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.

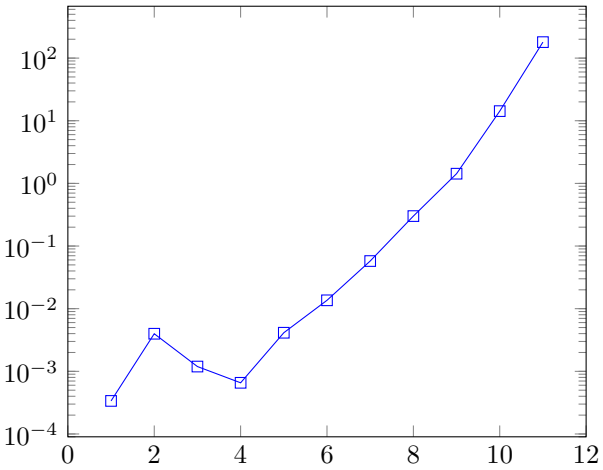
Capitolo 1

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 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 al 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 molte applicazioni ingegneristiche l'esperienza ha insegnato che la strategia che funziona meglio 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 non hanno nulla di male in sé, ma vanno implementati con attenzione per evitare che queste approssimazioni che sembrano innocue abbiano un grosso impatto sul risultato. Per esempio: eliminazione di Gauss, o anche semplici formule 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 spesso deve essere la parete di metallo che usate nelle ali? Se è troppo spessa, consumate più carburante del necessario. Se è troppo leggera, crac. Una volta c'era un solo modo di scoprirlo: costruisci un aereo di prova, 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 da modellare...), ed è fondamentale capire se i risultati sono affidabili e se questi errori diventano così grandi da compromettere il risultato.

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 nuovi problemi.

Capitolo 2

Aritmetica di macchina

Partiamo con la parte (un po' noiosa) di spiegare 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 2.1. *Fissata una base $\beta > 1$, ogni numero $x \neq 0$ si può scrivere come*

$$x = \pm\beta^p \sum_{i=1}^{\infty} c_i \beta^{-i}, \tag{2.1}$$

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

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

Due cambiamenti rispetto alla (2.1): t cifre (anziché infinite) nella mantissa, e un range finito per gli esponenti, da m a M.

Esempi di numeri di macchina con $\beta = 10, t = 3\ldots$

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 $\beta - 1$.

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 denormalizzati compresi tra 0 e ω; non ci interessano qui.

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 Dato un numero di macchina normalizzato positivo $x = \beta^p \sum_{i=1}^{t} c_i \beta^{-i}$, il numero di macchina 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.

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".

Approssimazione con numeri di macchina

Teorema 2.3. *Dato un numero reale $x \in [-\Omega, -\omega] \cup [\omega, \Omega]$, esiste un numero di macchina $\tilde{x}$ tale che*

$$\frac{|\tilde{x} - x|}{|x|} < \beta^{1-t}. \tag{2.2}$$

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 (2.1), 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 (2.2). In ogni caso, visto che $x \in (\underline{x}, \overline{x})$, si ha

$$|\tilde{x} - x| < \overline{x} - \underline{x} = \beta^{p-t}. \tag{2.3}$$

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

$$x \geq \beta^{p-1}. \tag{2.4}$$

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

$$\varepsilon = \frac{\tilde{x} - x}{x}. \tag{2.5}$$

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 (2.5) come $\tilde{x} = x(1 + \varepsilon)$. Quindi la (2.2) 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 $\mathtt{u} = \beta^{1-t}$ (*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

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

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 $\tilde{x} \oplus \tilde{y} = (x+y)(1+\varepsilon)$, 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 il sistema di numerazione con $\beta = 10, t = 3, m = -2, M = 2$. Quanto valgono ω e Ω? $[10^{-2}(0.100) = 0.001$ e $10^{2}(0.999) = 99.9]$

Qual è il risultato delle seguenti operazioni (arrotondando al più vicino)? $0.1 \ominus 0.001$; $1.01 \ominus 1.01$; $(0.007 \oplus 0.004) \oplus 1$; $0.007 \oplus 0.004 \oplus 1$; $1 \oslash 3 \otimes 3$; $50 \oplus 50$; $0.001 \odot 0.001$; $50 \oplus 0.01$.

Quali interi sono numeri 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 3

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

$$e_{in} = \frac{f(\tilde{x}) - f(x)}{f(x)}. \tag{3.1}$$

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 $g(\tilde{x})$. Per esempio, dando i comandi più sopra, invece di $f(\tilde{x}) = x^2 - 3/x$, il computer calcolerà $g(\tilde{x}) = \tilde{x} \otimes \tilde{x} \ominus 3 \oslash \tilde{x}$. La differenza tra queste due funzioni determina un *errore algoritmico*

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

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

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

"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 3.1.

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

Dimostrazione. Notiamo innanzitutto che riarrangiando la (3.1) si ottiene

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

Ora possiamo scrivere

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

□

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. Per esempio, $f(x,y) = x \cdot y$; cosa succede perturbando x in $x(1+\varepsilon_x)$? $\to$ ben condizionata.

Similmente per $f(x,y) = x + y$. (Questo include anche la differenza, perché $x - y = x + (-y)$.) $\to$ mal condizionata quando $x \approx y$.

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

Esempio: operazione di macchina (di nuovo con $\beta = 10, t = 3$), $1.25 \ominus 1.24$ e $1.24987 - 1.24367$; l'effetto combinato di approssimazioni precedenti e sottrazione è "cancellazione catastrofica".

Stima 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

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

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

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

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.

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 $h(\tilde{x},\tilde{y}) = (\tilde{x} \oplus \tilde{y}) \otimes (\tilde{x} \ominus \tilde{y})$ 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 l'esponenziale $\exp(x)$ tramite la sua serie di MacLaurin

$$\exp(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!},$$

non possiamo sommare un numero infinito di termini sul computer, ma dovremo arrestarci dopo un certo numero finito N. Il numero $\exp(x)$ in generale non è razionale, quindi non abbiamo speranza di calcolarlo esattamente con un numero finito di operazioni!

In generale, supponiamo di voler calcolare $\phi(x)$ (questa volta non per forza una funzione razionale), e di avere un algoritmo che calcola una sua approssimazione $f(x)$, per esempio $\phi(x) = \exp(x)$ e $f(x) = \sum_{n=0}^{N} \frac{x^n}{n!}$. Possiamo definire allo stesso modo un *errore analitico*

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

Con una dimostrazione analoga a quella sopra ma con tre termini anziché due, possiamo dimostrare che

$$\frac{g(\tilde{x}) - \phi(x)}{\phi(x)} \doteq e_{an} + e_{in} + e_{alg}.$$

Gli errori e_{in} ed e_{alg} sono dovuti all'aritmetica di macchina, 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. In un problema 'facile' 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.

Analizzeremo l'errore analitico di vari algoritmi che studieremo.

Capitolo 4

Equazioni non lineari (zeri di funzione)

Problema: data f, vogliamo calcolare x_* tale che $f(x_*) = 0$. Questo è un problema comune. 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. Difatti, il teorema degli zeri dice che $f \in \mathcal{C}^0([a, b])$ ha almeno uno zero in $[a, b]$ se esso è un intervallo di separazione.

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 $x \in (a, b)$).

4.1 Condizionamento del problema

Studiare il condizionamento di questo problema ci richiede di cambiare un po' le nostre definizioni, perché il suo "input" non è un numero ma una funzione f. Supponiamo qui che invece di ricevere in "ingresso" f riceviamo una funzione g che soddisfa $\max |g - f| \leq \delta$ (disegno). Può darsi che una piccola perturbazione sia sufficiente a trasformare un problema risolubile in uno non risolubile; per esempio se $f(x) = x^2$ e $g(x) = x^2 + \varepsilon$.

Sia x_* uno zero di f. Supponiamo che f sia derivabile con $f'(x_*) \neq 0$, quindi "taglia" l'asse delle ascisse trasversalmente (ed è invertibile in un intorno di x_*).

Uno zero $\tilde{x}$ di g (se c'è) deve stare per forza tra $f^{-1}(-\delta)$ e $f^{-1}(\delta)$. Sviluppando al primo ordine (e ricordando la derivata della funzione inversa),

$$f^{-1}(x_* + \delta) = f^{-1}(x_*) + \delta \frac{1}{f'(x_*)} + O(\delta^2)$$

Quindi $f^{-1}(\delta) - x_* \doteq \frac{1}{f'(x_*)}\delta$. Rifacendo lo stesso ragionamento su $-\delta$, si ha $f^{-1}(-\delta) - x_* \doteq -\frac{1}{f'(x_*)}\delta$, quindi

$$|\tilde{x} - x_*| \dot{\leq} \frac{1}{|f'(x_*)|}\delta.$$

Per parlare di "numero di condizionamento" nello stesso senso usato sopra, ci serve parlare di errori *relativi* sia sulla x_* che sulla f. Un modo di farlo è questo: scriviamo $M = \max_{[a,b]} |f|$, consideriamo una perturbazione g tale che $|f-g| \leq \varepsilon M$ (il nostro input), e chiamiamo $\tilde{x}$ uno zero della f (il nostro output). Allora,

$$\left|\frac{\tilde{x} - x_*}{x_*}\right| \dot{\leq} \frac{M}{|x_* f'(x_*)|}.$$

4.2 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 $f(a)$ e $f(b)$ abbiano segni opposti. Difatti questo assicura l'esistenza di *almeno* uno zero di f in $[a_1, b_1]$ (esistenza degli zeri). Chiamiamo "intervallo di separazione" un intervallo $[a,b]$ tale che $f(a)f(b) < 0$.

