici di test.
- Controllo della condizione di convergenza per le matrici date (con `M = tril(A)` e `M = diag(diag(A))`).
- Esercizio: scrivere una versione dei due metodi per matrici tridiagonali.

5.10 Sistemi di equazioni non lineari

Introduzione Facciamo qualche accenno anche a metodi per risolvere sistemi di equazioni *non* lineari: data una $F : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$, come possiamo trovare *una* soluzione del sistema?

Esempio (dalle slides di Magherini):

$$F(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} 2x_1 + \cos(x_2) \\ \sin(x_1) + 2x_2 - \pi \end{bmatrix}, \tag{5.10}$$

con una soluzione pari a $(0, \pi)$.

Un sistema di questo tipo è un problema molto più complicato che non un sistema lineare; tanto per cominciare, in generale si possono avere $0, 1$, più soluzioni, infinite soluzioni, e non c'è un criterio generale per capire quando questi casi si verificano.

Non c'è un modo facile di estendere il metodo di bisezione, più che altro perché non abbiamo un analogo facile del teorema di Weierstrass in più dimensioni: anche se $F(x_1, x_2)$ ha tutte le componenti < 0 e $F(y_1, y_2)$ ha tutte le componenti > 0, non è detto che ci sia una soluzione simultanea di tutte le equazioni né nella retta che congiunge (x_1, x_2) a (y_1, y_2) né nel "rettangolo" che li comprende. Un metodo che invece si generalizza in modo efficiente è il metodo di Newton.

Metodo di Newton multivariato Possiamo definire, generalizzando la derivata prima di una funzione, la *matrice Jacobiana*

$$J_F(\mathbf{x}) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1}(x) & \frac{\partial F_1}{\partial x_2}(x) & \dots & \frac{\partial F_1}{\partial x_n}(x) \\ \frac{\partial F_2}{\partial x_1}(x) & \frac{\partial F_2}{\partial x_2}(x) & \dots & \frac{\partial F_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1}(x) & \frac{\partial F_n}{\partial x_2}(x) & \dots & \frac{\partial F_n}{\partial x_n}(x) \end{bmatrix} \in \mathbb{R}^{n \times n}$$

(mnemonica: $F(\mathbf{x})$ produce un vettore colonna, e ogni colonna è una sua derivata rispetto a una variazione di x_j). Per esempio per la (5.10) abbiamo

$$J_F(\mathbf{x}) = \begin{bmatrix} 2 & -\sin(x_2) \\ \cos(x_1) & 2 \end{bmatrix}.$$

Vale (per funzioni sufficientemente regolari) una generalizzazione dello sviluppo di Taylor:

$$F(\mathbf{x} + \mathbf{h}) = F(\mathbf{x}) + J_F(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2);$$

con $\mathbf{x}, \mathbf{h} \in \mathbb{R}^n$; quello nel secondo termine è un prodotto matrice-vettore. Come nel metodo di Newton in una sola variabile, possiamo cercare uno zero dell'approssimazione di F ottenuta trascurando il termine di resto $O(\|\mathbf{h}\|^2)$. Questo produce il metodo

$$\begin{cases} \mathbf{x}^{(0)} \in \mathbb{R}^n & \text{assegnato} \\ \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(J_F(\mathbf{x}^{(k)})\right)^{-1} F(\mathbf{x}^{(k)}), & k = 0, 1, 2, \dots. \end{cases}$$

Valgono risultati analoghi a quelli del caso scalare: su funzioni sufficientemente regolari, il metodo converge a patto di prendere $\mathbf{x}^{(0)}$ sufficientemente vicino a una soluzione $\mathbf{x}$, e lo fa quadraticamente se $J_F(\mathbf{x})$ è invertibile.

Nella pratica, ad ogni passo, non vogliamo calcolare esplicitamente la matrice inversa, ma calcolare $J_F(\mathbf{x}^{(k)})$ e poi risolvere il sistema lineare associato (con una fattorizzazione LU o con altre varianti).

Metodo delle corde multivariato Nel caso multivariato, diventa interessante considerare alcune varianti come il metodo delle corde. Il motivo è che calcolare e fattorizzare la matrice $J_F(\mathbf{x}^{(k)})$ spesso è la parte più costosa del metodo, specialmente per n molto grande. Nella pratica, possiamo scegliere di tenerla fissa come nel caso del metodo delle corde; per esempio per più iterazioni. Pseudocodice: per ogni $k = 0, 1, 2, \ldots$

- Calcolo $F(\mathbf{x}^{(k)})$;
- In alcune iterazioni, valuto $A = J_F(\mathbf{x}^{(k)})$ e calcolo una sua fattorizzazione $PA = LU$; altrimenti, riutilizzo P, L, U da un passo precedente. Per esempio, posso scegliere di ricalcolare la fattorizzazione ogni 5 passi;
- Utilizzo la fattorizzazione per risolvere il sistema lineare $A\mathbf{h} = F(\mathbf{x}^{(k)})$;
- Pongo $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{h}$.

Costo computazionale: ad ogni passaggio dobbiamo fare una valutazione di F e una soluzione di un sistema lineare ($O(n^2)$). Inoltre, ogni volta che vogliamo ricalcolare la fattorizzazione facciamo $O(n^3)$ operazioni più una valutazione di J_F. Impossibile dire qualcosa in generale su come si confrontano senza sapere quanto costa calcolare F e J_F.

Quando il metodo converge, la velocità dipende da quanto spesso si ricalcola la fattorizzazione: se viene calcolata solo una volta all'inizio abbiamo convergenza lineare, se viene calcolata ad ogni iterazione quadratica.

5.11 Sistemi lineari sovradeterminati (minimi quadrati)

Supponiamo di avere un sistema del tipo

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{m \times n},\ \mathbf{b} \in \mathbb{R}^m,\ \mathbf{x} \in \mathbb{R}^n.$$

Se $m > n$ (A alta e stretta), la soluzione spesso non esiste: abbiamo più equazioni che incognite. Dall'algebra lineare, sappiamo che $A\mathbf{x}$ produce una combinazione lineare delle colonne di A, e queste sono solo un sottospazio di $\mathbb{R}^m$ (iperpiano).

[Figura in 3D: piano che corrisponde all'immagine di A, vettore $\mathbf{b}$ al di fuori di esso.]

Possiamo però risolvere un problema diverso, quello di calcolare la combinazione lineare delle colonne di A, cioè il vettore $A\mathbf{x}$, che va più vicino al vettore $\mathbf{b}$. In altre parole, cerchiamo

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|. \tag{5.11}$$

Equazioni normali Chiamiamo *residuo* il vettore $\mathbf{r} = A\mathbf{x} - \mathbf{b}$. Questo problema ha una soluzione particolarmente semplice se usiamo la norma-2. Difatti, minimizzare la norma-2 di $\mathbf{r}$ equivale a minimizzare $\sum r_i^2 = \mathbf{r}^T\mathbf{r}$, ovvero

$$\min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{r}^T\mathbf{r} = \min_{\mathbf{x}\in\mathbb{R}^n} (A\mathbf{x}-\mathbf{b})^T(A\mathbf{x}-\mathbf{b}). \tag{5.12}$$

Il problema (5.12) è detto *problema dei minimi quadrati*, visto che compaiono per l'appunto i quadrati delle componenti del residuo $\mathbf{r}$. Dimostriamo il seguente risultato.

Teorema 5.14. *Sia $\mathbf{x} \in \mathbb{R}^n$ un vettore tale che $A^T\mathbf{r} = A^T(A\mathbf{x}-\mathbf{b}) = 0$. Allora, $\mathbf{x}$ risolve il problema di minimo* (5.12)*.*

Dimostrazione. Prendiamo un qualunque vettore diverso da $\mathbf{x}$: possiamo scriverlo come $\mathbf{x}+\mathbf{z}$, con $\mathbf{z} \in \mathbb{R}^n, \mathbf{z} \neq 0$; il suo residuo vale

$$A(\mathbf{x}+\mathbf{z}) - \mathbf{b} = \underbrace{A\mathbf{x}-\mathbf{b}}_{=\mathbf{r}} + \underbrace{A\mathbf{z}}_{=\mathbf{s}} = \mathbf{r}+\mathbf{s}.$$

Vogliamo dimostrare che la norma di questo residuo è maggiore o uguale a quella di $\mathbf{r}$; questo ci permette di concludere che $\mathbf{x}$ è la soluzione del problema. Il quadrato di questa norma vale

$$\|A(\mathbf{x}+\mathbf{z})-\mathbf{b}\|^2 = (\mathbf{r}+\mathbf{s})^T(\mathbf{r}+\mathbf{s}) = \mathbf{s}^T\mathbf{s} + \mathbf{s}^T\mathbf{r} + \mathbf{r}^T\mathbf{s} + \mathbf{r}^T\mathbf{r}.$$

Notiamo però che il prodotto scalare $\mathbf{r}^T\mathbf{s} = \mathbf{s}^T\mathbf{r}$ si annulla: difatti,

$$\mathbf{s}^T\mathbf{r} = (A\mathbf{z})^T\mathbf{r} = \mathbf{z}^TA^T\mathbf{r} = 0.$$

Allora si ha

$$\|A(\mathbf{x}+\mathbf{z})-\mathbf{b}\|^2 = \mathbf{s}^T\mathbf{s} + \mathbf{r}^T\mathbf{r} \geq \mathbf{r}^T\mathbf{r}.$$

Difatti $\mathbf{s}^T\mathbf{s} = s_1^2 + s_2^2 + \cdots + s_m^2 \geq 0$. □

Quindi per trovare la soluzione $\mathbf{x}$ di questo problema ci basta risolvere il sistema di equazioni

$$A^TA\mathbf{x} = A^T\mathbf{b}, \tag{5.13}$$

che si ottiene moltiplicando per A^T il sistema originale (che era rettangolare e quindi potenzialmente senza soluzione) $A\mathbf{x} = \mathbf{b}$. La (5.13) si chiama *metodo delle equazioni normali*, perché le equazioni corrispondenti dicono (in termini geometrici) che il residuo $\mathbf{r} = A\mathbf{x} - \mathbf{b}$ è ortogonale (prodotto scalare nullo) alle colonne di A.

È semplice vedere che A^TA è quadrata e simmetrica; inoltre, si può dimostrare che è positiva definita (quindi invertibile!) tutte le volte che le colonne di A sono linearmente indipendenti. Possiamo allora risolvere (5.13) usando la fattorizzazione di Cholesky.

Questo metodo però in alcuni casi risulta instabile: il condizionamento del sistema lineare (5.13) può essere molto più alto di quello del problema originale (che non abbiamo studiato). Esiste un altro metodo più costoso ma più stabile.

Fattorizzazione QR Si può dimostrare (noi non lo facciamo) il seguente risultato.

Teorema 5.15. *Per ogni $A \in \mathbb{R}^{m\times n}$, esistono una matrice $Q \in \mathbb{R}^{m\times m}$* ortogonale *(cioè che soddisfa $Q^TQ = I$) e una matrice $R \in \mathbb{R}^{m\times n}$ triangolare superiore tali che $A = QR$.*

La matrice R è rettangolare: quando scriviamo "triangolare" intendiamo che

$$R = \begin{bmatrix} R_1 \\ O \end{bmatrix},$$

dove $R_1 \in \mathbb{R}^{n\times n}$ è una "normale" matrice quadrata triangolare, e $O \in \mathbb{R}^{(m-n)\times n}$ è un blocco di zeri. Se suddividiamo nello stesso modo

$$Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}, \quad Q_1 \in \mathbb{R}^{m\times n},\, Q_2 \in \mathbb{R}^{m\times(m-n)},$$

vediamo che gli elementi di Q_2 si "scontrano" con il blocco di zeri quando facciamo il prodotto, quindi $A = QR = Q_1R_1$.

Con alcune manipolazioni algebriche (che non vediamo nel dettaglio) si può vedere che la soluzione x delle (5.13) è una soluzione anche del sistema lineare triangolare

$$R_1x = Q_1^Tb. \tag{5.14}$$

Questo ci suggerisce un altro algoritmo per la soluzione del problema dei minimi quadrati:

- Calcoliamo la fattorizzazione $A = QR$ (o anche solo i blocchi $A = Q_1R_1$).
- Risolviamo per sostituzione all'indietro il sistema (5.14).

Questo metodo è più costoso del precedente (facendo un'analisi più accurata, è possibile dimostrare che quando $m \gg n$ richiede $2mn^2$ operazioni più termini di ordine inferiore, contro mn^2 per il metodo delle equazioni normali) ma è più stabile in alcuni problemi in cui A^TA è mal condizionata.

Soluzione tramite Matlab In Matlab, risolvere un problema ai minimi quadrati (5.11) è semplice: basta il comando `A \ b`, lo stesso che avreste usato per risolvere il sistema lineare se A fosse quadrata. Matlab usa il metodo più stabile basato sulla fattorizzazione QR.

Capitolo 6

Interpolazione, approssimazione, e integrazione numerica

6.1 Approssimazione e interpolazione

In diversi problemi applicativi, abbiamo una serie di misurazioni che corrispondono a punti del piano $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$, e siamo interessati a determinare una funzione che passa (esattamente o approssimativamente) per questi punti.

Ad esempio: supponiamo di avere un impianto industriale in cui stiamo eseguendo una reazione chimica, e di averne misurato la temperatura per 13 volte successive, alle 0:00, alle 1:00, alle 2:00, e così via fino alle 12:00. In che modo possiamo trovare una stima della temperatura a un'altra ora, ad esempio alle 5:45? O anche in un'ora al di fuori dell'intervallo di misurazione, per esempio alle 14:00? (In questo caso a volte si usa il termine *estrapolazione*.)

Magari abbiamo dei modelli che ci dicono che la temperatura decresce linearmente o esponenzialmente, ma questi modelli dipendono da parametri che dobbiamo calcolare. Oppure qualche volta non abbiamo proprio un modello e stiamo genericamente cercando una funzione 'semplice' che approssima i valori trovati.

- Si chiama *interpolazione* il problema di trovare una funzione ϕ (all'interno di una certa classe) tale che $\phi(x_i) = y_i$ per ogni $i = 1, 2, \ldots, n$.
- Si chiama *approssimazione* (o, dall'inglese, *fit*) il problema di trovare una funzione ϕ tale che $\phi(x_i) \approx y_i$, minimizzando un qualche tipo di residuo.

6.2 Interpolazione polinomiale

Vediamo più nel dettaglio il problema dell'*interpolazione polinomiale*. Supponiamo di avere $n+1$ punti dati nel piano $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ (attenzione: partiamo da 0, quindi i punti sono $n + 1$) e di voler trovare un polinomio di

grado minore o uguale a n il cui grafico passa esattamente per questi punti; cioè, stiamo cercando coefficienti incogniti $\alpha_0, \alpha_1, \ldots, \alpha_n$ tali che il polinomio

$$p(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_n x^n$$

soddisfa le relazioni

$$p(x_i) = y_i, \quad i = 0, 1, \ldots, n. \tag{6.1}$$

I punti x_i qui sono detti *nodi* dell'interpolazione, e i punti y_i *valori* da interpolare. Possiamo immaginare che questi punti y_i siano calcolati come $y_i = f(x_i)$ a partire da una funzione f più complicata da calcolare (perché è data da una formula più complicata, o perché è il risultato di misurazioni che non possiamo ripetere); in questo modo costruiamo un polinomio che fa da 'modello' più semplice di questa funzione.

Possiamo osservare che le relazioni (6.1) sono equivalenti a un sistema lineare di $n+1$ equazioni nelle $n+1$ incognite $\alpha_0, \ldots, \alpha_n$:

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots x_0^n \\ 1 & x_1 & x_1^2 & \ldots x_1^n \\ 1 & x_2 & x_2^2 & \ldots x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots x_n^n \end{bmatrix}}_{=X} \begin{bmatrix} \alpha_0 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

(Non ci facciamo ingannare dai nomi: gli x_i e gli y_i non sono variabili; qui sono noti, e sono i dati del problema.)

La matrice associata a questo sistema, che abbiamo indicato con X, si chiama *matrice di Vandermonde*.

Risolubilità È possibile dimostrare il seguente risultato.

Teorema 6.1. *Per ogni scelta di* $x_0, x_1, \ldots, x_n$ distinti*, la matrice di Vandermonde è invertibile.*

Dimostrazione. (*) Sia

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}$$

un vettore nel kernel della matrice X, cioè tale che $X\mathbf{a} = \mathbf{0}$. Vogliamo dimostrare che dev'essere $\mathbf{a} = \mathbf{0}$; questo implica che la matrice X è invertibile.

Se scriviamo l'uguaglianza $X\mathbf{a} = \mathbf{0}$ riga per riga, vediamo che essa corrisponde a

$$1 \cdot a_0 + x_i \cdot a_1 + x_i^2 \cdot a_2 + \cdots + x_i^n \cdot a_n = 0, \quad i = 0, 1, \ldots, d;$$

Questo equivale a dire che il polinomio $a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ si annulla nei punti $x_0, x_1, \ldots, x_n$. Ma dalle proprietà dei polinomi sappiamo che un polinomio di grado al più n che si annulla in $n+1$ punti dev'essere

necessariamente il polinomio nullo[1]. Questo quindi dimostra che $\mathbf{a} = 0$ e quindi che X è invertibile. □

Dall'invertibilità di questa matrice segue che il problema dell'interpolazione polinomiale è sempre risolubile, cioè vale questo risultato.

Teorema 6.2 (esistenza e unicità del polinomio di interpolazione)**.** *Date $n+1$ coppie di punti $(x_0, y_0), \dots, (x_n, y_n)$, con gli x_i tutti diversi tra loro, esiste uno e un solo polinomio $p(x)$ di grado minore o uguale a d tale che $p(x_i) = y_i$ per ogni $i = 0, 1, \dots, n$.*

Notare che sul grado abbiamo solo una disuguaglianza: nulla vieta che il coefficiente α_n (o anche quelli precedenti) sia uguale a zero per una particolare scelta dei punti (x_i, y_i). Per esempio, se scelgo $(0,0), (1,1), (2,2)$, allora il polinomio di grado $n \leq 2$ che passa esattamente per questi punti è $p(x) = x$, che in realtà ha grado 1.

Solitamente l'interpolazione polinomiale si usa con pochi nodi; $n \leq 10$ per esempio. Questo perché per valori grandi di n in molti casi la matrice di Vandermonde, seppure invertibile, risulta mal condizionata. Questo significa che a piccole variazioni dei dati x_i o y_i corrispondono grandi variazioni nei coefficienti del polinomio. Pertanto i coefficienti calcolati sono affetti da un errore inerente molto grande. Un altro problema è che, anche in aritmetica esatta, spesso al crescere di n il polinomio di interpolazione di una funzione f soddisfa sì $p(x_i) = f(x_i)$, ma i valori di p e di f al di fuori di questi valori possono differire anche di molto, con il polinomio che oscilla eccessivamente. [TODO: Immagine di esempio] In molti casi, un polinomio di approssimazione di grado basso (come quelli che vedremo più sotto) fornisce risultati graficamente migliori che non un polinomio di interpolazione. Questo fenomeno è detto *overfitting*.

Polinomi di Lagrange Possiamo dare una formula esplicita per la soluzione del problema dell'interpolazione polinomiale. Fissati i nodi distinti $x_0, x_1, \dots, x_n$, definiamo i *polinomi di Lagrange*

$$L_k(x) = \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}, \quad k = 1, 2, \dots, n.$$

Ad esempio, se i nodi sono $x_0 = 1, x_1 = 2, x_2 = 4$, abbiamo

$$L_0(x) = \frac{(x-2)(x-4)}{(1-2)(1-4)}, \quad L_1(x) = \frac{(x-1)(x-4)}{(2-1)(2-4)}, \quad L_2(x) = \frac{(x-1)(x-2)}{(4-1)(4-2)}.$$

Notare che il denominatore non si annulla mai se i nodi sono distinti, e che sono tutti polinomi di grado n, visto che il numeratore è il prodotto di n fattori di grado 1. Inoltre, vale il seguente risultato.

Lemma 6.3. *Si ha*

$$L_k(x_i) = \begin{cases} 1 & i = k, \\ 0 & \textit{altrimenti.} \end{cases}$$

[1] Per il teorema di Ruffini, un polinomio che si annulla in $x_0, x_1, \dots, x_n$ è un multiplo del polinomio $(x - x_0)(x - x_1) \dots (x - x_n)$; quindi o è il polinomio nullo, oppure ha grado almeno $n + 1$.

Dimostrazione. Sostituendo $x = x_k$, numeratore e denominatore diventano identici, quindi il polinomio vale 1. Sostituendo $x = x_i$ con $i \neq k$, uno dei fattori nella produttoria al numeratore diventa $(x_i - x_i)$, quindi $L_k(x_i) = 0$. □

Quindi il k-esimo polinomio di Lagrange fornisce la soluzione al problema di interpolazione con $\mathbf{y} = \mathbf{e}_k$ (k-esimo vettore della base canonica). Con queste soluzioni, possiamo ottenere la soluzione a un problema di interpolazione polinomiale generico.

Teorema 6.4. *Siano $(x_0, y_0), \ldots, (x_n, y_n)$ dati. La soluzione del problema di interpolazione polinomiale (cioè l'unico polinomio p di grado $\leq n$ tale che $p(x_i) = y_i$ per ogni $i = 0, 1, \ldots, n$) è data da*

$$p(x) = \sum_{k=0}^{n} y_k L_k(x).$$

Dimostrazione. Basta verificare che $p(x_i) = y_i$ per ogni i (cosa che segue dal Lemma 6.3) e che $p(x)$ ha grado minore o uguale a n (perché è una combinazione lineare dei polinomi di Lagrange, che hanno tutti grado n). □

Resto dell'interpolazione Vale il seguente risultato.

Teorema 6.5. *Sia $f \in \mathcal{C}^{n+1}([a, b])$, $x_0, x_1, \ldots, x_n$ nodi distinti in $[a, b]$, e $p(x)$ il polinomio di interpolazione (di grado al più n) di f sui nodi dati. Allora per ogni $x \in [a, b]$ esiste un punto $\xi \in (a, b)$ tale che*

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1)\ldots(x - x_n).$$

Non lo dimostriamo ma facciamo qualche commento. Notare che questo risultato "assomiglia" a uno sviluppo di Taylor con resto di Lagrange: abbiamo che $f(x)$ è uguale a un polinomio $p(x)$ (che arriva ad avere potenze fino al grado n) più un resto che dipende dalla derivata $n+1$-esima. Poiché abbiamo $n+1$ punti, anziché un "centro" solo dello sviluppo, nella formula per il resto $\frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}$, il termine $(x - x_0)^{n+1}$ viene rimpiazzato da $\prod_{j=0}^{n}(x - x_i)$.

Non ci stupisce che ci siano dei fattori $(x - x_i)$ nel membro di destra: difatti, se sostituiamo $x = x_i$, per un qualche $i = 0, 1, \ldots, n$, il termine di sinistra si annulla, quindi deve annullarsi anche quello di destra.

Un altro caso speciale in cui possiamo verificare il risultato direttamente è quello in cui $f(x)$ è essa stessa un polinomio di grado minore o uguale a n. In questo caso il polinomio di interpolazione coincide con $f(x)$ stessa, quindi il termine di sinistra è nullo per ogni x; e anche il termine di destra si annulla perché per un polinomio di grado al più n abbiamo $f^{(n+1)} \equiv 0$.

Da questa formula segue una stima per l'errore massimo (differenza tra $f(x)$ e $p(x)$), cioè

$$|f(x) - p(x)| \leq \frac{C_{n+1}}{(n+1)!}(b - a)^{n+1},$$

dove $C_{n+1} = \max_{x \in [a,b]} |f^{(n+1)}(x)|$.