Idea: calcolo il punto medio x_1, e controllo il segno di $f(x_1)$. Se $f(x_1) = 0$, ho vinto. Altrimenti, uno dei due intervalli $[f(a_1), f(x)]$ e $[f(x), f(b_1)]$ è di separazione; definisco $[a_2, b_2]$ gli estremi di questo intervallo e continuo. (pseudocodice).

Si può dimostrare formalmente (noi non lo faremo) che le successioni a_i, b_i, x_i convergono a un valore che è uno zero di f. A noi però interessa relativamente cosa succede al limite, perché possiamo fare solo un numero finito di passi. Ci serve un *criterio di arresto* che ci assicuri che abbiamo risolto il nostro problema con una certa accuratezza, vale a dire, che esiste uno zero $|x_* - x_n| \leq \varepsilon$ con un ε fissato.

Un modo è fermarsi quando l'ampiezza dell'intervallo di confidenza $[a_n, b_n]$ soddisfa $b_n - a_n \leq 2\varepsilon$. Difatti, in questo caso $|x_n - x_*| \leq \varepsilon$. Quante iterazioni ci servono?

$$b_n - a_n = \frac{b_{n-1} - a_{n-1}}{2} =\leq \frac{b_1 - a_1}{2^{n-1}},$$

quindi 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 $f(x_n)$ è "sufficientemente piccolo" (non possiamo usare "$f(x_n) = 0$" come criterio di arresto, perché in aritmetica di macchina rischia di non succedere mai!). Ma quanto piccolo è "sufficientemente piccolo"? Dipende dal valore di $f'(x_*)$, come abbiamo visto nell'analisi del condizionamento. Possiamo sviluppare un criterio euristico (a differenza del precedente non abbiamo una garanzia assoluta! Esempio con disegno).

Sappiamo dall'analisi precedente che se $x_n \in [x_* - \varepsilon, x_* + \varepsilon]$ allora $|f(x_n)| \dot{\leq} |f'(x_*)|\varepsilon$ (trascurando termini di ordine ε^2). Il valore di $f'(x_*)$ è ignoto, ma possiamo approssimarlo con $\frac{f(b_n)-f(a_n)}{b_n-a_n}$, che per il teorema di Lagrange è uguale a $f'(\xi)$ per un certo $\xi \in (a_n, b_n)$. Quindi un possibile criterio di arresto è: fermiamo il metodo quando

$$|f(x_n)| \leq \left|\frac{f(b_n) - f(a_n)}{b_n - a_n}\right| \varepsilon$$

(*criterio di arresto sul residuo*).

Vantaggi e svantaggi del metodo di bisezione Vantaggi:

- Richiede solo continuità di f.
- Converge *sicuramente* se è rispettata l'ipotesi $f(a)f(b) < 0$: questo non è vero per tutti i metodi, come vedremo.
- 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 trovare un intervallo di separazione da cui iniziare.
- Convergenza lenta. (Ora vediamo.)

Convergenza del metodo di bisezione Abbiamo una successione di approssimazioni x_n di una certa soluzione esatta x_*. La successione si dice *convergente* se l'errore $e_n := x_n - x_*$ tende a 0. Ma quanto velocemente? Uno dei casi più comuni è quello che e_n si comporti al limite come una successione geometrica:

$$|e_1| = a,\ |e_2| = ar,\ |e_3| = ar^2,\ |e_4| = ar^3, \dots$$

Ovviamente serve $r < 1$ per avere convergenza. Questo comportamento si verifica (al limite) quando riusciamo a dimostrare che

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} = r < 1.$$

Questa proprietà si chiama *convergenza lineare*. Il nome viene dal fatto che se tracciamo un grafico di $|e_n|$ in funzione di n *in scala logaritmica sulle* y, i punti tendono ad essere allineati lungo una retta di coefficiente angolare $\log r$.

Nel caso del metodo di bisezione non riusciamo a dimostrare esattamente questa relazione, ma una molto simile: $|e_n| \le \varepsilon_n$, dove ε_n è una successione che converge linearmente: difatti abbiamo visto che

$$|e_n| \le \frac{b_n - a_n}{2^n} =: \varepsilon_n,$$

e naturalmente

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

Si parla in questo caso di *convergenza pressoché lineare*.

4.3 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 x_* di $f(x_*) = 0$ è anche un *punto fisso* di Φ, cioè soddisfa $x_* = \Phi(x_*)$. 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 x_* 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

$$x_{n+1} = \Phi(x_n), \quad n = 0, 1, 2, \dots \tag{4.1}$$

Possiamo dimostrare un risultato di convergenza:

Teorema 4.1. *Sia $\Phi \in \mathcal{C}^1([a, b])$, e $x_* \in (a, b)$ un suo* punto fisso*, cioè un punto tale che $\Phi(x_*) = x_*$. Supponiamo che esista un intervallo chiuso $I = [x_* - \rho, x_* + \rho] \subseteq [a, b]$ tale che $|\Phi'(x)| < 1$ per ogni $x \in I$. Allora, il metodo* (4.1) *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);*
- *$\lim_{n\to\infty} x_n = x_*$.*

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

$$|x_n - x_*| \leq L^n \rho \quad \text{per ogni } n \geq 0. \tag{4.2}$$

Per $n = 0$, la (4.2) diventa $|x_0 - x_*| \leq \rho$, che è vera perché $x_0 \in I$. Supponiamo la (4.2) vera per un certo n, e dimostriamola per $n + 1$:

$$|x_{n+1} - x_*| = |\Phi(x_n) - \Phi(x_*)| \leq |\Phi'(\xi_n)||x_n - x_*| \leq LL^n r = L^{n+1}\rho, \tag{4.3}$$

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

Questo completa la dimostrazione della (4.2). Da questa relazione si ha che $x_n \in I$, visto che $|x_n - x_*| \leq L^n \rho \leq \rho$. Inoltre

$$0 \leq |x_n - x_*| \leq L^n \rho$$

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

Inoltre, possiamo dimostrare che il metodo converge (almeno) linearmente: dalla (4.3), si ha

$$\lim_{n\to\infty} \frac{|x_{n+1} - x_*|}{|x_n - x_*|} = \lim_{n\to\infty} |\Phi'(\xi_n)| = |\Phi'(x_*)| < 1.$$

L'ultima uguaglianza vale perché abbiamo $0 \le |\xi_n - x_*| \le |x_n - x_*|$, e quindi anche $\xi_n \to x_*$. 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.

Perché "almeno"? Perché se $|\Phi'(x_*)| = 0$, la convergenza può essere anche più veloce. In un grafico dell'errore in scala logaritmica, in questo caso le rette tendono ad essere verticali.

4.4 Convergenza locale

Osservazione 4.2. Se $\Phi(x) \in \mathcal{C}^1([a, b])$ e $|\Phi'(x_*)| < 1$ per un certo punto fisso x_*, allora per continuità possiamo prendere un ρ sufficientemente piccolo da avere $|\Phi'(x)| < 1$ per ogni $x \in I = [x_* - \rho, x_* + \rho]$. Quindi le ipotesi del teorema 4.1 sono verificate. In questo caso si dice che il metodo *converge localmente*, cioè, esiste un intervallo I centrato nella soluzione tale che il metodo converge per $x_0 \in I$.

Svantaggi:

- Convergenza non garantita se non in un intorno della soluzione. È difficile calcolare quanto è grande questo intorno.

Vantaggi del metodo:

- Converge davvero (almeno) linearmente, a differenza del metodo di bisezione; i residui non 'zig-zagano'.
- A seconda della scelta della Φ, può essere anche molto veloce (vedremo tra poco).

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 $|\Phi'(x_*)| > 1$, 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 $x_0 \in I$ esiste un n tale che $x_n \notin I$. Non vediamo questa dimostrazione.)

4.5 Metodo di Newton

Il metodo di Newton è un metodo per la ricerca di zeri che offre convergenza più veloce nella maggior parte dei casi, 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 della 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 $\Phi(x) = x - \frac{f(x)}{f'(x)}$. A questa funzione si applica la nostra Osservazione 4.2? Abbiamo

$$\Phi'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}; \tag{4.4}$$

pertanto se $f'(x_*) \neq 0$ abbiamo $\Phi'(x_*) = 0$. Possiamo allora trovare un intervallo I centrato in x_* in cui la $\Phi(x)$ esiste (il denominatore non si annulla) e $\Phi'(x) < 1$ per ogni $x \in I$ (per continuità).

Svantaggi:

- Richiede di saper calcolare non solo f ma anche f'.
- Costo computazionale maggiore: ad ogni passo, dobbiamo valutare f e f' una volta.
- Convergenza non garantita! Ci sono esempi in cui parto con x_1 lontano dalla soluzione e il metodo non converge.

Vantaggi:

- Nella maggior parte dei casi, converge molto più velocemente del metodo di bisezione.

Convergenza quadratica

Teorema 4.3. *Supponiamo $f \in \mathcal{C}^2([a,b])$. Se il metodo di Newton converge a $x_* \in (a,b)$ e se $f'(x_*) \neq 0$ (*radice semplice*), allora*

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

(cioè il limite esiste ed è finito.)

Qualche commento prima della dimostrazione. Questo implica (solitamente) convergenza molto più veloce che non quella del metodo di bisezione. Facciamo un conto approssimato, supponendo che sia valida quella relazione che vale solo al limite: se $e_n = 10^{-2}$, allora $e_{n+1} = c10^{-4}$, $e_{n+2} = c^3 10^{-8}$, $e_{n+3} = c^7 10^{-16}$, ... Se il valore di c non è troppo grande, questi valori vanno a zero molto più velocemente di una serie geometrica; e difatti

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} = \lim \frac{|e_{n+1}|}{|e_n|^2} \lim |e_n| = c \cdot 0 = 0. \tag{4.5}$$

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 x_* e l'errore e_n va sotto una certa soglia, esso comincia ad andare a zero molto velocemente.

Si dice che il metodo di Newton ha *convergenza (almeno) quadratica* o *di ordine (almeno) 2*. In generale, se riusciamo a dimostrare che un metodo produce successioni tali che

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} = c \in [0, \infty]$$

per un qualche $p > 1$, diciamo che il metodo ha *ordine di convergenza almeno* p. (Notare che per parlare di convergenza lineare, con $p = 1$, serviva assumere che $c < 1$; qui non più, il metodo converge comunque). Se $c \neq 0$, diciamo che *ha ordine di convergenza esattamente* p. Non lo dimostriamo, ma questi termini si comportano esattamente come vi aspettereste dalle parole che stiamo usando: per esempio se un metodo ha ordine esattamente 3, allora ha ordine almeno 2....

Dimostrazione del teorema 4.3. Usiamo uno sviluppo di Taylor (con resto di Lagrange) in x_n per scrivere l'uguaglianza

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

che vale per uno ξ_n compreso tra x_* e x_n.

Dividiamo per $f'(x_n)$ per ottenere

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

Allora

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

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

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

□

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 $|x_n - x_*| \leq \varepsilon$. 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 ha $f'(x_*) = 0$? Studiamo il caso più comune, che è quello di funzioni che si scrivono nella forma $f(x) = (x - x_*)^m g(x)$, per una funzione $g(x)$ che soddisfa $g(x_*) \neq 0$, e un $m \in \mathbb{N}$ (intero). Se una funzione si scrive in questo modo allora diciamo che x_* è uno *zero di molteplicità (esattamente)* m.

(Similmente agli ordini di convergenza, "esattamente" perché abbiamo raccolto tutti i fattori $x - x_*$ che potevamo...)

In termini di derivate, possiamo "vedere" la molteplicità in questo modo: se una funzione ha uno zero di molteplicità m (e $g(x)$ è derivabile almeno m volte), allora

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

Questo si può verificare per induzione: poiché

$$\frac{d}{dx}(x-x_*)^m g(x) = m(x-x_*)^{m-1}g(x)+(x-x_*)^m g'(x) = (x-x_*)^{m-1}(mg(x)+(x-x_*)g'(x)),$$

la derivata $f'(x)$ ha uno zero di molteplicità $m-1$. Allo stesso modo $f''(x)$ ha uno zero di molteplicità $m-2$, e così via.

Se inseriamo la forma $f = (x - x_*)^m g(x)$ nella formula (4.4) per la derivata di $\Phi(x)$ (e supponiamo $g \in \mathcal{C}^2$), otteniamo che Φ' si può estendere per continuità a

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

(vi invito a provare a fare i conti a casa, non sono complicati). Quindi in presenza di uno zero di molteplicità $m > 1$ il metodo di Newton converge, ma solo linearmente. Però se conosciamo il valore di m a priori, allora possiamo fare una modifica del metodo che converge di nuovo quadraticamente (*metodo di Newton modificato*):

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)} \quad n = 0, 1, 2, \ldots \tag{4.6}$$

(notare la m di fronte al secondo termine). In questo caso, ripetendo i calcoli otteniamo che $\Phi'(x_*) = 0$. Si può inoltre dimostrare che il metodo (4.6) converge (localmente) con ordine almeno 2.

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: guardare $\Phi'(x_*)$).

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 (in realtà una moltiplicazione, quando si va a scrivere il metodo per esteso). L'ordine di convergenza di questo metodo è $p = \frac{1+\sqrt{5}}{2} \approx 1.618$ (non lo dimostriamo, è più complesso: difatti non è un metodo del tipo $x_{n+1} = \Phi(x_n)$). Per applicarlo, serve partire da *due* punti iniziali x_0 e x_1.

Esercizi Esempio: metodo di Newton su $f(x) = \alpha(x - x_*)$ (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).

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 $x_* \approx 0.7391$ che non contenga $-\pi/2 \approx -1.5708$ né $\pi/2 \approx 1.5708$.

[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 5

Sistemi di equazioni lineari

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.

Prodotto matrice-matrice: AB definito come $(AB)_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$, prodotti scalari nello stesso modo. Ben definito solo quando $A \in \mathbb{C}^{m\times n}$ e $B \in \mathbb{C}^{n\times p}$ hanno la dimensione "interna" uguale, e produce una matrice $AB \in \mathbb{C}^{m\times p}$. L'ordine dei fattori conta! $AB \neq BA$ (e può anche darsi che uno sia ben definito e l'altro no).

A parte questo, valgono le "normali" proprietà di addizione e moltiplicazione: $(A+B)C = AC + BC$, $A(BC) = (AB)C$. Un'altra che non vale è l'annullamento del prodotto: $AB = 0$ può verificarsi anche se $A, B \neq 0$. (Esempio: $\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 0_{2\times 2}$.)

Possiamo moltiplicare matrici e vettori anche per degli *scalari* (cioè dei numeri $\in \mathbb{C}$). In questo caso, l'ordine non conta e possiamo "portare fuori" scalari da un prodotto, $A(\alpha B) = \alpha(AB)$.

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 $A^{-1}e_1, A^{-1}e_2, \dots A^{-1}e_n$; per questo è più lento.

Altri algoritmi da evitare sono quelli basati su determinanti: 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): $\|v\|_2 = \sqrt{\sum_{i=1}^{n} |v_i|} = \sqrt{v^* v}$.
- Norma infinito: $\|v\|_\infty = \max_{i \in \{1,2,\dots,n\}} |v_i|$.

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

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

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

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

(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. $f(A) \geq 0$ 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. $f(A + B) \leq f(A) + f(B)$ per ogni $A, B \in \mathbb{C}^{n\times n}$.
4. $f(AB) \leq f(A)f(B)$ 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 $\|Av\|_p \leq \|A\|_p\|v\|_p$ 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

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

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

5.3 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:

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

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*):

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

La complessità è di nuovo n^2 operazioni.

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

5.4 Eliminazione di Gauss e fattorizzazione LU

Nel caso di una matrice non triangolare, vediamo un algoritmo che è sostanzialmente una variante dell'eliminazione di Gauss che avete già visto ad algebra lineare.

Ricordiamo come funziona l'eliminazione di Gauss. Partiamo da $A \in \mathbb{C}^{n\times n}$, e vogliamo applicare ripetutamente delle trasformazioni in modo da ottenere una matrice 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. Nel k-esimo passo, introdurremo zeri anche nella k-esima colonna. Lo facciamo sommando ad 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 $-\frac{(A_k)_{ik}}{(A_k)_{kk}}$. Chiamiamo A_{k+1} la matrice ottenuta in questo modo, dopo aver creato uno zero in tutte le posizioni della colonna $k$esima al di sotto dela diagonale.

La parte che probabilmente non avete visto ad algebra lineare è che questa trasformazione si può vedere come una moltiplicazione: $A_{k+1} = L_k A_k$. La matrice L_k è

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

dove v_k è il vettore tale che

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

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

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

Supponiamo per ora di poter effettuare l'eliminazione di Gauss senza mai incontrare divisioni per zero sulla diagonale (e senza mai fare scambi di righe/-colonne). Dopo $n-1$ passi (l'ultimo passo $k = n$ non ha alcun effetto!), abbiamo ottenuto una matrice triangolare superiore

$$A_n = L_{n-1}A_{n-1} = L_{n-1}L_{n-2}A_{n-2} = \cdots = L_{n-1}L_{n-2}\dots L_2 L_1 A.$$

Notiamo che le L_k sono invertibili, perché sono triangolari inferiori con tutti 1 sulla diagonale. Possiamo riscrivere la relazione come

$$A = \underbrace{L_1^{-1}L_2^{-1}\dots L_{n-1}^{-1}}_{=L}\underbrace{A_n}_{=U} = A.$$

La matrice che abbiamo chiamato U (per "upper") è triangolare superiore, per come funziona l'eliminazione di Gauss. La matrice che abbiamo chiamato L (per "lower") è triangolare inferiore, perché prodotto di triangolari inferiori. Inoltre, dimostreremo tra poco che gli elementi sulla diagonale di L sono tutti uguali a 1.

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

Prima di indagare ulteriormente su quando questo calcolo si può portare a termine, dimostriamo alcuni risultati che ci portano a un'espressione esplicita della matrice L.

Una matrice L_k si chiama *matrice elementare di Gauss* se si scrive nella forma $I - v_k e_k^T$, dove $(v_k)_i = 0$ per ogni $i \leq k$.

Lemma 5.2. *Valgono le seguenti due proprietà:*

1. $L_k^{-1} = I + v_k e_k^T$.
2. $L_1^{-1} L_2^{-1} \dots L_k^{-1} = I + v_1 e_1^T + v_2 e_2^T + \cdots + v_k e_k^T$.

Dimostrazione. 1. Basta verificare che

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

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

$$L_1^{-1} L_2^{-1} \dots L_k^{-1} = (I + v_1 e_1^T + v_2 e_2^T + \cdots + v_{k-1} e_{k-1}^T)(I + v_k e_k^T).$$

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.