6.3 Altri problemi di interpolazione e approssimazione

Quello dell'interpolazione polinomiale è un problema che si rivela sorprendentemente lineare: anche se nella formulazione del problema compaiono polinomi di grado più alto di 1, la dipendenza dai coefficienti incogniti $\alpha_0, \dots, \alpha_n$ è lineare.

Possiamo risolvere nello stesso modo un problema più generale: fissiamo n funzioni $\phi_1(x), \dots, \phi_n(x)$, e cerchiamo coefficienti incogniti $\alpha_1, \dots, \alpha_n$ in modo da ottenere una funzione

$$\phi(x) = \alpha_1\phi_1(x) + \alpha_2\phi_2(x) + \cdots + \alpha_n\phi_n(x) \tag{6.2}$$

che rispetti certe condizioni.

Più nel dettaglio, supponiamo di avere a disposizione m coppie di valori $(x_1, y_1), \dots, (x_m, y_m)$ (e, di nuovo, assumiamo che gli x_i siano *distinti*.

Se imponiamo le condizioni $\phi(x_i) = y_i$ per $i = 1, 2, \dots, m$, abbiamo il seguente sistema lineare di m equazioni in n incognite

$$\underbrace{\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \dots & \phi_n(x_m) \end{bmatrix}}_{=X} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}}_{=\alpha} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}}_{=\mathbf{y}}$$

Se $m = n$, cioè se ho tanti punti quante funzioni incognite, questo è un sistema lineare quadrato e posso risolverlo esattamente (ammesso che la matrice X sia invertibile). Questa problema si chiama *interpolazione* di funzioni.

Approssimazione / fit Il caso più comune però è che $m > n$, e quindi ho più equazioni che incognite. Posso risolvere il sistema nel senso dei minimi quadrati come visto nella sezione precedente; questo corrisponde a risolvere questo problema: sono date delle coppie (x_i, y_i) che soddisfano approssimativamente $\phi(x_i) \approx y_i$, per una funzione ϕ come in (6.2), e voglio trovare i coefficienti $\alpha_1, \dots, \alpha_n$ che risolvono il problema di minimo

$$\min_{\alpha \in \mathbb{R}^n} (\phi(x_i) - y_i)^2.$$

Questo problema si chiama *approssimazione* di funzioni, o, dall'inglese, *fitting*.

I problemi di interpolazione e di approssimazione si risolvono utilizzando gli algoritmi per risolvere sistemi lineari e sistemi sovradeterminati, che abbiamo visto nella sezione precedente; non ci servono strumenti nuovi. Vediamo un po' di casi particolari.

Polinomi trigonometrici Se desideriamo interpolare (o approssimare) una funzione che sappiamo essere periodica di periodo π, una base "naturale" da

usare è

$$\begin{aligned}
&\phi_1(x) = 1, && \\
&\phi_2(x) = \sin x, && \phi_3(x) = \cos x, \\
&\phi_4(x) = \sin 2x, && \phi_5(x) = \cos 2x, \\
&\quad\vdots && \quad\vdots \\
&\phi_{2k}(x) = \sin kx, && \phi_{2k+1}(x) = \cos kx.
\end{aligned}$$

Queste sono, in un certo senso, le funzioni periodiche "più semplici" di periodo π. Non vediamo i dettagli qui, ma è un approccio molto usato, e collegato alla cosiddetta *trasformata discreta di Fourier*, che probabilmente avrete occasione di incontrare in futuro in altri corsi.

Retta dei minimi quadrati Corrisponde a trovare la retta che approssima meglio un insieme di punti dati (disegno, mostrando che lo scarto misurato è "in verticale"). In questo caso, le funzioni sono $\phi_1(x) = 1, \phi_2(x) = x$. A differenza dell'interpolazione polinomiale, però, i punti sono solitamente più di due, e quindi non esiste una retta che passa esattamente per tutti questi punti. Quindi

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix}, \quad \alpha = (X^T X)^{-1} X^T y,$$

dove l'ultima formula corrisponde ad usare le equazioni normali per risolvere questo problema. Possiamo ottenere una formula più esplicita espandendo i conti e usando la formula per l'inversa di una matrice 2×2; questo spesso viene fatto se avete già visto la retta di interpolazione in altri corsi; per noi rifare questo conto non è particolarmente interessante.

Approssimazione polinomiale Analogamente a quanto fatto con la retta dei minimi quadrati, possiamo fissare un grado massimo $n - 1$, e trovare il polinomio di grado al più $n - 1$ che meglio approssima una sequenza di punti. Questo corrisponde a scegliere le n funzioni di base

$$\phi_1(x) = 1, \quad \phi_2(x) = x, \quad \phi_3(x) = x^2, \ldots, \phi_n(x) = x^{n-1}. \tag{6.3}$$

6.4 Formule di integrazione

In questa sezione consideriamo il problema di approssimare numericamente l'integrale di una funzione, $I = \int_a^b f(x)dx$. Questo problema si chiama *integrazione numerica*, o *quadratura*.

Sappiamo dall'analisi che il problema di calcolare esattamente integrali (con una formula risolutiva) è un problema difficile; a differenza delle derivate, non ci sono delle formule da applicare meccanicamente ma una serie di tecniche per trovare primitive o integrali definiti, che a volte funzionano e a volte no. Per questo diventa prezioso avere degli algoritmi numerici approssimati che funzionino bene.

Le formule che vedremo sono tutte del tipo

$$I \approx \sum_{i=1}^{n} w_i f(x_i),$$

dove w_i sono detti *pesi* e x_i *nodi* della formula, e sono scelti indipendentemente da f (ma tipicamente dipendono dall'intervallo $[a, b]$).

Spesso a queste formule chiediamo che siano esatte su alcune funzioni semplici; per esempio, polinomi di grado basso. Notiamo che una formula è esatta sulla funzione $f(x) \equiv 1$ se $\sum_{i=1}^{n} w_n = b - a$, per esempio.

Partiamo analizzando due formule semplici.

Formula del punto medio È la formula

$$I_M = (b - a)f(c),$$

dove $c = \frac{a+b}{2}$ è il punto medio dell'intervallo di integrazione. Corrisponde a rimpiazzare l'integrale con l'area del rettangolo di base $[a, b]$ e altezza $f(m)$, quindi una somma di Riemann con un solo rettangolo. Notiamo che se $f(x)$ è una funzione lineare (polinomio di grado 1), allora questa formula calcola esattamente l'integrale. Difatti, [disegno] la differenza tra i due integrali è pari all'area di due triangoli uguali, una presa con il segno + e una presa con il segno −.

Diremo che una formula di integrazione numerica ha *grado di precisione* (o *di esattezza, di accuratezza*) d se fornisce il valore esatto dell'integrale per tutte le funzioni f che sono polinomi di grado d o inferiore. Come nel caso della convergenza, si dice *esattamente* d o *al più* d a seconda se imponiamo o no che la formula *non* sia esatta per grado $d + 1$.

Quindi la formula del punto medio ha grado di precisione 1. È facile verificare che questo grado è *esattamente* 1: se prendiamo un polinomio di grado 2, per esempio x^2, la formula non fornisce più valori esatti.

Possiamo dimostrare una formula per l'errore $I_M - I$ in termini della derivata seconda di f.

Teorema 6.6. *Sia $f \in \mathcal{C}^2([a, b])$. Allora,*

$$|I_M - I| \leq \frac{1}{24} C_2 (b - a)^3,$$

dove $C_2 = \max_{x \in [a,b]} |f''(x)|$.

Dimostrazione. Scriviamo uno sviluppo di Taylor in c di ordine 2,

$$f(x) = f(c) + f'(c)(x - c) + \frac{1}{2} f''(\xi)(x - c)^2.$$

Ora sostituiamo

$$\begin{aligned} I_M - I &= f(c)(b - a) - \int_a^b (f(c) + f'(c)(x - c) + \frac{1}{2} f''(\xi)(x - c)^2) dx \\ &= \underbrace{f(c)(b - a) - \int_a^b (f(c) + f'(c)(x - c)) dx}_{=0} - \int_a^b \frac{1}{2} f''(\xi)(x - c)^2 dx, \end{aligned}$$

dove il primo termine si annulla perché la formula ha grado di precisione 1, quindi calcola esattamente l'integrale della funzione $f(c) + f'(c)(x-c)$, che è un polinomio di grado 1 che assume il valore $f(c)$ per $x = c$. Possiamo allora stimare l'errore come

$$|I_M - I| = \left| \int_a^b \frac{1}{2} f''(\xi)(x-c)^2 dx \right| \leq \int_a^b \frac{1}{2} |f''(\xi)|(x-c)^2$$
$$\leq \frac{1}{2} \int_a^b C_2 (x-c)^2 dx = \frac{1}{24} C_2 (b-a)^3,$$

visto che $\int_a^b (x-c)^2 dx = \frac{1}{12}(b-a)^3$ (è un semplice conto sull'integrale di un polinomio di grado 2). □

Con un po' più di attenzione, possiamo dimostrare un risultato più forte: se $m \leq f''(x) \leq M$ (cioè m e M sono il minimo e il massimo di $f''(x)$) allora

$$\frac{m}{12}(b-a)^3 \leq \int_a^b m(x-c)^2 dx \leq \int_a^b f''(x)(x-c)^2 dx \leq \int_a^b M(x-c)^2 dx \leq \frac{M}{12}(b-a)^3,$$

e quindi esiste un valore di ξ per cui

$$I_M - I = -\frac{f''(\xi)}{24}(b-a)^3.$$

Questa formula più precisa ci dà anche informazioni sul segno: se la derivata seconda di f è sempre positiva (cioè f è convessa), allora l'errore è negativo, e I_M è più piccola dell'integrale vero; e al contrario se la funzione è concava, allora l'errore è positivo.

Formula dei trapezi È la formula $I \approx I_T = \frac{b-a}{2}(f(a) + f(b))$; cioè, rimpiazziamo l'integrale con l'area del trapezio con basi $f(a)$ e $f(b)$ (disegno sulla lavagna). Abbiamo due pesi $w_1 = w_2 = \frac{b-a}{2}$ e due nodi a, b. Questa formula calcola esattamente l'integrale I_T della retta di interpolazione per f con i due nodi $x_1 = a$, $x_2 = b$. In particolare, da questo segue che ha grado di precisione almeno 1. (Di nuovo, è facile vedere che il grado è *esattamente* 1). Possiamo dimostrare un risultato analogo a quello visto poco fa per il metodo del punto medio.

Teorema 6.7. *Sia $f \in \mathcal{C}^2([a,b])$. Allora,*

$$|I_T - I| \leq \frac{1}{12} C_2 (b-a)^3,$$

dove $C_2 = \max_{x \in [a,b]} |f''(x)|$.

Notare che il coefficiente è il doppio di quello ottenuto per la formula del punto medio.

Dimostrazione. Scriviamo la formula del resto dell'interpolazione,

$$f(x) - p(x) = \frac{f''(\xi)}{2}(x-a)(x-b),$$

dove $p(x)$ è la retta che interpola $f(x)$ nei due nodi a e b; abbiamo detto poco fa che $I_T = \int_a^b f(x)dx$. Allora,

$$I_T - I = \int_a^b (p(x) - f(x))dx = \int_a^b \frac{f''(\xi)}{2}(x-a)(b-x)dx,$$

dove abbiamo cambiato segno a $x - b$. Scegliendo i segni in questo modo, $(x-a)(b-x) \geq 0$ per ogni $x \in [a,b]$, e quindi è sempre uguale al suo valore assoluto. Ora possiamo scrivere

$$\begin{aligned} |I_T - I| &= \left| \int_a^b \frac{f''(\xi)}{2}(x-a)(b-x)dx \right| \leq \int_a^b \frac{|f''(\xi)|}{2}(x-a)(x-b)dx \\ &\leq \frac{C_2}{2}\int_a^b (x-a)(x-b)dx = \frac{C_2}{12}(b-a)^3, \end{aligned}$$

visto che $\int_a^b (x-a)(x-b)dx = \frac{1}{6}(b-a)^3$ (di nuovo è un conto su un integrale di un polinomio di grado 2). □

Come prima, un ragionamento più accurato porta a

$$I_T - I = \frac{f''(\xi)}{12}(b-a)^3.$$

Questa volta la formula calcola qualcosa di più *grande* per funzioni convesse, e più *piccolo* per funzioni concave; cosa che è evidente anche da un disegno. (disegno sulla lavagna).

Formule di Newton–Cotes La dimostrazione che abbiamo fatto suggerisce una strategia generale: per calcolare un integrale, possiamo scegliere $n+1$ nodi in $[a,b]$, calcolare il polinomio di interpolazione $p(x)$ su questi nodi, e rimpiazzare $I = \int_a^b f(x)dx$ con $I_n = \int_a^b p(x)dx$. Questo ci fornisce una formula con grado di esattezza almeno n, visto che se $f(x)$ è un polinomio di grado al più n allora coincide con il suo polinomio di interpolazione. Le *formule di Newton–Cotes* corrispondono a prendere $n+1$ nodi equispaziati $x_0 = a, x_1 = a+h, \ldots x_n = b$, con $h = \frac{b-a}{n}$. Notare che abbiamo cambiato leggermente gli indici rispetto a quanto fatto in precedenza, in modo da chiamare n il numero di intervalli, e quindi $n+1$ il numero di punti.

Utilizzando la forma di Lagrange del polinomio di interpolazione, abbiamo

$$I_n = \int_a^b p(x)dx = \int_a^b \sum_{k=0}^n L_k(x)f(x_k)dx = \sum_{k=0}^n f(x_k) \underbrace{\int_a^b L_k(x)dx}_{=w_k}.$$

Calcoliamo per esempio i pesi che risultano per $[a,b] = [-1,1]$ e $n = 2$ intervalli: i tre punti equispaziati sono $x_0 = -1, x_1 = 0, x_2 = 1$, e

$$\begin{aligned} L_0(x) &= \frac{(x-0)(x-1)}{(-1-0)(-1-1)} = \frac{x(x-1)}{2}, & w_0 &= \int_{-1}^1 L_0(x) = \frac{1}{3}, \\ L_1(x) &= \frac{(x+1)(x-1)}{(0+1)(0-1)} = 1-x^2, & w_1 &= \int_{-1}^1 L_0(x) = \frac{4}{3}, \\ L_2(x) &= \frac{(x+1)(x-0)}{(1+1)(1-0)} = \frac{(x+1)x}{2}, & w_2 &= \int_{-1}^1 L_0(x) = \frac{1}{3}. \end{aligned}$$

Cambio di variabile A priori, sembrerebbe che per ogni scelta dell'intervallo $[a, b]$ dobbiamo ricalcolare da capo queste formule. È possibile però fare un cambio di variabile lineare che ci permette di ricondurre un generico intervallo $[a, b]$ all'intervallo $[-1, 1]$. Definiamo come in precedenza $c = \frac{a+b}{2}$ il punto medio di $[a, b]$, e $x = c + \frac{b-a}{2}y$. È semplice verificare che $y = -1, y = 1$ corrispondono a $x = a, x = b$ rispettivamente.

Quindi abbiamo $dx = \frac{b-a}{2}dy$ e

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^1 f\left(\frac{2}{b-a}(x-c)\right)dy.$$

Questa formula di cambio di variabile si può utilizzare non solo per integrare la funzione f, ma anche per integrare i polinomi di Lagrange $L_k(x)$.

Inoltre, visto che il cambio di variabile è lineare, $n+1$ punti equispaziati $x_0, \ldots, x_n$ su $[a, b]$ diventano $n+1$ punti equispaziati $y_0, \ldots, y_n$ su $[-1, 1]$, e il polinomio di Lagrange $L_k(x)$ costruito sui punti $x_0, \ldots, x_n$ diventa il polinomio di Lagrange $L_k(y)$ costruito sui punti $y_0, \ldots, y_n$. Questo ci dice che per un intervallo generico i pesi di I_2 saranno

$$w_0 = \frac{1}{6}(b-a), \quad w_1 = \frac{4}{6}(b-a), \quad w_2 = \frac{1}{6}(b-a).$$

Otteniamo quindi, per $n = 2$, la formula

$$I \approx I_2 = \frac{b-a}{6}\left(f(a) + 4f(c) + f(b)\right),$$

che è detta *formula di Cavalieri–Simpson*. Analogamente è possibile costruire formule di Newton–Cotes di grado più alto.

Grado di precisione delle formule di Newton–Cotes La formula di Cavalieri–Simpson che abbiamo costruito ha necessariamente grado di esattezza almeno 2: difatti, corrisponde a prendere il polinomio di interpolazione $p(x)$ a $f(x)$ di grado al più 2, e calcolarne l'integrale; quindi se $f(x)$ è già di partenza un polinomio di grado al più 2 riotteniamo l'integrale di partenza. La cosa sorprendente è che questa formula in realtà ha grado di esattezza 3, cioè restituisce il risultato esatto anche per polinomi di grado 3. Possiamo verificare facilmente che per $[a, b] = [-1, 1]$ e $f(x) = x^3$, per simmetria $I = \int_a^b f(x)dx = 0$, e la formula di Cavalieri–Simpson restituisce, correttamente, $I_2 = 0$. Non vediamo i dettagli, ma poiché sia gli integrali che le nostre formule di integrazione sono lineari nella f, questo è sufficiente per concludere che la formula è esatta per *ogni* polinomio di grado 3.

Più in generale, per le formule di Newton–Cotes valgono espressioni dell'errore simili a quella dimostrata per il metodo dei trapezi.

Teorema 6.8. *La formula di Newton–Cotes di grado n (con $n+1$ punti equispaziati) ha grado di esattezza*

$$d = \begin{cases} n & n \text{ dispari}, \\ n+1 & n \text{ pari}. \end{cases}$$

Inoltre, per l'errore vale la formula

$$I_n - I = \kappa_n C_{d+1}(b-a)^{d+2},$$

dove κ_n è un'opportuna costante.

Per il metodo dei trapezi, già sappiamo che $\kappa_1 = \frac{1}{12}$. Per il metodo di Cavalieri–Simpson, vale $\kappa_2 = \frac{1}{90}$ (ma non lo dimostriamo, così come non dimostriamo il teorema).

6.5 Formule di integrazione composite

In realtà, utilizzare formule di Newton–Cotes con valori grandi di n non è una buona idea, visto che si incorre negli stessi problemi di cattivo condizionamento che abbiamo visto con l'interpolazione polinomiale. Tipicamente, si scelgono formule di grado al massimo 4, e se si vuole ridurre ancora l'errore si usa una strategia diversa.

Dividiamo l'intervallo $[a, b]$ in N intevalli uguali, scegliendo gli $N + 1$ punti equispaziati

$$x_0 = a, x_1 = a + h, \ldots, x_n = a + nh, \ldots, x_N = b,$$

con $h = \frac{b-a}{N}$ la lunghezza di ogni intevallo. (Stiamo riutilizzando la stessa notazione della sezione precedente, anche se qui utilizziamo in modo diverso gli N sottointervalli uguali in cui abbiamo diviso $[a, b]$.)

Abbiamo

$$\int_a^b f(x)dx = \sum_{n=1}^{N} \int_{x_{n-1}}^{x_n} f(x)dx.$$

Poi approssimiamo ognuno degli n integrali nel termine di destra con una delle formule viste al passo precedente. In questo modo otteniamo i seguenti metodi.

Metodo del punto medio composito

$$I_{M,N} = \sum_{n=1}^{N} \underbrace{(x_n - x_{n-1})}_{h} f\left(\frac{x_{n-1} + x_n}{2}\right) = \frac{b-a}{n} \sum_{n=1}^{N} f\left(\frac{x_{n-1} + x_n}{2}\right).$$

Metodo dei trapezi composito

$$I_{T,N} = \sum_{n=1}^{N} \frac{x_n - x_{n-1}}{2} \left(f(x_{n-1}) + f(x_n)\right) = \frac{b-a}{2n} \sum_{n=1}^{N} \left(f(x_{n-1}) + f(x_n)\right).$$

Metodo di Cavalieri–Simpson composito

$$\begin{aligned} I_{2,N} &= \sum_{n=1}^{N} \frac{x_n - x_{n-1}}{6} \left(f(x_{n-1}) + 4f\left(\frac{x_{n-1} + x_n}{2}\right) + f(x_n)\right) \\ &= \frac{b-a}{6n} \sum_{n=1}^{N} \left(f(x_{n-1}) + 4f\left(\frac{x_{n-1} + x_n}{2}\right) + f(x_n)\right) \\ &= \frac{2}{3} I_{M,N} + \frac{1}{3} I_{T,N}. \end{aligned}$$

Il costo del metodo del punto medio composito è N valutazioni di funzione, nei punti medi di ognuno degli intervalli della suddivisione. (Più $O(N)$ somme che sono solitamente trascurabili rispetto alle valutazioni di funzione.)

Il costo del metodo dei trapezi apparentemente è di $2N$ valutazioni di funzione, però notiamo che tutti i punti a parte il primo e l'ultimo compaiono due volte: una volta come estremo destro di un intervallo, una volta come estremo sinistro. Quindi possiamo riscrivere la sommatoria che compare in $I_{T,N}$ come

$$\frac{1}{2}\sum_{n=1}^{N}(f(x_{n-1})+f(x_n)) = \frac{1}{2}f(x_0)+\frac{1}{2}f(x_N)+\sum_{n=1}^{N-1}f(x_n),$$

che richiede solo $N+1$ valutazioni di funzione. Sarà un'accortezza necessaria quando lo implementeremo.

Analogamente, il metodo di Cavalieri–Simpson richiede $2N+1$ valutazioni di funzione. Vedremo però che questo costo maggiore è compensato da un errore minore.

Le formule che, come quella dei trapezi e di Cavalieri–Simpson, includono gli estremi a e b come nodi, sono dette *chiuse*.

Errore delle formule composite

Teorema 6.9. *Se per una formula di integrazione vale una stima sull'errore del tipo $|\tilde{I}-I| \le \kappa C_{d+1}(b-a)^{d+2}$, allora per la sua versione composita con N sottointervalli vale la stima*

$$|\tilde{I}_N - I| \le \kappa C_d \frac{(b-a)^{d+2}}{N^{d+1}} = \kappa C_{d+1}(b-a)h^{d+1}.$$

Dimostrazione. È sufficiente sommare tra loro le stime sugli N sottointervalli, ognuno di lunghezza $h = \frac{b-a}{N}$:

$$|\tilde{I}_N - I| \le \sum_{n=1}^{N}\left|\tilde{I}_{[x_{n-1},x_n]} - \int_{x_{n-1}}^{x_n} f(x)dx\right| \le \sum_{n=1}^{N}\kappa C_{d+1}h^{d+2} = N\kappa C_{d+1}\frac{(b-a)^{d+2}}{N^{d+2}}.$$

□

Ordine di convergenza Al crescere del numero di punti N, quindi, l'errore di una formula composita $e_N = |I_N - I|$ tende a zero come una potenza dell'inverso del numero di punti $\frac{1}{N}$, o, equivalentemente, come una potenza di h. Possiamo quindi scrivere

$$e_N = O(\frac{1}{N^p}) = O(h^p),$$

dove $p = d+1$. Questo numero p è detto *ordine di convergenza* della formula di integrazione. Quindi, per esempio, la formula dei trapezi composita e la formula del punto medio composita hanno ordine di convergenza $p = 2$, mentre la formula di Cavalieri–Simpson ha ordine $p = 4$.

L'ordine di convergenza ci permette di prevedere come decresce l'errore al crescere del numero di punti N: se il numero di punti raddoppia, passando da N a $2N$, cioè la distanza tra due punti successivi dimezza, passando da h a $\frac{h}{2}$, allora l'errore si riduce di un fattore 2^p (approssimativamente, visto che la relazione è vera solo nel limite per $N \to \infty$).