□

In particolare, il punto 2 ci dice che abbiamo

$$L_1^{-1} L_2^{-1} \dots L_{n-1}^{-1} = \begin{bmatrix} 1 & & & & & \\ v_{21} & 1 & & & & \\ v_{31} & v_{32} & 1 & & & \\ v_{41} & v_{42} & v_{43} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \ddots & \\ v_{n1} & v_{n2} & v_3 & \dots & v_{n,n-1} & 1 \end{bmatrix},$$

dove abbiamo indicato con v_{ik} l'entrata i-esima del vettore v_k. Questo mostra che per calcolare il prodotto $L_1^{-1} L_2^{-1} \dots L_{n-1}^{-1}$ non dobbiamo effettuare nessuna ulteriore operazione aritmetica: basta "scrivere" gli elementi v_{ik} nel punto giusto della matrice.

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

```
    A(k+1:n, k) = 0;
    A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - L(k+1:n,k)*A(k,k+1:n);
end
U = A;
```

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 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.

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 $b = Ax = LUx$.

L'algoritmo è composto di tre passi:

1. Calcolo la fattorizzazione $A = LU$.
2. Calcolo il vettore $y = Ux$ risolvendo il sistema lineare $b = Ly$ (sostituzione in avanti).
3. Calcolo il vettore x risolvendo il sistema lineare $y = Ux$ (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 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.

Notare che nel nostro codice non scriviamo mai esplicitamente moltiplicazioni `Lk*Ak`, ma scriviamo del codice che usa la struttura particolare di queste matrici L_k. Se invece facessimo nell'altro modo, 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!

Esistenza della fattorizzazione LU Cosa può andare storto? Una sola cosa: quando dobbiamo dividere per $(A_k)_{kk}$ (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 5.3. *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.*

Dimostrazione. Supponiamo di aver effettuato i primi $k-1$ passi dell'eliminazione di Gauss, ed essere pronti ad effettuare il k-esimo. Abbiamo $L_{k-1}L_{k-2}\dots L_1 A = A_{k-1}$, cioè riordinando

$$A = L_1^{-1}\dots L_{k-1}^{-1}A_{k-1} = \begin{bmatrix} \star & 0 \\ \star & I \end{bmatrix}\begin{bmatrix} \star & \star \\ 0 & \star \end{bmatrix},$$

dove abbiamo zeri nelle prime $k-1$ colonne di U e non-zeri nelle prime $k-1$ colonne di L.

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

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

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

Teorema 5.4. *Una matrice dominante diagonale è invertibile.*

Dimostrazione. Supponiamo invece che $Ax = 0$ per un qualche vettore $x \neq 0$. Sia x_m l'elemento maggiore in valore assoluto, $|x_m| = \max_{i=1,2,\ldots,n}|x_i|$. Chiaramente $x_m \neq 0$ (altrimenti x sarebbe il vettore nullo). Allora possiamo scrivere

$$0 = (Ax)_m = \sum_{j=1}^{n} A_{mj}x_j.$$

Portiamo dall'altro lato l'elemento diagonale,

$$-A_{mm}x_m = \sum_{j\neq m} A_{mj}x_j,$$

dividiamo per x_m e prendiamo i valori assoluti:

$$|A_{mm}| = \left|\sum_{j\neq m} A_{mj}\frac{x_j}{x_m}\right| \leq \sum_{j\neq m}|A_{mj}|\frac{|x_j|}{|x_m|} \leq \sum_{j\neq m}|A_{mj}|.$$

Questo 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 valgono 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 catastrofico. Non facciamo un'analisi dettagliata dell'errore algoritmico, ma vediamo direttamente un esempio problematico. 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. 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 $(A_k)_{k:n,k}$.

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.

Eliminazione di Gauss con pivoting (parziale) 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_1A, dove P_1 è la matrice che differisce dall'identità solo per lo scambio delle righe 1 e p.

Similmente, prima ogni passo k, individuiamo la riga dove sta il pivot $|a_{pk}| = \max|a_{k:n,k}|$, e scambiamola con la riga k (sia nella matrice U che nella matrice L su cui stiamo lavorando). Difatti, ripercorrendo l'algoritmo è possibile vedere che queste due matrici sono quelle che avremmo prodotto se avessimo rimpiazzato la matrice di partenza A con una che ha le righe k e p scambiate, operazione che è rappresentata da un prodotto a sinistra P_kA.

Rimettendo tutto insieme, otteniamo che quella che abbiamo calcolato è una fattorizzazione LU della matrice $P_{n-1}P_{n-2}\dots P_2P_1A$. Il prodotto $P = P_{n-1}P_{n-2}\dots P_2P_1$ è una matrice che esegue una permutazione delle righe, una cosiddetta *matrice di permutazione*. Tali matrici hanno come righe le righe della matrice identità (in un qualche ordine).

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 dev''essere quadrata');
end
```

```
L = eye(n);
P = eye(n);
for k = 1:n-1
   [valore, posizione] = max(abs(A(k:n,k)));
   p = posizione + k-1;
   if valore == 0
       error('matrice esattamente singolare');
   end
   A([p k], 1:n) = A([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);
   L(k+1:n, k) = A(k+1:n, k) / A(k, k);
   A(k+1:n, k) = 0;
   A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - L(k+1:n,k)*A(k,k+1:n);
end
U = A;
```

Le uniche operazioni che abbiamo aggiunto sono la ricerca del massimo e gli scambi di righe. Queste sono operazioni che richiedono comunque 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 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 "ribaltarla" rispetto alla diagonale principale.

Pertanto possiamo convertire la scrittura in una fattorizzazione $P^T LU = A$, e usarla per risolvere sistemi lineari: da $P^T LUx = b$ possiamo chiamare $y = Ux$, $z = Ly$ e risolvere nell'ordine

$$P^T z = b, \quad Ly = z, \quad Ux = y.$$

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

Matrici simmetriche e 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 A_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 $A_2 = L_1 A_1$, definiamo $A_2 = L_1 A_1 L_1^T$ (moltiplicando a destra e a sinistra). Questa matrice è simmetrica, difatti $(L_1 A_1 L_1^T)^T = (L_1^T)^T (A_1)^T L_1^T = L_1 A_1 L_1^T$. In più, 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

$$A_2 = \begin{bmatrix} A_{11} & 0 & 0 & 0 \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \end{bmatrix}.$$

Il blocco indicato con $\star$ è simmetrico, e possiamo continuare nello stesso modo, definendo $A_3 = L_2 A_2 L_2^T$, e così via. Alla fine della procedura, otteniamo una matrice diagonale, che possiamo chiamare D. Più nel dettaglio, abbiamo

$$D = L_{n-1} L_{n-2} \dots L_2 L_1 A L_1^T L_2^T \dots L_{n-2}^T L_{n-1}^T.$$

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

$$A = \underbrace{(L_1^{-1} L_2^{-1} \dots L_{n-1}^{-1})}_{=L} D \underbrace{(L_{n-1}^T)^{-1} (L_{n-2}^T)^{-1} \dots (L_2^T)^{-1} (L_1^T)^{-1}}_{=L^T} = LDL^T.$$

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. La differenza maggiore è che questa volta tutti gli aggiornamenti della forma

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

coinvolgono matrici simmetriche. Quindi non è necessario calcolare tutti gli elementi: basta calcolare quelli della parte triangolare inferiore (inclusa la diagonale), e riempire quelli della parte triangolare superiore 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 non è semplicissimo applicare quest'ultimo trucco di riempire la parte superiore per simmetria senza scrivere cicli `for` direttamente. 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.

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 $x^TAx > 0$ per ogni vettore $x \in \mathbb{R}^n$, $x \neq 0$.

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

Si può dimostrare che la seconda condizione è equivalente a chiedere che tutti gli autovalori di A siano positivi (sono comunque sicuramente reali per il teorema spettrale).

Valgono le seguenti proprietà.

1. Una matrice definita positiva è invertibile, visto che non ha autovalori 0.
2. Una matrice definita positiva ha tutti gli elementi sulla diagonale (strettamente) positivi; difatti $a_{kk} = e_k^TAe_k > 0$. (Notare che queste sono condizioni necessarie ma non sufficienti!)
3. Se A è semidefinita positiva e M è invertibile, allora lo è anche MAM^T: difatti $x^TMAM^Tx = (M^Tx)^TA(M^Tx) > 0$.

Con 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 questa fattorizzazione 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

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

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)`.

5.5 Laboratorio su LDL ed esistenza LU

Esercizio (teorico): per quali valori di α esiste la fattorizzazione LU di

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

Oppure, in alternativa, procediamo (con carta e penna) con l'eliminazione di Gauss.

Esercizio: scriviamo una `function s = somma_fuori_diagonale(A, i)` che, data in input una matrice A, calcola $\sum_{\substack{j=1 \\ j\neq i}}^{n} |A_{ij}|$. Mostrare soluzioni fatte sia con `for j=1:n; if j ~= i` che con
`sum(abs(A(i,1:i-1))) + sum(abs(A(i,i+1:n)))`.

Poi usiamola per scrivere una funzione che controlla se una matrice è predominante diagonale per righe. Come facciamo a restituire valori vero/falso?