Diverse nozioni di ordine di convergenza Attenzione: è importante notare che questo è un concetto di "ordine di convergenza" diverso da quello visto per i metodi iterativi. Nel caso dei metodi iterativi, la convergenza si riferiva a un numero di iterazioni n: convergenza di ordine p significa che

$$e_{n+1} = O(e_n^p),$$

cioè, l'errore al passo $n+1$ è dell'ordine della potenza p-esima di *quello al passo precedente*.

Nel caso di questi metodi, invece, non c'è un concetto di "iterazioni successive", e non ha molto senso confrontare e_{N+1} con e_N: il risultato di una formula di integrazione con $N+1$ punti non si può calcolare facilmente a partire da quello con N punti, visto che tutti i punti necessari per calcolarla sono diversi.

Anche il comportamento dei due errori al crescere di n è diverso. Ad esempio, supponiamo di avere un metodo iterativo con ordine di convergenza 1; esso soddisfa

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n} = r < 1,$$

quindi $e_n \approx Cr^n$. In particolare, passando dall'iterazione n all'iterazione $2n$ l'erore si riduce di un fattore r^n, non di un fattore 2.

Stima dell'errore per le formule composite Sia N un intero pari. Applicando una formula di integrazione composita con ordine di convergenza p prima con N intervalli e poi con $N/2$ sottointervalli, ci aspettiamo quindi che

$$I_N = I + e_N, \quad I_{N/2} = I + e_{N/2} \approx I + 2^p e_N.$$

In particolare da questa relazione possiamo ricavare approssimativamente il valore di e_N come

$$\frac{I_{N/2} - I_N}{2^p - 1} \approx \frac{I + 2^p e_N - (I + e_N)}{2^p - 1} = e_N.$$

Per le formule dei trapezi e di Cavalieri–Simpson (ma non per tutti i metodi!) il calcolo di $I_{N/2}$ non richiede ulteriori valutazioni rispetto a quelle già utilizzate per il calcolo di I_N; e, a patto di fare le somme in un ordine appropriato durante l'implementazione, neppure somme aggiuntive. Quindi calcolare questa stima dell'errore ha un costo aggiuntivo trascurabile, per questi due metodi.

6.6 Quadratura Gaussiana

Ci poniamo ora (insieme a Gauss) il problema di quanto alto può essere il grado di precisione di una formula di quadratura, se scegliamo bene pesi e nodi. Perché una formula $\tilde{I}$ con n nodi sia esatta su tutti i polinomi di grado $\leq m$, è necessario e sufficiente che sia esatta su $1, x, x^2, \ldots, x^m$, visto che i polinomi di grado $< m$ sono combinazioni lineari di queste funzioni. Questo porta a $m+1$ equazioni in $2n$ incognite: per esempio, con $[a, b] = [-1, 1]$ abbiamo

$$\sum_{k=1}^{n} w_k 1 = 2, \quad \sum_{k=1}^{n} w_k x_k = 0, \quad \sum_{k=1}^{n} w_k x_k^2 = \frac{2}{3}, \ldots$$

Queste equazioni (nelle incognite $w_1, x_1, \ldots, w_n, x_n$) non sono lineari, quindi non è chiaro che siano risolubili neppure quando il sistema è quadrato, ma nella pratica risulta essere così: esiste una scelta di $x_1, \ldots, x_n, w_1, \ldots, w_n$ che fornisce una formula con grado di precisione $2n - 1$. Queste $x_1, \ldots, x_n, w_1, \ldots, w_n$ sono delle costanti universali, che dipendono solo dall'intervallo $[a, b]$, e qualcuno le ha già calcolate per noi. Pesi e nodi di queste formule, dette di *quadratura Gaussiana*, si possono trovare su diverse fonti online, per esempio `https://en.wikipedia.org/wiki/Gaussian_quadrature`. Per esempio con $n = 2$ otteniamo

$$x_1 = \frac{1}{\sqrt{3}} = 0.57735\ldots, \quad x_2 = -\frac{1}{\sqrt{3}} = -0.57735\ldots, w_1 = w_2 = 1$$

come pesi e nodi dell'unica formula di quadratura su $[-1, 1]$ con grado di precisione 3. Con un cambio di variabile simile a quello visto sopra è possibile adattarle ad altri intervalli. Non essendo formule chiuse, tipicamente non vengono usate in versione composita, ma sono particolarmente utili nella loro versione semplice, quando basta un'approssimazione dell'integrale con bassa precisione ma calcolabile con poche valutazioni di funzione.

Capitolo 7

Equazioni differenziali ordinarie

Il problema In questo capitolo, ci poniamo il problema di risolvere numericamente un'equazione differenziale, o più precisamente un *problema ai valori iniziali* (o *problema di Cauchy*)

$$\begin{cases} y'(t) = f(t, y(t)), \quad t \in [a, b], \\ y(a) = y_0. \end{cases} \tag{7.1}$$

Qui $y : [a, b] \to \mathbb{R}^m$ è una funzione a valori vettori (anche se nella maggior parte degli esempi che vedremo $m = 1$), e $f(t, y(t))$ è una funzione $f : [a, b] \times \mathbb{R}^m \to \mathbb{R}^m$ che specifica il problema. Il simbolo y' indica la derivata di y rispetto al tempo, e $y_0 \in \mathbb{R}^m$ è un valore iniziale dato.

Uno degli esempi che vedremo molto spesso è il seguente (*problema test*)

$$\begin{cases} y'(t) = \lambda y(t), \\ y(0) = y_0, \end{cases} \tag{7.2}$$

che corrisponde alla funzione $f(t, y) = \lambda y$. La soluzione di questo problema è $y(t) = \exp(\lambda t)$.

Equazioni che contengono derivate di ordine superiore si possono sempre trasformare in problemi in questa forma introducendo variabili ausiliarie: per esempio,

$$\begin{cases} y'' = 3y' + 5y \\ y(a) = 1,\ y'(a) = 0 \end{cases}$$

diventa, ponendo $z(t) = \begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix}$,

$$\frac{d}{dt} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} z_2 \\ 3z_2 + z_1 \end{bmatrix}.$$

La maggior parte degli algoritmi che vedremo mirano a calcolare un'approssimazione dei valori assunti dalla soluzione $y(t)$ su una griglia di punti

equispaziati: useremo la notazione già vista nel capitolo scorso

$$h = \frac{b-a}{N}, \quad t_n = a + nh, \quad n = 0, 1, 2, \ldots, N.$$

Chiameremo questi valori $y_i \approx y(t_i)$, per $t = 1, 2, \ldots, N$.

Vediamo subito alcuni algoritmi particolarmente semplici.

7.1 Metodi a un passo

Metodo di Eulero esplicito Facendo uno sviluppo di Taylor in t_n, abbiamo

$$y(t_{n+1}) = y(t_n) + y'(t_n)h + \frac{1}{2}y''(\xi)h^2 = y(t_n) + f(t_n, y_n)h + \frac{1}{2}y''(\xi)h^2. \quad (7.3)$$

Se ignoriamo il secondo termine e rimpiazziamo $y(t_n), y(t_{n+1})$ con le loro approssimazioni sui punti della griglia, otteniamo

$$y_{n+1} = y_n + hf(t_n, y_n).$$

Questa può essere vista come una formula che ci permette di calcolare y_{n+1} a partire da y_n. Otteniamo quindi il *metodo di Eulero esplicito*

$$y_{n+1} = y_n + hf_n, \quad n = 0, 1, 2, \ldots, N-1, \quad (7.4)$$

dove abbiamo posto $f_n = f(t_n, y_n)$ per brevità.

Il costo computazionale è di N valutazioni di f, più $O(mN)$ operazioni aritmetiche. Come per altri metodi, l'operazione più costosa qui è la valutazione della funzione, quindi sostanzialmente il costo è di N valutazioni della funzione, una per passo.

Metodo di Eulero implicito Facendo invece uno sviluppo di Taylor in t_{n+1}, abbiamo

$$y(t_n) = y(t_{n+1}) - y'(t_{n+1})h + O(h^2),$$

che operando nello stesso modo conduce alla relazione

$$y_{n+1} = y_n + h\underbrace{f(t_{n+1}, y_{n+1})}_{f_{n+1}}. \quad (7.5)$$

La differenza importante è che questa volta la y_{n+1} compare anche al secondo termine, quindi non possiamo calcolare direttamente il suo valore. Invece, è necessario qualche metodo per risolvere l'equazione (7.5) e calcolare y_{n+1} da essa. Un possibile metodo, che funziona per ogni scelta di f, è vederla come un'equazione di punto fisso: per ogni n fissato generiamo una successione

$$z_0 = y_n, \quad z_{k+1} = y_n + hf(t_{n+1}, z_k), \quad k = 0, 1, 2, \ldots$$

che (sperabilmente) converge a una soluzione $\lim_{k\to\infty} z_k = y_{n+1}$ dell'equazione. Oppure possiamo vedere la (7.5) come un'equazione non-lineare nell'incognita y_{n+1} e risolverla con il metodo di Newton.

Per alcune equazioni particolari esistono strategie che permettono di ottenere direttamente una soluzione. Per esempio, per il problema test (7.2) si ha

$$y_{n+1} = y_n + h\lambda y_{n+1};$$

questa è un'equazione lineare nella y_{n+1}, quindi possiamo risolverla facilmente ottenendo

$$y_{n+1} = \frac{1}{1 - h\lambda} y_n.$$

In ogni caso, si chiama *metodo di Eulero implicito* il metodo in cui si calcola y_{n+1} a partire da y_n risolvendo la (7.5) (in qualche modo) ad ogni passo per $n = 0, 1, 2, \ldots, N-1$. Il costo computazionale dipende dal modo in cui risolviamo la (7.5).

Metodo dei trapezi È il metodo

$$y_{n+1} = y_n + h\left(\frac{1}{2} f_n + \frac{1}{2} f_{n+1}\right). \tag{7.6}$$

È una sorta di "media" tra il metodo di Eulero esplicito e di quello implicito. Ha questo nome perché si può ottenere scrivendo

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} y'(t)dt = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt$$

e approssimando l'integrale con la formula dei trapezi otteniamo la (7.6).

Convergenza del metodo di Eulero esplicito (*) Vedremo tra poco l'enunciato di un risultato generale che dice che le sequenze di approssimazioni prodotte da questi metodi convergono alla soluzione esatta dell'equazione (7.1). Vogliamo però dare una dimostrazione esplicita della convergenza almeno per il più semplice, il metodo di Eulero esplicito.

Prima di tutto, definiamo cosa intendiamo per "convergenza". Data un'equazione differenziale (7.1) con soluzione $y(t)$, e una sequenza di approssimazioni $y_n \approx y(t_n)$ su una griglia $(t_0, t_1, \ldots, t_N)$, chiamiamo *errore globale* la quantità

$$E_N = \max_{n=1,2,\ldots,N} e_n, \quad e_n = \|y_n - y(t_n)\|, \tag{7.7}$$

cioè il massimo errore (in norma, o in valore assoluto se $m = 1$) tra la successione e la funzione che vuole approssimare. Intuitivamente, l'errore e_{n+1} al passo $n+1$ viene da due fonti: la prima è che abbiamo un errore e_n "ereditato" dai passi precedenti, per cui non partiamo a risolvere l'equazione da $y(t_n)$, ma dalla sua approssimazione y_n. La seconda è che, anche partendo dal valore esatto, la formula che usiamo per calcolare il passo successivo è approssimata; difatti l'abbiamo ottenuta eliminando il resto $\frac{f''(\xi)}{2} h^2$ dallo sviluppo di Taylor (7.3).

Al crescere del numero di punti N, ci aspettiamo che l'errore globale E_N ottenuto con il metodo di Eulero diminuisca. Per dimostrarlo, chiediamo un'ipotesi sulla funzione $f(t, y)$ che compare nella 7.1. Diciamo che f è *Lipschitziana* nella variabile y (con costante L) se esiste un numero reale $L \geq 0$ tale che per ogni $t \in [a, b]$ e $y_1, y_2 \in \mathbb{R}^m$ vale

$$\|f(t, y_1) - f(t, y_2)\| \leq L\|y_1 - y_2\|.$$

Questa proprietà ricorda un po' le proprietà di buon condizionamento: facendo una piccola perturbazione (assoluta, questa volta) dell'input y, l'output $f(t, y)$ varia di al più L volte questa perturbazione. Forse l'avete già vista ad analisi, perché è la stessa che serve per dimostrare l'esistenza di soluzioni di un problema ai valori iniziali (7.1).