```
function result = isdominant(A)
  % cicli, condizioni, eccetera
  result = true;
  return

  result = false;
  % "return" non serve alla fine
```

Altro esercizio: scriviamo una funzione che controlla se una matrice è predominante diagonale per colonne. Facile se riutilizziamo il codice dell'altra funzione!

Buona norma di programmazione: se potete, meglio riutilizzare che riscrivere! E meglio usare funzioni corte che fanno una cosa sola, come stiamo facendo qua. Sono più facili da testare.

Altro esercizio: proviamo a scrivere una `function result = issymmetric(A)` che testa se la matrice A è simmetrica.

Come facciamo a farlo? Occhio che `A == A'` non fa quello che volete: in Matlab, `A == B` restituisce una matrice $n \times n$ di valori vero/falso (o 0/1) che ci dice quali elementi A_{ij} sono uguali a quali elementi B_{ij}. Ci servono due cicli for, oppure `isequal(A, A')`.

Altro esercizio: calcoliamo la fattorizzazione LDL di una matrice simmetrica A (senza pivoting).

Il punto delicato è come calcolare l'update $A_{k+1:n,k+1:n} \leftarrow A_{k+1:n,k+1:n} - L_{k+1:n,k}A_{k,k+1:n}$ facendo operazioni solo per parte triangolare inferiore. Per questo, andiamo a scrivere questo update come

$$A_{ij} \leftarrow A_{ij} - L_{ik}A_{kj}, \quad i,j = k+1, k+2, \ldots, n$$

ed usiamo dei cicli for che effettuano il calcolo solo su una delle due parti. Definiamo una funzione ausiliaria `symmetric_update(A, l, u)`.

Possibile errore: un ciclo del tipo