Quando $m = 1$ (e le norme diventano valori assoluti), un modo di assicurare questa proprietà, per esempio, è provare che la f ha derivata parziale limitata $\left|\frac{\partial f}{\partial y}(t, y)\right| \leq L$ per ogni valore di t, y: difatti, il teorema di Lagrange ci assicura che per un certo ξ compreso tra y_1 e y_2 vale

$$f(t, y_1) - f(t, y_2) = \frac{\partial f}{\partial y}(t, \xi)(y_1 - y_2).$$

Inoltre, supponiamo di avere un'equazione differenziale la cui soluzione $y(t)$ è una funzione di classe $\mathcal{C}^2$ su $[a, b]$. Allora (per il teorema di Weierstrass) esiste

$$C_2 = \max_{t \in [a,b]} \|y''(t)\|.$$

Andiamo quindi a enunciare il nostro risultato di convergenza.

Teorema 7.1. *Sia dato un problema ai valori iniziali* (7.1) *in cui la funzione f è Lipschitziana nella y con costante L, e in cui la soluzione $y(t)$ è di classe $\mathcal{C}^2$ su $[a, b]$. Allora, $\lim_{N\to\infty} E_N = 0$, e più precisamente $E_N = O(h)$ (cioè, esiste una costante $C > 0$ tale che $E_N \leq Ch$).*

Dimostrazione. Siamo interessati a stimare la quantità $e_n = \|y_n - y(t_n)\|$ per ogni $n = 0, 1, \ldots, N$. Chiaramente $e_0 = \|y_0 - y(t_0)\| = 0$. Possiamo scrivere

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n), \\ y(t_{n+1}) &= y(t_n) + h \underbrace{y'(t_n)}_{=f(t_n, y(t_n))} + \frac{h^2}{2} y''(\xi), \end{aligned}$$

dove la seconda equazione è uno sviluppo di Taylor. Sottraendo membro a membro e prendendo valori assoluti abbiamo

$$\begin{aligned} e_{n+1} = \|y_{n+1} - y(t_{n+1})\| &= \left\| y_n - y(t_n) + h(f(t_n, y_n) - f(t_n, y(t_n))) - \frac{h^2}{2} y''(\xi) \right\| \\ &\leq \|y_n - y(t_n)\| + h\|f(t_n, y_n) - f(t_n, y(t_n))\| + \frac{h^2}{2}\|y''(\xi)\| \\ &\leq e_n + hLe_n + \frac{h^2}{2} C_2 = (1 + hL)e_n + \frac{h^2}{2} C_2. \end{aligned}$$

Poniamo $M = \frac{h^2}{2} C_2$; iterando, abbiamo

$$\begin{aligned}
e_n &\le M + (1+hL)e_{n-1} \\
&\le M + (1+hL)M + (1+hL)^2 e_{n-2} \\
&\le M + (1+hL)M + (1+hL)^2 M + (1+hL)^3 e_{n-3} \\
&\le \dots \\
&\le M\left(1 + (1+hL) + (1+hL)^2 + \cdots + (1+hL)^{n-1}\right) + (1+hL)^n \underbrace{e_0}_{=0} \\
&= M \frac{(1+hL)^n - 1}{(1+hL) - 1} \\
&= \frac{M}{hL}\left(\left(1 + \frac{t_n - t_0}{n} L\right)^n - 1\right);
\end{aligned}$$

l'ultima uguaglianza segue dal fatto che $t_n - t_0 = nh$. Possiamo riconoscere il limite notevole che produce l'esponenziale; in realtà questo limite è anche una disuguaglianza, in quanto si ha

$$\left(1 + \frac{t_n - t_0}{n} L\right)^n \le e^{(t_n - t_0)L}.$$

Quindi rimettendo tutto insieme abbiamo per ogni $n = 1, 2, \dots, N$

$$e_n \le \frac{C_2 h}{2L}(e^{(t_n - t_0)L} - 1) \le \frac{C_2 h}{2L}(e^{(b-a)L} - 1) = O(h). \qquad \square$$

Metodi generali a un passo In generale possiamo scrivere un metodo a un passo come

$$y_{n+1} = y_n + h\Phi(t_n, y_n)$$

per un'opportuna funzione Φ: anche se nella sua espressione compaiono t_{n+1}, y_{n+1}, ecc., possiamo comunque considerarla come una funzione di quei due argomenti, perché si può calcolare a partire dai soli t_n e y_n (oltre che h e f); non serve conoscere altre quantità come per esempio il valore di y_{n-1}.

Vogliamo dare un risultato generale di convergenza per questi metodi. Per questo ci serve introdurre alcune definizioni. Oltre all'errore globale E_N definito più sopra, definiamo anche l'*errore locale di troncamento* come

$$T_N = \max_{n=0,1,\dots,N-1} \|\tau_n\|, \quad \tau_n = \frac{y(t_{n+1}) - y(t_n) - h\Phi(t_n, y(t_n))}{h},$$

cioè $1/h$ volte la differenza tra il valore esatto della soluzione $y(t_{n+1})$ e il valore $y(t_n) + h\Phi(t_n, y(t_n))$ calcolato tramite un passo dell'algoritmo a partire dal valore esatto $y(t_n)$.

Per un intero $p > 0$, un metodo si dice *consistente di ordine p* se $T_N = O(h^p)$ nel limite quando $N \to \infty$ (e quindi $h \to 0$); e si dice *convergente di ordine p* se $E_N = O(h^p)$. Queste definizioni si applicano quando abbiamo un'equazione differenziale con soluzione $y(t)$ sufficientemente regolare; per esempio, ci serve avere una funzione di classe almeno C^2 per scrivere la (7.3).

Il metodo di Eulero esplicito è consistente di ordine 1; difatti da uno sviluppo di Taylor abbiamo

$$y(t_{n+1}) = y(t_n) + y'(t_n)h + \frac{1}{2} y''(\xi) h^2,$$

quindi

$$\left\| \frac{y(t_{n+1}) - y(t_n) - hf(t_n, y(t_n))}{h} \right\| = \left\| \frac{1}{2} y''(\xi) h \right\| \leq \frac{1}{2} C_2 h = O(h).$$

Stessa cosa per il metodo di Eulero implicito. Invece il metodo dei trapezi è consistente di ordine 2; si può dimostrarlo facendo uno sviluppo di Taylor nel punto medio $\frac{y(t_n)+y(t_{n+1})}{2}$.

Convergenza dei metodi a un passo Vale il seguente risultato (che non dimostriamo).

Teorema 7.2. *Supponiamo che la funzione $\Phi(t, y)$ sia continua per $t \in [a, b]$, e Lipschitziana nella y, con una costante L che non dipende da h. Allora, un metodo è convergente di ordine p se e solo se è consistente di ordine p.*

Metodi di Runge–Kutta I metodi di Runge–Kutta sono una classe di metodi a un passo (quindi definiti da una $\Phi(t_n, y_n)$ come sopra) più generale, che permette di raggiungere ordini di consistenza e convergenza maggiori. Un metodo di Runge–Kutta si definisce tramite alcuni valori che vengono convenzionalmente raccolti in una tabella, detta *tavola* (o *tableau*) *di Butcher*; essa ha la forma

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}.$$

L'intero s è detto *numero di stadi* del metodo. A partire dai coefficienti in questa tabella possiamo definire la funzione $\Phi(t_n, y_n)$ in questo modo: definiamo

$$k_i = f\left(t_n + c_i h, y_n + h \left(\sum_{j=1}^{s} a_{ij} k_j \right) \right), \qquad i = 1, 2, \dots, s,$$

$$\Phi(t_n, y_n) = \sum_{j=1}^{s} b_j k_j.$$

I metodi che abbiamo visto finora sono casi particolari dei metodi di Runge–Kutta. Per il metodo di Eulero esplicito abbiamo $s = 1$ e tavola di Butcher

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array},$$

per il metodo di Eulero implicito

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array},$$

e per il metodo dei trapezi $s = 2$ (difatti ci sono due diverse valutazioni di funzione) e

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$

Per derivare le tavole degli ultimi due metodi abbiamo dovuto risostituire la relazione finale per scrivere il termine y_{n+1} che compare dentro f_{n+1} nella forma $y_n + h(\dots)$.

Un metodo di Runge–Kutta è *esplicito* se la parte triangolare superiore di A (inclusa la diagonale) contiene zeri; in questo caso è possibile calcolare esplicitamente i valori di k_i uno per volta a partire dal primo (come nel caso del metodo di Eulero esplicito). Altrimenti (metodi *impliciti*) ogni k_i dipende anche dai valori dei successivi, ed è necessario risolvere il sistema di equazioni non-lineari come abbiamo visto per il metodo di Eulero implicito.

Esistono anche metodi di Runge–Kutta con un numero di stadi maggiore; per esempio Matlab utilizza in una delle sue funzioni per risolvere equazioni differenziali (`ode45`) un metodo con tavola di Butcher

0							
1/5	1/5						
3/10	3/40	9/40					
4/5	44/45	−56/15	32/9				
8/9	19372/6561	−25360/2187	64448/6561	−212/729			
1	9017/3168	−355/33	46732/5247	49/176	−5103/18656		
1	35/384	0	500/1113	125/192	−2187/6784	11/84	
	35/384	0	500/1113	125/192	−2187/6784	11/84	0

,

che ha $s = 7$ stadi e ordine di convergenza $p = 5$ (*metodo di Dormand–Prince*).

Problemi stiff Il risultato di convergenza che abbiamo visto ci dice cosa succede al limite $h \to 0$; però ci sono differenze importanti tra un metodo e l'altro che riguardano quello che succede con un valore *finito* di h. In particolare, non è difficile osservare in esperimenti numerici che prendendo h troppo grande il metodo di Eulero esplicito produce sequenze y_n oscillanti anche quando la soluzione vera converge a zero in modo monotono.

Un modo per cominciare a investigare cosa succede è attraverso il "problema test", (7.2). Prendiamo un problema test con $\lambda < 0$; la soluzione esatta $y(t) = e^{\lambda t}$ quindi è decrescente e ha $\lim_{t\to+\infty} y(t) = 0$. Applicando il metodo di Eulero esplicito a questo problema, come abbiamo già visto, otteniamo $y_n = (1 + h\lambda)^n$. Se $\lambda < -\frac{2}{h}$, abbiamo $|1 + h\lambda| > 1$, quindi il metodo produce iterate y_n che diventano sempre più grandi e oscillano con segni alterni. Questo succede quando scegliamo un passo h troppo grande, e la limitazione esatta dipende dal valore di λ.

Le cose si complicano per problemi in più variabili, in cui possono comparire contemporaneamente valori diversi di λ; per esempio pensiamo al problema

$$\begin{bmatrix} x'(t) \\ y'(t) \\ z'(t) \end{bmatrix} = \begin{bmatrix} \lambda_1 x(t) \\ \lambda_2 y(t) \\ \lambda_3 z(t) \end{bmatrix} = \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \lambda_3 \end{bmatrix} \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix},$$

con $\lambda_1 < \lambda_2 < \lambda_3 < 0$: la soluzione del problema tende a zero come l'esponente più vicino a zero $e^{\lambda_3 t}$, ma il passo che possiamo scegliere dipende dall'esponente più lontano da zero λ_1. Quindi per approssimare la soluzione con il metodo di Eulero esplicito dobbiamo scegliere un passo molto piccolo rispetto alla scala del problema.

Le cose diventano ancora più complicate per problemi non lineari; spesso non è possibile fare un'analisi esatta e possiamo solo confrontarli con problemi lineari dal comportamento simile. In generale, esistono problemi per cui il metodo di Eulero (e con lui molti altri metodi) presenta oscillazioni eccessive a meno che la scelta del passo sia estremamente piccola; questi si chiamano *problemi stiff*, o in italiano *rigidi*. Una definizione precisa è complicata; in generale questo fenomeno è associato a:

- componenti della soluzione y che hanno improvvisi cambiamenti;
- fenomeni che avvengono a diverse scale di tempi;
- funzioni del tipo $f(t, y) = Ay$ con A mal condizionata, o funzioni non lineari $f(t, y)$ con Jacobiano $\frac{\partial f}{\partial y}$ mal condizionato.