```
for i = k+1:n
  for j = k+1:n
    if j>i
      A(i,j) = A(j,i);
    else
      A(i,j) = ... ;
    end
  end
end
```

rischia di copiare un valore dalla parte triangolare inferiore prima che questo venga assegnato; quindi copia solo uno zero!

Domanda aperta: come fareste a controllare se una matrice è positiva definita, oltre che simmetrica? Impossibile testare che $x^T Ax > 0$ su un numero infinito di vettori! Un modo è testare solo su alcuni vettori casuali. Oppure calcolare gli autovalori di A e controllare se sono strettamente maggiori di zero. In alternativa: è vero che se portiamo a termine la fattorizzazione LDL (e otteniamo solo valori positivi sulla diagonale) allora A è SPD? Abbiamo (brevemente) dimostrato l'altra implicazione a lezione; ora vediamo che anche questa è un "se e solo se".

Esercizio extra se avanza tempo: fattorizzazione LU di una matrice di Hessenberg superiore.

5.6 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 molto bassa, 10% o qualcosa di simile; non correte ad usarlo non appena vedete uno zero in una matrice. Ve lo dico solo per conoscenza, ma non vediamo dettagli su come gestirle qui.

I metodi che abbiamo visto finora per risolvere sistemi lineari non sono l'ideale per matrici sparse, perché spesso "eliminano" 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, $x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(3)}, \dots$ (ogni $x^{(i)}$ è un vettore; mettiamo l'indice in alto per non confonderci con gli elementi). Sotto condizioni opportune (che vedremo), questa successione converge alla soluzione x di $Ax = b$.

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 $x \in \mathbb{C}^{n \times n}$ del sistema lineare soddisfa

$$Ax = b \iff (M-N)x = b \iff Mx = Nx+b \iff x = M^{-1}(Nx+b) \quad (5.2)$$

Questo suggerisce di impostare un'iterazione di punto fisso

$$\begin{cases} x^{(0)} \in \mathbb{C}^n & \text{fissato;} \\ x^{(k+1)} = M^{-1}(Nx^{(k)} + b) & k = 0, 1, 2, \dots \end{cases} \quad (5.3)$$

Si tratta di una versione "multidimensionale" del metodo di punto fisso $x_{k+1} = \Phi(x_k)$ che abbiamo già visto. Parlando di quel metodo, avevamo visto che spesso la convergenza avviene solo quando $x^{(0)}$ è sufficientemente vicino alla soluzione esatta 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 $x^{(0)} \in \mathbb{C}^n$ la successione generata dal metodo converge alla soluzione x del sistema lineare.

Per studiare la convergenza, definiamo $e^{(k)} = x^{(k)} - x$ l'errore (assoluto) al passo k. Usando la (5.2) e la (5.3) abbiamo per ogni $k = 0, 1, \dots$

$$e^{(k)} = x^{(k)} - x = M^{-1}(Nx^{(k-1)} - b) - M^{-1}(Nx - b) = M^{-1}(Nx^{(k-1)} - Nx) = M^{-1}Ne^{(k-1)}. \quad (5.4)$$

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

Teorema 5.5. *Il metodo iterativo* (5.3) *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 (5.4) is ha

$$e^{(k)} = He^{(k-1)} = HHe^{(k-2)} = \cdots = H^k e^{(0)}.$$

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

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 $Hv = v\lambda$, $|\lambda| \geq 1$, allora esiste una scelta di $x^{(0)}$ per cui l'errore $e^{(k)}$ non converge a zero. Per farlo, prendiamo $x^{(0)} = x + v$. Allora abbiamo $e^{(0)} = x^{(0)} - x = v$, e

$$e^{(k)} = H^k v = v\lambda^k \tag{5.5}$$

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

Se $|\lambda| \geq 1$ (e $v \neq 0$, visto che v è un autovettore), allora la quantità $v\lambda^k$ 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 5.6. *Se* $\|H\|_p \leq 1$ *per una qualunque norma matriciale indotta, allora il metodo iterativo* (5.3) *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

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

con uguaglianza per quasi tutti i vettori di partenza $x^{(0)}$. Pertanto il metodo converge linearmente. La convergenza è tanto più veloce (pendenza della retta in scala logaritmica più vicina a $-\infty$) quanto $\rho(H)$ è piccolo.

Criteri di arresto per metodi iterativi per sistemi lineari [TODO: forse questa osservazione va introdotta in una versione scalare prima, quando si parla del metodo del punto fisso.]

Possiamo arrestare un metodo iterativo quando $\|x^{(k-1)} - x^{(k)}\|$ è al di sotto di una certa tolleranza ε, oppure quando $\|Ax^{(k)} - b\| \leq \varepsilon$. Vediamo cosa implica il primo criterio. Abbiamo

$$x^{(k-1)} - x^{(k)} = (x^{(k-1)} - x) - (x^{(k)} - x)) = e^{(k-1)} - e^{(k)} = e^{(k-1)} - He^{(k-1)} = (I - H)e^{(k-1)},$$

e quindi quando ci arrestiamo

$$\|e^{(k)}\| = \|He^{(k-1)}\| = \|H(I-H)^{-1}(x^{(k-1)} - x^{(k)})\| \leq \|H(I-H)^{-1}\| \|x^{(k-1)} - x^{(k)}\| \leq \|H(I-H)^{-1}\|\varepsilon.$$

Qui la norma matriciale che compare è quella indotta dalla norma vettoriale scelta.

Il risultato vale supponendo che $I - H$ sia invertibile. Si può dimostrare che quando $\rho(H) < 1$ (e quindi il metodo converge) vale l'identità

$$(I - H)^{-1} = I + H + H^2 + H^3 + \dots,$$

che è una versione matriciale della formula per la somma della serie geometrica. Questo mostra anche che la stima per $\|e^{(k)}\|$ è tanto peggiore quanto più lentamente converge il metodo.

Metodi di Jacobi e Gauss–Seidel 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 $Mx^{(k+1)} = b + Nx^{(k)}$:

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

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 $x^{(k+1)}$ a partire da $x^{(k)}$. 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 $Mx^{(k+1)} = b + Nx^{(k)}$:

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

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

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{n} A_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} A_{ij} x_j^{(k)}}{A_{ii}}, \quad i = 1, 2, \dots, n. \tag{5.6}$$

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

La (5.6) 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 freschi" $x_j^{(k+1)}$ appena 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 e quello di Gauss–Seidel no.

Notare che il metodo di Gauss–Seidel può essere implementato con *una sola* variabile (vettore) x che contiene ad ogni passo le approssimazioni "più fresche" degli elementi della soluzione; ad ogni passo $i = 1, 2, \dots, n$ rimpiazziamo un

elemento $x_i^{(k)}$ con $x_i^{(k+1)}$:

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

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 5.7. *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 $A_{ii} \neq 0$:

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

Mostreremo che $\rho(H) < 1$. Supponiamo invece per assurdo che H abbia un autovalore λ con $|\lambda| \geq 1$. Allora

$$0 = \det(H - \lambda I) = \det(M^{-1}(N - \lambda M)) = \det M^{-1} \det(N - \lambda M).$$

Poiché $\det M^{-1} \neq 0$ (la matrice M^{-1} è invertibile!), dev'essere $\det(N - \lambda M) = 0$.

La matrice $N - \lambda M$ ha elementi

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

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

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

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, andando a scrivere le disuguaglianze corrispondenti notiamo che sono "più forti" di quelle che danno la predominanza diagonale di A, se $|\lambda| \geq 1$. Quindi, in particolare, $N - \lambda M$ è invertibile, ed è impossibile che abbia determinante 0. Questo completa la dimostrazione per assurdo. □

5.7 Laboratorio su metodi iterativi per sistemi lineari

- Fatto insieme: scrivere una funzione che esegue k iterazioni del metodo di Jacobi. Lasciato a loro: 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.

5.8 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?

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.

Esempio (dalle slides di Magherini):

$$F(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} 2x_1 + \cos(x_2) \\ \sin(x_1) + 2x_2 - \pi \end{bmatrix}, \tag{5.7}$$

con soluzione $(0, \pi)$. 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 Possiamo definire, generalizzando la derivata prima di una funzione, la *matrice Jacobiana*

$$J_F(x) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1}(x) & \frac{\partial F_1}{\partial x_2}(x) & \dots & \frac{\partial F_1}{\partial x_n}(x) \\ \frac{\partial F_2}{\partial x_1}(x) & \frac{\partial F_2}{\partial x_2}(x) & \dots & \frac{\partial F_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1}(x) & \frac{\partial F_n}{\partial x_2}(x) & \dots & \frac{\partial F_n}{\partial x_n}(x) \end{bmatrix}$$

(mnemonica: $F(x)$ produce un vettore colonna, e ogni colonna è una sua derivata rispetto a una variazione di x_j). Per esempio per la (5.7) abbiamo

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

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

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

questa volta $x, h \in \mathbb{R}^n$, e nel secondo termine c'è un prodotto matrice-vettore. Come nella derivazione del metodo di Newton, possiamo rimpiazzare F con una sua approssimazione lineare (piano tangente), e trovare uno zero di quella, ottenendo il metodo

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

Valgono risultati analoghi a quelli del caso scalare: su funzioni sufficientemente regolari, il metodo converge a patto di prendere $x^{(0)}$ sufficientemente vicino a una soluzione, e lo fa quadraticamente. Nella pratica, non vogliamo calcolare un'inversa esplicita, ma ad ogni passo otteniamo $J_F(x^{(k)})$ e calcoliamo una sua fattorizzazione LU (o altre varianti).

Metodo delle corde Nel caso multivariato, diventa interessante considerare alcune varianti come il metodo delle corde, che non erano troppo interessanti nel caso scalare. Il motivo è che calcolare e fattorizzare la matrice $J_F(x^{(k)})$ 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 $k = 0, 1, 2, \dots$

- Calcolo $F(x^{(k)})$;
- In alcune iterazioni, valuto $A = J_F(x^{(k)})$ e calcolo una sua fattorizzazione $A = P^T LU$; 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 $Ah = F(x^{(k)})$;
- Pongo $x^{(k+1)} = x^{(k)} - h$.

Costo computazionale: ad ogni passaggio dobbiamo sicuramente 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 di nuovo dipende da quanto spesso si ricalcola la fattorizzazione; se viene calcolata solo una volta all'inizio è lineare, se viene calcolata ad ogni iterazione è quadratica.

5.9 Sistemi sovradeterminati

Equazioni normali Supponiamo di avere un sistema del tipo

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

Se $m > n$ (A alta e stretta), la soluzione spesso non esiste: abbiamo più equazioni che incognite. Dall'algebra lineare, sappiamo che Ax 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 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 Ax, che va più vicino al vettore b. In altre parole, cerchiamo

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|.$$

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

$$\min_{x\in\mathbb{R}^n} r^T r = \min_{x\in\mathbb{R}^n} (Ax - b)^T (Ax - b). \tag{5.8}$$

Il problema (5.8) è detto *problema dei minimi quadrati*, visto che compaiono per l'appunto i quadrati di r. Dimostriamo il seguente risultato.

Teorema 5.8. *Sia $x \in \mathbb{R}^n$ un vettore tale che $A^T r = A^T(Ax - b) = 0$. Allora, x (e il residuo r corrispondente) risolvono il problema di minimo* (5.8).

Dimostrazione. Sia $y \in \mathbb{R}^n$ un altro vettore; allora il suo residuo vale

$$Ay - b = \underbrace{A(y - x)}_{=s} + \underbrace{Ax - b}_{=r} = s + r.$$

Abbiamo

$$\|Ay - b\|^2 = (Ay - b)^T (Ay - b) = (s + r)^T (s + r) = s^T s + s^T r + r^T s + r^T r.$$

Notiamo che il prodotto scalare $r^T s = s^T r$ si annulla: difatti,

$$s^T r = (A(x - y))^T r = (x - y)^T A^T r = 0.$$

Allora si ha

$$\|Ay - b\|^2 = s^T s + r^T r \geq r^T r.$$

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

Quindi per trovare la soluzione x del problema ci basta risolvere il sistema di equazioni

$$A^T A x = A^T b, \tag{5.9}$$