A differenza del mal condizionamento di sistemi lineari, non è un fenomeno che riusciamo a quantificare esattamente e misurare con un numero.

In ogni caso, alcuni metodi sono più adatti ai problemi *stiff* del metodo di Eulero; tipicamente si tratta dei metodi impliciti. Facciamo un'analisi di stabilità più generale per stabilirlo.

Funzione di stabilità e A-stabilità Possiamo replicare l'analisi fatta per il metodo di Eulero e applicarla a un metodo di Runge–Kutta generico. Scrivendo esplicitamente un qualunque metodo di Runge–Kutta per il problema test (7.2), si vede che questo assume sempre la forma

$$y_{n+1} = R(q)y_n, \quad q = h\lambda.$$

Per esempio, per il metodo dei trapezi abbiamo

$$y_{n+1} = y_n + h\frac{1}{2}(\lambda y_n + \lambda y_{n+1}) \iff y_{n+1} = \underbrace{\frac{1 + \frac{1}{2}q}{1 - \frac{1}{2}q}}_{=:R(q)} y_n$$

La successione $y_n = R(q)^n$ converge a zero se e solo se $|R(q)| < 1$. Si definisce *regione di (assoluta) stabilità* del metodo l'insieme $S_A = \{z \in \mathbb{C} : |R(z)| < 1\}$. Quindi y_n converge a zero (per il problema test) se e solo se h è scelto in modo che $h\lambda \in S_A$. La soluzione esatta del problema test $y(t) = e^{\lambda t}$ invece converge a zero se λ sta nel semipiano sinistro. Questo motiva una definizione: un metodo di Runge–Kutta si dice *A-stabile* se la sua regione di stabilità contiene tutto il semipiano sinistro $\{z \in \mathbb{C} : \mathrm{Re}(z) < 0\}$. Esistono varianti di questa definizione: per esempio un metodo si dice A_0-*stabile* se la regione di stabilità contiene la semiretta reale negativa $\{z \in \mathbb{R} : z < 0\}$; imparare tutti i nomi non è importante.

La regione di stabilità del metodo dei trapezi è precisamente il semipiano sinistro; possiamo dimostrarlo con un ragionamento geometrico. Si ha

$$|R(q)| = \frac{|1 + \frac{1}{2}q|}{|1 - \frac{1}{2}q|} = \frac{d(-1, \frac{1}{2}q)}{d(1, \frac{1}{2}q)},$$

il rapporto tra le distanze del punto $\frac{1}{2}q$ da 1 e da -1. Se q sta nel semipiano sinistro, allora è più vicino a -1 che a 1, e quindi il rapporto è minore di 1.

La regione di stabilità del metodo di Eulero esplicito è un cerchio di centro -1 e raggio 1; difatti già abbiamo visto che non è A-stabile.

Potete dimostrare per esercizio che la regione di stabilità del metodo di Eulero implicito è l'*esterno* di un cerchio di centro 1 e raggio 1; in particolare è A-stabile. Anzi, è anche "troppo stabile", visto che y_n converge a zero anche in casi in cui λ sta nel semipiano destro e quindi $y(t) = e^{\lambda t}$ dovrebbe tendere a infinito per $t \to +\infty$.

Il risultato principale (che non dimostriamo) è il seguente:

Teorema 7.3. *Nessun metodo di Runge–Kutta esplicito è A-stabile.*

Quindi in presenza di un problema stiff è solitamente meglio usare un metodo implicito; altrimenti il metodo spesso richiede un passo h molto piccolo per fornire risultati accettabili. Questo giustifica l'utilità dei metodi impliciti, che richiedono un metodo più complicato per calcolare y_{n+1} ad ogni passo, visto che è definito implicitamente come la soluzione di un'equazione.

7.2 Metodi a più passi

Vediamo ora un'altra famiglia di metodi, i *metodi a più passi* (o *multistep* in inglese). In particolare ci concentriamo sui *metodi lineari a più passi*.

L'idea è la seguente: i metodi di Runge–Kutta ci consentono di aumentare l'ordine di convergenza del metodo, ma al costo di avere $s > 1$ valutazioni della f all'interno di ogni passo n. Possiamo però ottenere ordini più alti anche con un'altra strategia, quella di riutilizzare anche i valori precedenti già calcolati di $y_{n-1}, y_{n-2}, \dots$ e $f_{n-1}, f_{n-2}, \dots$ per ottenere una migliore approssimazione. Per esempio, il metodo

$$y_{n+2} = y_{n+1} + h(\frac{3}{2} f_{n+1} - \frac{1}{2} f_n) \tag{7.8}$$

è tale che $\frac{y(t_{n+2}) - y_{n+2}}{h} = O(h^2)$ [1], se y_{n+1} e y_n sono approssimazioni sufficientemente accurate di $y(t_{n+1})$ e $y(t_n)$. Se supponiamo di avere a disposizione *due* valori iniziali y_0, y_1, anziché semplicemente uno, possiamo usare la (7.8) per $n = 0, 1, 2, \dots$ per calcolare tutti i valori successivi con un errore locale di troncamento migliore del metodo di Eulero (ordine 2 anziché 1), mantenendone comunque il costo di una sola nuova valutazione della f per passo.

In generale, definiamo un *metodo lineare a k passi* con una formula del tipo

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j}, \quad n = 0, 1, 2, \dots, \tag{7.9}$$

dove abbiamo posto $f_i := f(t_i, y_i)$ come già fatto in passato, per un'opportuna scelta delle costanti reali α_i, β_i. Per evitare casi degeneri, supponiamo che α_0, β_0 non siano entrambi nulli, così come α_k e β_k. Se $\beta_k = 0$, allora possiamo calcolare esplicitamente y_{n+k}, e il metodo si dice *esplicito*; altrimenti il metodo si dice *implicito* e abbiamo bisogno di un algoritmo per risolvere l'equazione (7.9) ad ogni passo n.

[1] Una dimostrazione sta su `https://en.wikiversity.org/wiki/Adams-Bashforth_and_Adams-Moulton_methods`. L'idea è usare uno sviluppo di Taylor di $y'(t)$ in t_{n+1} per scrivere f_n, e uno sviluppo di Taylor di ordine 2 in t_{n+1} per scrivere $y(t_{n+2})$.

Scelta dei valori iniziali Per poter applicare il metodo (7.9), abbiamo bisogno di opportuni valori iniziali $y_1, y_2, \ldots, y_{k-1}$, in aggiunta a y_0. Questi non sono tra i dati iniziali del problema, quindi vanno calcolati in qualche modo. Solitamente si utilizza un metodo a un passo per calcolarli; quindi ogni metodo a più passi ha bisogno di essere "inizializzato" calcolando questi primi valori tramite un opportuno metodo a un passo.

Perché il metodo possa convergere alla soluzione esatta, è necessario imporre una condizione su come questi valori iniziali vengono scelti. Se $h \to 0$, anche $t_1, t_2, \ldots, t_{k-1} \to t_0$, e quindi $y(t_1), \ldots, y(t_k) \to y_0$. Diciamo che un modo di scegliere i dati iniziali $y_1, y_2, \ldots, y_{k-1}$ in un metodo a più passi è *compatibile* se soddisfa

$$\lim_{N\to\infty} y_i = y_0, \quad i = 1, 2, \ldots, k-1.$$

Famiglie di metodi Ci sono diverse famiglie di metodi lineari a più passi: tra queste i *metodi di Adams–Bashforth* (tra cui ricade anche il nostro primo esempio (7.8)), espliciti, e i *metodi di Adams–Moulton* e le *Backward differentiation formulas (BDF)*, entrambi impliciti. Per esempio, i primi metodi della famiglia BDF sono

$$y_{n+1} - y_n = hf_{n+1} \qquad \text{(Eulero implicito)}$$
$$y_{n+2} - \frac{4}{3}y_{n+1} + \frac{1}{3}y_n = 2hf_{n+2}$$

e i successivi sono fatti in modo simile, con un solo valore $\beta_k \neq 0$ a destra dell'uguale e opportune combinazioni lineari delle y_{n+j} a sinistra.

Per altri metodi, potete vedere per esempio `https://en.wikipedia.org/wiki/Linear_multistep_method` e `https://en.wikipedia.org/wiki/Backward_differentiation_formula`.

Consistenza e convergenza Come nei metodi a un passo, definiamo l'*errore locale di troncamento*

$$\tau_n = \frac{1}{h}\left(\sum_{j=0}^{k} \alpha_j y(t_{n+j}) - h\sum_{j=0}^{k} \beta_j f(t_{n+j}, y(t_{n+j}))\right)$$

Questa quantità a volte viene normalizzata diversamente dividendola per α_k, o per la somma dei β_j; in ogni caso, quello che misura è di quanto $y(t_{n+k})$ differisce dalla quantità y_{n+k} calcolata dal metodo assumendo che tutti i passi precedenti siano esatti. Un metodo si dice *consistente di ordine* p se $T = \max_n |\tau_n|$ è $O(h^p)$.

Un metodo si dice *convergente di ordine* p se l'errore globale (7.7) (definito esattamente come nei metodi a un passo) è $O(h^p)$; qui nessuna nuova definizione.

Possiamo dimostrare che (sotto opportune ipotesi) questi due concetti sono equivalenti, proprio come nei metodi a un passo.

Ci serve però prima introdurre uno strumento tecnico che permette di studiare le successioni prodotte dal metodo.

Equazioni alle differenze Consideriamo innanzitutto cosa succede applicando il metodo a più passi al problema $y' = 0$, con condizioni iniziali $y_0, y_1, \ldots, y_{k-1}$ qualunque. La successione delle y_i allora soddisfa la relazione

$$\alpha_0 y_n + \alpha_1 y_{n+1} + \cdots + \alpha_k y_{n+k} = 0. \tag{7.10}$$

Relazioni della forma (7.10) si chiamano *equazioni alle differenze (lineari a coefficienti costanti)*; sono una sorta di analogo discreto delle equazioni differenziali lineari a coefficienti costanti. Forse l'esempio più celebre è quello dei numeri di Fibonacci, che soddisfano la relazione

$$y_n + y_{n+1} - y_{n+2} = 0.$$

Esiste una teoria per calcolare le soluzioni delle (7.10) che è molto simile a quella per calcolare le soluzioni delle equazioni differenziali lineari a coefficienti costanti.

Teorema 7.4. *Data un'equazione alle differenze* (7.10)*, definiamo*

$$p_1(z) = \alpha_0 + \alpha_1 z + \cdots + \alpha_k z^k$$

il suo polinomio associato.

- *Se $p_1(z)$ ha k zeri distinti (nel piano complesso $\lambda_1, \lambda_2, \ldots, \lambda_k$), allora ogni soluzione della* (7.10) *si scrive come combinazione lineare delle soluzioni di base $\lambda_1^n, \lambda_2^n, \ldots, \lambda_k^n$.*
- *Se uno zero (o più) λ di $p_1(z)$ ha molteplicità $m > 1$, allora come soluzioni di base prendiamo invece $\lambda^n, n\lambda^n, n^2\lambda^n, \ldots, n^{m-1}\lambda^n$.*

[Esempi]
In particolare, vale il seguente risultato.

Teorema 7.5. *Per un'equazione alle differenze* (7.10)*:*

- *Se tutti gli zeri del polinomio $p_1(z)$ hanno valore assoluto strettamente minore di* 1*, allora tutte le soluzioni della* (7.10) *tendono a zero quando $n \to \infty$.*
- *Se tutti gli zeri del polinomio $p_1(z)$ hanno valore assoluto minore o uguale a* 1*, e in più tutte le radici di modulo* 1 *sono semplici (molteplicità $= 1$), allora tutte le soluzioni della* (7.10) *sono limitate, cioè $\sup_{n\in\mathbb{N}} |y_n|$ è finito.*

La seconda di queste condizioni si chiama *condizione delle radici* (root condition).

Zero-stabilità Il polinomio $p_1(z)$ definito sopra, quando gli α_i sono i coefficienti di un metodo a più passi (7.9), si chiama *primo polinomio caratteristico* del metodo.

Un metodo lineare a più passi si dice *zero-stabile* se il suo primo polinomio caratteristico $p_1(z)$ soddisfa la condizione delle radici; questo in particolare implica che le successioni che genera per la soluzione del problema $y' = 0$ sono sempre limitate (uniformemente in n). Questo è quello che ci aspettiamo se il metodo deve "funzionare bene" su questa equazione, visto che tutte le soluzioni di $y' = 0$ sono costanti (e quindi limitate).

Notiamo che $\alpha = 1$ è sempre una radice del primo polinomio caratteristico, se il metodo è consistente; non è difficile vedere che altrimenti è impossibile che $\tau_n \to 0$, considerando per esempio una soluzione costante del problema $y' = 0$.

La zero-stabilità sembra un concetto di poco rilievo, visto che ci concentriamo su un problema molto facile e di poco interesse; in realtà ha una conseguenza importante, perché ci assicura che eventuali piccoli errori nei dati iniziali $y_1, \ldots, y_{k-1}$ non causano errori più consistenti sulle iterate successive.

Teorema di equivalenza di Dahlquist Possiamo ora enunciare un teorema di convergenza per i metodi a più passi.

Teorema 7.6. *Consideriamo un metodo lineare a più passi* (7.9)*, con una scelta compatibile dei dati iniziali. Se il metodo è zero-stabile e consistente, allora è convergente. Inoltre, su un problema con soluzione* $y \in \mathcal{C}^{p+1}([a,b])$*, se un metodo è consistente di ordine* p *allora è anche convergente di ordine* p*.*

La dimostrazione è complicata, quindi ci acconentiamo dell'enunciato.

Ovviamente le famiglie di metodi che vengono usate in pratica sono tutte zero-stabili.

A-stabilità Come nel caso dei metodi a un passo, la sola convergenza non ci dice nulla riguardo al passo h che serve per avere un'approssimazione accettabile della soluzione. Possiamo quindi studiare la A-stabilità dei metodi a più passi, esattamente nello stesso modo in cui abbiamo affrontato quelli a un passo; questo ci dà informazioni su quali metodi sono adatti per problemi stiff.

Applicando il metodo (7.9) al problema test (7.2), otteniamo

$$\sum_{j=0}^{k}(\alpha_j - \underbrace{\lambda h}_{:=q}\beta_j)y_{n+j} = 0.$$

Analogamente a quanto richiesto per i metodi a un passo, vogliamo vedere per quali valori di q le soluzioni di questa equazione alle differenze tendono a zero, in modo da replicare il comportamento della soluzione esatta $y(t) = e^{\lambda t}$ quando $\mathrm{Re}(\lambda) < 0$.

Di nuovo per le proprietà delle equazioni alle differenze, questo succede (fissato un valore di q) quando tutte le radici del polinomio associato

$$\pi_q(z) = \sum_{j=0}^{k}(\alpha_j - q\beta_j)z^j$$

hanno modulo *strettamente* minore di 1. Il polinomio $\pi_q(z)$ (che dipende dal valore di q, che supponiamo fissato) si chiama *polinomio di stabilità*; possiamo scriverlo volendo come

$$\pi_q(z) = p_1(z) - qp_2(z)$$

in termini del primo polinomio caratteristico (che già abbiamo incontrato) e del *secondo polinomio caratteristico*

$$p_2(z) = \beta_0 + \beta_1 z + \cdots + \beta_k z^k.$$

Si dice *regione di (assoluta) stabilità* di un metodo a più passi l'insieme

$$S_A = \{q \in \mathbb{C}\text{: tutte le radici del polinomio } \pi_q(z) \text{ hanno modulo minore di } 1\}.$$

Un metodo a più passi si dice *A-stabile* se S_A contiene tutto il semipiano sinistro. Anche in questo caso la A-stabilità è una condizione abbastanza stringente, che in particolare esclude tutti i metodi espliciti. Questo è dimostrato in una serie di risultati noti come "barriere di Dahlquist".

Teorema 7.7. *Valgono i seguenti risultati.*

- *Non esistono metodi lineari a più passi* espliciti *A-stabili.*
- *I metodi lineari a più passi* impliciti *A-stabili hanno ordine* $p \leq 2$*.*
- *Tra i metodi lineari a più passi A-stabili impliciti di ordine* $p = 2$*, quello per cui l'errore converge a zero più velocemente è il metodo dei trapezi.*

Quindi nessun metodo lineare a più passi A-stabile migliora in termini di accuratezza il metodo dei trapezi (che in realtà è a un passo, $k = 1$). Una famiglia di metodi (quelli BDF) in realtà si avvicina molto ad essere A-stabile, visto che le loro regioni di stabilità includono tutto il semipiano negativo tranne una regione molto piccola[2]. Questi metodi sono tra quelli più usati per problemi stiff.

7.3 Solutori di equazioni differenziali in Matlab

Matlab contiene diverse funzioni che possono essere usate per risolvere numericamente problemi ai valori iniziali (7.1). Una delle più comuni è `function [t, Y] = ode45(f, [a,b], y0)`. Essa prende in input una *function handle* alla funzione $f(t, y)$ (che dev'essere *sempre* una funzione di due variabili), un vettore di due elementi $[a, b]$ (attenzione!), e un valore iniziale $y0$ (scalare o vettore). Essa restituisce un vettore t e una matrice Y che contiene le iterate y_n prodotte dal metodo come *righe* (quindi la trasposta di quella prodotta dalle funzioni che abbiamo scritto noi nel laboratorio).

La funzione `ode45` utilizza un metodo di Runge–Kutta esplicito di ordine 5, ma contiene uno stimatore dell'errore e lo usa per modificare la lunghezza del passo di integrazione h, adattandola in modo da non fare mai passi più corti o più lunghi del necessario per ottenere una soluzione accurata (metodo di Dormand–Prince o RK45). Pertanto il vettore `t` restituito non contiene una sequenza di punti equispaziati, ma una sequenza di punti opportunamente scelti. Tipicamente sono necessari più punti negli intervalli in cui la $y(t)$ varia più velocemente. Le y_n restituite corrispondono alla funzione valutata su questa sequenza di punti, cioè $y_n \approx y(t_n)$.

È possibile usare il comando `odeset` per specificare alcune opzioni, per esempio la tolleranza cercata sulla soluzione. Si usa in questo modo:

```
>> opzioni = odeset('AbsTol', 1e-6, 'RelTol', 1e-3);
```

specifica una tolleranza assoluta sulla soluzione calcolata di 10^{-6} (cioè, vogliamo che $\|y_n - y(t_n)\|_\infty \leq 10^{-6}$) e una relativa di 10^{-3} (cioè, $\frac{\|y_n - y(t_n)\|}{\|y(t_n)\|} \leq 10^{-3}$).

Le opzioni impostate vengono salvate in una variabile `opzioni` che possiamo poi passare al solutore come ultimo argomento:

```
>> ode45(f, [a,b], y0, opzioni);
```

Guardando la documentazione di `odeset` trovate altre opzioni.

Indipendentemente da questi miglioramenti, `ode45` usa un metodo esplicito, quindi ha sempre il problema di richiedere passi h molto corti (e quindi un alto numero di iterazioni e costo computazionale) per risolvere adeguatamente problemi stiff.

[2]Queste regioni di stabilità sono raffigurate per esempio su `https://en.wikipedia.org/wiki/Backward_differentiation_formula`

Matlab contiene anche funzioni per risolvere equazioni differenziali che utilizzano metodi impliciti; la più comune è la funzione `ode15s`. Essa accetta e ritorna gli stessi argomenti di `ode45`, ma utilizza un metodo implicito e quindi è adatta anche a problemi stiff (come indica la `s` nel nome della funzione). Utilizza diversi metodi impliciti a più passi, di ordine variabile da 1 a 5, cambiando il metodo e la lunghezza del passo a seconda dell'accuratezza richiesta. Notare che cambiare la lunghezza del passo è più complicato per metodi a più passi, visto che non è più possibile riutilizzare direttamente i valori di $f(t_{n+j}, y_{n+j})$ calcolati nei passi precedenti.

```
>> A = [-10 -10; -10 -11];
>> f = @(t, y) A*y;
>> ode15s(f, [0,1], [1,0])
```

Nel caso dei metodi impliciti, un'opzione aggiuntiva particolarmente utile per migliorare le performance dei metodi è fornire a Matlab lo Jacobiano $\frac{\partial f}{\partial y}$. Difatti `ode15s` (e varianti) utilizzano un metodo tipo-Newton per calcolare y_{n+k} dall'equazione implicita che lo definisce; e se lo ricordate questi metodi necessitano al loro interno della derivata della funzione che definisce l'equazione da risolvere. Lo Jacobiano può essere inserito tramite un'altra coppia di parametri `'Jacobian', J` passati ad `odeset`; qui `J` può essere una matrice costante oppure una function handle `J(t, y)`.

```
>> ode15s(f, [0,1], [1,0], odeset('Jacobian', A));
```

In questo caso il problema è comunque semplice, quindi le performance dei due metodi non cambiano di molto.

Esistono anche altre funzioni, per esempio `ode23` che utilizza un metodo di Runge–Kutta di ordine inferiore, o `ode23s` che (come indica la `s`) è un metodo adatto per problemi stiff.

7.4 Esempio: problema di Robertson

Un esempio classico di problema stiff deriva dalla modellizzazione del sistema di reazioni chimiche

$$\begin{cases} A \xrightarrow{k_1} B \\ B + B \xrightarrow{k_2} B + C \\ B + C \xrightarrow{k_3} A + C. \end{cases}$$

Qualitativamente, la prima e la terza reazione convertono la specie A nella specie B e viceversa, mentre la seconda reazione "rimuove" la specie B dal sistema trasformandola irreversibilmente nella specie C. Quindi sul lungo periodo ci aspettiamo che le specie A e B vengano convertite interamente in C. Inoltre, la quantità totale $[A] + [B] + [C]$ è conservata.

Il sistema di equazioni differenziali che fornisce la quantità delle tre specie presente rispetto al tempo è

$$\frac{d}{dt}\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \end{bmatrix} = \begin{bmatrix} -k_1 y^{(1)} + k_3 y^{(2)} y^{(3)} \\ k_1 y^{(1)} - k_2 (y^{(2)})^2 - k_3 y^{(2)} y^{(3)} \\ k_2 (y^{(2)})^2 \end{bmatrix}.$$

Solitamente, si sceglie il valore iniziale $y_0 = [1, 0, 0]^T$ e tre valori su scale molto diverse per le tre costanti cinetiche, $k_1 = 0.04, k_2 = 3 \cdot 10^7, k_3 = 10^4$. Questa differenza di scale rende il problema fortemente stiff.

Osservazioni numeriche:

- `ode45` richiede un numero estremamente alto di passi intermedi: anche su $[a, b] = [0, 100]$ sono necessari più di 400.000 punti intermedi. Tentare periodi più lunghi è senza speranza.
- `ode15s` risolve il problema con molti meno passi.
- È necessaria una simulazione su un intervallo di tempo molto lungo prima di confermare le proprietà viste teoricamente che tutta la massa viene convertita da A a C; si ha $y(10^6) \approx [0.002, 0, 0.998]$.
- Si apprezza molto meglio il comportamento disegnando il grafico in scala semilogaritmica (`semilogx(t, y, '-x')`).
- La seconda componente (quantità di B) resta sempre molto bassa e va plottata a parte perché si veda qualcosa (`semilogx(t,y(:,2), '-x')`).
- Ci sono poche speranze con metodi "semplici" come Eulero esplicito o implicito.
- Possiamo anche calcolare lo Jacobiano del sistema e fornirlo come argomento a `ode15s`.