che si ottiene moltiplicando per A^T il sistema originale (insolubile) $Ax = b$. La (5.9) si chiama *metodo delle equazioni normali*, perché le equazioni corrispondenti dicono (in termini geometrici) che il residuo $r = Ax - b$ è ortogonale (prodotto scalare nullo) alle colonne di A.

Si può dimostrare che $A^T A$ è quadrata e SPD (quindi invertibile!) tutte le volte che le colonne di A sono linearmente indipendenti. Possiamo allora risolvere (5.9) usando la fattorizzazione di Cholesky. Questo metodo però può risultare instabile a volte: il condizionamento del sistema lineare (5.9) può essere molto più alto di quello del problema originale (che non abbiamo studiato). Esiste un altro metodo più stabile.

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

Teorema 5.9. *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 spezziamo 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 (5.9) è una soluzione anche del sistema lineare triangolare

$$R_1x = Q_1^Tb. \tag{5.10}$$

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 (5.10).

Questo metodo è più costoso del precedente ($2mn^2$ 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.

Capitolo 6

Interpolazione, approssimazione, e integrazione numerica

6.1 Approssimazione e interpolazione

Nel problema dell'interpolazione, supponiamo di sapere che una certa funzione è ottenuta come una combinazione lineare di n funzioni fissate,

$$\phi(x) = \alpha_1\phi_1(x) + \alpha_2\phi_2(x) + \cdots + \alpha_n\phi_n(x).$$

Le funzioni $\phi_i(x) : U \subseteq \mathbb{R} \to \mathbb{R}$ sono date; i coefficienti α_i invece sono ignoti e vogliamo determinarli.

Per fare questo, abbiamo a disposizione m coppie di valori $(x_1, y_1), \ldots, (x_m, y_m)$ tali che $\phi(x_i) = y_i$ (e inoltre assumiamo che gli x_i siano *distinti*, cioè, $x_i \neq x_j$ per ogni $i \neq j$).

Ho n coefficienti incogniti e m equazioni

$$\begin{aligned}
y_1 &= \alpha_1\phi_1(x_1) + \alpha_2\phi_2(x_1) + \cdots + \alpha_n\phi_n(x_1),\\
y_2 &= \alpha_1\phi_1(x_2) + \alpha_2\phi_2(x_2) + \cdots + \alpha_n\phi_n(x_2),\\
&\vdots \quad \vdots\\
y_m &= \alpha_1\phi_1(x_m) + \alpha_2\phi_2(x_m) + \cdots + \alpha_n\phi_n(x_m).
\end{aligned}$$

Possiamo vedere che questo è un sistema lineare: se poniamo

$$X = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \ldots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \ldots & \phi_n(x_m) \end{bmatrix} \in \mathbb{R}^{m\times n}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m, \quad \alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n,$$

queste equazioni diventano il sistema lineare $X\alpha = y$. Occhio alla notazione: qui X e y sono note, e α è la nostra incognita. Se $m = n$, questo è un sistema lineare

quadrato e posso pensare di risolverlo esattamente (ammesso che la matrice X sia invertibile). Questa si chiama *interpolazione* di funzioni. 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: mi sono date delle coppie (x_i, y_i) che soddisfano approssimativamente $\phi(x_i) \approx y_i$, 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 si chiama *approssimazione* di funzioni, o, dall'inglese, *fitting*. Per fare questo, ci basta usare uno degli algoritmi visti nella sezione precedente, quindi non ci servono strumenti nuovi. Vediamo però un po' di casi particolari.

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 $\phi_1(x) = 1, \phi_2(x) = x$. Quindi

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

Possiamo ottenere una formula più esplicita, espandendo i conti e usando la formula per l'inversa di una matrice 2×2, ma la cosa non è particolarmente interessante.

Polinomi trigonometrici Se abbiamo una funzione che sappiamo essere periodica di periodo π, una base "naturale" da usare è

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

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.

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

$$\phi_1(x) = 1, \quad \phi_2(x) = x, \quad \phi_3(x) = x^2, \dots, \phi_n(x) = x^{n-1}. \tag{6.1}$$

6.2 Interpolazione polinomiale

Vediamo più nel dettaglio il problema dell'interpolazione polinomiale, cioè il problema di trovare un polinomio di grado al più $n-1$ che passa (esattamente) per $m = n$ punti dati, $x_1, x_2, \ldots, x_n$. (I punti x_i sono detti *nodi*.) Questo corrisponde a scegliere la base (6.1), e quindi a risolvere (esattamente!) il seguente sistema lineare quadrato

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots x_n^{n-1} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

La matrice associata a questo sistema si chiama *matrice di Vandermonde*.

Risolubilità È possibile dimostrare il seguente risultato.

Teorema 6.1. *Per ogni scelta di $x_1, x_2, \ldots, x_n$ distinti, la matrice di Vandermonde è invertibile.*

Questo teorema implica che il problema dell'interpolazione polinomiale è sempre risolubile, cioè il risultato seguente.

Teorema 6.2. *Date n coppie di punti $(x_1, y_1), \ldots, (x_n, y_n)$, con gli x_i distinti, esiste uno e un solo polinomio $p(x)$ di grado $d \leq n-1$ tale che $p(x_1) = y_1, \ldots, p(x_n) = y_n$.*

Notare che sul grado abbiamo solo una disuguaglianza: nulla vieta che il coefficiente α_n di fronte a $\phi_n(x) = x^{n-1}$ sia uguale a zero per una particolare scelta dei punti (x_i, y_i). Per esempio, se scelgo $(1,1), (2,2), (3,3)$, allora il polinomio di grado $d \leq 2$ che passa esattamente per questi punti è $p(x) = x$, che in realtà ha grado 1.

Purtroppo in molti casi la matrice di Vandermonde, seppure invertibile, risulta mal condizionata, specialmente per valori grandi di n. Solitamente l'interpolazione polinomiale si usa con *pochi* nodi; $n \leq 10$ per esempio.

Polinomi di Lagrange Possiamo dare una formula esplicita per la soluzione del problema dell'interpolazione polinomiale. Fissati i nodi distinti $x_1, x_2, \ldots, x_n$, definiamo i *polinomi di Lagrange*

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

Ad esempio, se i nodi sono $x_1 = 1, x_2 = 2, x_3 = 4$, abbiamo

$$L_1(x) = \frac{(x-2)(x-4)}{(1-2)(1-4)}, \quad L_2(x) = \frac{(x-1)(x-4)}{(2-1)(2-4)}, \quad L_3(x) = \frac{(x-1)(x-2)}{(4-1)(4-2)}.$$

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

Lemma 6.3. *Si ha*

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

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 $y = e_k$ (k-esimo vettore della base canonica). Con queste soluzioni, possiamo ottenere la soluzione a un problema di interpolazione polinomiale generico.

Teorema 6.4. *Siano $(x_1, y_1), \ldots, (x_n, y_n)$ dati. La soluzione del problema di interpolazione polinomiale (cioè l'unico polinomio p di grado $d < n$ tale che $p(x_i) = y_i$ per ogni $i = 1, 2, \ldots, n$) è data da*

$$p(x) = \sum_{k=1}^{n} y_k L_k(x).$$

Dimostrazione. Basta verificare che $p(x_i) = y_i$ per un i generico (cosa che segue dal Lemma 6.3) e che $p(x)$ ha grado minore o uguale a $n - 1$ (perché è una combinazione lineare dei polinomi di Lagrange, che hanno tutti grado $n-1$). □

Approssimazione di funzioni Con le tecniche viste possiamo approssimare una funzione più complicata, f, con una funzione più semplice, ϕ. Per questo prendiamo nodi distinti $x_1, x_2, \ldots, x_n$, e scegliamo $y_i = f(x_i)$. Questo produce una funzione più semplice ϕ (retta, polinomio di grado basso, polinomio trigonometrico...) che interseca il grafico di f nei punti x_i dati.

[disegnare un esempio]

Non è un problema nuovo, ma solo un modo comune di usare il problema dell'interpolazione/approssimazione.

Resto dell'interpolazione Vale il seguente risultato.

Teorema 6.5. *Sia $f \in \mathcal{C}^{n+1}([a, b])$, $x_1, x_2, \ldots, x_n$ nodi distinti in $[a, b]$, e $p(x)$ il polinomio di interpolazione (di grado al più $n - 1$) di f sui nodi dati. Allora per ogni $x \in [a, b]$ esiste un punto $\xi \in (a, b)$ tale che*

$$f(x) - p(x) = \frac{f^{(n)}(\xi)}{n!}(x - x_1)(x - x_2)\ldots(x - x_n).$$

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 $n - 1$) più un resto che dipende dalla derivata n-esima. Nella forma $\frac{f^{(n)}(\xi)}{n!}x^n$ del resto dello sviluppo di Taylor, abbiamo rimpiazzato x^n, o $(x - x_0)^n$, con $\prod(x - x_i)^n$.

Non ci stupisce che ci siano dei fattori $(x - x_i)$: difatti, se sostituiamo $x = x_i$, per un qualche $i = 1, 2, \ldots, n$, 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 $n-1$. 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 $n-1$ abbiamo $f^{(n)} \equiv 0$.

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

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

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

6.3 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 $I \approx I_M = (b-a)f(c)$, 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 sulla lavagna) 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

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

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 $x - b$. Scegliere i segni in questo modo è abbastanza naturale perché $(x-a)(b-x) \geq 0$ 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 $x_0 = a, x_1 = a+h, \dots x_n = b$, 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 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 $x_0, \dots, x_n$ diventano $n+1$ punti equispaziati $y_0, \dots, y_n$ su $[-1,1]$, e il polinomio di Lagrange $L_k(x)$ costruito sui punti $x_0, \dots, x_n$ diventa il polinomio di Lagrange $L_k(y)$ costruito sui punti $y_0, \dots, y_n$. 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 utilizzando il fatto che sia gli integrali che le nostre formule di integrazione sono lineari nella f è possibile dimostrare che questo è sufficiente per dimostrare 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*

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

Inoltre, per l'errore vale la formula

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

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).

6.4 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

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

Metodo dei trapezi composito

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

Metodo di Cavalieri–Simpson composito

$$\begin{aligned} I_2 &= \sum_{n=1}^{N} \frac{x_n - x_{n-1}}{6} \left(f(x_{n-1}) + 4f\left(\frac{x_{n-1} + x_n}{2}\right) + f(x_n)\right) \\ &= \frac{b-a}{6n} \sum_{n=1}^{N} \left(f(x_{n-1}) + 4f\left(\frac{x_{n-1} + x_n}{2}\right) + f(x_n)\right) \\ &= \frac{2}{3} I_M + \frac{1}{3} I_T. \end{aligned}$$

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 come

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

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* $|\tilde{I} - I| \leq \nu_n C_{m+1}(b-a)^{m+2}$*, allora per la sua versione composita con* N *sottointervalli vale la stima*

$$|\tilde{I}_c - I| \leq \nu_n C_m \frac{(b-a)^{m+2}}{N^{m+1}} = \nu_n C_{m+1}(b-a)h^{m+1}.$$

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

$$|\tilde{I}_c - I| \leq \sum_{n=1}^{N} \left| \tilde{I}_n - \int_{x_{n-1}}^{x_n} f(x)dx \right| \leq \sum_{n=1}^{N} \nu_n C_{m+1} h^{m+2} = N\nu_n C_{m+1} \frac{(b-a)^{m+2}}{N^{m+2}}.$$

In particolare, per la formula dei trapezi composita abbiamo

$$|I_{Tc} - I| \leq \frac{(b-a)^3}{12N^2} C_2 = O(\frac{1}{N^2});$$

raddoppiando il valore di N ci aspettiamo una decrescita dell'errore di un fattore $2^2 = 4$. Stessa cosa per la formula del punto medio composita, mentre per la formula di Cavalieri–Simpson composita ci aspettiamo una decrescita dell'errore di un fattore 2^4. □

Stima dell'errore per le formule composite Sia N un intero pari. Applicando una formula di integrazione composita con grado di esattezza m con N intervalli e con $N/2$ sottointervalli, ci aspettiamo quindi che, per un certo valore dell'errore E_N,

$$I_N = I + E_N, \quad I_{N/2} = I + E_{N/2} \approx I + 2^{m+1} E_N.$$

In particolare da questa relazione possiamo ricavare approssimativamente il valore di E_N come

$$\frac{I_{N/2} - I_N}{2^{m+1} - 1} \approx \frac{I + 2^{m+1}E_N - (I + E_N)}{2^{m+1} - 1} = E_N.$$

Si noti che per le formule dei trapezi e di Cavalieri–Simpson (ma non per altri 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.

6.5 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

$$\sum_{k=1}^{n} w_k 1 = 2, \quad \sum_{k=1}^{n} w_k x_k = 0, \quad \sum_{k=1}^{n} w_k x_k^2 = \frac{2}{3}, \dots$$

Queste equazioni (nelle incognite $w_1, x_1, \dots, w_n, x_n$) non sono lineari, quindi non è chiaro che siano risolubili, ma nella pratica risulta essere così: esiste una scelta di $x_1, \dots, x_n, w_1, \dots, w_n$ formula con grado di precisione $2m-1$. Queste 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

$$x_1 = \frac{1}{\sqrt{3}} = 0.57735\dots, \quad x_2 = -\frac{1}{\sqrt{3}} = -0.57735\dots, w_1 = w_2 = 1$$

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*)

$$\begin{cases} y'(t) = \lambda y, \\ y(0) = y_1, \end{cases} \tag{7.2}$$

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:

di nuovo, introduciamo la notazione

$$h = \frac{b-a}{N}, \quad t_n = a + nh, \quad n = 0, 1, 2, \ldots, N.$$

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

$$y(t_{n+1}) = y(t_n) + y'(t_n)h + \frac{1}{2}y''(\xi)h^2 = y(t_n) + f(t_n, y_n)h + \frac{1}{2}y''(\xi)h^2.$$

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*

$$y_{n+1} = y_n + hf_n, \quad n = 0, 1, 2, \ldots, N-1, \tag{7.3}$$

dove abbiamo posto $f_n = f(t_n, y_n)$ per brevità.

Il costo computazionale è di $N + 1$ valutazioni di f, più $O(mN)$ operazioni aritmetiche.

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

$$y_{n+1} = y_n + h \underbrace{f(t_{n+1}, y_{n+1})}_{f_{n+1}}. \tag{7.4}$$

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 (7.4) 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. In alcuni casi però esistono altre strategie che permettono di ottenere direttamente una soluzione. Per esempio, per il problema test (7.2) si ha

$$y_{n+1} = y_n + h\lambda y_{n+1},$$

che possiamo risolvere direttamente in quanto è lineare, come

$$y_{n+1} = \frac{1}{1 - h\lambda} y_n.$$

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 (7.4) (in qualche modo) ad ogni passo per $n = 0, 1, 2, \ldots, N-1$. Il costo computazionale dipende dal modo in cui risolviamo la (7.4).

Metodo dei trapezi È il metodo

$$y_{n+1} = y_n + h\left(\frac{1}{2} f_n + \frac{1}{2} f_{n+1}\right).$$

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 il metodo dei trapezi.

Metodi generali a un passo In generale possiamo scrivere un metodo a un passo come

$$y_{n+1} = y_n + h\Phi(t_n, y_n).$$

(Anche se nell'espressione compaiono t_{n+1}, y_{n+1}, ecc., possiamo comunque vederla come una funzione di quei due argomenti, dipendente anche implicitamente da h e f.)

Vogliamo dare un risultato generale di convergenza per questi metodi. Per questo ci serve introdurre alcune definizioni. Definiamo l'*errore globale* come

$$E = \max_{n=1,2,\ldots,N} |e_n|, \quad e_n = y(t_n) - y_n, \tag{7.5}$$

e l'*errore locale di troncamento* come

$$T = \max_{n=0,1,\ldots,N-1} |\tau_n|, \quad \tau_n = \frac{y(t_{n+1}) - y(t_n)}{h} - \Phi(t_n, y(t_n)),$$

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 discretizzato a partire dal valore esatto $y(t_n)$. Per un intero $p > 0$, un metodo si dice *consistente di ordine* p se $T = O(h^p)$ nel limite quando $N \to \infty$ (e quindi $h \to 0$); e si dice *convergente di ordine* p se $E = O(h^p)$.

Vale il seguente risultato (che non dimostriamo).

Teorema 7.1. *Supponiamo che la funzione $\Phi(t, y)$ sia continua per $t \in [a, b]$, e che esista una costante L per cui (indipendentemente da h)*

$$\|\Phi(t, y_1) - \Phi(t, y_2)\| \leq L\|y_1 - y_2\| \tag{7.6}$$

Allora, un metodo è convergente di ordine p se e solo se è consistente di ordine p.

Qualche commento: la (7.6) si chiama *condizione di Lipschitz*, o *Lipschitzianità nella y*, ed è molto simile alla condizione $\|f(t,y_1) - f(t,y_2)\| \leq L\|y_1 - y_2\|$ che compare nel teorema di esistenza e unicità della soluzione per le equazioni differenziali, che avete probabilmente visto ad analisi. Per una funzione Φ differenziabile, la (7.6) vale se esiste finito il massimo

$$L = \max_{t,y} \left\| \frac{\partial \Phi(t,y)}{\partial y} \right\|.$$

Tipicamente queste condizioni di Lipschitzianità (sulla f e sulla Φ) sono soddisfatte per le equazioni differenziali e per i metodi che vedremo.

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

$$\frac{y(t_{n+1}) - y(t_n)}{h} - f(t_n, y(t_n)) = \frac{1}{2}y''(\xi)h = O(h).$$

Stessa cosa per il metodo di Eulero implicito. Invece il metodo dei trapezi è consistente di ordine 2, come si può dimostrare con uno sviluppo di Taylor nel punto medio $\frac{y(t_n)+y(t_{n+1})}{2}$.

Metodi di Runge–Kutta I metodi di Runge–Kutta sono una classe di metodi a un passo (quindi definiti da una $\Phi(t,y)$ come sopra) più generale dei metodi visti, che permette di raggiungere ordini maggiori. In generale 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

$$\begin{array}{cccccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ & b_1 & b_2 & \dots & b_s \end{array}.$$

L'intero s è detto *numero di stadi* del metodo. Per trasformare i coefficienti di questa tabella in una funzione Φ, definiamo

$$k_i = f\left(t_n + c_i h, y_n + h\left(\sum_{j=1}^{s} a_{ij}k_j\right)\right), \qquad i = 1, 2, \dots, s,$$

$$\Phi(t_n, y_n) = \sum_{j=1}^{s} b_j k_j.$$

Ad esempio, per il metodo di Eulero esplicito abbiamo $s = 1$ e tavola di Butcher

$$\begin{array}{cc} 0 & 0 \\ & 1' \end{array}$$

per il metodo di Eulero implicito

$$\begin{array}{cc|c} 1 & 1 & \\ \hline & & 1' \end{array}$$

e per il metodo dei trapezi $s = 2$ (visto che abbiamo due valutazioni di funzione) e

$$\begin{array}{ccc|c} 0 & 0 & 0 & \\ 1 & \frac{1}{2} & \frac{1}{2} & \\ \hline & \frac{1}{2} & \frac{1}{2} & . \end{array}$$

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 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 ovviamente 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

$$\begin{array}{c|ccccccc} 0 & & & & & & & \\ 1/5 & 1/5 & & & & & & \\ 3/10 & 3/40 & 9/40 & & & & & \\ 4/5 & 44/45 & -56/15 & 32/9 & & & & \\ 8/9 & 19372/6561 & -25360/2187 & 64448/6561 & -212/729 & & & \\ 1 & 9017/3168 & -355/33 & 46732/5247 & 49/176 & -5103/18656 & & \\ \hline 1 & 35/384 & 0 & 500/1113 & 125/192 & -2187/6784 & 11/84 & \end{array} ,$$

che ha 7 stadi e ordine $p = 5$ (*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). Applicando il metodo di Eulero esplicito a questo problema, si ottiene $y_{n+1} = y_n(1+h\lambda)$, e quindi tenendo conto della condizione iniziale $y_0 = 0$ otteniamo la formula generale $y_n = (1 + h\lambda)^n$. Se $\lambda < -\frac{2}{h}$, la soluzione esatta del problema $y(t) = e^{\lambda t}$ è un esponenziale con base negativa, quindi converge a 0 monotonamente; però $|1 + h\lambda| > 1$ quindi il metodo produce iterate y_n che diventano sempre più grandi (con segni alterni). Quindi non possiamo scegliere 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 potremmo avere 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 piccolo in valore assoluto $e^{\lambda_3 t}$, ma il passo che possiamo scegliere dipende dall'esponente più grande in valore assoluto λ_1. E diventano ancora più complicati per problemi non lineari, in cui 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, o funzioni del tipo $\Phi(t, y) = Ay$ con A mal condizionata. A differenza del mal condizionamento, non c'è un numero che possiamo associare a questo fenomeno.

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 in 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 il metodo 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 converge a zero se e solo se $|R(q)| < 1$. Si definisce *regione di (assoluta) stabilità* del metodo l'insieme $S_A = \{z \in \mathbb{C} : |R(q) < 1|\}$. Quindi y_n converge a zero (per il problema test) se e solo se h è scelto $h\lambda \in S_A$. 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 $\{z \in \mathbb{C} : \mathrm{Re}(q) < 0\}$. 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. Ma questo è meno importante.

Il risultato principale (che non dimostriamo) è il seguente:

Teorema 7.2. *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 (come abbiamo ricordato) 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'intero 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

$$y_{n+2} = y_{n+1} + h(\frac{3}{2} f_{n+1} - \frac{1}{2} f_n) \tag{7.7}$$

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 (7.7) 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

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j}, \quad n = 0, 1, 2, \dots, \tag{7.8}$$

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 (7.8) 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 (7.8), 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 (7.7)), espliciti, e i *metodi di Adams–Moulton* e le *Backward differentiation formulas (BDF)*, entrambi impliciti. Non vediamo i dettagli; sostanzialmente, con tutte queste famiglie possiamo costruire metodi di ordine maggiore aumentando il numero di passi.

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 $y(t_{n+j})$ differisce dalla quantità y_{n+j} 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 (7.5) (definito esattamente come nei metodi a un passo) è $O(h^p)$; qui nessuna nuova definizione.

Per dimostrare che questi due concetti sono equivalenti, ci serve un'ipotesi aggiuntiva (proprio come nei metodi a un passo ci serviva la Lipschitzianità della Φ).

Zero-stabilità 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

$$\alpha_0 y_n + \alpha_1 y_{n+1} + \cdots + \alpha_k y_{n+k} = 0. \tag{7.9}$$

Relazioni della forma (7.9) 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 (7.9) che è molto simile a quella per calcolare le soluzioni delle equazioni differenziali lineari a coefficienti costanti. L'idea base è che le soluzioni sono combinazioni lineari di soluzioni del tipo z^n, dove z è una soluzione del polinomio associato

$$p_1(z) = \alpha_0 + \alpha_1 z + \cdots + \alpha_k z^k.$$

Questo polinomio si chiama *primo polinomio caratteristico* del metodo a più passi (7.8). Enunciamo però solo un risultato di nostro interesse.

Teorema 7.3. *Ogni soluzione y_n dell'equazione* (7.9) *è limitata (cioè* $\sup_{n\in\mathbb{N}}|y_n|$ *è finito) se e solo se valgono le seguenti due condizioni:*

- *Tutte le radici di $p_1(z)$ soddisfano $|z| \leq 1$;*
- *Tutte le radici z tali che $|z| = 1$ sono semplici.*

La combinazione di queste due proprietà si chiama *condizione delle radici* (root condition).

Possiamo quindi passare a enunciare questa condizione come una proprietà del metodo a più passi.

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 7.4. *Consideriamo un metodo lineare a più passi* (7.8)*, 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 (7.8) 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 7.5. *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.

[2]Queste regioni di stabilità sono raffigurate per esempio su `https://en.wikipedia.org/wiki/Backward_differentiation_formula`

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 `[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 modifica la lunghezza del passo di integrazione h ad ogni passo, 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 cambiare l'accuratezza con il comando
`ode45(f, [a,b], y0, 'AbsTol', 1e-6, odeset('AbsTol', 1e-6, 'RelTol', 1e-3))`,
che modifica la massima tolleranza (relativa e/o assoluta) accettata sulla soluzione. Notare l'uso di `odeset` che permette di specificare opzioni aggiuntive.

Nonostante questo miglioramento, essendo un metodo esplicito `ode45` 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.

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, ma utilizzando un metodo implicito 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 i 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.

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; se ricordate questi metodi (visti all'inizio del corso) utilizzano lo Jacobiano al loro interno. 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)`.

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')`).
- Poche speranze con metodi "semplici" come Eulero esplicito o implicito.